University of Copenhagen

New Ideas on Labeling Schemes

Author:
Noy Rotbart

Supervisors:
Prof. Jakob Grue Simonsen and Prof. Christian Wulff-Nilsen

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Department of Computer Science

October 18, 2016

UNIVERSITY OF COPENHAGEN

Abstract

Faculty of Sciences
Department of Computer Science

Doctor of Philosophy

New Ideas on Labeling Schemes

by Noy Rotbart

With ever increasing size of graphs, many distributed graph systems emerged to store, preprocess and analyze them. While such systems ease up congestion on servers, they incur certain penalties compared to centralized data structure. First, the total storage required to store a graph in a distributed fashion increases. Second, attempting to answer queries on vertices of a graph stored in a distributed fashion can be significantly more complicated.

In order to lay theoretical foundations to the first penalty mentioned a large body of work concentrated on labeling schemes. A labeling scheme is a method of distributing the information about the structure of a graph among its vertices by assigning short labels, such that a selected function on vertices can be computed using only their labels. Using labeling schemes, specific queries can be determined using little communication and good running times, effectively eliminating the second penalty mentioned.

We continue this theoretical study in several ways. First, we dedicate a large part of the thesis to the graph family of trees, for which we provide an overview of labeling schemes supporting several important functions such as ancestry, routing and especially adjacency. The survey is complemented by novel contributions to this study, among which are the first asymptotically optimal adjacency labeling scheme for bounded degree trees, improved bounds on ancestry labeling schemes, dynamic multifunctional labeling schemes and an experimental evaluation of fully dynamic labeling schemes.

Due to a connection between adjacency labeling schemes and the graph theoretical study of induced universal graphs, we study these in depth and show novel results for bounded degree graphs and power-law graphs. We also survey and make progress on the related implicit representation conjecture. Finally, we extend the concept of labeling schemes to allow for a better understanding of the space cost incurred by information dissemination.

Acknowledgements

This thesis is dedicated to my son Noah Poralla Rotbart, who was born halfway through its writing, and who learned to walk faster than I understood Noga Alon's new result.

I would like to thank both my supervisors Jakob Grue Simonsen and Christian Wulff-Nilsen. It was only in my last paper that I finally got to have a fruitful work with both (along with Casper Petersen), but they were each supportive, fun to talk to and incredibly smart. My first publication was the result of a long lasting friendship with David Adjiashvili. I can't start describing what it meant to me, and I'm very grateful for his support. My second publication was born in the office I shared with Marcos Vaz Zalles, and a follow up Masters student thesis with Iasonas Zotos. Our conversations were not only amusing, but taught me a lesson about the huge gap existing between industry and research, in particular in algorithms. Follow up conversations led to the idea behind my third publication, and upon moving to an office with Soren Dalgaard, it was only appropriate that we will work on it together. Soren's part in our paper was a difficult one, and when discussing the matter with Mathias Knudsen, a solution was quickly found. The latter proved much more dominant in the follow up paper. I am very pleased to have had papers with these two.

I would like to thank Pierre Fraigniaud, Amos Korman and Adi Rozen for good talks and hosting when I was in Paris, and Viktor Zamarev and Vadim Lozin for a nice hosting in Warwick, Mike and Fran Fellows for inviting me to their cool conferences, and Oren Weimann and Ofer Freedman for a nice week in Haifa.

Patrick Hagge Cording and Brian Brost kept me in good shape throughout this period, and I thank them for that and for being great friends. I had a great mentoring from Julia Lawall, and to many extents she can be considered a co-supervisor of this thesis. I would also like to thank Benny Chor for keeping my spirit up with some top notch Hummus.

I would like to thank my parents Yaron and Carmia, as well as my brothers Nativ and Tzlil for their love and support. My last thanks is for my girlfriend and the mother of my child Henrike Poralla, who keeps me both grounded and inspired.

Contents

List of Tables

Chapter 1

Introduction

The Internet is a large geographically dispersed network, and information about its structure is needed by all its participants. Decentralizing the structural information is a key part in the scalability and performance thereof.

Labeling schemes are methods of distributing the structure of a graph among its vertices, such that certain queries can be answered in a short time. Moreover, the information needed to answer the queries should be contained in the labels of the questioned vertices themselves. On the one hand, storing the entire data structure at each vertex would result in short answer time for each query. On the other hand, such replications would defy the aim of distributing information. Therefore, the primary indicator of the quality of a labeling scheme is the size of the labels it produces in the worst-case scenario.

Labeling schemes are tailored for specific types of queries. *Adjacency* labeling schemes were the first to be investigated in the literature. The vertices in a graph are labeled in such a way that the *Adjacency* between two vertices can be determined directly from their labels. A restricted form of *Adjacency* labeling schemes for graphs was studied almost 50 years ago by Brauer [1]. The more general concept was defined 25 years later by Kannan, Naor and Rudich [2] as well as by Muller [3]. Following that, the idea laid 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*. This observation revived interest in labeling schemes and triggered an abundance of subsequent publications, including many variants of the concept. Further details on the historical development of labeling schemes can be found in [4].

We investigate labeling schemes for the restricted graph families, and most notably, the family of trees with at most n vertices. This particular family is chosen for the following reasons: When graphs are considered, labeling schemes for all of the functions we study require a label size that is in the order of the size of the graph itself. Even the most basic query, *Adjacency*, labeling schemes requires at least $n/2$ bits to support the family of graphs [5]. It is thus not surprising that the focus of the body of work surveyed, and the early definitions of labeling schemes, require labels of size poly-logarithmic in the size of the graph. On the other hand, techniques introduced in the papers surveyed contributed directly to labeling schemes for several families of graphs. Among others, bounded tree-width [6], bounded degree [7], bounded arboricity [8], and planar graphs [9] labeling schemes for various functions are obtained directly from their corresponding labeling schemes for trees.

1.1 An example

Before diving into practical applications and 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 vertex a *label*, which is just some data, such that, given the labels of two vertices, one can determine if the vertices are adjacent or not.

Consider an arbitrary rooted tree with n vertices. Enumerate the vertices in the tree with the numbers 0 through $n-1$ as binary strings, and let, for each vertex v, $\mathrm{Id}(v)$ be the number associated with v and $p(v)$ be the parent of v in T. The encoding of the labels is demonstrated in Figure 1.1. Given the labels $\mathcal{L}(v) = (\mathrm{Id}(v), \mathrm{Id}(p(v))$ and $\mathcal{L}(u) = (\mathrm{Id}(u), \mathrm{Id}(P(u))$ for two vertices v and u, they are adjacent if and only if either $\mathrm{Id}(p(v)) = \mathrm{Id}(u)$ or $\mathrm{Id}(p(u)) = \mathrm{Id}(v)$ but not both, so that the root is not adjacent to itself.

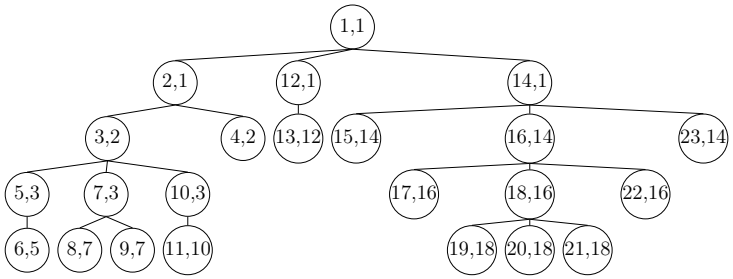


FIGURE 1.1: A tree with $n = 23$ vertices. Each vertex is assigned a label of size $2 \log n$ supporting *Adjacency* queries. A vertex v with parent u is labeled $(\mathrm{Id}(v), \mathrm{Id}(u))$ and the root r is labeled $(\mathrm{Id}(r), \mathrm{Id}(r))$.

This is a labeling scheme. It consists of an *encoder* algorithm that labels the vertices 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.

Recall that the quality of a labeling scheme is measured by the size of the label it produces in the worst case. In the above example, the pair $(\mathrm{Id}(u), \mathrm{Id}(v))$ can be represented as a binary string of length $2\lceil \log n \rceil$ [1]. The goal is to find an *optimal* labeling scheme: that is, one where the worst-case label size is as small as theoretically possible.

There are two main characteristics of a labeling scheme. First, there is the type of query for which the labeling scheme has been constructed. In our example, the type of query is "*Adjacency*", but it could also have been something else, for example "*Ancestry*" (is one vertex an ancestor of the other?) or "*Distance*" (what is the distance between two vertices?). 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

[1] From hereon, unless stated otherwise, $\log n$ stands for $\log_2 n$, and n is assumed to be a power of 2.

choice of graph family may lead to an entirely different labeling scheme with different properties and a different optimal worst-case label size.

1.2 Discussed functions

The thesis provides some overview of labeling schemes where the maximum (worst-case) label size is as small as possible. The problem of finding such labeling schemes for a graph family $\mathcal{G}$ for a given function f is known as the *f-labeling problem for $\mathcal{G}$*. An overview of applications for labeling scheme is provided in Section 1.4, and formal definitions in Section 1.5.

Let $k \in \mathbb{N}$, and let $G = (V, E) \in Graphs(n)$ a simple graph. We investigate labeling schemes supporting the following functions for any $u, v \in V$:

1. $Adjacency : (V \times V) \to \{false, true\}$ (Chapters 3 to 6)
   $Adjacency_G(u, v) = true$ if and only if u and v are adjacent in G.
2. $Distance : (V \times V) \to \mathbb{N}$ (Chapter 5)
   $Distance_G(u, v)$ returns the length of the shortest path $u \rightsquigarrow v$ in G, which is equal to the number of edges in $u \rightsquigarrow v$ if G in unweighted.

We provide special detail for functions on trees: Let $k \in \mathbb{N}$, and let $T = (V, E) \in Trees(n)$ be a tree rooted in r. We investigate labeling schemes supporting the following functions for any $u, v \in V$:

3. $Siblings : (V \times V) \to \{false, true\}$ (Chapter 8)
   $Siblings_T(u, v) = true$ if and only if the parent of u is the parent of v in T for $u \neq r$ and $v \neq r$. A vertex is its own sibling.
4. $Connectivity : (V \times V) \to \{false, true\}$ (Chapter 8)
   $Connectivity(u, v) = true$ if and only if u and v are in the same component in G.
5. $Ancestry : (V \times V) \to \{false, true\}$ (Chapter 7)
   $Ancestry_T(u, v) = true$ if and only if u is an ancestor of v in T.
6. $Routing : (V \times V) \to \mathbb{N}$ (Chapter 10)
   $Routing_T(u, v)$ returns the port number (Definition 18) leading to the next vertex on the path $u \rightsquigarrow v$ in T.
7. $NCA : (V \times V) \to \{0, 1\}^+$ (Appendix A)
   $NCA_T(u, v)$ returns the label of the first vertex in common for the paths $v \rightsquigarrow r$ and $u \rightsquigarrow r$ in T.

Additional functions for labeling schemes were studied in the literature. Some of these are also defined for the family of edge weighted n vertex trees, i.e. trees where each edge is assigned an integer at most 2^M for some integer M. We denote this family as $WeightedTrees(n, 2^M)$.

8. $Center : (V \times V \times V) \to \{0, 1\}^+$
   $Center_T(u, v, w)$ returns the unique vertex z such that the three paths $z \rightsquigarrow u$, $z \rightsquigarrow v$, and $z \rightsquigarrow w$ in T are edge-disjoint.
9. $MaxFlow : (V \times V) \to \mathbb{N}$
   $MaxFlow_T(u, v)$ returns the smallest edge weight on $u \rightsquigarrow v$ in T.

10. *SepLevel* : $(V \times V) \to \{0,1\}^+$ (Appendix A)
    $SepLevel_T(u,v)$ returns the length of the path $r \rightsquigarrow w$ in T where r is the root and w is $NCA_T(u,v)$.

11. *Small-Distance* : $(V \times V \times \mathbb{N}) \to \mathbb{N}$
    $Small\text{-}Distance_T(u,v,k) = true$ if and only if $Distance_T(u,v) \leq k$ in T, otherwise, it returns ∞.

A full report of known results for the aforementioned functions is found in Table 1.1.

1.3 The structure of the thesis

The work on this thesis begun with a survey for labeling schemes on trees, and parts of it are included therein. The following sections of this thesis belong to this survey: Chapter 2 for an overview of techniques, Chapter 3 for the function *Adjacency*, Chapter 7 includes a survey on *Ancestry*, Chapter 10 for the function *Routing*[2] and Appendix A for the function *NCA*. This thesis includes the following novel contributions:

1. Chapter 4 appeared in ICALP 2014' [7]. It concerns asymptotically optimal labeling schemes for bounded degree outer-plannar graphs and trees. In this work we also introduce *Adjacency* labeling schemes for bounded (non constant) degree graphs. The version presented in the thesis includes the complete proofs, including the ones omitted in [7], along with additional results for planar graphs in Section 4.8.

2. Chapter 5 appeared in ICALP 2016' [10] (and in PODC 2016' [11] as an announcement). In this work we pioneer the study of *Adjacency* labeling schemes for power-law graphs, and show some other results on *Adjacency* labeling scheme for this family as well as a careful analysis of *Adjacency* labeling schemes for sparse graphs. The version in the thesis includes an experimental analysis of our labeling scheme.

3. Chapter 6 is an extract from an upcoming survey on the implicit representation conjecture along with Prof. Vadim Lozin and Dr. Viktor Zamarev from University of Warwick. It contains several novel results regarding the implicit representation conjecture.

4. Chapter 7 contains a result from ICALP 2015' [12] improving the best known *Ancestry* labeling scheme for trees from $\log n + 4 \log\log n$ to $\log n + 2 \log\log n$ using a significantly simpler method.

5. Chapter 8 appeared in ISAAC 2014' [13] (and in DISC 2014' [14] as an announcement). We studied labeling schemes incorporating several the aforementioned labeling schemes[3] as well as extension of these to a dynamic case.

6. Chapter 9 appeared in SEA 2014' [15] . It is an experimental evaluation of a previously studied dynamic labeling schemes with permitted relabeling.

[2] We stress that Chapter 10 includes a repair of the main theorem of the best known upper bound for the function.

[3] We describe such results as *multifunctional* labeling schemes.

7. Chapter 11 consists of (i) a dedicated to a novel concept that generalize the notion of labeling schemes to better understand the cost of information dissemination, and (ii) an overview of open questions.

1.4 Practical and theoretical applications of labeling schemes

Labeling schemes have contributed directly to a number of areas, including XML querying, graph theory, shortest paths in road networks and routing schemes.

XML querying. Extensible Markup Language (XML) documents are a popular and ubiquitous standard for exchanging structured data on the Internet [16]. An XML document can be viewed as a rooted tree in which each vertex corresponds to a semantic element, enclosed by matching beginning 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, certain 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, and has been studied extensively [16–21].

To achieve a good performance for queries on XML documents, it is important that a large part of the document's indexed structure can reside in main memory. Since these structures can be extremely large, every single bit counts. Much of the work in this particular direction focused on seemingly small benefits. For example, both relations used by practitioners, namely, *Adjacency* (illustrated in Figure 1.1) and ancestry had strikingly simple labeling schemes with labels of at most $2 \log n$ bits [2]. In order to identify the XML element specifically the labels of an element in an XML file with at most n elements requires at least $\log n$ bits label size. The effort to reduce the single additive $\log n$ contributed not only to the performance of XML queries but also to the subject of implicit representation of trees.

Graph theory. If the query type supported by a labeling scheme allows the entire graph to be reconstructed from the labels, then the collection of labels can be seen as a *disseminated 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 vertices is determined by consulting the relevant entry in a single entity, and where the index of each vertex 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 structure must be disseminated may lead to one that is larger in total. We explore this overhead further in Section 11.1.

Kannan et al. [2] noticed the tight correlation between *Adjacency* labeling schemes and the induced-universal graphs, a topic researched extensively since the 1960's [5]. For various numbers of families, the goal is to find the smallest number of vertices required for a graph that contains each of the n-vertex graphs of the family as induced subgraphs. We defer further discussion on the nature of this connection to Chapter 6.

Shortest paths in road networks. Another practical aspect of labeling schemes concerns distributed shortest paths on road networks. Gavoille et al. [22] first investigated *Distance* labeling schemes. Abraham et al. [23, 24] modified the labeling schemes to achieve a distributed system to answer reachability and shortest path queries on road networks.

Routing schemes. Perhaps the most practical application of labeling schemes arises from routing schemes. In a large network, information sent from one participant to another must visit other participants in the network according to the network topology. A routing scheme is a description of a system designed to support the operation of transferring packets through the network. According to Peleg [25], labeling schemes assist in the design of "memory-free" routing schemes, which support fast and simple switch architectures by storing little data locally. A significant effort by the community yielded numerous papers dealing with labeling schemes for routing [4, 9, 26–32].

1.5 Preliminaries

In the remainder of the thesis we use the following terms: k is a positive integer, a binary string is a member of the set $\{0,1\}^*$. We denote $\lceil \log_2 n \rceil$ as $\log n$. We denote by $\mathbb{N} = \{1, 2 \dots\}$ the set of natural numbers and by $\mathbb{N}_0 = \{0, 1, 2 \dots\}$ the extended set that includes 0. For $x \in \mathbb{N}$ we denote by $\text{bin}(x)$ its standard binary representation, and $|x|$ as the number of bits in $\text{bin}(x)$. The concatenation of two bit strings a and b is denoted $a \circ b$.

1.6 Graph families related definitions

For a graph G we denote the set of vertices and edges by $V(G)$ and $E(G)$, respectively. For any graph family $\mathcal{F}$, let $\mathcal{F}(n) \subseteq \mathcal{F}$ denote the subfamily containing the graphs of at most n vertices. Unless stated otherwise, the number of vertices in a graph n is assumed to be a power of 2. The family of all graphs is denoted $\mathcal{G}$. The collection of unweighted forests in $\mathcal{G}$ respectively $\mathcal{G}(n)$ is denoted $Forests$ respectively $Forests(n)$. The collection of unweighted trees in $Forests$ respectively $Forests(n)$ is denoted $Trees$ respectively $Trees(n)$. Similarly, the collection of edge weighted trees, with weights in $\{1 \dots 2^M\}$ in $\mathcal{G}$ respectively $\mathcal{G}(n)$ is denoted $WeightedTrees(2^M)$ respectively $WeightedTrees(n, 2^M)$. A *caterpillar* is a tree in which all non-leaf vertices lie on a single path, denoted the *main path*. The collection of unweighted caterpillars in $Trees$ respectively $Trees(n)$ is denoted $Caterpillars$ and $Caterpillars(n)$. The collection of trees with bounded depth δ in $Trees$ respectively $Trees(n)$ is denoted $Trees(\delta)$ respectively $Trees(n, \delta)$. Trees of Bounded degree Δ are marked similarly $Trees(\Delta)$ and $Trees(n, \Delta)$. As a shorthand, trees of bounded degree three are marked $BinaryTrees$ and

$BinaryTrees(n)$. For each of the (unbounded) families described the bounded degree and depth variants are denoted in according. From hereon, unless stated otherwise, we assume trees to be rooted.

1.7 Basic tree related definitions

Let $T = (V, E) \in Trees(n)$ be a tree rooted in r. The number of edges in a tree is always $|E| = |V| - 1$. The vertices of degree 1 other than the root are called *leaves*, and all other vertices are called *internal* vertices.

Let u and v be vertices in tree T. If $(v, u) \in E$ we say that v and u are *neighbours*, and denote The set of all neighbours of v as $N(v)$, and $|N(v)|$ as the degree of v, or $\deg(v)$. A non-root vertex with degree at most 1 is called a *leaf*, and an *inner vertex* is a non-leaf vertex. The parent of v, if v is not the root, is denoted $p(v)$, and v is the *child* of $p(v)$. A *sibling* ($v \neq u$ of u) is a child of $p(u)$. We denote by $v \rightsquigarrow u$ the sequence $\langle v_0, v_1 \dots v_k \rangle$ of vertices such that $v = v_0$, $u = v_k$, and $(v_{i-1}, v_i) \in E$ for $i = 1, 2 \dots k$ between v and u in T. This sequence is typically called the *path* between v and u, and k is called the *length* of the path $v \rightsquigarrow u$. Note that every two vertices in a tree are connected by a unique path. The distance between u and v in T, denoted $Distance(u, v)$, is the number of edges on the path $u \rightsquigarrow v$. The distance from v to the root is called the *depth* of v, denoted $\text{depth}(v)$, and the depth of the tree, $\text{depth}(T)$, is the maximum depth among its vertices. If u is a vertex on the path from the root of a rooted tree to a vertex v, then u is an *ancestor* of v, and v is a *descendant* of u. Note, in particular, that a vertex is its own ancestor, descendant and sibling, but not its own child or parent. A *common ancestor* of two vertices is a vertex that is an ancestor of both vertices, and their *nearest common ancestor* (NCA) is the unique common ancestor with maximum depth. Given a vertex v, the descendants of v form an induced subtree T_v with v as root. Finally, the *size* of v, denoted $\text{size}(v)$, is the number of vertices in T_v.

1.8 Definitions and generalizations of labeling schemes

We present definitions of labeling schemes, approximate labeling schemes and three related definitions.

Definition 1. *Let $f : V(G)^k \to S$ be a k-ary function over vertices in $G \in \mathcal{G}$ into S.*

A label assignment *e_G for $G \in \mathcal{G}$ is a mapping of each $v \in V(G)$ into a bit string $e_G(v) = \mathcal{L}(v)$, called the* label *of v.*

An f-labeling scheme *for $\mathcal{G}$, denoted $\langle e, d \rangle$, consists of the following:*

1. *An* encoder *e which is an algorithm that receives $G \in \mathcal{G}$ as input and computes the label assignment e_G.*
2. *A* decoder *d which is an algorithm that gets any sequence of k labels $\mathcal{L}(v_1) \dots \mathcal{L}(v_k)$ and computes the query $d(\mathcal{L}(v_1) \dots \mathcal{L}(v_k))$. If $d(\mathcal{L}(v_1) \dots \mathcal{L}(v_k)) = f(v_1 \dots v_k)$ we say that d is an* exact *decoder.*

Remark 1.

i. If $\forall G \in \mathcal{G}$, e_G is an injective mapping, i.e. for all distinct $u, v \in V(G)$, $e_G(u) \neq e_G(v)$, we say that the labeling scheme has a unique *encoder.*

ii. The encoding of a vertex depends on the graph to which the vertex belongs, whereas the decoding of a k-tuple of labels is oblivious to the graph from which the labels come.

iii. By Definition 1 labels of unrestricted size can be used to encode the entire graph structure.

Definition 2. *Let $\langle e, d\rangle$ be an f-labeling scheme for $\mathcal{G}(n)$.*

1. *$\langle e, d\rangle$ is a $\rho(n)$* f-labeling scheme *if for any $G \in \mathcal{G}(n)$, the size of any label in the label assignment e_G is bounded by some function $\rho(n)$.*
2. *e is* computed in time $t(n)$ *if for any $G \in \mathcal{G}(n)$, e computes the label assignment e_G in time at most $t(n)$.*
3. *d is* computed in time $t(n)$ *if for any $G \in \mathcal{G}(n)$ and any $v_1 \dots v_k \in G$ the query $d(\mathcal{L}(v_1) \dots \mathcal{L}(v_k))$ is computed in time at most $t(n)$.*
4. *$\langle e, d\rangle$ is an* average ρ f-labeling scheme *if for any $G \in \mathcal{G}(n)$, the sum of lengths of all labels in the label assignment e_G is bounded by $\rho(n) \cdot n$ (See [33]).*

We turn to define labeling schemes that provide an approximate solution.

Definition 3. *Let $f : V(G)^k \to \mathbb{N}$ be a k-ary function over vertices in $G \in \mathcal{G}$ to $\mathbb{N}$. An* R-approximate f*-labeling scheme for $\mathcal{G}$, denoted $\langle e, d\rangle$, is an* f-labeling scheme *with the following property. Consider any graph $G = (V, E) \in \mathcal{G}$ which receives a label assignment e_G from e. Then for any set of vertices $v_1 \dots v_k \in V$ with labels $\mathcal{L}(v_1) \dots \mathcal{L}(v_k)$ the value computed by d satisfies*

$$\frac{1}{R} \cdot d(\mathcal{L}(v_1) \dots \mathcal{L}(v_k)) \leq f(v_1 \dots v_k) \leq R \cdot d(\mathcal{L}(v_1) \dots \mathcal{L}(v_k)).$$

1.8.1 Generalizations

We present definitions of three generalizations to the concept of labeling schemes, namely query, one-sided error and forbidden-set labeling schemes.

Labeling schemes with a query [34] extend the decoder's operation such that given two labels, it may call a third label to determine the query.

Definition 4. *Let f be a function over vertex sets in $G \in \mathcal{G}$ to $\mathbb{N}$. A (polynomial time) query algorithm Q receives the labels $\mathcal{L}(u)$ and $\mathcal{L}(v)$ of vertices u and v in G and outputs the label $\mathcal{L}(w)$ of vertex $w \in G$. An* f-labeling scheme with a query *is an f-labeling scheme (Definition 1) where the decoder may use Q to compute the query.*

For result concerning this generalization see Section 5.7.

Labeling schemes with one-sided error [35] allow the labels to produce incorrect results for boolean functions, as long as they are one-sided.

Definition 5. *Let f be a boolean function over vertex sets in $G \in \mathcal{G}$ to $\mathbb{N}$. A (probabilistic)* one-sided error f-labeling scheme with guarantee p *is an f-labeling scheme (Definition 1) with the following property: Consider any graph $G = (V, E) \in \mathcal{G}$ which receives a label assignment e_G from e. Then for any set of vertices $v_1 \dots v_k \in V$ with labels $\mathcal{L}(v_1) \dots \mathcal{L}(v_k)$*

- *If* $f(v_1 \dots v_k) = true$, *then* $prob(d(\mathcal{L}(v_1) \dots \mathcal{L}(v_k)) = true) \geq p$.
- *If* $f(v_1 \dots v_k) = false$, *then* $d(\mathcal{L}(v_1) \dots \mathcal{L}(v_k)) = false$.

An in depth review of results on this generalization is available in Section 3.4, and additional probabilistic labeling scheme results can be found in Section 5.5. From hereon, unless stated otherwise, encoder and decoder stands for exact encoder and deterministic decoder.

Forbidden-set labeling schemes [36] aim to assign labels which are *robust* to partial network failure. This particularly interesting body of work [37–40] was not researched in the scope of this thesis.

Definition 6. *Let* f *be a function over vertex sets in* $G \in \mathcal{G}$ *to* $\mathbb{N}$, *and let* X *be a subgraph of* G. *A* forbidden-set f-labeling scheme *is an* f*-labeling scheme (Definition 1) where the decoder receives the subgraph* X *and returns* $d(\mathcal{L}(v_1) \dots \mathcal{L}(v_k))$ *where the function is defined over the graph* $G \setminus X$.

1.8.2 A subtle point regarding the size of the graphs

We denote the family of trees with at most n vertices $total(n)$ and the family of trees with exactly n vertices $exact(n)$. In the literature reviewed, some results [2, 16] are defined for $total(n)$, while other [8, 29, 41, 42] are defined for $exact(n)$. Naturally, for $n > 1$, $exact(n) \subset total(n)$. Therefore, upper bounds defined for $total(n)$ trivially hold for $exact(n)$. In contrast, lower bounds on labeling schemes for $exact(n)$ hold for $total(n)$. It may be conjectured that for labeling schemes, the definitions are equivalent, but there exist no such proof in the literature surveyed. The subject is expended in [43]. We were unable to find a labeling scheme or a lower bound for a labeling scheme in which the result provided only holds for either $exact(n)$ or $total(n)$.

1.8.3 Predefined naming

Suppose a tree $T = (V, E)$ has a predefined label assignment of $\log n$ bits from a preset name domain, and denote the product of such an assignment for a vertex $v \in V$ as the *vertex identifier* of v, or simply $\mathrm{Id}(v)$. We can extend most labeling schemes presented so far to support vertex identifiers by modifying their encoder to concatenate the vertex identifier to each label. In the case of *NCA* such an extension is not as straightforward since it should return, for two vertices in the tree, the vertex identifier of a third vertex. Moreover, such an extension was proven to incur an asymptotical penalty for the label size required [44]. We report results on both variants, and denote labeling schemes for *NCA* specifically designed to support vertex identifiers as *Id-NCA*, and those that do not as *Label-NCA*.

Recall that the function *Routing* returns an edge identifier[4]. In a similar manner, if each vertex in a tree $T = (V, E)$ has a predefined (local) label assignment on its edges, it was shown [25] that returning this pre-assigned edge label also incurs an asymptotical penalty on the label size. If such a restriction exist we report the result as *fixed port Routing*, and *designer port Routing* otherwise.

[4] The first edge on the distinct path leading from the first to the second vertex.

TABLE 1.1: Known upper/lower bounds for labeling schemes on trees. Labeling schemes that do not necessarily provide unique labels are marked with *. The families $Trees(n), Forests(n), BinaryTrees(n), Trees(n,\delta)$ and $Trees(n,\Delta)$ are defined in Section 1.5. Results implied, but not specified in the literature are marked with †.

Reference	Variant	Upper bound	Lower bound	Encoder	Decoder
		Adjacency			
[45]	$Trees(n)$	$\log n + O(1)$	$\log n + 1$	$O(n)$	$O(1)$
[46]	$BinaryTrees(n)$	$\log n + O(1)$	$\log n + 1$	$O(n)$	$O(1)$
[47]	$Trees(n,\delta)$	$\log n + 3\log\delta + O(1)$	$\log n + 1$	$O(n)$	$O(1)$
[7]	$Trees(n,\Delta)$	$\log n + O(\log\Delta)$	$\log n + 1$	$O(n\log n)$	$O(\log\log n)$
		Non-Adjacency			
[35]	One sided error*	$2k+1$ with $p = 1 - \frac{1}{2^k}$	$---$	$O(n)$	$O(1)$
		Ancestry			
[12]	$Trees(n)$	$\log n + 2\log\log n + 3$	$\log n + \log\log n$	$O(n)$	$O(1)$
[47]	$Trees(n,\delta)$	$\log n + 2\log\delta + O(1)$		$O(n)$	$O(1)$
[35]	One sided error*	$\log n - \frac{k}{2} + O(\log\log n)$ with $p = \frac{1}{2^k}$	$\log n + \log p -$ $O(1)$	$O(n)$	$O(1)$
		Non-Ancestry			
[35]	One sided error*	$\lceil \log n \rceil$ with $p = \frac{1}{2}$	$\log n + \log p -$ $O(1)$	$O(n)$	$O(1)$
		Center			
[25]	$Trees(n)$, fixed model	$\Theta(\log^2 n)$	$\Theta(\log^2 n)$	$O(n\log n)$	$O(1)^\dagger$
		Connectivity			
[42]	$Forests(n)$	$\log n + \log\log n$	$\log n + \log\log n$	$O(n)$	$O(1)$
[42]	$Forests(n)$*	$\log n$	$\log n$	$O(n)$	$O(1)$
		Distance			
[48]	$Trees(n)$	$\frac{1}{4}\log^2 n + o(\log^2 n)$	$\frac{1}{4}\log^2 n - O(\log n)$	$O(n\log n)$	$O(1)^\dagger$
[22]	$WeightedTrees(n, 2^M)$	$\Theta(M\log n + \log^2 n)$	$\Theta(M\log n + \log^2 n)$	$O(n\log n)$	$O(\log n)$
		Distance, approx. $1 + 1/n$			
[49]	2^M weighted diameter	$\Theta(\log n \cdot \log M)$	$\Theta(\log n \cdot \log M)$	$O(nm)$	$O(1)$
[48]	$Trees(n)$	$O(\log(1/\epsilon)\log n)$	$\Omega(\log(1/\epsilon)\log n)$	$O(n)$	$O(1)$
		MaxFlow			
[50]	$WeightedTrees(n, 2^M)$	$\Theta(M\log n + \log^2 n)$	$\Theta(M\log n + \log^2 n)$	$O(n)$	$O(1)$
		NCA			
[51]	$Trees(n)$, *Label-NCA*	$3\log n$	$1.008\log n$	$O(n)$	$O(1)$
[25]	$Trees(n)$, *Id-NCA*	$O(\log^2 n)$	$\Omega(\log^2 n)$	$O(n\log n)$	$O(1)^\dagger$
		Routing			
[9]	$Trees(n)$, designer port	$(1+o(1))\log n$	$\log n + \log\log n$	$O(n\log n)$	$O(1)$
[26]	$Trees(n)$, fixed port	$\Theta(\log^2 n/\log\log n)$	$\Theta(\log^2 n/\log\log n)$	$O(n)$	$O(1)$
		Siblings			
[52]	$Trees(n)$	$\log n + \log\log n$	$\log n + \log\log n$	$O(n)$	$O(1)$
[42]	$Trees(n)$*	$\log n$	$\log n$	$O(n)$	$O(1)$
		Small-Distance			
[48]	$Trees(n)$,distance k	$\min \log n + O(k\log(\log n/k))$ $O(\log n \cdot \log(k/\log n))$	$\log n + \Omega(k\log(\log n/(k\log k)))$	$O(n)^\dagger$	$O(1)$

Chapter 2

Common Algorithm Techniques

In this section we describe various algorithmic techniques that are either folklore, used in several papers, or ones that originate from papers not covered by the thesis. The purpose of the section is to outline a common toolbox useful for labeling schemes, and to point out the similarity between some of the techniques.

2.1 Binary strings and bit tricks

2.1.1 Number representation

Labels and words can be seen both as integers or as boolean strings. In order to represent any possible number in the range $\{1 \dots n\}$, $\lceil \log n \rceil$ bits are required. Therefore, the number of bits in a binary string $\text{bin}(x)$ is $\lceil \log x \rceil$, and we occasionally denote it by $|\text{bin}(x)|$.

A method used in most of others work as well as our own is the concatenation of meaningful bits. Given two strings α and β, we denote the concatenated string $\alpha \circ \beta$. Both α and β may be extracted from the string $\alpha \circ \beta$ by a naïve *separating* string. A *separating string* is a string of $m = |\alpha| + |\beta|$ bits with $'1'$ in bit number $|\alpha|$ and $'0'$ in the rest. A label with m bits of two or more parts, each of variable size, may be described as a corresponding label of size $2m$, such that each part can be extracted from it using the added separating string. When we are interested in reducing $2m$ further, and have a fixed number of parts c, we may use a label of size $m + c \log m$. Moreover, suppose we have a label containing two parts α and β, then we may extract both parts using an improved separating string with $\log \min(|a|, |b|)$ bits. It follows that any constant number of bits attached to a string does not change its size asymptotically. Most labeling schemes reviewed utilise this property to attach a constant number of bits to be used later by the decoder.

2.1.2 Padding

The following bit-trick allows for parts of a label to consistently have the same size by adding a single bit to the largest resulting label. Suppose we want to represent x where $|\text{bin}(x)| < \log n$, then we can use *exactly* $\log n + 1$ bits to describe x using the bit string $\text{bin}(x) \circ 1 \circ 0^i$ where $i = n + 1 - |\text{bin}(x)|$, and 0^i is the binary string composed of i times 0.

2.1.3 Approximation using $O(\log \log n)$ bits

Let $k \in \{1 \dots n\}$ be an integer, and recall that $\text{bin}(k)$ requires at most $\lceil \log n \rceil$ bits. Using $\lceil \log \log n \rceil$ bits we can represent a number k' such that $\lceil k/2 \rceil <$

$2^{k'} \leq k$ or, alternatively such that $k \leq 2^{k'} < 2k$. We set $k' = k[p]$, where $k[p]$ is the position of the most significant bit set to $'1'$ in $\text{bin}(k)$. Now $\text{bin}(k')$ is interpreted as a 1/2-approximation of k by computing the appropriate binary string $1 \circ 0^{k'-1}$. Similarly, we can produce a 2-approximation of k by the binary string $1 \circ 1^{k'-1}$. Any additional bit stored in k' now increases the accuracy of the approximation by a factor of two. The latter is in particular useful since by storing additional $\lceil \log \log n \rceil$ bits we can represent a number k^* such that $k \leq k^* < \lfloor (1 + 1/\log n)k \rfloor$, respectively $\left\lfloor \frac{\log n}{\log n+1} k \right\rfloor < k^* \leq k$.

2.1.4 Word storing in a string

We conclude this section with the following bit-trick, which is based on the following facts. For any two integers j and k, the value of the bit string $\text{bin}(j) \circ \text{bin}(k)$ is $j \cdot 2^{|\text{bin}(k)|} + k$, and in addition $j < 2^{|\text{bin}(j)|}$.

Lemma 1. *Let w and z be two integers. One can compute an integer $x \in [z, z + 2^{|\text{bin}(w)|})$ such that $\text{bin}(w)$ is a suffix of $\text{bin}(x)$.*

Proof. We compute $d = \left\lfloor z/2^{|\text{bin}(w)|} \right\rfloor$ and denote the difference $m = z - d$. Consider now the number x represented by the bit-string $\text{bin}(d) \circ \text{bin}(w)$. If $w \geq m$ then $x \geq z$ and also $x = z - m + w \leq z + w < z + 2^{|\text{bin}(w)|}$. If $w < m$ then x may be smaller than z. We therefore increase the value of x by $2^{|\text{bin}(w)|}$, represented by $\text{bin}(d+1) \circ \text{bin}(w)$. Now, $x = z - m + 2^{|\text{bin}(w)|} + w > z + w \geq z$ since $m < 2^{|\text{bin}(w)|}$, and also $z - m + 2^{|\text{bin}(w)|} + w < z + 2^{|\text{bin}(w)|}$ since $w < z$. □

2.2 Efficient encodings

2.2.1 A dfs_i traversal

We denote a depth first traversal of a tree as dfs. A substantial number of the results surveyed use a vertex numbering by a particular depth first traversal of a tree. A depth-first traversal of the tree is denoted dfs_i if children of small size[1] are visited before children of larger size. Given a tree $T = (V, E)$ every vertex $v \in V$ receives the number $\text{dfs}_\text{i}(v)$ from dfs_i, and we occasionally refer to it as the dfs_i identifier of v.

2.2.2 Suffix-free codes

A *code* is a set of words, and a code is *suffix-free*[2], if no word in the code is the suffix of another word. As a concrete example consider the following collection of words: $code_0(x) = 1 \circ 0^x$, where 0^x is the binary string composed of x times 0. This suffix-free code is inefficient since the number of bits required to store $code_0(x)$ is $2^{|\text{bin}(x)|} + 1$. We can extended this code to an efficient recursively constructed suffix-free code by $code_{i+1}(x) = \text{bin}(x) \circ code_i(|bin(x)|-1)$ for every $i \geq 0$. For example, $code_0(8) = 100000000$, $code_1(8) = 10001000$, $code_2(8) = 10001110$. In order to store $code_1(x)$ and $code_2(x)$ we use $2\lfloor \log x \rfloor + 2$ and $\lfloor \log x \rfloor + 2\lfloor \log \log x \rfloor + 3$ bits respectively. The method can be applied recursively up to $\log^* x$ times such that the length of $code_i(x)$ is at most $\lfloor \log x \rfloor + \lfloor \log \log x \rfloor + \cdots + O(\log^* x)$[3].

[1] The *size* of a vertex in a tree is the number of its descendants.

[2] suffix-free codes are also known as suffix codes.

[3] $\log^*$ is the number of times log should be iterated before the result is at most 1.

Suffix-free codes are useful for labeling schemes for one important property: a concatenation of suffix codes is by itself a suffix code. For labels constructed by two or more parts of variable sizes, suffix codes allow for an encoding of those parts. It is worth noting that, by Kraft's inequality [53], no uniquely decipherable encoding for boolean string s can enjoy a size of less than $s + \log s$, for a sufficiently large s.

2.2.3 Alphabetic sequences

The following Lemma is useful to assign labels to the vertices of a rooted path that achieves the following two properties: First, vertices of a large size are assigned a small label, and second, the labels maintain a total order among the path's vertices.

Let $<_{\text{lex}}$ denote the lexicographical order of binary strings. A finite sequence (a_i) of nonempty, binary strings $a_i \in \{0,1\}^*$ is *alphabetic* if $a_i <_{\text{lex}} a_j$ for $i < j$.

Lemma 2. *Given a finite sequence (w_i) of positive numbers with $w = \sum_i w_i$, there exists an alphabetic sequence (a_i) with $|a_i| \leq \lfloor \log w - \log w_i \rfloor + 1$ 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 $a_1 = 0$, which satisfies $|a_1| = 1 = \lfloor \log w_1 - \log w_1 \rfloor + 1$. 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 $a_k = 0$. 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 alphabetic sequences $(b_i)_{i<k}$ and $(b_i)_{i>k}$ with $|b_i| \leq \lfloor \log(w/2) - \log w_i \rfloor + 1 = \lfloor \log w - \log w_i \rfloor$ for all $i \neq k$. Now, define a_i for $i < k$ by $a_i = 0 \circ b_i$ and for $i > k$ by $a_i = 1 \circ b_i$. Then (a_i) is an alphabetic sequence with $|a_i| \leq \lfloor \log w - \log w_i \rfloor + 1$ for all i. □

Viewed differently, alphabetic sequence is possible since each number w_j $(1 \leq j \leq i)$ can be represented by a number between $\sum_{i=1}^{j-1} w_i$ and $\sum_{i=1}^{j-1} w_i + 2^{\lfloor \log w_j \rfloor}$ with at least $\lfloor \log w_j \rfloor$ $'0'$ in its least significant bits that can be discarded. As an example, the sequence $(w_5) = 2, 8, 1, 1, 4$ with $w = 16$, may be represented by $0000, 1000, 1010, 1100, 1110$ and in according $(a_i) = 0, 1, 101, 11, 111$. Numbers in (a_i) now have a total order with respect to the order in which they were in (w_i). Given two numbers, a decoder may equalise their size by adding zeros to the number with less digits.

2.3 Tree decompositions

This section is dedicated to recursive tree decomposition techniques that were found useful in the results surveyed. The *heavy-light* and *separator* are two well known techniques, and in this section we expose their similarity. The *splines* decomposition can be seen as an extension and generalisation of *heavy-light* decomposition. The *clustering* decomposition was only used once in the surveyed literature, but bears potential for further use. Some of the techniques create a *path-decomposition* of a tree T, which is a collection of paths in T such that every vertex $v \in T$ is a member of exactly one path.

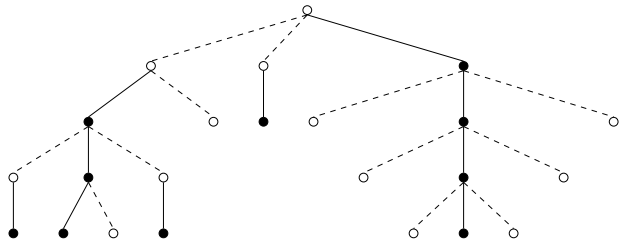


FIGURE 2.1: A tree in which light and heavy vertices have been marked with "○" and "●", respectively, and heavy and light edges have been drawn with solid and dashed lines, respectively.

2.3.1 Heavy-light decomposition

Harel and Tarjan [54] show how to create a path-decomposition of a rooted tree where each vertex of a path, except its top-vertex[4], has maximum size among its siblings. The decomposition allows the location of a vertex in the tree to be described by the sequence of paths that must be traversed in order to reach the vertex from the root.

Let T be a tree with root r. The vertices of T are classified as either *heavy* or *light* as follows. The root is light. For each internal vertex $v \in T$, 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 $\text{hchild}(v)$ and the set of light children of v, if exist, by $\text{lchildren}(v) = v_1 \dots v_k$. The *light size* of a vertex v is the number

$$\text{lsize}(v) = 1 + \sum_{i=1}^{k} \text{size}(v_i).$$

Note that if v is a leaf or has only a single child then $\text{lsize}(v) = 1$, and if v is internal then $\text{lsize}(v) = |v| - |\text{hchild}(v)|$.

An edge connecting a light vertex to its parent is a *light edge*, and edge connecting a heavy vertex to its parent is a *heavy edge*. By removing the light edges, T is divided into a collection of *heavy paths*. The set of vertices on the same heavy path as v is denoted $\text{hpath}(v)$. See Figure 2.1 for an example.

Given a vertex v in T, consider the list of light vertices $u_0 \dots u_k$ encountered on the path from the root r to v. The first of these light vertices is the root, $r = u_0$, and we denote the list $u_0 \dots u_k$ as $\text{lpath}(v)$. The number k is the *light depth* of v, denoted $\text{ldepth}(v)$. The light depth of T, $\text{ldepth}(T)$, is the maximum light depth among the vertices in T. Since the size of every light vertex 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 $\text{ldepth}(v) \leq \lfloor \log n \rfloor$ for all vertices v, where n is the number of vertices in T.

Note that the results surveyed which use this path-decomposition performs also a dfs_i traversal of the tree (see Section 2.2.1).

2.3.2 Separator

Two sets of vertices in a graph are separated if no vertex in one is adjacent to any vertex in the other. A separator is typically defined as a subset of vertices

[4] The top-vertex of a path in a tree T rooted in r is the vertex closest to r.

in a graph G whose removal from G separates the graph to two subsets U and V such that $|U| \leq |V| < 2|U|$ [55]. For all trees there exist a separator that consist of a single vertex.

Theorem 1. *[56] Given a tree T rooted in vertex r, we can find, in linear time, a single vertex whose removal separates T into subtrees of at most $n/2$ vertices each.*

Proof. Consider any vertex $v \in T$. If v does not divide T into components of size at most $n/2$ each, then v has a neighbour u in a subtree of size more than $n/2$. Move into the subtree rooted by u and recurse. This process will never traverse the same edge twice, thus, the separator is found in at most n steps. □

The result holds naturally for forests. Korman and Peleg [57] use Theorem 1 to define a *separator tree* described briefly hereafter. Given a tree $T = (V, E) \in Trees(n)$ we construct a *separator tree* denoted T^{sep} in the following manner. The root r of T^{sep} is the separator, its children are the separators of the subtrees described in Theorem 1 for r. The subtree rooted by each of those children is defined recursively in the same manner. The depth of the corresponding separator tree T^{sep} is $\log n$. A vertex $v \in V$ on the path $r \rightsquigarrow v$ in T^{sep} with depth i $(0 \leq i \leq \log n - 1)$ has now some interesting properties. First, it is a separator for a subtree in T of size at most $2^{\log n - i}$, in which both v and all its children in T^{sep} are present. Second, v is a vertex in all of the subtrees associated with v's ancestors in T^{sep}. For a demonstration, see Figure 2.2.

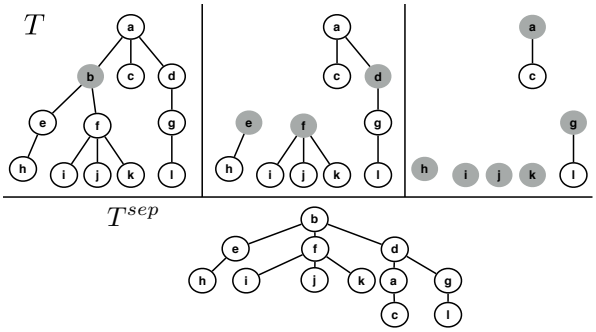


FIGURE 2.2: Separator tree for the tree T, with separator vertices marked in grey. Top row: T rooted at a (left). The forest resulted by removing the separator b from T (center). The remaining forest after removing the separators e, f, d (right). Bottom row: the resulting T^{sep}.

2.3.3 Spines decomposition

This path-decomposition was invented by Thorup and Zwick [9], and used by Fraigniaud and Korman [58]. Similarly to heavy-light decomposition, a *spines decomposition* decomposes an n vertex tree T into a collection of paths, using an additional parameter, an integer $1 \leq b \leq n$. A vertex v is $heavy_s$ if $size(v) \geq n/b$ and $light_s$ otherwise. Note that the heavy vertices in T induce a subtree rooted by the root of T, which is always a $heavy_s$ vertex. T_h is the subtree of T which spans the heavy vertices, and we note that T_h has at

most b leaves. We create a path-decomposition of T_h by removing every edge (u, v) where u has more than one child in T_h. This results in at most $2b - 1$ paths $P_1 \dots P_l$, $(1 \leq l \leq 2b - 1)$ which we denote as $heavy_s$ paths (some of which may consist of a single vertex). When $b = 1$, the single $heavy_s$ path is referred to as the *spine* of T.

The spines decomposition of T is constructed recursively for subtrees in the forest $T \setminus T_h$ by using the above path-decomposition. The removal of T_h from T results in a forest F. Note that a $light_s$ vertex v in T adjacent to a vertex in T_h is a root of a tree $T_v \in F$ of size size$(v) \leq n/b$. The decomposition stops when all vertices in T are $heavy_s$ vertices at some recursive step, and both vertex and path are of *level* i if they occur at step i in the decomposition. Using arguments similar to those for heavy-light decomposition, it follows that within at most $\log_b n$ steps all vertices in T are $heavy_s$ vertices. See Figure 2.3 for a demonstration.

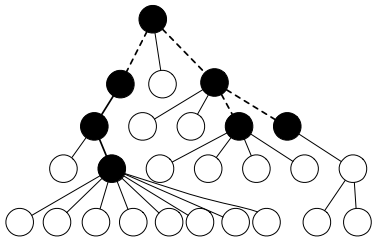


FIGURE 2.3: The first step of a spines decomposition of T with $b = 6$ for a tree with $n = 24$ vertices. Black vertices are $heavy_s$ vertices, and the rest are $light_s$ vertices for the first step. Dotted lines are removed from T_h and the emphasised edges mark the $heavy_s$ paths.

2.3.4 Clustering

The following definitions are used to prove that given a number x between 1 and n, every tree can be decomposed into at most n/x clusters with $O(x)$ vertices each, such that every cluster has at most two vertices in common with other clusters. Moreover, such clusters can be governed by a so-called macro tree.

Definition 7. *Let T be a tree of size $n = |V(T)| > 1$. For a connected subtree C of T, we call a vertex in $V(C)$ incident with a vertex in $V(T) \setminus V(C)$ a* boundary vertex. *The boundary vertices of C are denoted by δC. A* cluster *is a connected subtree of T where $|\delta C| \leq 2$. We denote $C(u, v)$ a cluster with the boundary vertices u and v, where depth$(u) <$ depth(v). If a cluster has only one boundary vertex, we associate its rightmost leaf [5] as the second boundary vertex. Such clusters are called* leaf *clusters. The remaining clusters are called* internal *clusters. A set of clusters $\mathcal{CS}$ is a* cluster partition *of a tree T with root r if and only if $V(T) = \cup_{C \in \mathcal{CS}} V(C)$, $E(T) = \cup_{C \in \mathcal{CS}} E(C)$, and for any distinct $C_1, C_2 \in \mathcal{CS}$, $E(C_1) \cap E(C_2) = \emptyset$, $|E(C_1)| \geq 1$, and $r \in V(C)$ if $r \in \delta C$.*

[5] The rightmost leaf of a tree T is the last leaf encountered in the dfs traversal (see Section 2.2.1) of T.

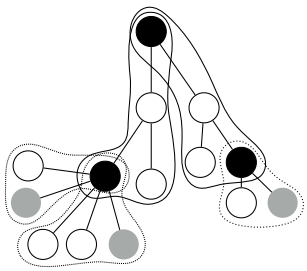


FIGURE 2.4: A demonstration of a nice cluster partition with $n = 14$ and $x = 4$. The dotted clusters are leaf clusters and the rest are internal clusters. The black vertices are boundary vertices, and the grey vertices are the second boundary vertices added.

Definition 8. *Given a tree T rooted in r, $n > 1$, and a parameter $1 \leq x \leq n$. A cluster partition $\mathcal{CS}$ is a* nice cluster partition *if $|\mathcal{CS}| \leq n/x$ and $|V(C)| \leq cx$ for all $C \in \mathcal{CS}$, for some constant c.*

Lemma 3. *For any tree T with n vertices, and any $1 \leq x \leq n$, there exist a nice cluster partition. Moreover, such a partition can be computed in linear time.*

A proof for the lemma can be found in [59] (Lemma 12, Appendix A), and an illustration of the decomposition in Figure 2.4.

To make use of these partitions, the following meta data structure is useful.

Definition 9. *A cluster partition $\mathcal{CS}$ has a* macro tree T_m *defined as follows. The vertices $V(T_m)$ are the set of boundary vertices of $\mathcal{CS}$. The edges $E(T_m)$ are the set of pairs (u, v), where u, v belong to the same cluster $C(u, v)$. The root of T_m is the root r of the original tree T. This is possible since r is defined to be a boundary vertex of some cluster in $\mathcal{CS}$.*

2.4 Boxes and groups

This section presents a single lemma for lower bounds on label sizes. Introduced by Alstrup, Bille and Rauhe [42] and refined by Dahlgaard et al. [13]. The technique uses a division into "boxes and groups" of the set that is to be labeled.

Lemma 4. *Let X be a set with $|X| = nk$, where n is a power of 3 and $k \leq \log_3 n$. Further, let $e: X \to S$ be a function that labels the elements from X with labels from some set S. Assume that we have partitioned X into $k+1$ disjoint subsets of the same size:*

$$X = X_0 \cup \cdots \cup X_k, \quad \text{where } |X_i| = n,$$

and that we have further partitioned X_i into 3^i partitions of size $n/3^i$:

$$X_i = X_{1,i} \cup \cdots \cup X_{n/3^i,i}, \quad \text{where } |X_{ij}| = 3^i.$$

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 $|S| \geq n + (n/3)k$.

Proof. We show the claim by induction on $b \leq k$. For $b = 0$, by Property (i) each of the elements in X_0 must have a distinct label. Assume that the claim holds for $b - 1$. Let X_{prior} be the set of all boxes $X_0 \cup \cdots \cup X_{b-1}$. We show that the number of elements in $X_{prior} \cup X_b$ sharing a label is at most $2n/3$: By Property (i), the labels assigned to X_b are distinct. The number of groups in X_{prior} is $\sum_{i=0}^{b-1} 3^i < 2 \cdot 3^{b-1}$, and the number of groups in X_b is $3 \cdot 3^{b-1}$. From property (ii) it follows that there are at least $(3 - 2) \cdot 3^{b-1}$ groups in X_b, each of size $n/3^b$ which may not share any element with X_{prior}, and thus, box X_b contains at least $3^{b-1}n/3^b = n/3$ items of labels distinct in $X_{prior} \cup X_b$. □

Chapter 3

Introduction to *Adjacency* Labeling Schemes on Trees

In this chapter we survey three key results of *Adjacency* labeling schemes, and start with a literature overview in Section 3.1. In Section 3.2 we summarize the work by Alstrup and Rauhe [8] which was the best known bound at the time of the writing of the thesis. It begins with a simple $\log n + O(\log \log n)$ result for trees, and followed by an improvement to $\log n + O(\log^* n)$. We then show a $\log n + O(1)$ labeling scheme by Gavoille and Labourel [46] for caterpillars in Section 3.3. Lastly, in Section 3.4 we discuss an intriguing one-sided error variant by Fraigniaud and Korman [35] .

3.1 Literature overview

Adjacency labeling schemes were introduced by Breuer [1], and revisited by Kannan et al. [2] for trees and graphs. They were also independently defined by Muller [3] in a model that does not require polynomiality of encoding and decoding. Following the $2 \log n$ *Adjacency* labeling schemes for trees (Section 1.1), Abiteboul, Kaplan, and Milo [16] improved the label size to $1.5 \log n + O(\log \log n)$. Alstrup and Rauhe [8] proved that both $Forests(n)$ and $Trees(n)$ have an *Adjacency* labeling scheme of $\log n + O(\log^* n)$. Fraigniaud and Korman [47] showed that trees with bounded depth δ have a labeling scheme of size $\log n + 3 \log \delta + O(1)$. Gavoille and Labourel [46] proved that caterpillars and binary trees enjoy a labeling scheme of size $\log(n) + O(1)$ using a method called "Traversal and jumping". As we wrote this section, it became clear that bounded degree trees could be a stepping stone for solving the problem. In 2014, the along with David Adjiashvili [7] we showed that trees with bounded degree Δ have a labeling scheme of size $\log n + \log \Delta + O(1)$. We also showed that this bound holds for bounded degree outerplanar graphs, and a result concerning bounded (not necessarily constant) degree. For a description of the result see Chapter 4. Fraigniaud and Korman [35] showed that in a one-sided error labeling scheme, *Non-Adjacency* labeling schemes require $k+1$ bits to have a guarantee of $1 - \frac{1}{2^k}$. In 2015, Alstrup, Dahlgaard and Knudsen [45] showed an asymptotically optimal $\log n + O(1)$ adjacency labeling scheme for the most general case, namely, forests. This chapter was written prior to [45] and plays a historical role as it was available as reading material to its authors[1].

[1] The careful reader will have noticed that the first step in the proof is a better analysis of traversal and jumping, a technique we describe later on.

Adjacency labeling schemes for trees are useful for general graphs due to an observation by Nash-Williams [60], which states that a graph of arboricity[2] d can be decomposed into at most d forests[3]. Kannan et al. [2] proved that graphs with arboricity d can be labeled using $(d+1)\log n$ bit labels. Therefore, the labeling scheme of [45] yields a label size of $d\log n + O(d)$ for graphs with arboricity d. Graphs in $\mathcal{G}(n,\Delta)$ have arboricity of $\lceil \Delta/2 \rceil$ [2]. It follows that graphs in $\mathcal{G}(n,\Delta)$ have a labeling scheme of $(\lceil \Delta/2 \rceil)\log n + O(1)$ bits if Δ is constant. For a description of current labeling schemes results on various graph families see Table 6.1.

3.2 A $\log n + O(\log^* n)$ labeling scheme for $Trees(n)$

For general trees, Alstrup and Rauhe [8] showed a $\log n + O(\log^* n)$ labeling scheme for *Adjacency* with query time of $O(\log^* n)$ and encoding time of $O(n \log^* n)$. In this section we describe their result in detail.

3.2.1 A $\log n + O(\log\log n)$ simple labeling scheme

We first introduce two labeling schemes for $Trees(n)$. The first favours equal size labeling scheme for all vertices, and the other achieves a smaller label size for leaves, at the expanse of bigger label size for internal vertices.

Lemma 5. *[8] The following* Adjacency *labeling schemes are available for* $Trees(n)$. $\langle e_\alpha, d_\alpha \rangle$, *whose worst-case label size is* $\log n + 3\log\log n$ *for all vertices, and* $\langle e_\beta, d_\beta \rangle$, *which has a label size of* $\log n + O(1)$ *for the leaves and* $\log n + 4\log\log n$ *for internal vertices.*

Both labeling schemes are based on a heavy-light decomposition (Section 2.3.1) as well as a dfs_i traversal (Section 2.2.1), in which children of small size are visited before children of larger size. They also use 1/2-approximation (see Section 2.1) of numbers in $\{1 \dots n\}$ for different parts of the labels.

Consider a non-leaf vertex v with parent $p(v)$ and heavy child h. The encoder of $\langle e_\alpha, d_\alpha \rangle$ labels v as a concatenation of the bit strings: *I.)* $\text{dfs}_\text{i}(v)$; *II.)* $\text{ldepth}(v)$; *III.)* a 1/2-approximation of $\text{dfs}_\text{i}(v) - \text{dfs}_\text{i}(p(v))$; and *IV.)* a 1/2-approximation of $\text{dfs}_\text{i}(h) - \text{dfs}_\text{i}(v)$. Leaves are labeled similarly, with the last part set to 0, and the root is marked especially.

Alstrup and Rauhe [8] prove that the information is sufficient to determine adjacency. For completeness we attach a revised proof in Appendix B.1.

We create $\langle e_\beta, d_\beta \rangle$ by extending $\langle e_\alpha, d_\alpha \rangle$ such that each internal vertex receives additional $\log\log n$ bits, describing a 1/2-approximation of the number of its leaf children, which we denote $\lfloor l \rfloor_2$. Suppose that v and u are two leaf children of a vertex with l children, and v is one of the first $\lfloor l \rfloor_2$ leaf children traversed, and u is not. The encoder then sets $\mathcal{L}(v) = (\text{dfs}_\text{i}(v), 0)$ and $\mathcal{L}(u) = (\text{dfs}_\text{i}(u) - \lfloor l \rfloor_2, 1)$. See Figure 3.1 for a demonstration. In conclusion, $\langle e_\beta, d_\beta \rangle$ produces labels of size $\log n + 4\log\log n$ for internal vertices and $\log n + O(1)$ for leaves.

[2] The arboricity of a graph G is the minimum number of edge-disjoint acyclic subgraphs whose union is G.

[3] See also [61] for a simplified proof of the claim.

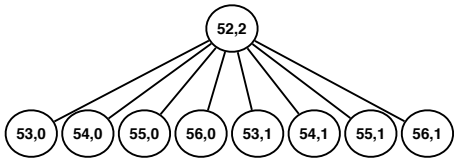


FIGURE 3.1: An internal vertex with (simplified) label (52,2). The second number indicate that its 1/2-approximation is $2^2 = 4$. The children are given labels in $\{53 \dots 56\}$ and an additional bit.

3.2.2 A modified cluster tree

To further reduce the size of the label, the authors use a special and recursive clustering on the tree T. The nice clustering technique (Section 2.3.4) is modified to suit the needs of the labeling scheme. Recall that for every $1 \leq x \leq n$ and any tree T, there exist a (nice) cluster partition dividing T to at most n/x clusters, each having $O(x)$ vertices.

Definition 10. *A cluster $C(u, v)$ (Definition 7) is a* single child cluster *if I.) it is a leaf cluster; or II.) it contains at most two vertices; or III.) $u \rightsquigarrow v$ contains at least 5 vertices, v has no children in $C(u, v)$, and u has only one, i.e., the vertex on the path to v. A nice cluster partition (Definition 8) is a* single child cluster partition *if all its clusters are single child clusters.*

We complete an omitted proof of Lemma 7 in [8].

Lemma 6. *Let T be a rooted tree with n vertices. For every $1 \leq x \leq n/7$ there exist a single child cluster partition of T of at most n/x clusters, each containing $O(x)$ vertices.*

Proof. Let $\mathcal{CS}$ be a nice cluster partition for T with $|\mathcal{CS}| \leq n/7x$ and each cluster contains $O(x)$ vertices. We decompose every internal cluster $C(u, v) \in \mathcal{CS}$ which is not a single child cluster to at most seven single child clusters as described below.

Assume that $u \rightsquigarrow v$ contains 5 or more vertices, and that v has children in the cluster $C(u, v)$. Denote an arbitrary child of u not on $u \rightsquigarrow v$ as u'. $C(u, v)$ is decomposed to the clusters $C'(u, v)$ and $C'(u, u')$, where $C'(u, u')$ is a leaf cluster consisting of the children of u excluding the one in the path $u \rightsquigarrow v$, and $C'(u, v)$ contains the rest of the vertices. We use a similar procedure on a cluster $C(u, v)$ where v has more than one child. It follows that a cluster $C(u, v)$ is decomposed to at most two leaf clusters and one internal cluster $C'(u, v)$ that contains the path $u \rightsquigarrow v$. Hence, all three clusters are single child clusters.

If $C(u, v)$ contains more than two vertices, and $u \rightsquigarrow v$ contains less than five, we decompose the cluster in the following manner (illustrated in Figure 3.2). An internal 2 vertex cluster $C(i, j)$ is created for every two adjacent vertices i and j on the path $u \rightsquigarrow v$. In addition, a leaf cluster $C(a, b)$ is created for every vertex a on $u \rightsquigarrow v$ with at least one child not on the path. Let $\mathcal{F}_d$ be the forest created by removing all edges on the path $u \rightsquigarrow v$ from the cluster $C(u, v)$. Each cluster contains all the vertices in the tree rooted in a from $\mathcal{F}_d$, and $b \neq a$, an arbitrary vertex in $\mathcal{F}_d$. To conclude, $C(u, v)$ is transformed to at most three single child internal clusters, and at most four leaf clusters. □

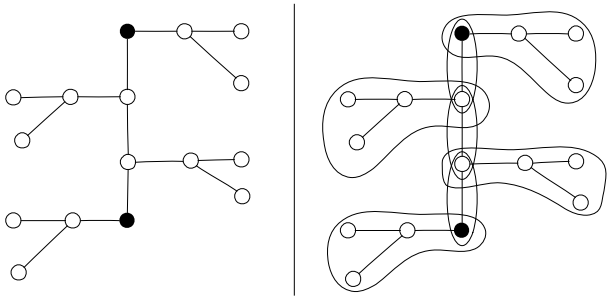


FIGURE 3.2: Left: A nice cluster, Right: the cluster decomposed to 7 single child nice clusters. Black vertices are boundary vertices.

The cluster partitioning guaranteed in Lemma 6 is used to create a correlating *macro tree* as seen in Definition 9.

3.2.3 The final labeling scheme

Theorem 2. *[8] There exist an* Adjacency *labeling scheme for* $Trees(n)$ *whose worst-case label size is at most* $\log n + O(\log^* n)$.

We show a simplified proof of the theorem.

Proof. We begin by labeling the macro tree T_m (Definition 9) of a single child cluster partition of T, using the leaf favouring encoder e_β (defined in Lemma 5). Every edge $(u, v) \in E(T_m)$ represents a *micro tree*, a cluster of at most cx vertices, for some c, not including its boundary vertices. In addition, T_m has at most $n/x + 1$ vertices. We can use e_β to encode the leaves of T_m using $\log(n/x) + O(1)$ bits, and the inner vertex of T_m using $\log(n/x) + 4\log\log(n/x)$. By choosing $x = \log^5 n$ both label sizes are bounded by $\log n + O(1)$. For the remainder of the proof, an edge $(u, v) \in E(T_m)$ where $\text{dfs}_\text{i}(u) < \text{dfs}_\text{i}(v)$ is labeled in v's label, $\mathcal{L}(v)$.

Consider the vertices in the internal cluster $C(a, b) \setminus \{a, b\}$. These vertices maintain adjacency relation only between themselves and possibly the two boundary vertices a and b. By storing only a unique identifier of the edge they belong to in the macro tree, we can safely reject a query of two vertices from different clusters. Such a unique label is already given to every micro-tree labeled $\mathcal{L}(b)$ by $\text{dfs}_\text{i}(b)$, which is just the first $\log(n/x)$ bits of the label given by e_β. We stress that the non-boundary vertices of the internal clusters are exempt from maintaining the additional $4\log\log(n/x)$ bits required by d_β to determine *Adjacency* in T_m. We mark the boundary vertices, and their immediate (single) neighbours from each side by a finite number of types (such as root of an internal cluster, a child of a root of a cluster etc.).

A full label for the tree is achieved by concatenating those labels with the *Adjacency* labels of the micro trees, using the original $\log x + 3\log\log x$ bits labeling scheme for trees of size x. This time the boundary vertices are the one exempt from storing this information. The decoder can detect if two vertices are in the same micro tree using the first part, and check for *Adjacency* if the two vertices are in the same micro part, using the second part of the labels.

To summarize, there are four types of vertices. Vertices in leaf clusters, boundary vertices of the internal clusters, immediate neighbors of the boundary vertices, and vertices in micro trees of internal clusters. All four enjoy a labeling scheme of at most $\log n + 3\log\log\log^5 n + O(1)$.

We utilise the structure of the new micro trees and label those by applying the labeling scheme recursively. The encoder assigns $\mathcal{L}(u)$ to $u \in T$ as a concatenation of at most i decreasing size macro trees that contain u. The last element concatenated to $\mathcal{L}(u)$ marks that u is either be a boundary vertex in some level prior to i, or a full label of u in the last micro tree. Denote $f(n) = log^5(n)$ and $f^i(n)$ operating f $i-1$ times recursively on n. e_β is applied $i-1$ times, and vertices in the (final) micro trees are labeled by the second encoder e_α. The description of the labeling scheme yields labels of size at most:

$$\sum_{j=1}^{i} \left(\log\left(f^{j-1}(n)/f^j(n)\right) + O(1)\right) + 3\log\log\left(f^{i-1}(n)/f^i(n)\right) + O(1) = \log n + \log f^i(n) + i \cdot O(1) + 3\log\log\left(f^{i-1}(n)/f^i(n)\right). \tag{3.1}$$

Assigning $i = \log^* n$ we have that the maximum label size is $\log n + O(\log^* n)$ as guaranteed. □

3.3 Traversal and jumping

Caterpillars were shown to have a labeling scheme of $\log n + O(1)$ by Gavoille and Labourel [46], using a technique called "traversal and jumping". In the remainder of this section we occasionally refer to the labels as the integers they represent.

A caterpillar tree T rooted in x_1 consists of the path of *internal vertices* $P = x_1 \dots x_p$ where each $x_i \in P$ has $l(i)$ *leaf children* $L(i) = y_1^i \dots y_{l(i)}^i$. The only two cases where two vertices are adjacent in a caterpillar is either when one is a leaf child of the other or if they are adjacent internal vertices. We also note that there exist a straightforward non-unique *Adjacency* labeling scheme for this family.

Recall that a $\mathrm{dfs_i}$ traversal (Section 2.2) of a caterpillar T rooted in x_1 visits vertices in $L(i)$ before x_{i+1} for all $1 \le i \le p$. As with the labeling scheme in Section 3.2, the leaves do not contain additional information other than a unique identifier. For every $1 \le i \le p$ and $1 \le j \le l(i)$, we set $\mathcal{L}(y_j^i) = \mathcal{L}(x_i) + j$. In other words, the leaf children of x_i are given labels consecutively after x_i's label, and a single bit is prefixed to each label to determine the vertex's type.

We can construct a $2\log n$ *Adjacency* labeling scheme by assigning $\mathrm{dfs_i}(v)$ to every vertex v, and concatenate $l(i)$ to every internal vertex. An internal vertex u and a leaf child v are adjacent if and only if $\mathrm{dfs_i}(u) \le \mathrm{dfs_i}(v) \le \mathrm{dfs_i}(u) + l(i)$, and two internal vertices u, and v where $\mathrm{dfs_i}(v) > \mathrm{dfs_i}(u)$ are adjacent if and only if $\mathrm{dfs_i}(u) + l(u) + 1 = \mathrm{Id}(v)$.

Rather than storing $l(i)$ in x_i, we maintain a 2-approximation (Section 2.1.3) of both l_i and l_{i+1} denoted $\lceil l_i \rceil_2$ and $\lceil l_{i+1} \rceil_2$, respectively in x_i. For $1 \le i \le p$ we compute

$$t_i := max\{2\log t_{i+1}, \log(\lceil l_i \rceil_2)\} + O(1),$$

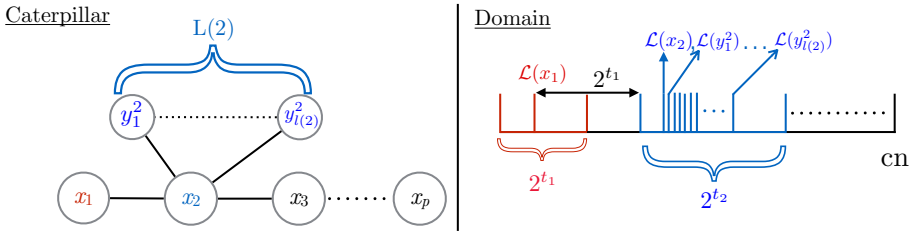


FIGURE 3.3: Label assignment of the inner vertices and leaves of the caterpillar depicted on the left using traversal and jumping. The label $\mathcal{L}(x_1)$ is chosen from $\{1 \dots 2^{t_1}\}$, $\mathcal{L}(x_2)$ from $\{\mathcal{L}(x_1) + 2^{t_1} \dots \mathcal{L}(x_1) + 2^{t_1} + 2^{t_2}\}$ and the leaves $y_1^2 \dots y_{l(2)}^2 \in L_2$ are labeled s.t. $\mathcal{L}(y_1^2) = \mathcal{L}(x_2) + 1 \dots \mathcal{L}(y_{l(2)}^2) = \mathcal{L}(x_2) + l_2$.

where $t_p = \log(\lceil l_p \rceil_2)$. An internal vertex $x_i \in P$ is assigned a range of size 2^{t_i} in which the label of x_i and the labels of $L(i)$ will reside. In order to store both t_i and t_{i+1} in the label of x_i, we first construct the suffix code (Section 2.2.2) $C(x_i)$, which is defined as:

$$C(x_i) = code_1(t_{i+1}) \circ code_0(t_i - |code_1(t_{i+1})|).$$

We then choose the label of x_{i+1} to be the number in the range $\{\mathcal{L}(x_i) + 2^{t_i} \dots \mathcal{L}(x_i) + 2^{t_i} + 2^{t_{i+1}}\}$ that contains $C(x_i)$ as a suffix. We find this particular number using Lemma 1. Due to the selection of t_i, it is guaranteed that there are at least $l(i)$ consecutive numbers in the range after that number. To prove that the size of the label is at most $\log n + O(1)$ the authors prove that for some constant c': $\sum_{i=1}^{p} 2^{t_i} \leq c' \cdot \sum_{i=1}^{p} l(i)$. The avid reader will find the proof for this claim in Appendix B.2. Since $\sum_{i=1}^{p} l(i) < n$, it follows that the maximum label size used by any internal vertex is $\log n + O(1)$. The largest label is assigned to the last leaf $\mathcal{L}(y_{l(p)}^p)$, and since the leaves are assigned numbers consecutively after their parents, the number of labels is at most doubled, requiring only one additional bit. For an illustration of the technique see Figure 3.3.

3.4 One-sided error *Adjacency* labeling scheme

Consider the function *Non-Adjacency* for any graph $G = (V, E)$ that returns true if and only if vertices $u, v \in V$ are *not* adjacent in G. Suppose we demand our query to be precise only when the vertices are in fact adjacent, and allow two non adjacent vertices to be declared adjacent. It is of course helpful to be able to predict and control how often such a "false positive" would occur. [35] demonstrate the usefulness of one-sided error labeling scheme (Definition 5) for *Adjacency* in the following.

Theorem 3. *For any $1 \leq k$ there exist a one-sided error* Non-Adjacency *labeling scheme $\langle e, d \rangle$ for $Trees(n)$ with guarantee $p = 1 - \frac{1}{2^k}$ of size $2(k+1)$. Furthermore, e is computed in $O(n)$, and d is computed in $O(1)$*

Proof. For each vertex $v \in V$ in the tree $T = (V, E)$, e assigns $\mathcal{L}(v) = \langle \mathcal{L}_1(v), \mathcal{L}_2(v) \rangle$ as follows. The root r is assigned $\langle \mathcal{L}_1(r), \mathcal{L}_2(r) \rangle$, where $\mathcal{L}_1(r)$, $\mathcal{L}_2(r)$ are chosen uniformly at random in $\{0 \dots 2^{k+1} - 1\}$. A vertex u with parent v in the tree is assigned $\mathcal{L}_1(u) = \mathcal{L}_2(v)$ and $\mathcal{L}_2(u)$ chosen uniformly

at random in $\{0 \dots 2^{k+1} - 1\}$. e is clearly computed in $O(n)$ with $n+1$ calls to a random function, and may assign non-unique labels.

The decoder d will decode $\langle \mathcal{L}_1(u), \mathcal{L}_2(u) \rangle, \langle \mathcal{L}_1(v), \mathcal{L}_2(v) \rangle$ by evaluating the predicate $(\mathcal{L}_1(v) = \mathcal{L}_2(u)) \vee (\mathcal{L}_1(u) = \mathcal{L}_2(v))$. If $u, v \in V$ are adjacent, the predicate is trivially satisfied. If $u, v \in V$ are non adjacent, $prob(\mathcal{L}_1(v) = \mathcal{L}_2(u)) = prob(\mathcal{L}_2(v) = \mathcal{L}_1(u)) = \frac{1}{2^k+1}$. Therefore, the predicate is satisfied with probability $\frac{1}{2^{k+1}} + (1 - \frac{1}{2^{k+1}}) \cdot \frac{1}{2^{k+1}} < \frac{2}{2^{k+1}} = \frac{1}{2^k}$, and thus $p = 1 - \frac{1}{2^k}$. □

Notice that one bit is sufficient to avoid false determination of two adjacent vertices as non adjacent. The latter theorem shows that for "false positive" guarantee of 99.99%, as few as 30 bits are sufficient. Choosing $k = \log n + O(1)$ ensures a guarantee $p = 1 - \frac{1}{2^{O(1)}\sqrt{n}}$. In contrast Fraigniaud and Korman [35] prove that any one-sided error *Adjacency* labeling scheme for $Trees(n)$ with guarantee p requires labels of size at least $\log n + \log p - O(1)$.

Chapter 4

Adjacency for Bounded degree: Trees, Planar graphs and Graphs

In this chapter we describe an optimal (up to an additive constant) $\log n + O(1)$ size *Adjacency* labeling scheme for bounded-degree outerplanar graphs. A graph is *outerplanar* if it admits a planar embedding with the property that all vertices lie on the unbounded face. graphs with bounded degree Δ. As a special case thereof, we obtain an optimal labeling scheme for bounded degree trees. We summarize this result in the following theorem.

Theorem 4. *For every fixed $\Delta \geq 1$, the class $\mathcal{O}(n, \Delta)$ of bounded-degree outerplanar graphs admits an* Adjacency *labeling scheme of size $\log n + O(1)$, with encoding complexity $O(n \log n)$ and decoding complexity $O(\log \log n)$.*

Our labeling scheme utilizes a novel technique based on *edge*-universal graphs[1] for bounded degree outerplanar graphs. To the best of our knowledge, our labeling scheme is the first to use the total label size to separate the different components of the label. In contrast, other labeling schemes, such as [9, 44], introduce an extra overhead to support such separation.

As a corollary of Theorem 4, we also obtain an efficient $(\lfloor \Delta/2 \rfloor + 1) \log n$ size labeling scheme for graphs with maximum degree Δ. For the case of bounded degree planar graphs we construct a $\frac{\lceil \Delta/2 \rceil + 1}{2} \log n$ size labeling scheme with average label size of $(1 + o(1)) \log n + O(\log \log n)$, improving the best known construction for $\Delta \leq 4$. Finally, we observe that a simple application of combinatorial number systems [65] gives an adjacency labeling scheme for all graphs with maximum degree $\Delta(n)$, improving the known bounds for $\Delta(n) \in [(e+1)\sqrt{n}, n/5]$.

We summarize our contributions in Table 4.1. Our results for outerplanar graphs, planar graphs and general graphs are presented in Section 4.2, Section 4.8 and Section 4.9, respectively.

Our method relies on an embedding technique of Bhatt, Chung, Leighton and Rosenberg [62] for bounded degree outerplanar graphs. In their paper, the authors were concerned with *edge-universal graphs* for various families of bounded degree graphs. In particular, they show that for every $n \in \mathbb{N}$ there exists a graph H_n with $O(n)$ vertices and $O(n)$ edges that contains every bounded degree outerplanar graph $G \in \mathcal{O}(n, \Delta)$ as a subgraph (not necessarily induced).

[1] A graph G is *edge-universal* for a class $\mathcal{R}$ of graphs, if every graph in $\mathcal{R}$ appears as a subgraph in G (not necessarily induced).

TABLE 4.1: Our contribution for families of graphs with bounded degree Δ.

Family	Upper bound	Tight	Encoding	Decoding	Ref.
Trees	$\log n + O(1)$	yes	$O(n \log n)$	$O(\log \log n)$	Cor. 1
Outerplanar	$\log n + O(1)$	yes	$O(n \log n)$	$O(\log \log n)$	Thm. 4
Planar ($\Delta \leq 4$)	$\frac{3}{2} \log n + O(\log \log n)$	no	$O(n \log n)$	$O(\log \log n)$	Thm. 7
Planar	$2 \log n + O(1)$	no	$O(n \log n)$	$O(\log \log n)$	Cor. 2
Graphs, $\Delta(n)$	$\log \binom{n}{\lceil \Delta(n)/2 \rceil} + 2 \log n$	no	$O(n^2)$	$O(\Delta(n) \log n)$	Thm. 8
Graphs	$\lceil \frac{\Delta}{2} \rceil \log n + O(1)$	no	$O(n \log n)$	$O(\log \log n)$	Cor. 3

4.1 Our methods

Our main tool is an embedding technique due to Bhatt et al. [62] of outerplanar graphs into H_n. On the one hand, the embedding is simple to compute. This fact will lead to an efficient $O(n \log n)$ time encoder. On the other hand, the embedding satisfies a useful locality property. This property allows our labels to contain both unique vertex identifiers of the graph H_n and edge identifiers, without exceeding the desired label size $\log n + O(1)$.

To obtain the latter label size via an embedding into H_n we need to overcome several difficulties. Although H_n has a linear number of edges, its maximum degree is $\Omega(\log n)$, thus, unique edge identifiers require $\Omega(\log \log n)$ bits, in general. Since also $|V(H_n)| = \Omega(n)$, it follows that a label cannot contain an arbitrary combination of vertex identifiers in V_n and edge identifiers at the same time, as it would lead to labels with size $\log n + \Omega(\log \log n)$. This difficulty is overcome by exploiting the structure of H_n further and constructing unique vertex identifiers in a particular way that allows reducing the encoding length. This solution creates an additional difficulty of separating the different parts of the label in the decoding phase. This difficulty is overcome by designing careful encoding lengths that minimize the ambiguity, and storing an additional constant amount of information to eliminate it altogether.

4.2 A compact edge-universal graph for bounded-degree outerplanar graphs

We describe next the edge-universal graph $H_n = (V_n, E_n)$ constructed by Bhatt et al. [62] for $\mathcal{O}(n, \Delta)$. We let $k = \min\{s \in \mathbb{N} : 2^s - 1 \geq n\}$ and set $N = 2^k - 1$. The construction uses two constants $c = c(\Delta), g = g(\Delta)$, that depend only on Δ.

The graph H_n is constructed from the complete binary tree T on N vertices as follows. To obtain the vertex set V_n, split every vertex $v \in V(\mathrm{T})$ at level t in T into $\gamma_t = c \log(N/2^t)$ vertices $w_1(v), \cdots, w_{\gamma_t}(v)$. The latter set of vertices is called the *cluster* of v. For $w \in V_n$ we denote by $t(w)$ the *level* of w, that is the level in the binary tree T of the cluster to which w belongs.

The edge set E_n is defined as follows. Two vertices $w_i(v), w_j(u) \in V_n$ are adjacent if and only if the clusters they belong to in T are at distance at most g in T. Note that $w_1 w_2 \in E_n$ implies $|t(w_1) - t(w_2)| \leq g$. This completes the construction of the graph. One can easily check that $|V_n| = O(n)$. $|E_n| = O(n)$ also holds, but we do not use this fact directly. The graph H_n is illustrated in Figure 4.1. Our labeling scheme relies on the following result.

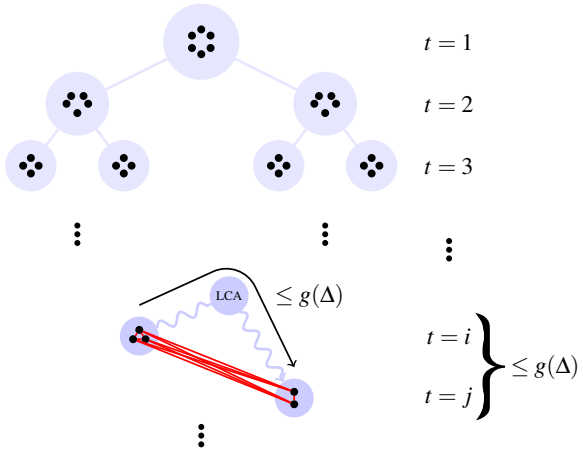


FIGURE 4.1: An illustration of the graph H_n.

Theorem 5. *[62] H_n is edge-universal for the class of bounded degree outerplanar graphs $\mathcal{O}(n, \Delta)$.*

4.3 Warm-up: a $\log n + O(\log \log n)$ Labeling Scheme

We briefly describe a simple $\log n + O(\log \log n)$ labeling scheme. First, assign unique identifiers Id to the vertices in H_n. Since $|V_n| = O(n)$ we can assume that $|\mathrm{Id}(v)| = \log n + O(1)$ for every $v \in V_n$. Next, for every $v \in V_n$ assign unique identifiers to the edges incident to v, Since every vertex in H_n has $O(\log n)$ neighbours, each edge can be encoded using $\log \log n + O(1)$ bits.

To obtain labels for a given outerplanar graph $G = (V, E)$, compute first an embedding $\phi : V \to V_n$ of G into H_n. Next, define the label of vertex $v \in V$ to be the concatenation of $\mathrm{Id}(\phi(v))$ with the identifiers of all edges in E_n leading to images under the embedding ϕ of neighbouring vertices, namely all $\phi(u)\phi(v)$ such that $uv \in E$. Since the maximum degree in G is bounded by the constant Δ, this results in a $\log n + O(\log \log n)$ label size. It is not difficult to see that encoding and decoding can be performed efficiently (we elaborate on it later on).

4.4 The encoder

To reduce the size of the labels to $\log n + O(1)$ we need to refine the latter scheme significantly. As a first step we employ *differential sizing*, a technique first used in the context of labeling schemes by Thorup and Zwick [9]. In differential sizing some parts of the label do not have a fixed number of allocated bits across all labels. Concretely, we use differential sizing for both vertex and edge identifiers.

The resulting labels will have the desired length, but will also contain an undesired ambiguity, that will prohibit correct decoding. We will then append a short prefix to the label that will resolve this ambiguity.

4.4.1 Differential sizing - the suffix of a label

Let us first formally define our naming scheme for vertex and edge identifiers in H_n.

Definition 11. *A* naming *of H_n is an injective function* $\mathrm{Id} : V_n \to \mathbb{N}$ *and a collection of injective functions* $EId_v : E_v \to \mathbb{N}$ *for every* $v \in V_n$*. A naming is* coherent *if for every* $v, v_1, v_2 \in V_n$*:*

1. $\mathrm{Id}(v_1) > \mathrm{Id}(v_2)$ *implies that* $t(v_1) > t(v_2)$*, or* $t(v_1) = t(v_2)$ *and the cluster of* v_2 *appears to the left of the cluster of* v_1 *in* T.

2. $EId_v(vv_1) > EId_v(vv_2)$ *implies that* $\mathrm{Id}(v_1) > \mathrm{Id}(v_2)$.

We compute a coherent naming by assigning the identifiers 1 through $|V_n|$ to V_n level by level, traversing the clusters in any single level in T from left to right, and then naming the edges incident to $v \in V_n$ from 1 to $|E_v|$ in a way that is consistent with the vertex naming. For $v, v' \in V$ define $\alpha(v) := \lceil \log \mathrm{Id}(v) \rceil$ and $\beta_v(vv') := \lceil \log EId_v(vv') \rceil$ and let

$$\alpha_t = \max_{v\,:\,t(v)\leq t} \alpha(v) \quad \text{and} \quad \beta_t = \max_{vv'\,:\,t(v)\leq t} \beta_v(vv')$$

be the maximal number of bits required to encode a vertex name and an edge name for vertices with level at most t. The simple labels described in the beginning of this section store $\log n + O(1)$ and $\log\log n + O(1)$ bits for *every* vertex name, and every edge name, respectively. In contrast, our label for a vertex in level t stores α_t bits for a vertex name, and β_t for edge names. In the following lemma we prove that the new labels have the desired size.

Lemma 7. *For every* $t \leq \log N$ *it holds that* $\alpha_t \leq t + \lceil \log(\log N - t) \rceil + O(1)$*,* $\beta_t \leq \lceil \log(\log N - t) \rceil + O(1)$ *and* $\alpha_t + \Delta\beta_t = \log n + O(1)$.

Proof. We start with the following simple observation stating a couple of facts about the size of different parts of H_n. We let $V_n^t \subset V_n$ and $\Gamma(v) \subset V_n$ denote the set of vertices in H_n with level at most t, and the set of neighbors of $v \in V_n$ in H_n, respectively. This properties will provide bounds on α_t and β_t.

Property 1. *The following properties hold for every level* t *and every vertex* v *in level* t.

i. $|V_n^t| \leq c2^t(\log N - t + 1)$.

ii. $|\Gamma(v)| \leq 6c\Delta^2(\log N - t + g(\Delta))$.

Proof. i. Note that the number of vertices in level i is $c2^{i-1}(\log N - i)$, thus we obtain

$$|V_n^t| = \sum_{i=1}^{t} 2^{i-1}c(\log N - i) \leq c2^t \log N - c\sum_{i=1}^{t} 2^{i-1}i.$$

Using $\sum_{i=1}^{t} 2^{i-1}i = t2^{t+1} - (t+1)2^t + 1 \geq 2^t(t-1)$ and substituting in the expression above we obtain the desired result.

ii. We notice that, by definition of H_n, the set of neighbors of v are exactly all vertices whose cluster is at distance at most $g(\Delta)$ from the cluster of v. It follows that every such cluster has at most $c\log(N/2^{t-g(\Delta)}) = O(\log N - t + g(\Delta))$ vertices. Finally, notice that the number of clusters that are at distance at most $g(\Delta)$ away from a given cluster is at most $3 \cdot 2^{g(\Delta)-1} = 6\Delta^2$. Putting things together we obtain a bound of $6\Delta^2(\log N - t + g(\Delta))$, as desired. □

We can now use property 1 to prove the bounds

$$\alpha_t \leq t + \lceil \log(\log N - t) \rceil + \log c + 2.$$

and

$$\beta_t \leq \lceil \log(\log N - t) \rceil + 2 \log \Delta + \log c + 4.$$

We turn to proving the bound on $\alpha_t + \Delta\beta_t$ for an arbitrary $t \leq \log N$. We drop the additive terms $\log c + 2$ and $2 \log \Delta + \log c + 4$ as they do not depend on n, and hence only contribute a constant additive term.

Substituting the bounds above for α_t and β_t and defining $Q = \log N - t(v)$ we obtain

$$\alpha_t + \Delta\beta_t = t + (\Delta + 1) \log(\log N - t) = \log N - Q + (\Delta + 1) \log \log Q.$$

Using the fact that $(\Delta+1) \log \log Q - Q \leq 0$ whenever $Q \geq (\Delta+1) \log(\Delta+1)$ and $(\Delta + 1) \log \log Q - Q \leq (\Delta + 1) \log \log(\Delta + 1)$ whenever $Q < (\Delta + 1) \log(\Delta+1)$, we have $\alpha_t + \Delta\beta_t \leq \log N + (\Delta+1) \log \log(\Delta+1) = \log n + O(1)$. This concludes the proof of the lemma.

□

We henceforth denote the part of the label containing the vertex name and all its edge identifiers as the *suffix*.

4.4.2 Resolving ambiguity.

Since the vertex name does not occupy a fixed number of bits across all labels, it is a priori unclear which part of the label contains it. To resolve this ambiguity we analyze the following function, which represents the final length of our labels for vertices in level t (up to a fixed constant). Let $D = [\lceil \log N \rceil]$ and $f : D \to \mathbb{N}$ be defined as

$$f(t) = \alpha_t + \Delta\beta_t = t + (\Delta + 1) \lceil \log(\log N - t)) \rceil.$$

The following lemma states that all but a constant number (depending on Δ) of values in D have at most two pre-images under f. This observation is useful, since it implies that the knowledge of the level $t(v)$ of the vertex v can resolve all remaining ambiguities in its label, as the vertex name occupies exactly $\alpha_{t(v)}$ bits.

Lemma 8. *Let $r(\Delta) = \lceil 8(\Delta+1) \log(\Delta+1) \rceil$. For every $t \in [\lceil \log N \rceil - r(\Delta)]$ the number of integers $t' \in D \setminus \{t\}$ that satisfy $f(t) = f(t')$, is at most one.*

Proof. Consider first the behavior of the term $h(t) = (\Delta + 1) \lceil \log(\log N - t) \rceil$ in the definition of f. Let $t_1 \in [D]$ be the smallest integer such that $\log(\log N - t_1)$ is integer. It follows that $\log N - t_1 = 2^s$ for some integer s. Consider the smallest integer $t_2 \in D$ with $t_2 > t_1$ such that $\log(\log N - t_2)$ is integer. This value clearly satisfies $\log N - t_2 = 2^{s-1}$. The smallest integer $t_3 \in D$ with $t_3 > t_2$ such that $\log(\log N - t_3)$ is integer is $\log N - t_3 = 2^{s-2}$ and so on. Let $t_1 < t_2 < \cdots < t_p$ be the obtained sequence of $p = O(\log \log N)$ points that satisfies $t_i = \log N - 2^{s-i+1}$ for $i \in [p]$. We call such points *jumps*. See Figure 4.2 for an illustration of f. Observe next that for every $k \in [p-1]$ the term $h(t)$ is constant within the interval $I_k = [t_k + 1, t_{k+1}]$, while at the point $t_{k+1} + 1$ it decreases by $\Delta + 1$.

We rewrite f as

$$f(t) = t - \sum_{k\,:\,t_k \le t-1} (\Delta+1) + (\Delta+1)\lceil \log(\log N - 1)\rceil,$$

The last term does not depend on t, thus it suffices to prove the claim for $\hat{f}(t) = t - \sum_{k\,:\,t_k \le t-1}(\Delta+1)$. Consider next some $t \in D$ and assume that $\hat{f}(r) \neq \hat{f}(t)$ for every $r < t$. Define $K = \lceil \log N \rceil - t$, and let $t_i \ge t$ be the smallest jump larger or equal to t. Define $\gamma = t_i - t$. Observe that the number of jumps within the interval $[t_i, \lceil \log N \rceil]$, whose length is $K - \gamma$ is at most $\log(K-\gamma)+1$, while the next jump occurring after t_i appears exactly in the middle of the latter interval, namely $t_{i+1} - t_i = \frac{K-\gamma}{2}$. To this end assume that $\hat{f}(t)$ has at least two other pre-images $t'' > t' > t$ under $\hat{f}$. Since $\hat{f}$ is monotonic between jumps, there must be at least one jump between every consecutive pair of pre-images of $\hat{f}$. It follows that $t \le t_i \le t'$ and $t' \le t_j \le t''$ for some $j \ge i+1$. In particular, it follows that $[t_i, t_{i+1}] \subset [t_i, t'']$, which implies $t'' - t_i \ge \frac{K-\gamma}{2}$. Now, using the fact that within the interval $[t_i, \lceil \log N \rceil] \supset [t_i, t'']$ there are at most $\log(K-\gamma)$ jumps, we can write

$$\begin{aligned}
\hat{f}(t'') &= t'' - \sum_{k\,:\,t_k \le t''-1} (\Delta+1) \\
&\ge t - \sum_{k\,:\,t_k \le t-1} (\Delta+1) + (t''-t) - \sum_{k\,:\,t_k \in [t_i, t''-1]} (\Delta+1) \\
&\ge \hat{f}(t) + (\gamma + \frac{K-\gamma}{2}) - (\Delta+1)(\log(K-\gamma)+1) \\
&\ge \hat{f}(t) + \frac{K}{2} - (\Delta+1)(\log K + 1),
\end{aligned}$$

which, using $\hat{f}(t) = \hat{f}(t'')$, implies

$$\frac{K}{2} - (\Delta+1)(\log K + 1) \le 0.$$

Rearranging the terms we obtain $\frac{K}{2(\log K+1)} \le \Delta + 1$, which clearly implies $K \le r(\Delta)$ and $t \in [\lceil \log N \rceil - r(\Delta)]$. This concludes the proof.

□

Remark 2. *It is natural to ask if having equal label lengths for vertices in different levels can be avoided altogether (thus making Lemma 8 unnecessary). This seems not to be the case for the following reason. The number of vertices in every level is at least $\Omega(\log N)$, thus, with label size $\ell = o(\log\log N)$ one can not uniquely represent all vertices in* any *level. Furthermore, the label length is also restricted to $\log n + O(1)$, and the number of levels is $\lceil \log N \rceil$. Thus, a function assigning levels to label lengths would need to have domain $[\lceil \log N \rceil]$ and range $[g(N), \lceil \log N \rceil]$ for $g(N) = \Omega(\log\log N)$, implying that it cannot be one-to-one.*

Recall that the length of the suffix of vertex $v \in V$ is exactly $\alpha_{t(v)} + \Delta\beta_{t(v)}$. We next show how the structural property proved here allows to construct a constant size *prefix*, that will eliminate the ambiguity caused by differential sizing.

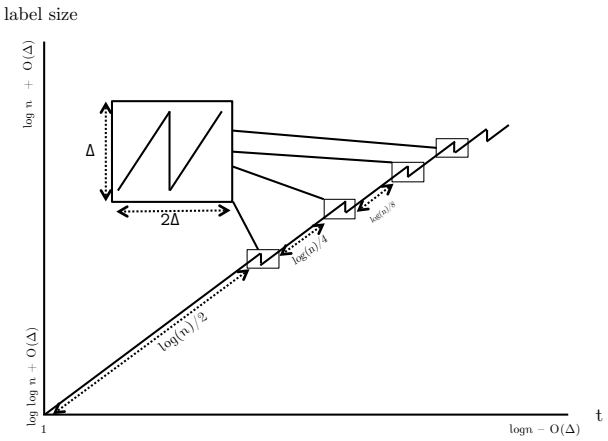


FIGURE 4.2: An illustration of Lemma 8

4.4.3 Constructing the prefix.

For a formal description of the prefix we need the following definition. We let $r(\Delta) = \lceil 8(\Delta+1)\log(\Delta+1)\rceil$, as in Lemma 8.

Definition 12. *A vertex $v \in V$ is called* shallow *if its level $t(v)$ is at most $\lceil \log N\rceil - r(\Delta)$. We call a shallow vertex* early *if $t(v)$ is the smallest pre-image of $f(t(v))$. A shallow vertex that is not early is called* late.

A vertex $v \in V$ that is not shallow is called deep. *A deep vertex is of* type τ, *if its level satisfies $t(v) = \lceil \log N\rceil - \tau$.*

It is easy to verify the following properties. Lemma 8 guarantees that if v is shallow, then $f(t(v))$ has at most two pre-images under f. If v is shallow and there is only one pre-image for $f(t(v))$, then v is early. Finally, observe that the type of deep vertices ranges in the interval $[1, r(\Delta)]$.

We are now ready to define the prefix of a label $\mathcal{L}(v)$ for a vertex $v \in V$. Every prefix starts with a single bit $D(v)$ that is set to 0 if v is shallow, and to 1 if v is deep. The second bit $R(v)$ in every prefix indicates whether a shallow vertex is early, in which case it is set to 0, or late, in which case it is set to 1. The bit $R(v)$ is always set to 0 in labels of deep vertices. The next part $Type(v)$ of the prefix contains $\lceil \log r(\Delta)\rceil$ bits representing the type of the vertex v, in case v is deep. If v is shallow this field is set to zero. This concludes the definition of the prefix. Observe that the prefix contains $O(\log\Delta) = O(1)$ bits. We stress that length s_p of the prefix is identical across all labels.

It is evident that the prefix of a label eliminates any remaining ambiguity. This follows from the fact that the level $t(v)$ of a vertex $v \in V$ can be computed from the length of the suffix and the additional information stored in the prefix. The level, in turn, allows to decompose the suffix into the vertex name and the incident edge names, which can then be used for decoding. We elaborate on the decoding algorithm later on.

4.4.4 The final labels.

The final label is obtained by concatenating the suffix to the prefix, namely $\mathcal{L}(v)$ is defined as follows.

$$\mathcal{L}(v) = \underbrace{D(v) \circ R(v) \circ Type(v)}_{\text{prefix}} \circ \underbrace{\mathrm{Id}(\phi(v)) \circ EId_{\phi(v)}(e_1) \circ \cdots \circ EId_{\phi(v)}(e_\Delta)}_{\text{suffix}}.$$

Figure 4.3 illustrates the label structure as a function of the level of the vertex. Note that $|\mathcal{L}(v)| = s_p + f(t(v))$, thus the level $t(v)$ of v determines $|\mathcal{L}(v)|$. Note that Lemma 7 and the fact that the prefix has constant size guarantees that $|\mathcal{L}(v)| = \log n + O(1)$, as desired. We also pad each label with sufficiently many 0's and a single '1', to arrive at a uniform length. The latter simple modification allows the decoder to work without knowing n in advance (see [47] for details).

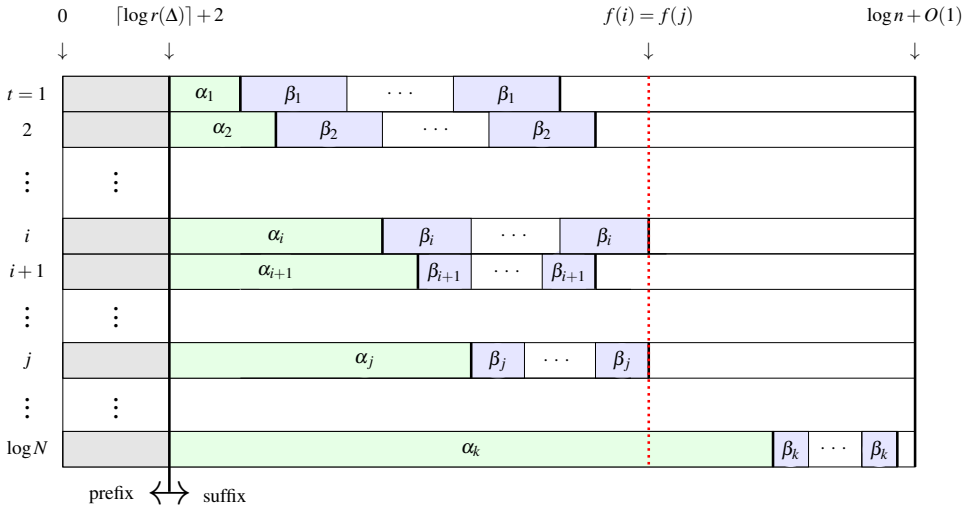


FIGURE 4.3: The composition of labels in our labeling scheme for vertices in different levels $t = 1, \cdots, \log N$. The size of the prefix is seen to be constant in every level, while the fields of lengths α_t and β_t, used to store vertex and edge identifiers, respectively, have variable sizes. The levels i and j comprise a collision with respect to the function f, thus labels of vertices in these levels have the same length.

Although it is not necessary for the correctness of our labeling scheme, we prove here uniqueness of the labels. In other words, we show that two different vertices in G necessarily get different labels.

Lemma 9. *For every two distinct $u, v \in V(G)$ it holds that $\mathcal{L}(u) \neq \mathcal{L}(v)$.*

Proof. Consider two vertices $u, v \in V(G)$ and assume $\mathcal{L}(u) = \mathcal{L}(v)$. Since the prefix of every label has the same length and the same parts for every vertex in $V(G)$, it must hold that $D(u) = D(v)$, $R(u) = R(v)$ and $Type(u) = Type(v)$.

Assume first that both u and v are shallow. In this case $\mathcal{L}(u) = \mathcal{L}(v)$ implies that $f(t(u)) = f(t(v))$. Now, since $R(u) = R(v)$, both v and u are either early, or late, implying by Lemma 8 that $t(u) = t(v) =: t$. It follows by definition of the labels, that the first α_t bits of the suffices of $\mathcal{L}(u)$ and $\mathcal{L}(v)$ contain $\mathrm{Id}(\phi(u))$ and $\mathrm{Id}(\phi(v))$, respectively, implying $\mathrm{Id}(\phi(u)) = \mathrm{Id}(\phi(v))$. From the correctness of the embedding ϕ and uniqueness of the identifiers Id it follows that $u = v$.

Finally, assume that both u and v are deep. In this case $Type(u) = Type(v)$ implies again that $t(u) = t(v)$. We now reach the conclusion $u = v$ as in the previous case. □

4.5 Decoding

Consider two labels $\mathcal{L}(u)$ and $\mathcal{L}(v)$ for vertices $u, v \in V$. The decoder first extracts the levels $t(u)$ and $t(v)$ of u and v respectively, using the following simple procedure, which we describe for v. If $D(v) = 0$, v is shallow. To this end, the decoder computes all pre-images of the length of the suffix, $|\mathcal{L}(v)|-s_p$, under f. Recall, that by Lemma 8, the number of pre-images is at most two. Let $t_1 \leq t_2$ be the computed pre-images. Next, the decoder inspects the bit $R(v)$. According to the definition of the labels, $t(v) = t_1$ if $R(v) = 0$, and $t(v) = t_2$, otherwise. Consider next the case $D(v) = 1$, namely that v is deep. In this case, the decoder inspects $Type(v)$. The level of v is $t(v) = \lceil \log N \rceil - Type(v)$, by definition of a type of a deep vertex. It is obvious by the definition of the labels that the decoder extracts $t(v)$ correctly. Assume next that the decoder extracted $t(u)$ and $t(v)$. The decoder can now extract $\mathrm{Id}(\phi(u))$ and $\mathrm{Id}(\phi(v))$, by inspecting the first $\alpha_{t(u)}$ and $\alpha_{t(v)}$ bits of the suffix of $\mathcal{L}(u)$ and $\mathcal{L}(v)$, respectively. Next, the decoder checks if $\phi(u)\phi(v) \notin E_n$, in which case it reports **false**. Finally, if $\phi(u)\phi(v) \in E_n$ the decoder scans all Δ blocks of $\beta_{t(u)}$ bits each, succeeding $\mathrm{Id}(\phi(u))$ in the suffix of $\mathcal{L}(u)$, checking if one of them contains the edge-identifier $EId_{\phi(u)}(\phi(u)\phi(v))$. If this identifier is found the decoder reports **true**. Otherwise, it reports **false**. The correctness of the decoding is clear from the label definition and Lemma 8.

Lemma 10. *The decoding of the labels can be performed in time* $O(\log \log n)$.

Proof. All operations performed by the decoder take $O(1)$ time except for the computation of pre-images of an integer under f, and steps that inspect properties in H_n. The former can obviously be performed in $O(\log \log n)$ time as follows. If the label corresponds to a deep label, then the decoder checks all $r(\Delta) = O(1)$ possible preimages, so assume next that the vertex is shallow. Notice that $t - f(t) = O(\log \log n)$ for every $t \in D$. Thus, given $f(t)$, one can compute in $O(\log \log n)$ time all $f(y)$ for all integers y with $|y - f(t)| = O(\log \log n)$. All preimages of $f(t)$ under f will be found this way.

We focus hereafter on the complexity of deciding if $\phi(u)\phi(v) \in E_n$ and the computation of the edge-identifier corresponding to two vertices in H_n. To this end we need to elaborate on the way labeling is performed for the graph H_n. Recall that H_n is built from the complete binary tree T with $\log N$ levels by splitting each vertex at level t into $\gamma_t = c \log(N/2^t)$ vertices, thus forming clusters, and then connecting two vertices if they belong to clusters at distance at most $g(\Delta) = 2 \log \Delta + 2$ apart in T. The labeling of vertices is performed by assigning a *range* R_C of size γ_t to every cluster at level t in a single breadth-first search traversal, and then ordering the vertices in the cluster arbitrarily using unique values in the range R_c. More precisely, the ranges are constructed starting from the root of T, processing T level by level as follows. The Range of the root cluster is simply $[1, \gamma_1]$. The range of the left child of the root is assigned the range $[\gamma_1 + 1, \gamma_1 + \gamma_2]$, while the right child is assigned the range $[\gamma_1 + \gamma_2 + 1, \gamma_1 + 2\gamma_2]$ and so on. At level

t, the assignment of ranges starts from the left descendant of the parent at level $t-1$ with the lowest range (the range with the smallest lower bound), followed by the right one, and so on. This process defines an ordering of the clusters, in each level. We denote by $POS(C)$ the position of the cluster C in this order.

Given a label L of a vertex $w \in V_n$ at level $t = t(w)$, its clusters range (which identifies the cluster) can be computed as follows. By definition of the labeling, we have that L is contained in the range $I(w) = [A+1, A+\gamma_t]$ with

$$A = \sum_{i=1}^{t-1} 2^{i-1}\gamma_i + (POS(C(w)) - 1)\gamma_t,$$

where $C(w)$ is the cluster to which w belongs. We now notice that

$$\sum_{i=1}^{t-1} 2^{i-1}\gamma_i = \sum_{i=1}^{t-1} 2^{i-1}(\log n - i) = c \log n \sum_{i=1}^{t-1} 2^{i-1} - \frac{c}{2}\sum_{i=1}^{t-1} 2^i i,$$

where both sums in the last formula have simple closed form expressions. It follows that $POS(C(w))$ can be easily computed from t and L in $O(1)$ time.

Finally, the decoder can compute both the parent and the descendants of a cluster C at level t whose range is $[A+1, A+\gamma_t]$. The parent is computed using the fact that it is at level $t-1$ and its position in this level is $\lceil POS(C)/2 \rceil$. The descendants can be computed using the fact that their level is $t+1$ and their positions in this level are $2POS(C)-1$ and $2POS(C)$. Both computations can clearly be performed in $O(1)$ time, once presented with the values γ_t. We conclude that it is possible to obtain in time $O(1)$ the ranges of all neighbours of a given cluster C in T.

We turn to the problem of deciding whether $\phi(u)\phi(v) \in E_n$. The decoder starts by computing the clusters C_u and C_v of u and v, respectively in time $O(\log n \log n)$. Then the decoder computes paths P_u and P_v of length at most $g(\Delta)$ from C_u and C_v, respectively in the tree T. Each path is obtained by successively moving at most $g(\Delta)$ steps from a cluster to its parent in T. The computation of these paths takes $O(1)$ time, using the aforementioned method for computing a parent of a cluster. By definition of H_n, we have that $\phi(u)\phi(v) \in E_n$ if and only if the clusters of $\phi(u)$ and $\phi(v)$ are at distance at most $g(\Delta)$ from one another in T. This can be now easily tested by inspecting the paths P_u, P_v, which must contain the least common ancestor (LCA) of $\phi(u)$ and $\phi(v)$, in case they are neighbours in H_n (see Figure 4.1 for an illustration).

With a similar argument one shows that each edge identifier can be decoded in $O(1)$ time. The details are similar, and thus omitted. Performing the decoding to all Δ edge identifiers gives a total running time of $O(1)$. We conclude that the decoder can be implemented to have running time $O(\log \log n)$. □

4.6 Computing the embedding ϕ

All the labels can clearly be computed from the graph G, the embedding ϕ and the graph H_n in time $O(n \log n)$. It is also straightforward to compute H_n in $O(n)$ time. It remains to discuss how to compute an embedding ϕ, for which we provide a high-level overview. For a detailed description, see [62].

The algorithm uses a subroutine for computing bisectors of a graph. A *bisector* of a graph $G = (V, E)$ is a set $S \subset V$ of vertices with the property that the connected components of the graph $G - S$ can be partitioned into two parts, such that the sum of the vertices in each part is the same, and no edge connects two vertices in different parts. If S is a bisector in G we say that S *bisects* $V \setminus S$.

Given a k-coloring $V = V_1 \cup \cdots \cup V_k$ of V (for some $k \in \mathbb{N}$), one can define a *k-bisector* of G as a set $S \subset V$ that bisects *every* color class, namely one that bisects $V_i \setminus S$ for all $i \in [k]$. An important property of outerplanar graphs is that they admit $O(\log n)$ size k-bisectors, for every fixed k.

The algorithm works by assigning vertices in the graph G to clusters in T. The root of T is assigned up to $c \log n$ vertices that form a bisector of G with parts G_1, G_2. In the next iteration, vertices adjacent to vertices in the bisector are given a new color. Next, two new 2-bisectors are found, one in each part G_1, G_2, and they are assigned to the corresponding decedents of the root of T.

Let $k(\Delta) = \log \Delta + 1$. In general, the vertices stored at a vertex of T at level t correspond to a $k(\Delta)$-bisector. The colors of this bisector correspond to the neighbors of vertex-sets stored at $k(\Delta) - 1$ nearest ancestors of the current vertex in T. The last color is reserved to the remaining vertices. Also stored in this vertex are all neighbors of the vertex-set stored in the ancestor of the current vertex at distance exactly $k(\Delta)$, that were not yet assigned to some other cluster. We refer the reader to [62] for an analysis of the sizes of clusters.

Let $T(n)$ be the running time of the latter algorithm in a graph with n vertices. $T(n)$ clearly satisfies $T(n) \leq 2T(n/2) + O(h(n))$, where $h(n)$ is the complexity of finding an $O(1)$-bisector of $O(\log n)$ size in an n-vertex graph. For outerplanar graphs the latter can be done in linear time [55,62], thus the labels of our labeling scheme can be computed in $O(n \log n)$ time.

4.7 Improvements and special cases

Several well-known techniques can be easily applied on top of our construction to reduce the additive constant in the label size. First, since graphs of maximum degree Δ have arboricity $\lfloor \frac{\Delta}{2} \rfloor + 1$, one can reduce the number of edge identifiers stored in each label to the latter number (see [2]). We later show a simpler procedure that works for bounded degree graphs.

Finally, for bounded-degree trees $\mathcal{T}(n, \Delta)$, it suffices to store a single edge identifier (corresponding to the edge connecting a vertex to its parent in G). We summarize this result in the following corollary of Theorem 4.

Corollary 1. *For every fixed $\Delta \geq 1$, the class $\mathcal{T}(n, \Delta)$ of bounded-degree trees admits a labeling scheme of size $\log n + O(1)$, with encoding complexity $O(n \log n)$ and decoding complexity $O(\log \log n)$.*

4.8 Planar graphs

We present here our labeling scheme for the class of bounded degree planar graphs $\mathcal{P}(n, \Delta)$. First, [63] proved that planar graphs can be edge-partitioned into two outerplanar graphs in linear time. Combining this fact with Theorem 4, yields the following.

Corollary 2. *For every fixed $\Delta \geq 1$, the class $\mathcal{T}(n, \Delta)$ of bounded-degree planar graphs admits a labeling scheme of size $2\log n + O(1)$, with encoding complexity $O(n \log n)$ and decoding complexity $O(\log \log n)$.*

In the remainder of this section we show improved bounds for $\Delta < 6$. We rely on an embedding of the given planar graph G into a graph $PL_n = (W_n, A_n)$, obtained from the complete binary tree T. The only difference between PL_n and H_n is that in PL_n, every vertex v in level t of T is divided into a cluster of $\delta_t = c'\sqrt{\frac{N}{2^{-t}}}$ vertices $z_1(v), \cdots, z_{\delta_t}(v)$ (instead of γ_t in H_n), for some constant c'. As for H_n, two vertices in W_n are connected if and only if the distance between their clusters in T is at most $\hat{g}(\Delta) = O(\log \Delta)$. [62] showed the following.

Theorem 6 ([62]). *PL_n is edge-universal for the class of bounded degree planar graphs $\mathcal{P}(n, \Delta)$. Furthermore, PL_n has $O(n)$ vertices and $O(n \log n)$ edges.*

We use the latter theorem to prove the following result.

Theorem 7. *For every fixed $\Delta \geq 1$, the class $\mathcal{P}(n, \Delta)$ of bounded-degree planar graphs admits a labeling scheme of size $\frac{\lceil \Delta/2 \rceil + 1}{2} \log n + O(\log \log n)$, with encoding complexity $O(n \log n)$ and decoding complexity $O(\log n \log n)$. The average label size of the latter scheme is $(1 + o(1)) \log n + O(\log \log n)$.*

Proof. We use a simplified version of the method used in Section 4.2, thus many details are omitted. Our labeling scheme for planar graphs works with an embedding $\psi : V \to W_n$ of the given graph G into PL_n. Levels of vertices $w \in W_n$ and $u \in V$ are defined as before. Analogously to our previous scheme, we assign unique identifiers $\mathrm{Id}(v)$ to vertices $v \in W_n$ level by level, starting from the highest level, and use them to define edge identifiers. Let $W_n^t \subset W_n$ denote the set of vertices in PL_n with level at most t. Then we have the following lemma.

Lemma 11. *The following properties hold for every level t and every vertex v in level t.*

i. $|W_n^t| = O(\sqrt{N 2^t})$.

ii. $|\Gamma(v)| = O(\sqrt{N 2^{-t}})$.

Proof. i. Note that the number of vertices in level i is $O(2^i \sqrt{\frac{N}{2^i}})$, thus for some constant d,

$$|W_n^t| = c' \sum_{i=1}^{t} 2^{i-1} \sqrt{\frac{N}{2^i}} \leq c' \sqrt{N} \sum_{i=1}^{t} \sqrt{2}^{i-1} = O(\sqrt{N 2^t}).$$

ii. Since v is connected to vertices in clusters at distance at most $O(\log \Delta) = O(1)$ from its cluster, there are $O(1)$ such clusters. Furthermore, the highest level of such a cluster is at least $t - O(1)$, thus every such cluster contains $O(\delta_t) = O(\sqrt{N 2^{-t}})$ vertices, which concludes the proof. □

Lemma 11 implies that a the encoding length of label containing a vertex identifier $\mathrm{Id}(\psi(v))$ together with $s \in \mathbb{N}$ edge identifiers corresponding to $\psi(v)$ is

$$\log(\sqrt{N 2^t}) + s \log(\sqrt{N 2^{-t}}) + O(1) = \log n + \frac{s-1}{2}(\log n - t(v)) + O(1). \quad (4.1)$$

Additionally storing the level $t(v)$ using $O(\log\log n)$ bits allows to perform decoding easily, without needing the complications arising in our scheme for outerplanar graphs. Finally, combined with the label splitting technique of Kannan et al. [2] the statement of the theorem follows.

To prove the bound on the average label length we perform the following simple computation. Consider a constant $B \gg \Delta$ and let $t^* = (1-B^{-1})\log n$. By choice of t^* and using Lemma 11 it follows that $|W_n^{t^*}| = c'n^{1-\epsilon}$ and $|W_n \setminus W_n^{t^*}| = n - c'n^{1-\epsilon}$ for some constants $c', \epsilon > 0$. Now, the length of every label $\mathcal{L}(v)$ for $v \in W_n^{t^*}$ can be trivially bounded by $\Delta \log n$, while for vertices $v \in W \setminus W_n^{t^*}$, whose level is at least t^*, we use (Equation (4.1)) to upper-bound the label size by

$$\log n + \frac{\Delta-1}{2}(\log n - t^*) + O(\log\log n) = \left(1 + B^{-1}\frac{\Delta-1}{2}\right)\log n + O(\log\log n).$$

Finally, combining the bounds one obtains the following bound on the sum of the label sizes.

$$\sum_{v\in V}|\mathcal{L}(v)| \leq c'n^{1-\epsilon}\Delta\log n + (n - c'n^{1-\epsilon})\left(1 + B^{-1}\frac{\Delta-1}{2}\right)\log n + O(n\log\log n).$$

The term $dn^{1-\epsilon}\Delta\log n$ is clearly negligible, since Δ is fixed. We conclude that the average label length is at most $\left(1 + B^{-1}\frac{\Delta-1}{2}\right)(1 + h(n))\log n + O(\log\log n)$ for some function $h(n) = o(1)$. Repeating the argument with $B = O(\log\log n)$ leads to a bound of

$$(1 + o(1))\log n + O(\log\log n)$$

on the average label size, which is asymptotically the best possible average label size.

□

In light of the $2\log n + O(\log\log n)$ labeling scheme of [6] for general planar graphs, our result improves the best known bounds for $\Delta \leq 4$, which includes grid graphs, and matches it for $\Delta \leq 6$.

4.9 Graph of bounded, but not constant degree

First we note that Theorem 4 implies almost directly a $\lceil\Delta/2\rceil\log n$ labeling scheme for graphs of fixed bounded degree Δ. The result follows from the technique of [8], Lemma 9 and the fact that any subtree of a bounded degree graph has bounded degree.

Corollary 3. *For every $\Delta \geq 1$, the class $\mathcal{G}(n,\Delta)$ of bounded-degree graphs admits labeling schemes of size $\lceil\Delta/2\rceil\log n + O(1)$, with encoding complexity $O(n\log n)$ and decoding complexity $O(\log\log n)$.*

From here on, we discuss labeling schemes for graphs of non-constant bounded degree $k = \Delta(n)$. *Adjacency* relation between any two vertices may be reported in only one of the labels representing them. For bounded degree graphs, the method of [2] of decomposition into forests can be replaced with a simpler procedure, using Eulerian circuits, as we prove in the following.

Lemma 12. *Let $G = (V, E)$ be a graph with degree bounded by k. It is possible to partition E into sets S_v, $v \in V$, with the properties that all edges in S_v are incident to v and $|S_v| \leq \lceil \frac{k}{2} \rceil$ for all $v \in V$. This partition implies a labeling scheme with size $(\lceil \frac{k}{2} \rceil + 1) \log n$ for graphs with degree bounded by k.*

Proof. We first create an Eulerian multigraph $G' = (V, E')$ from G by adding at most $|V|/2$ new edges, comprising a matching that connects pairs of vertices in G with odd degree. The degrees of all vertices in G' are even, hence it is Eulerian. We proceed by finding an Eulerian circuit P in G' and directing every edge according to the direction along P. Every vertex in G' with degree d has now exactly $\frac{d}{2}$ incoming and outgoing edges. The label of a vertex will only correspond to the vertices which are adjacent through outgoing edges in P. This number is at most $\frac{d}{2}$. The identifiers of the edges in M are not included in the resulting label. The labeling scheme obtained will assign unique labels to each of the vertices, and concatenate the selected labels to each vertex. since $d \leq k$ this yields a label of size $(\lceil \frac{k}{2} \rceil + 1) \log n$. □

The current best labeling schemes for graphs works in two modes, according to the range of k. If $k \leq n/\log n$, a $k/2 \log n$ labeling scheme can be achieved [2], essentially by encoding an adjacency list. For larger k, labels defined through the adjacency matrix of the graph have size $n/2 + O(1)$ [64]. Our improved labels use the well-known *combinatorial number system* (see e.g. [65]).

Lemma 13. *Let $L = \binom{n}{k}$. There is a bijective mapping $\sigma : S_k \to [0, L-1]$ between the set of strictly increasing sequences S_k of the form $0 \leq t_1 < t_2 \cdots < t_k < n$ and $[0, L-1]$ given by*

$$\sigma(t_1, \cdots, t_k) = \sum_{i=1}^{k} \binom{t_i}{i}.$$

We use Lemma 13 to prove the following theorem. For this purpose we assume that the labeling scheme presented in Lemma 12 returns a subset of vertices for every vertex $v \in V$ according to the partition instead of a final label.

Theorem 8. *For $1 \leq k \leq n$, there exist an* Adjacency *labeling scheme for $\mathcal{G}(n, k)$ with size:* $\log \binom{n}{\lceil k/2 \rceil} + \lceil \log n \rceil + \lceil \log k \rceil$.

Proof. We assume the vertices of the graph $G = (V, E)$ to be numbered from 0 to $n-1$. We call a binary vector C of length n an *adjacency vector* of a vertex $v \in V$ if it satisfies that $C_i = 1$ if and only if v_i is a neighbor of v in G. We denote by $\bar{C}$ the binary vector with $\bar{C}_i = 1$ if and only if $C_i = 0$. We interpret the vectors C and $\bar{C}$ and sets $S \subset V$ as sequences of integers in the range $[0, n-1]$, as in Lemma 13. Let $v \in V$ be a vertex with $k' \leq k$ neighbors, and let C be its adjacency vector. Our labeling scheme assigns every $v \in V$ to an appropriate $\mathcal{L}(v)$ as follows.

We first use the encoder from the labeling scheme described in Lemma 12 to obtain a temporary subset S_v of neighbors of v. If $k' < \frac{2n}{3}$ we set the last bit of $\mathcal{L}(v)$ to 0, and append it to the number mapped to the sequence of integers corresponding to S_v under the bijection in Lemma 13. According to Lemma 12, we are assured that $|S_v| \leq \lceil \frac{k}{2} \rceil \leq \lceil \frac{n}{3} \rceil$. If $k' \geq \frac{2n}{3}$, we set the last bit of $\mathcal{L}(v)$ to 1, and append to it the number corresponding to $\bar{C}$

under the bijection in Lemma 13. Since $k' > \frac{2n}{3}$, the number of 1's in $\bar{C}$ is at most $\frac{n}{3}$. Finally, we attach to every label a binary representation of k' and a unique vertex identifier using exactly $\log n$ bits each. The encoder performs the operation in polynomial time. It is straightforward to verify the claimed label length.

The decoder receives $\mathcal{L}(u)$ and $\mathcal{L}(v)$ and extracts the corresponding vectors C_u and C_v, the adjacency vectors of u and v, resp., using Lemma 13, and possibly a bit inversion operation. The decoder returns **true** if and only if either $C_u(v) = 1$ or $C_v(u) = 1$. We note that the decoding can be implemented in $O(k \log \binom{n}{k})$ time. □

The labeling scheme suggested in Theorem 8 implies a label size of approximately $(k+2)\log n$ bits, when k is small and $\Theta(n)$ when $k = \Theta(n)$. The following lemma identifies the range of k for which our labeling scheme improves on the best known bounds.

Lemma 14. *For* $(e+1)\sqrt{n} \leq k \leq \frac{n}{5}$, *and* $f(n,k) = \binom{n}{\lceil k/2 \rceil} + \log k + \log n$ *it holds that a)* $f(n,k) < \frac{n}{2}$; *and b)* $f(n,k) < \lceil k/2 \rceil + 2\log(n)$.

Proof. Stirling's approximation yields the following asymptotic approximation.

$$\log \binom{cn}{n} \sim \log \left(\frac{c^c}{(c-1)^{c-1}} \right) \cdot n.$$

Accordingly, $\log \binom{n}{\frac{1}{3}n} \sim \frac{2.75}{3} n$. Since $\log(\frac{10^{10}}{9^9})/10 < 0.5$, and since the function $\binom{n}{k}$ is increasing for $\sqrt{n} \leq k \leq \frac{n}{3}$ our labeling scheme will incur strictly less than $0.5n$ label size for the proposed labeling scheme over the range specified.

Let x_n be defined by $x_n = \log(n!) - (n\log n - n + 1/2(\log 2\pi n))$, and by its definition $x_n \to 0$ as $n \to \infty$. In addition:

$$\log \binom{k}{n} = \log(n!) - \log((n-k)!) - \log(k!)$$

$$= (x_n - x_{n-k} - x_k) + n\log n - (n-k)\log(n-k) - k(\log k) + 1/2\log(n/k(n-k)2\pi)).$$

Define $f(n,k)$ by $f(n,k) = k/2\log n - \log \binom{k}{n}$. We are now interested in the smallest k for which $k_n = f(n,k) \leq 0$ for fixed n. $f(n, \lfloor \sqrt{n} \rfloor) < 0$, so $k_n > \sqrt{n}$, so we can assume $\sqrt{n} \leq k \leq n/2$, which implies that $1/2\log(n/(k(n-k)2\pi)) = O(\log n)$. Furthermore $x_n - x_{n-k} - x_k = O(1)$ so

$$f(n,k) = k/2\log n - n\log n + (n-k)\log(n-k) + k\log k + O(\log n)$$

$$= k/2\log n + n\log((n-k)/n) + k\log(k/(n-k)) + O(\log n).$$

Note that as $n\log((n-k)/n) = O(k)$ this means that

$$\frac{f(n,k)}{k} = \frac{1}{2}\log n - \log((n-k)/k) + O(1) = \log\left(\frac{k\sqrt{n}}{n-k}\right) + O(1)$$

This means that there exists some constant $c > 0$ such that if $\log\left(\frac{k\sqrt{n}}{n-k}\right) \geq c$ then $f(n,k) \geq 0$. But

$$\log\left(\frac{k\sqrt{n}}{n-k}\right) \geq c \Longleftrightarrow k \geq c\frac{n}{\sqrt{n}+c}.$$

If $k > c\sqrt{n}$ then $f(n,k) \geq 0$. We can conclude $k_n \leq c\sqrt{n}$, and we may assume that $c\sqrt{n} \geq k \geq \sqrt{n}$.

Now we can conclude that $nlog((n-k)/n) = -k+O(k^2/n) = -k+O(1)$. Hence

$$\frac{f(n,k)}{k} = \frac{1}{2}\log n - 1 + k\log(k/(n-k)) + O(\frac{\log n}{\sqrt{n}}),$$

and thus

$$\frac{f(n,k)}{k} = \log\left(\frac{k\sqrt{n}}{e(n-k)}\right) + O(\frac{\log n}{\sqrt{n}}).$$

We can now conclude with $c_n = exp(O(\frac{\log n}{\sqrt{n}}))$

$$k_n = c_n e\frac{n}{\sqrt{n}+c_n e} + O(1) = e\sqrt{n} + O(\log n).$$

□

We conclude from Lemma 14 that our labeling scheme is preferable to [64] for graphs of bounded degree k for $(e+1)\sqrt{n} \leq k \leq \frac{n}{5}$.

4.10 Concluding remarks

Firstly, we note that [47] result for *Ancestry* in bounded depth trees served as basis for a labeling scheme for the unbounded case [58]. It would be interesting to see if there exist an alternative asymptotically optimal *Adjacency* labeling schemes for forests using a variant of Theorem 4.

Secondly, there exists a large gap in our results for bounded degree planar graph (Section 4.8) between the size of the labeling scheme and the average label length. This suggests the possibility of improving labeling schemes for bounded degree planar graphs. It is also conceivable that there exists a general transformation from labeling schemes with small average label length to good "ordinary" labeling schemes.

Lastly, our labeling scheme for general graphs (Section 4.9) bears an expensive decoding process, and would benefit from a more efficient decoding.

Chapter 5

Adjacency Labeling Schemes for Power-Law Graphs

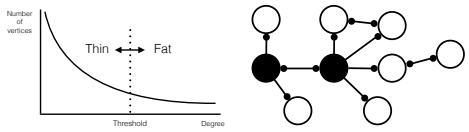
Though many graph families have been meticulously studied for this problem, a non-trivial labeling scheme for the important family of power-law graphs has yet to be obtained. This family is particularly useful for social and web networks as their underlying graphs are typically modelled as power-law graphs. Using simple strategies and a careful selection of a parameter, we show upper bounds for such labeling schemes of ($\sqrt[\alpha]{n}$) for power law graphs with coefficient α, as well as nearly matching lower bounds. We also show two relaxations that allow for a label of logarithmic size, and extend the upper-bound technique to produce an improved *Distance* labeling scheme for power-law graphs.

One class of graphs extensively used for modelling real-world networks is *power-law graphs*: roughly, n-vertex graphs where the number of vertices of degree k is proportional to n/k^{α} for some positive α. Power-law graphs (also called scale-free graphs in the literature) have been used to model the Internet AS-level graph [67,68], and many other types of network (see, e.g., [69,70] for overviews). The adequacy of fit of power-law graph models to actual data, as well as the empirical correctness of the conjectured mechanisms giving rise to power-law behaviour, have been subject to criticism (see, e.g., [69,71]). In spite of such criticism, and because their degree distribution affords a reasonable approximation of the degree distribution of many networks, the class of power-law graphs remains a popular tool in network modelling. In this chapter, we perform the first theoretical and practical study of *Adjacency* labeling schemes for classes of graphs whose statistical properties–in particular their *degree distribution*–more closely resemble that of real-world networks.

Our contributions

In this chapter we contribute the following:

A discrete and simple characterisation of power-law graphs An n-vertex graph is power-law if the number of its vertices of degree k is proportional to n/k^{α} for some positive α. To solidify this somewhat vague definition, numerous probabilistic and deterministic definitions of power-law graphs are given in the literature. In Section 5.2, we define and prove useful properties for two simple families of graphs, $\mathcal{P}_h$ and $\mathcal{P}_l$, where $\mathcal{P}_h$ contains and $\mathcal{P}_l$ is contained by the standard definitions of power-law graphs in the literature, including recent ones [72]. We use $\mathcal{P}_h$ and $\mathcal{P}_l$ to study upper and lower bounds respectively.



(A) (B)

FIGURE 5.1: Two illustrations of the main idea: Figure (a) demonstrates the threshold assignment, figure (b) demonstrates the label assignment, in which fat (black) vertices do not store *Adjacency* to thin (white) vertices.

An $O(\sqrt[\alpha]{n}(\log n)^{1-1/\alpha})$ *Adjacency* labeling scheme In Section 5.4, we describe our labeling scheme, which is based on two ideas: (i) a labeling *strategy* that partitions the vertices of G into high ("fat") and low degree ("thin") vertices based on a threshold degree, and (ii) a threshold *prediction* that depends only on the coefficient α of a power-law curve fitted to the degree distribution of G. These ideas are illustrated in Figure 5.1. Using the same ideas, we get an asymptotically near-tight $O(\sqrt{n \log n})$ *Adjacency* labeling scheme for sparse graphs. As real-world power-law graphs have $2 \leq \alpha \leq 3$ and rarely exceed 10^{10} vertices, this implies labels of the order of $10^4 - 10^5$ bits. That, and the simplicity of our labeling scheme suggests that our labeling schemes may be appealing in practice. To stress this point, we offer an experimental evaluation of our labeling scheme in Section 5.9.

A lower bound of $\Omega(\sqrt[\alpha]{n})$ for any *Adjacency* labeling scheme In Section 5.6, We use our restrictive subclass of power-law graphs and show that it requires label size $\Omega(\sqrt[\alpha]{n})$ for n-vertex graphs. This lower bound shows that our upper bound above is asymptotically optimal, bar a $(\log n)^{1-1/\alpha}$ factor. By the connections between *Adjacency* labeling schemes and universal graphs, we also obtain upper and lower bounds for induced universal graphs for power-law graphs. We also show, in Section 5.7, two scenarios in which this lower bound can be bypassed.

A $o(n)$ *Distance* labeling scheme In Section 5.8, we demonstrate the usefulness of our strategy to arrive at a $o(n)$ *Distance* labeling scheme for power-law graphs. Our labeling scheme is designed to outperform competing labeling schemes for small distances, in accordance with Chung and Lu's findings [73] on the small expected diameter of power-law graphs.

5.1 Preliminaries

In this chapter we consider n-vertex, undirected graphs. For a real $c > 0$, a graph is *c-sparse* if it has at most cn edges and *sparse* if it is c-sparse for some constant c. For $0 < c \leq n-1$, the set of c-sparse graphs with n vertices is denoted by $\mathcal{S}_{c,n}$. If $\mathcal{F}$ is a set of graphs, $\mathcal{F}_n$ denotes the subset of graphs in $\mathcal{F}$ having exactly n vertices. The degree of a vertex v in a graph is denoted by $\Delta(v)$, and for non-negative integers k, the set of vertices in a graph G of degree k is denoted by V_k. The length of a binary string $x \in \{0,1\}^*$ is denoted by $|x|$.

Let $\mathcal{F}$ be a set of graphs. An Adjacency *labeling scheme* (from hereon just *labeling scheme*) for $\mathcal{F}$ is a pair consisting of an *encoder* and a *decoder*. The encoder is an algorithm that receives $G \in \mathcal{F}$ as input and outputs a bit string $\mathcal{L}(v) \in \{0,1\}^*$, called the *label* of v, for each vertex v in G. The decoder is an algorithm that receives any two labels $\mathcal{L}(v), \mathcal{L}(u)$ as input and outputs `true` if u and v are adjacent in G and `false` otherwise. Note that the graph G is not an input to the decoder. The *size* of a labeling scheme is the map size : $\mathbb{N} \longrightarrow \mathbb{N}$ such that size(n) is the maximum length of any label assigned by the encoder to any vertex in any graph $G \in \mathcal{F}_n$. The *degree distribution* of a graph $G = (V, E)$ is the mapping $\text{ddist}_G(k) : \mathbb{N}_0 \longrightarrow \mathbb{Q}$ defined by $\text{ddist}_G(k) := \frac{|V_k|}{n}$.

5.2 Defining power-law graphs

In the literature *power-law* graphs are usually defined as the class of n vertex graphs G such that $\text{ddist}_G(k)$ is proportional to $k^{-\alpha}$ for some real number $\alpha > 1$. Ideally, and ignoring rounding, $\text{ddist}_G(k) = Ck^{-\alpha}$ for all k for constant C. As the degree distribution of a graph must be a probability distribution, we have $\sum_{k=1}^{\infty} Ck^{-\alpha} = C\sum_{k=1}^{\infty} k^{-\alpha} = 1$, hence $C = 1/\zeta(\alpha)$ where ζ is the Riemann zeta function. However, in the literature, concessions are usually made that relax the restrictions on $\text{ddist}_G(k)$, for example that the power-law property need only hold for high-degree vertices ("above a cutoff"), or that $\text{ddist}_G(k)$ is only *approximately* equal to $Ck^{-\alpha}$, with some approximation error that falls off with n. To ensure that our results hold for all these variations of power-law graphs, we define two families of graphs $\mathcal{P}_h$ and $\mathcal{P}_l$ with $\mathcal{P}_l \subsetneq \mathcal{P}_h$. Family $\mathcal{P}_h$ is rich enough to contain the graphs whose degree distribution is approximately, or perfectly, power-law distributed, and our upper bound on the label size for our labeling scheme holds for any graph in $\mathcal{P}_h$. Family $\mathcal{P}_l$ is used to show our lower bound and is restrictive enough that most definitions of power-law graph occurring in the literature will contain it.

In the following, let $i_1 = \Theta(\sqrt[\alpha]{n})$ be the smallest integer such that $\lfloor Cn/i_1^{\alpha} \rfloor \leq 1$, and let $C' \geq (\frac{C}{\alpha-1} + \frac{i_1}{\sqrt[\alpha]{n}} + 5)^{\alpha} + \frac{C}{\alpha-1}$ be a constant; we shall use C' in the remainder of the chapter.

Definition 13. *Let $\alpha > 1$ be a real number and let $\chi : \mathbb{N} \to \mathbb{N}$ be a function. $\mathcal{P}_{h,\chi,\alpha}$ is the family of graphs G such that if $n = |V(G)|$ then for all integers k between $\chi(n)$ and $n-1$, $\sum_{i=k}^{n-1} |V_i| \leq C'(\frac{n}{k^{\alpha-1}})$. We shall usually suppress χ and α, writing merely $\mathcal{P}_h$.*

The function χ captures the notion of a cutoff as defined in [69, Sec. 3.1]; the intuition is that for an n-vertex graph the power-law distribution need only apply for vertices of degree higher than $\chi(n)$, rather than for all degrees. Setting $\chi(n) = 1$ corresponds to the case where the entire range of degrees follows a power-law distribution, hence even for small values of $\chi(n)$, $\mathcal{P}_h$ morally contains all graphs with power-law degree distribution. We will later prove upper bounds that hold for *all* χ bounded from above by some function; in particular for the upper bound for *Adjacency* labeling schemes, the bound holds for $\chi(n)$ as high as $\sqrt[\alpha]{n/\log n}$.

The class $\mathcal{P}_l$ contains graphs where the number of vertices of degree k must be $C\frac{n}{k^{\alpha}}$ rounded either up or down and the number of vertices of degree

k is non-increasing with k. Note that the function $k \mapsto C\frac{1}{k^\alpha}$ is strictly decreasing.

Definition 14. *Let $\alpha > 1$ be a real number and let $C = 1/\zeta(\alpha)$ where ζ is the Riemann zeta function. $\mathcal{P}_{l,\alpha}$ is the set of graphs $G = (V, E)$ such that*

1. $\lfloor Cn \rfloor - i_1 - 1 \leq |V_1| \leq \lceil Cn \rceil$,
2. $\lfloor C\frac{n}{2^\alpha} \rfloor \leq |V_2| \leq \lceil C\frac{n}{2^\alpha} \rceil + 1$,
3. *for every i with $3 \leq i \leq n$:* $|V_i| \in \{\lfloor C\frac{n}{i^\alpha} \rfloor, \lceil C\frac{n}{i^\alpha} \rceil\}$, *and*
4. *for every i with $2 \leq i \leq n-1$:* $|V_i| \geq |V_{i+1}|$.

We usually suppress α, writing just $\mathcal{P}_l$.

Note that we allow slightly more noise in the sizes of V_1 and V_2 than in the remaining sets; without it, it seems tricky to prove a better lower bound than $\Omega(\sqrt[\alpha+1]{n})$ on label sizes.

We show the following properties of $\mathcal{P}_l$.

Proposition 1. *The maximum degree in an n-vertex graph in $\mathcal{P}_l$ is at most $\left(\frac{C}{\alpha-1} + 2\right)\sqrt[\alpha]{n} + i_1 + 3 = \Theta(\sqrt[\alpha]{n})$.*

Proof. Let $n > 0$ be an integer and let $k' = \lfloor \sqrt[\alpha]{n} \rfloor$. Furthermore, let $S_{k'} = \sum_{i=1}^{k'} |V_i|$, that is $S_{k'}$ is the number of vertices of degree at most k'. Let $S^-_{k'} = (\sum_{i=1}^{k'} \lfloor Cni^{-\alpha} \rfloor) - i_1 - 1$. Then $S_{k'} \geq S^-_{k'}$. We now bound $S^-_{k'}$ from below. For every i with $1 \leq i \leq k'$,

$$
\begin{aligned}
S^-_{k'} + k' &= -i_1 - 1 + \sum_{i=1}^{k'} \left(\left\lfloor Cni^{-\alpha} \right\rfloor + 1 \right) \\
&\geq -i_1 - 1 + \sum_{i=1}^{k'} Cni^{-\alpha} = -i_1 - 1 + Cn \sum_{i=1}^{k'} i^{-\alpha} \\
&\geq n \left(1 - C \sum_{i=k'+1}^{\infty} i^{-\alpha} \right) - i_1 - 1 \geq n \left(1 - C \int_{k'}^{\infty} x^{-\alpha} dx \right) - i_1 - 1 \\
&= n \left(1 - C \left[\frac{1}{\alpha - 1} x^{-\alpha+1} \right]_{\infty}^{k'} \right) - i_1 - 1 = n \left(1 - \frac{C}{\alpha - 1} \left(\lceil \sqrt[\alpha]{n} \rceil \right)^{-\alpha+1} \right) - i_1 - 1 \\
&\geq n \left(1 - \frac{C}{\alpha - 1} \left(\sqrt[\alpha]{n} \right)^{-\alpha+1} \right) - i_1 - 1 = n - \frac{Cn}{\alpha - 1} n^{-1+\frac{1}{\alpha}} - i_1 - 1 \\
&= n - \frac{C}{\alpha - 1} \sqrt[\alpha]{n} - i_1 - 1,
\end{aligned}
$$

giving $S_{k'} \geq S^-_{k'} \geq n - \frac{C}{\alpha-1}\sqrt[\alpha]{n} - \lceil \sqrt[\alpha]{n} \rceil - i_1 - 1$. There are thus at most $\frac{C}{\alpha-1}\sqrt[\alpha]{n} + \lfloor \sqrt[\alpha]{n} \rfloor + i_1 + 1$ vertices of degree strictly more than $k' = \lceil \sqrt[\alpha]{n} \rceil$. Since for every $1 \leq i \leq n-1$: $|V_i| \geq |V_{i+1}|$, it follows that the maximum degree of any graph in $\mathcal{P}_l$ is at most $\left(\frac{C}{\alpha-1} + 2\right)\sqrt[\alpha]{n} + i_1 + 3$. □

Proposition 2. *For $\alpha > 2$, all graphs in $\mathcal{P}_l$ are sparse.*

Proof. By Proposition 1, the maximum degree of an n-vertex graph in $\mathcal{P}_l$ graph is at most $k' \triangleq \left(\frac{C}{\alpha-1}+2\right)\sqrt[\alpha]{n}+i_1+3$, whence the total number of edges is at most $\frac{1}{2}\sum_{k=1}^{k'} k|V_k|$. By definition, $|V_k| \leq \lceil \frac{Cn}{k^\alpha} \rceil \leq \frac{Cn}{k^\alpha}+1$ for $k \neq 2$ and $|V_2| \leq \lceil \frac{Cn}{2^\alpha} \rceil + 1$, and thus

$$\begin{aligned}\frac{1}{2}\sum_{k=1}^{k'} k|V_k| &\leq 1+\frac{1}{2}\sum_{k=1}^{k'} k\left(\frac{Cn}{k^\alpha}+1\right) \leq 1+\frac{k'(k'+1)}{4}+Cn\sum_{k=1}^{\infty} k^{-\alpha+1} \\ &= O(n^{2/\alpha})+Cn\zeta(\alpha-1)=O(n).\end{aligned}$$

□

Proposition 3. *For any χ and $\alpha > 1$, $\mathcal{P}_{l,\alpha} \subseteq \mathcal{P}_{h,\chi,\alpha}$.*

Proof. Let $d = \lfloor (\frac{C}{\alpha-1}+2)\sqrt[\alpha]{n}+i_1+3 \rfloor$. For any graph in $\mathcal{P}_l$ with n vertices and for any k, $|V_k| \leq Ck^{-\alpha}n+1$ and by Proposition 1, $|V_k|=0$ when $k > d$.

Let k be an arbitrary integer between $\chi(n)$ and $n-1$. We need to show that $\sum_{i=k}^{n-1}|V_i| \leq C'(\frac{n}{k^{\alpha-1}})$. It suffices to show this for $k \leq d$. We have:

$$\begin{aligned}\sum_{i=k}^{n-1}|V_i| &\leq \sum_{i=k}^{d}(Ci^{-\alpha}n+1)=d-k+1+Cn\sum_{i=k}^{d} i^{-\alpha} \\ &\leq \left(\frac{C}{\alpha-1}+\frac{i_1}{\sqrt[\alpha]{n}}+5\right)\sqrt[\alpha]{n}+Cn\int_k^d x^{-\alpha}dx \\ &\leq \left(\frac{C}{\alpha-1}+\frac{i_1}{\sqrt[\alpha]{n}}+5\right)\sqrt[\alpha]{n}+Cn\left[\frac{1}{\alpha-1}x^{-\alpha+1}\right]_\infty^k \\ &\leq \left(\left(\frac{C}{\alpha-1}+\frac{i_1}{\sqrt[\alpha]{n}}+5\right)\left(\frac{\sqrt[\alpha]{n}d^{\alpha-1}}{n}\right)+\frac{C}{\alpha-1}\right)nk^{-\alpha+1} \\ &\leq \left(\frac{C}{\alpha-1}+\frac{i_1}{\sqrt[\alpha]{n}}+5\right)\left(\frac{C}{\alpha-1}+\frac{i_1}{\sqrt[\alpha]{n}}+5\right)^{\alpha-1}nk^{-\alpha+1}+\left(\frac{C}{\alpha-1}\right)nk^{-\alpha+1} \\ &\leq C'nk^{-\alpha+1},\end{aligned}$$

as desired. □

5.3 Comparison to other deterministic models

Numerous probabilistic and deterministic definitions of power-law graphs are given in the literature. A recent deterministic model, called shifted power-law distribution [74] has recently proven to capture a vast number of such definitions, both in theory and experimentally in [72]. We show that our definition of $\mathcal{P}_h$ contains graphs that adhere to the model, which is defined as follows. Let $c_1 > 0$ be a constant. A graph G is *power-law bounded* for parameters $\alpha > 1$ and $t \geq 0$ if for every integer $d \geq 0$, the number of vertices of G of degree in $[2^d, 2^{d+1})$ is at most

$$c_1 n(t+1)^{\alpha-1}\sum_{i=2^d}^{2^{d+1}-1}(i+t)^{-\alpha}.$$

As experimentally verified in [72], the value of t is typically very small. If $t = O(1)$, the bound above becomes $O(n\sum_{i=2^d}^{2^{d+1}-1} i^{-\alpha})$. In this case, our family $\mathcal{P}_h(\chi,\alpha)$ is rich enough to contain these power-law bounded graphs

for sufficiently large C' and any choice of χ and α. This follows since for any power-law bounded graph with n vertices and any integer k between 1 and $n-1$, $\sum_{i=k}^{n-1}|V_i| = O(\sum_{d=\lfloor \log k \rfloor}^{\lceil \log(n-1) \rceil} n \sum_{i=2^d}^{2^{d+1}-1} i^{-\alpha}) = O(\frac{n}{k^{\alpha-1}})$. Thus our upper bound also applies to power-law bounded graphs. It is possible to extend our upper bound to super-constant t where the bound is stronger the smaller t is; we omit the details. Conversely, our family $\mathcal{P}_l$ is restrictive enough that $\mathcal{P}_l$ is contained in the family of power-law bounded graphs when $t = O(1)$, and the lower bound we derive thus also holds in that setting.

5.4 The labeling schemes

We now construct algorithms for labeling schemes for c-sparse graphs and for the family $\mathcal{P}_h$. Both labeling schemes partition vertices into *thin* vertices which are of low degree and *fat* vertices of high degree. The *degree threshold* for the scheme is the lowest possible degree of a fat vertex. We start with c-sparse graphs.

Theorem 9. *There is a* $\sqrt{2cn \log n} + 2\log n + 1$ *labeling scheme for* $\mathcal{S}_{c,n}$.

Proof. Let $G = (V, E)$ be an n-vertex c-sparse graph. Let $\tau(n)$ be the degree threshold for n-vertex graphs; we choose $\tau(n)$ below. Let k denote the number of fat vertices of G, and assign each fat vertex a unique identifier between 1 and k. Each thin vertex is given a unique identifier between $k+1$ and n.

For a $v \in V$, the first part of the label $\mathcal{L}(v)$ is a single bit indicating whether v is thin or fat followed by a string of $\log n$ bits representing its identifier. If v is thin, the last part of $\mathcal{L}(v)$ is the concatenation of the identifiers of the neighbors of v. If v is fat, the last part of $\mathcal{L}(v)$ is a *fat bit string* of length k where the ith bit is 1 iff v is incident to the (fat) vertex with identifier i.

Decoding a pair $(\mathcal{L}(u), \mathcal{L}(v))$ is straightforward: if one of the vertices, say u, is thin, u and v are adjacent iff the identifier of v is part of the label of u. If both u and v are fat then they are adjacent iff the ith bit of the fat bit string of $\mathcal{L}(u)$ is 1 where i is the identifier of v. Both decoding processes can be computed in $O(\log n)$ time using standard assumptions.

Since $|E| \leq cn$, we have $k \leq 2cn/\tau(n)$. A fat vertex thus has label size $1 + \log n + k \leq 1 + \log n + 2cn/\tau(n)$ and a thin vertex has label size at most $1 + \log n + \tau(n)\log n$. To minimize the maximum possible label size, we solve $2cn/x = x \log n$. Solving this gives $x = \sqrt{2cn/\log n}$ and setting $\tau(n) = \lceil x \rceil$ gives a label size of at most $1 + \log n + (\sqrt{2cn/\log n} + 1)\log n \leq 1 + 2\log n + \sqrt{2cn \log n}$. □

By Proposition 2, graphs in $\mathcal{P}_l$ are sparse for $\alpha > 2$. This gives a label size of $O(\sqrt{n \log n})$ with the labeling scheme in Theorem 9. We now show that this label can be significantly improved, by constructing a labeling scheme for $\mathcal{P}_h$ which contains $\mathcal{P}_l$.

Theorem 10. *There is a* $\sqrt[\alpha]{C'n}(\log n)^{1-1/\alpha} + 2\log n + 1$ *labeling scheme for* $\mathcal{P}_h$.

Proof. The proof is very similar to that of Theorem 9. We let $\tau(n)$ denote the degree threshold. If we pick $\tau(n) \geq \sqrt[\alpha]{n/\log n}$ then by Definition 13 there

are at most $C'n/\tau(n)^{\alpha-1}$ fat vertices. Defining labels in the same way as in Theorem 9 gives a label size for thin vertices of at most $1+\log n+\tau(n)\log n$ and a label size for fat vertices of at most $1+\log n+C'n/\tau(n)^{\alpha-1}$. We minimize by solving $x\log n = C'n/x^{\alpha-1}$, giving $x = \sqrt[\alpha]{C'n/\log n}$. Setting $\tau(n) = \lceil x \rceil$ gives a label size of at most $\sqrt[\alpha]{C'n}(\log n)^{1-1/\alpha} + 2\log n + 1$. □

5.5 A labeling scheme for random graphs

There are schemes using randomness to "grow" graphs that, with high probability, have an approximate power-law degree distribution for a range of degrees (see e.g. [75]). For graphs obtained from such models, their degree sequences are instead probability distributions. We now show that applying our labeling scheme for $\mathcal{P}_h$ to random graphs with the power-law distribution results in a small expected worst-case label size.

Using the definition of Mitzenmacher [70], a random variable X is said to have the *power-law* distribution (w.r.t. $\alpha > 1$) if

$$Pr[X \geq x] \sim cx^{-\alpha+1},$$

for a constant $c > 0$, i.e., $\lim_{x\to\infty} Pr[X \geq x]/cx^{-\alpha+1} = 1$.

Let $\epsilon > 0$ be fixed. Consider a graph G picked from a family $\mathcal{F}$ of random graphs whose degree sequences have the power-law distribution. Order the vertices of G arbitrarily as $v_1, \ldots, v_n$. For $i = 1, \ldots, n$, let indicator variable X_i be 1 iff v_i has degree at least $d = \sqrt[\alpha]{n/\log n}$. There is a constant $N_0 \in \mathbb{N}$ (depending on ϵ) such that if $n \geq N_0$ then for all i,

$$E[X_i] = \Pr[X_i = 1] \leq (1+\epsilon)cd^{-\alpha+1}.$$

With the same labeling scheme as for $\mathcal{P}_h$ with degree threshold $\tau(n) = d$, denote by E_n the expected label size of an n-vertex graph from $\mathcal{F}$. Then for all $n \geq N_0$,

$$\begin{aligned} E_n &= \sum_{x=0}^{n} \Pr\left[\sum_{i=1}^{n} X_i = x\right] O((x + d\log n)) = O\left(d\log n + E\left[\sum_{i=1}^{n} X_i\right]\right) \\ &= O\left(d\log n + \sum_{i=1}^{n} E[X_i]\right) = O\left(d\log n + nd^{-\alpha+1}\right) = O\left(\sqrt[\alpha]{n}(\log n)^{1-1/\alpha}\right). \end{aligned}$$

Thus, we have:

Theorem 11. *Let $\mathcal{F}$ be a family of graphs with degree sequences having the power-law distribution w.r.t. $\alpha > 1$. Then there is a labeling scheme for $\mathcal{F}$ such that the expected worst-case label size of any graph $G \in \mathcal{F}$ is $O(\sqrt[\alpha]{n}(\log n)^{1-1/\alpha})$ where n is the number of vertices of G.*

5.6 Lower bounds

We now derive lower bounds for the label size of any labeling schemes for both $\mathcal{S}_{c,n}$ and $\mathcal{P}_l$. Our proofs rely on Moon's [76] lower bound of $\lfloor n/2 \rfloor$ bits for labeling scheme for general graphs. We first show that the upper bound achieved for sparse graphs is close to the best possible. The following

proposition is essentially a more precise version of the lower bound suggested by Spinrad [77].

Proposition 4. *Any labeling scheme for $\mathcal{S}_{c,n}$ requires labels of size at least $\left\lfloor \frac{\sqrt{cn}}{2} \right\rfloor$ bits.*

Proof. Assume for contradiction that there exists a labeling scheme assigning labels of size strictly less than $\lfloor \frac{\sqrt{cn}}{2} \rfloor$. Let G be an n-vertex graph. Let G' be the graph resulting by adding $\left\lfloor \frac{n^2}{c} \right\rfloor - n$ isolated vertices to G, and note that now G' is c-sparse. The graph G is an induced subgraph of G'. It now follows that the vertices of G have labels of size strictly less than $\left\lfloor \frac{\sqrt{c\lfloor n^2/c \rfloor}}{2} \right\rfloor \leq n/2$ bits. As G was arbitrary, we obtain a contradiction. □

In the remainder of this section we are assuming that $\alpha > 2$ and prove the following:

Theorem 12. *For any n, any labeling scheme for n-vertex graphs of $\mathcal{P}_{h,\chi,\alpha}$ requires label size $\Omega(\sqrt[\alpha]{n})$.*

More precisely, we present a lower bound for $\mathcal{P}_l$ which is contained in $\mathcal{P}_h$. Let $n \in \mathbb{N}$ be given and let $H = (V(H), E(H))$ be an arbitrary graph with i_1 vertices where $i_1 = \Theta(\sqrt[\alpha]{n})$ is defined as in Section 5.2. We show how to construct a graph $G = (V, E)$ in $\mathcal{P}_l$ with n vertices that contains H as an induced subgraph. Observe that a labeling of G induces a labeling of H. As H was chosen arbitrarily and as any labeling scheme for k-vertex graphs requires $\lfloor i_1/2 \rfloor$ label size in the worst case, Theorem 12 follows if we can show the existence of G.

We construct G incrementally where initially $E = \emptyset$. Partition V into subsets $V_1, \ldots, V_n$ as follows. The set V_1 has size $\lfloor Cn \rfloor - i_1$. For $i = 2, \ldots, i_1 - 1$, V_i has size $\lfloor Cn/i^\alpha \rfloor$. Letting $n' = \sum_{i=1}^{i_1-1} |V_i|$, we set the size of V_i to 1 for $i = i_1, \ldots, i_1 + n - n' - 1$ and the size of V_i to 0 for $i = i_1 + n - n', \ldots, n$, thereby ensuring that the sum of sizes of all sets is n. Observe that $\sum_{i=1}^{i_1} \lfloor Cn/i^\alpha \rfloor \leq n$ so that $n' \leq n - i_1$, implying that $n - n' \geq i_1$. Hence we have at least i_1 size 1 subsets $V_{i_1}, \ldots, V_{i_1+n-n'-1}$ in each of which the vertex degree allowed by Definition 14 is at least i_1.

Let $v_1, \ldots, v_{i_1}$ be an ordering of $V(H)$, form a set $V_H \subseteq V$ of i_1 arbitrary vertices from the sets $V_{i_1}, \ldots, V_{i_1+n-n'-1}$, and choose an ordering $v'_1, \ldots, v'_{i_1}$ of V_H. For all $i, j \in \{1, \ldots, i_1\}$, add edge (v'_i, v'_j) to E iff $(v_i, v_j) \in E(H)$. Now, H is an induced subgraph of G and since the maximum degree of H is $i_1 - 1$, no vertex of V_i exceeds the degree bound allowed by Definition 14 for $i = 1, \ldots, n$.

We next add additional edges to G in three phases to ensure that it is an element of $\mathcal{P}_l$ while maintaining the property that H is an induced subgraph of G. For $i = 1, \ldots, n$, during the construction of G we say that a vertex $v \in V_i$ is *unprocessed* if its degree in the current graph G is strictly less than i. If the degree of v is exactly i, v is *processed*.

Phase 1 Let $V' = V \setminus (V_1 \cup V_H)$. Phase 1 is as follows: while there exists a pair of unprocessed vertices $(u, v) \in V' \times V_H$, add (u, v) to E.

When Phase 1 terminates, H is clearly still an induced subgraph of G. Furthermore, all vertices of V_H are processed. To see this, note that the sum

of degrees of vertices of V_H when they are all processed is $O(i_1^2) = O(n^{2/\alpha})$ which is $o(n)$ since $\alpha > 2$. Furthermore, prior to Phase 1, each of the $\Theta(n)$ vertices of V' have degree 0 and can thus have their degrees increased by at least 1 before being processed.

Phase 2 While there exists a pair of unprocessed vertices $(u, v) \in V' \times V'$, add (u, v) to E. At termination, at most one vertex of V' remains unprocessed. If such a vertex exists we process it by connecting it to $O(\sqrt[\alpha]{n})$ vertices of V_1; as $|V_1| = \Theta(n)$ there are enough vertices of V_1 to accommodate this. Furthermore, prior to adding these edges, all vertices of V_1 have degree 0, and hence the bound allowed for vertices of this set is not exceeded.

Phase 3 We add edges between pairs of unprocessed vertices of V_1 until no such pair exists. If no unprocessed vertices remain we have the desired graph G. Otherwise, let $w \in V_1$ be the unprocessed vertex of degree 0. We add a single edge from w to another vertex w' of V_1, thereby processing w and moving w' from V_1 to V_2. Note that the sizes of V_1 and V_2 are kept in their allowed ranges due to the first two conditions in Definition 14. This proves Theorem 12.

5.7 Bypassing the lower bound

The lower bound presented can be avoided in two interesting cases. The first, for random graphs generated by a popular model, and the second using an extension of the concept of labeling schemes from the literature.

BA model As discussed in Section 5.5, generative models play an important role in the study of power-law graphs. Perhaps the most well-known generative model is the Barabási-Albert (BA) which, roughly, grows a graph in a sequence of time steps by inserting a single vertex at each step and attaching it to m existing vertices with probability weighted by the degree of each existing vertex [78]. The BA model generates graphs that asymptotically have a power-law degree distribution ($\alpha = 3$) for low-degree vertices [79]. However, graphs created by the BA model have low arboricity [1] [80]. We use this fact to devise the following highly efficient labeling scheme for such graphs.

Proposition 5. *The family of graphs generated by the BA model has an* $O(m \log n)$ Adjacency *labeling scheme.*

Proof. Let $G = (V, E)$ be an n-vertex graph resulting by the construction by the BA model with some parameter m (starting from some graph $G_0 = (V_0, E_0)$ with $|V_0| \ll n$). While it is not known how to compute the arboricity of a graph efficiently, it is possible in near-linear time to compute a partition of G with at most twice[2] the number of forests in comparison to the optimal [82]. We can thus decompose the graph to $2m$ forests in near linear time and label each forest using the recent $\log n + O(1)$ labeling scheme for trees [45], and achieve a $2m(\log n + O(1))$ labeling scheme for G. □

[1] the arboricity of a graph is the minimum number of spanning forests needed to cover its edges.

[2] More precisely, for any $\epsilon \in (0, 1)$ there exist an $O(|E(G)|/\epsilon)$ algorithm [81] that computes such partition using at most $(1 + \epsilon)$ times more forests than the optimal one.

If the encoder operates at the same time as the creation of the graph, Proposition 5 can be tightened to yield a $m \log n$ labeling scheme, by storing the identifiers of the vertices to the vertex introduced. Theorem 12 and Proposition 5 strongly suggest that local properties of power-law graphs are very different from those of a randomly generated graph using the BA model. In contrast, other generative models such as Waxman's [83], N-level Hierarchical [84]. and Chung and Liu's [75] (Chapter 3) do not seem to have an obvious smaller label size than the one in Theorem 10.

Labeling schemes with a query. The concept of labeling scheme limits the number of vertices participating in a query severely. A relaxed variant thereof, called 1-query labeling scheme [34], assumes that the decoder receives both labels queried, and may access the label of a third vertex in order to answer the query. If this is allowed, we can construct an $O(\log n)$ 1-query *Adjacency* labeling scheme for sparse (and power-law) graphs as follows: We assign each vertex v with an identifier $ID(v)$, then produce a classic [85] chaining perfect hash-function[3] from $\{1 \dots cn\}$ to $\{1 \dots n\}$, with the guarantee that the worst case number of collisions is constant. We then compute the hash function for all edges (u, v) and store the tuple $\langle ID(v), ID(u) \rangle$ in the label of the corresponding vertex. The decoder first computes the hash value resulting from $ID(v)$ and $ID(u)$ and proceed to examine if on the label corresponding to the result of the function the tuple appears. The decoder needs only to know the primary and secondary hash functions used, description thereof amount to logarithmic number of bits, which can be concatenated to each label.

5.8 A *Distance* labeling scheme

In this section we extend the usefulness of our strategy by showing a labeling scheme for small distances in power-law graphs.

For sparse graphs, Alstrup et al. [86] obtain a *Distance* labeling scheme with maximum label size $O(\frac{n}{D} \log^2 D)$ where $D = (\log n)/(\log \frac{m+n}{n})$ and m is the number of edges in the graph. Gawrychowski et al. obtain an upper bound of [87] $O(\frac{n}{D} \log D)$ with sub-linear decoding time. Few general results on lower bounds exist. The lower bound of $\Omega(\sqrt{n})$ for *Adjacency* given hereafter is trivially also a lower bound for *Distance*; for total label size, the best known lower bound remains $\Omega(n^{3/2})$ as proved by Gavoille et al. [22].

Lemma 15. *For any computable $f : \mathbb{N} \longrightarrow \mathbb{N}$ such that $f(n) \leq n - 1$ for all n, and for any $\chi(n) \geq n^{1/(alpha-1+f(n))}$ there is an $f(n)$-*Distance *labeling scheme for $\mathcal{P}_{h,\chi,\alpha}$ that assigns labels of length at most $O(n^{f(n)/(f(n)+1)} \log f(n))$.*

Proof. Let G be a graph in $\mathcal{P}_{h,\chi,\alpha}$. A vertex of G is *fat* if it has degree at least $n^{1/(\alpha-1+f(n))}$ and *thin* otherwise. The label of each vertex v contains (i) a table of distances to all fat vertices (if the distance is more than $f(n)$, it is simply ignored), (ii) a table of distances to all thin vertices w that are at most distance $f(n)$ away from v where the shortest path between v and w does not pass through any fat vertex, and (iii) a single bit signifiying whether the vertex is fat or thin. Clearly, as $f(n)$ is computable and distances in G are computable, there is a computable encoder assigning labels. A decoder

[3] To this end, we may for example first partition the domain into c parts.

can now compute the distance between any two vertices u, v as follows: If both u or v are fat, the distance can be directly read off part (i) of the label of any vertex. If at least one of u and v is fat, the distance can be read off part (i) of the label of the thin vertex. If both vertices are thin, the decoder can check if the distance is in part (ii) of the label of either vertex; if the distance is not present, either the distance is strictly greater than $f(n)$, or the shortest path between u and v passes through a fat vertex; in this case, the decoder may brute-force check the distances from u and v to each fat vertex, and output the smallest sum of these two distances.

Furthermore, as all vertices of G are either thin or fat, it is clearly possible for an encoder to compute all distances less than or equal to $f(n)$ between any pair of vertices. Note that as all distances we care for are bounded above by $f(n)$, each such distance can be stored using at most $\log f(n)$ bits.

As $G = G(V, E)$ is in $\mathcal{P}_{h,\chi,\alpha}$, we have

$$\begin{aligned}\sum_{i=\chi(n)}^{n-1} |V_i| \leq \sum_{i=n^{\frac{1}{\alpha-1+f(n)}}}^{n-1} |V_i| &\leq C' \left(\frac{n}{\left(n^{\frac{1}{\alpha-1+f(n)}}\right)^{\alpha-1}} \right) \\ &\leq C' n^{1-(\alpha-1)/\alpha-1+f(n)} = C' n^{f(n)/(\alpha-1+f(n))}\end{aligned}$$

Thus, a table of distances to all fat vertices takes up at most

$$O\left(n^{\frac{f(n)}{\alpha-1+f(n)}} \log f(n)\right) \text{ bits.}$$

Similarly, for each vertex v there are at most $\left(n^{1/(\alpha-1+f(n))}\right)^{f(n)}$ $= n^{f(n)/(\alpha-1+f(n))}$ vertices at distance at most $f(n)$ away from v where the shortest path consists only of thin vertices. Hence, the associated table of distances takes up at most
$O(n^{f(n)/(\alpha-1+f(n))} \log n)$ bits.

In total, each label thus has size at most $O(n^{f(n)/(f(n)+1)} \log n)$ bits. □

For $f(n) = \log n$, Lemma 15 yields labels of size $O\left(n^{(\log n)/(\alpha-1+\log n)} \log \log n\right)$. Unsurprisingly, as we are only considering distances up to $f(n)$, this label size is asymptotically smaller than for the labeling schemes working for all distances in *sparse* graphs, e.g. the largest label sizes of [87] for sparse graphs is $O(n\frac{\log \log n}{\log n})$. For power-law random graphs, Chung and Lu show in [73] that, subject to mild conditions, the diameter of power-law graphs with $\alpha > 2$ is almost surely $\Theta(\log n)$. We thus expect our labeling scheme to have superior performance for such graphs.

5.9 Experimental study

We now perform an experimental evaluation of our labeling scheme on a number of large networks. The source code for our experiments can be found at: www.diku.dk/~simonsen/suppmat/podc15/powerlaw.zip

Experimental framework

Performance indicators. Recall that our labeling scheme consists of two ideas: separation of the vertices according to some threshold, and selecting a

threshold depending on the power-law parameter α. In our labeling scheme, the threshold is $\lceil \sqrt[\alpha]{Cn/(\alpha-1)} \rceil$. We call this the *predicted* threshold; it is an approximation to the theoretically optimal threshold choice when degree distributions follow the power-law curve $k \mapsto Cn/k^\alpha$ perfectly. The approximation uses integration similar to what is done in, e.g., the proof of Proposition 3. For a concrete graph G, it is conceivable that some other threshold n_0, different from the predicted threshold, would yield a labeling scheme with smaller size. Let $\max_t(n_0)$ and $\max_f(n_0)$ be the maximum label sizes of thin, resp. fat vertices in G when the threshold is set at $1 \leq n_0 \leq n-1$. Clearly the maximum label size with the threshold n_0 is $\max\{\max_t(n_0), \max_f(n_0)\}$. Observe further that $\max_t(n_0)$ and $\max_f(n_0)$ are monotonically increasing, resp. decreasing functions of n_0. Hence, the n_0 for which $\max\{\max_t(n_0), \max_f(n_0)\}$ is minimal is where the curves of $\max_t(n_0)$ and $\max_f(n_0)$ intersect. We call this n_0 the *empirical* threshold. We set up the following performance indicators to gauge (1) the difference in label size with predicted and empirical threshold, and (2) the label size obtained by our labeling scheme on several data sets, compared to other labeling schemes.

Performance Indicator 1: We measure the label sizes for the labeling schemes with the predicted and empirical thresholds. We interpret a small relative difference between these label sizes means that the predicted threshold can achieve small label sizes without examining the global properties of the network other than the power-law parameter α.

Performance Indicator 2: We measure the label sizes attained by our labeling schemes to other labeling schemes, namely state-of-the art labeling schemes for the classes of bounded-degree, sparse and general graphs using the labeling schemes suggested in [7], Theorem 9 and [64]. We interpret small label sizes for our scheme, especially in comparison with "small" classes like the class of bounded-degree graphs, as a sign that our labeling scheme efficiently utilizes the extra information about the graphs: namely that their degree distribution is reasonably well-approximated by a power-law.

Test sets. We employ both real-world and synthetic data sets.

The six *synthetic* data sets are created by first generating a power-law degree sequence using the method of Clauset et al. [69, App. D], subsequently constructing a corresponding graph for the sequence using the Havel-Hakimi method [88]. We use the range $2 < \alpha < 3$ as suggested in [69] as this range of α occurs most commonly in modeling of real-world networks. We generate graphs of $300,000$ and $1M.$ vertices denoted $\text{s300}^{\alpha=x}$ and $\text{s1M}^{\alpha=x}$ respectively, for $x \in \{2.2, 2.4, 2.6, 2.8\}$.

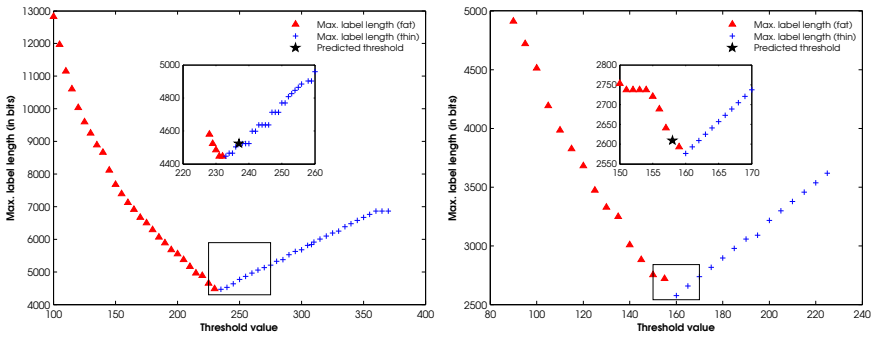
The three *real-world* data sets originate from articles that found the data to be well-approximated by a power-law. The WWW data set [89] contains information on links between webpages within the nd.edu domain. The ENRON data set [90] contains email communication between Enron employees (vertices are email addresses; there is a link between two addresses if a mail has been sent between them). The INTERNET data set [91] provides a snapshot the Internet structure at the level of autonomous systems, reconstructed from BGP tables. For all of these sets, we consider the underlying simple, undirected graphs. For each set, standard maximum likelihood methods were used to compute the parameter α of the best-fitting power-law curve [69]. Additional information on the data sets can be found in Table 5.1.

Real-Life					
Data set	$\|V\|$	$\|E\|$	α	$\Delta_{\max}$	Source
WWW	325,729	1,117,563	2.16	10,721	[89]
ENRON	36,692	183,830	1.97	1,383	[90]
INTERNET	22,963	48,436	2.09	2,390	[91]
Synthetic					
$s1M^{\alpha=2.4}$	1,000,000	1,127,797	2.4	42,683	–
$s1M^{\alpha=2.6}$	1,000,000	878,472	2.6	12,169	–
$s1M^{\alpha=2.8}$	1,000,000	751,784	2.8	1,692	–
$s300^{\alpha=2.2}$	300,000	491,926	2.2	10,906	–
$s300^{\alpha=2.4}$	300,000	327,631	2.4	3,265	–
$s300^{\alpha=2.6}$	300,000	261,949	2.6	1,410	–
$s300^{\alpha=2.8}$	300,000	227,247	2.8	1,842	–

TABLE 5.1: Data sets and their properties. All graphs are undirected and simple. $\Delta_{\max}$ is the maximum degree of any vertex in the data set.

Findings

Figure 5.2 shows the distribution of maximum label sizes for one synthetic and one real-world data set. The maximum label size for the predicted and empirical thresholds as well as upper bounds on the label sizes from different label schemes in the literature can be seen in Table 5.2 for two synthetic data sets and all three real-world data sets.



(A) $syn300^{\alpha=2.2}$ (B) ENRON

FIGURE 5.2: Maximum label sizes of different threshold values for the $syn300^{\alpha=2.2}$ and ENRON data sets. The triangles and crosses represent that for the tested threshold the largest label belong to fat, resp. thin vertex. The star indicate the position of the predicted threshold.

Table 5.2 shows the maximum label sizes achieved using different labeling schemes on our data sets. "Predicted" shows the experimental maximum label size obtained by running our scheme on the graphs, "Empirical" is the label size attained by using the empirical threshold. The remaining columns show non-experimental upper bounds for different label schemes: "Bound" is the upper bound guaranteed in Theorem 10, "C-sparse" is the labeling scheme for sparse graphs defined in Theorem 9, "BD" is the $\lceil \frac{\Delta}{2} \rceil \lceil \log n \rceil$ bounded degree graph labeling of [7], and AKTZ is the $\lceil n/2 \rceil + 6$ general graph labeling

of [64]. Both "Empirical" and "Bound" using simple concatenation of labels to represent the fat bit string[4].

Data set	Predicted	Empirical	Bound	C-sparse	BD [7]	AKTZ [64]
s1M$^{\alpha=2.4}$	4,841	4,821	25,012	30,079	426,820	500,006
s1M$^{\alpha=2.6}$	3,361	3,201	15,282	26,551	121,680	500,006
s1M$^{\alpha=2.8}$	2,101	2,061	10,081	24,566	16,920	500,006
s300$^{\alpha=2.2}$	4,523	4,447	24,878	18,885	103,607	150,006
s300$^{\alpha=2.4}$	2,775	2,680	14,404	15,420	31,008	150,006
s300$^{\alpha=2.6}$	1,958	1,920	9,151	13,792	13,395	150,006
s300$^{\alpha=2.8}$	1,350	1,312	6,244	12,849	17,499	150,006
WWW	5,245	3,060	29,225	28,445	101,840	162,870
ENRON	2,609	2,577	15,835	9,735	11,056	18,352
INTERNET	1,426	1,156	8,181	4,700	17,925	11,487

TABLE 5.2: Label size in bits of labeling schemes. The two leftmost columns are experimental results; the remaining are upper bounds on label sizes computed from the characteristics of the data sets.

Our findings are as follows. For Performance Indicator (i), our labeling scheme obtains maximum label size at most 3% larger than what would have been obtained by using the empirical threshold for all synthetic data sets. This is expected—the synthetic data sets are graphs generated specifically to have power-law distributed degree distribution. For the real-world data sets, the labeling scheme obtains maximum label size at most 23% larger than by using the empirical threshold; this larger deviation is likely due to degree distributions of the data sets being close to, but not quite, power-law distributions due to natural phenomena or noise. E.g., for the ENRON data set there is sudden drop in frequency between vertices of degree < 158 and ≥ 158.

For Performance Indicator (ii), both our experimental results and theoretical upper bounds for our labeling scheme are several orders of magnitudes lower than for labeling schemes aimed at more general classes of graphs, as expected. Of the more general classes of graphs, it is most interesting to compare the upper bound of bounded degree graphs—the most restrictive class of graphs that both contains the class of power-law graphs and has an efficient labeling scheme described in the literature [7]. As seen in Table 5.2, the upper bound on our labeling schemes for both power-law graphs and sparse graphs have better upper bounds on label sizes, but only marginally so for data sets with low maximum degree and low values of the power-law parameter α, e.g. ENRON ($\alpha = 1.97$). It is interesting to note that the actual label sizes obtained in the experiments (the two leftmost columns of Table 5.2) are substantially lower than the upper bounds, that is, the labeling scheme performs much better in practice than suggested by theory (down to less than a kilobyte per vertex for all data sets). This phenomenon may be due to the degree distribution of the graphs of the data sets having only minor deviation from a power-law for small vertex degrees; our upper bounds on the label size are derived by using the very rich family $\mathcal{P}_h$ that allows very large deviation from a power-law for degrees between 1 and $\sqrt[\alpha]{n/\log n} - 1$.

[4]Our labeling schemes introduced in this paper all make use of a succinctly represented "fat bit string"; for our experiments, we use simple concatenation of labels instead of a bit string; this incurs a $(\log n)/\alpha$ factor on the label size, but simplifies the implementation.

Finally, note that our labeling scheme supports *Adjacency* for *directed* graphs by using one more bit per edge in each label to store the edge orientation. For data sets whose natural interpretation is as a directed graph (e.g., the WWW set where edges are outgoing and incoming links), the results of Table 5.2 thus carry over with just one more bit added to the numbers in the two leftmost columns.

5.10 Conclusion and future work

We have devised *Adjacency* and *Distance* labeling schemes for sparse graphs and graphs whose degree distribution approximately follows a power-law distribution. We have proven lower bounds for the class of power-law graphs showing that our strategy for *Adjacency* labeling scheme is almost optimal, and showed two relaxations that allow for logarithmic size labels.

We propose the following directions:

- Our labeling schemes are designed for static networks, and while it seems not difficult to extend our idea to dynamic networks, an analysis is required to account for the communication and number of re-labels incurred by such an extension.
- Labeling schemes for power-law graphs can likely be devised for the realistic case where the scheme only has incomplete knowledge of the graph, for example when the expected frequency of vertices of each degree is known, but not the exact frequency of each vertex.
- Closing the gap of the multiplicative logarithmic factor may be of interest to the theory community. A more interesting gap exists for *Distance* labeling schemes. As we have seen, there is a large gap between labeling schemes for short *Distance* and *Adjacency* for power-law (and sparse) graphs. This gap effectively deemed the *Distance* labels uninteresting for practical applications.
- Finally, while power-law distributions may model the degree distribution of real-world networks, other distributions may fit better (see, e.g., [69]); it is interesting to see whether refinements of our labeling scheme that utilize knowledge about such distributions would result in superior labeling schemes for real-world data.

Chapter 6

On the Implicit Representation Conjecture

We begin this section with an overview of a connection between labeling schemes and induced universal graph in Section 6.1. We then introduce the implicit graph conjecture in Section 6.2 and present three unpublished and perhaps somewhat unpolished results on this conjecture in Sections 6.3 to 6.5.

6.1 Introduction

A graph U is induced universal for an n vertex graph family $\mathcal{F}$ if it contains each of the graphs in $\mathcal{F}$ an induced subgraphs[1]. A large body of work established in the attempt to find graphs of smallest number of vertices for various graph, most notably graphs [76], and trees [92].

Kannan Naor and Rudich stated the following theorem:

Theorem 13. *If a graph family has a $k \log n$* Adjacency *labeling scheme, then it has a universal graph of size n^k constructible in polynomial time.*

To prove this theorem, the authors construct an n^k vertex graph such that each vertex corresponds to one of the possible strings produced using $k \log n$ bits. The edges of this induced universal graph are inferred by querying all pairs of vertices on the decoder of the labeling scheme. As illustration, we demonstrate the process in Figure 6.1 for the labeling scheme in Section 1.1.

[1] To understand the difference between subgraph and induced subgraph: an edge from a graph creates a subgraph, removal of a vertex and all edges adjacent to it creates an induced subgraph.

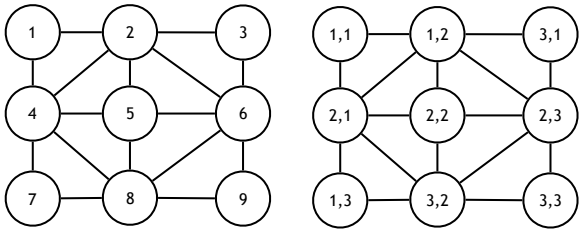


FIGURE 6.1: Constructing the 3^2 induced universal graph from Theorem 13 for 3 vertex trees according to the encoder of the *Adjacency* labeling scheme in Section 1.1. Left: the original numbering and right: the numbers parsed according to Section 1.1.

This theorem implies that *Adjacency* labeling schemes are explicit constructors of induced universal graphs. It is thus not surprising that this important graph theoretic concept served as a major incentive to determine the precise label size required.

The opposite relation holds in a weaker sense. The existence of an induced-universal graph with $2^{f(n)}$ vertices for a family $\mathcal{F}$ of graphs implies the existence of labeling scheme with size $f(n)$. Such transformation is however not efficient, namely the resulting scheme has exponential running time. As seen in Chapter 4, it can be highly non trivial to construct an efficient $f(n)$ *Adjacency* labeling scheme for a graph family, even when an induced universal graph of size $2^{f(n)}$ exists. We shall also see furhter in Section 6.4 a case in which an exponential decoding time is necessary.

A variant of this concept is the concept of induced universal trees, which are trees that contain all n-vertex trees as induced subgraphs [92]. In this case an induced universal tree of size $2^{f(n)}$ implies a *Distance* labeling scheme of size $f(n)$. Recently, Freedman et al. [48] proved that the converse is not true, namely that there exists a *Distance* labeling scheme for trees small enough that it beats the lower bound [93] on the size of an induced universal tree for n vertex trees. Lastly, a connection between universal matrices and *Distance* labeling schemes was studied by Korman et al. [94] and by Gavoille and Paul [95].

Zeitgeist

This thesis was written between 2013 – 2016. As evident in Chapters 3 and 4, a good amount of progress was done in this period for labeling schemes/induced universal graphs for trees. We also pioneered the study for power law graphs in Chapter 5. We mention that this was also a fruitful period for other important families, namely general and bounded degree graphs:

- General graphs: Alstrup, Kaplan, Thorup and Zwick successfully shaved the additive $\log n$ factor from the label size for general graphs and showed an *Adjacency* labeling scheme of size $\lceil n/2 \rceil + 4$ [64]. Two years later Alon showed [98], using the probabilistic method, a non constructive proof that a $\lceil n/2 \rceil - 1$ *Adjacency* labeling scheme is possible for general graphs, essentially proving that the lower bound from 1965 [76] was in fact the correct answer. The techniques in both papers other improved also the best known results for directed graphs, tournaments and bipartite graphs.

- Bounded degree graphs: Following our results for bounded degree $k(n)$ graphs (Section 4.9), Abrahamsen, Alstrup, Holm, Knudsen and Stockel [96] improved the bound from $\log \binom{n}{\lceil k/2 \rceil} + \lceil \log n \rceil + \lceil \log k \rceil$ to $\binom{\lfloor n/2 \rfloor}{\lfloor k/2 \rfloor} \cdot 2^{O(\sqrt{k \log n} \log(n/k))}$ and showed a fairly close lower bound of $\binom{\lfloor n/2 \rfloor}{\lfloor k/2 \rfloor} \cdot 2^{-O(\sqrt{k \log n} \log(n/k))}$. In the same time, the label size on bounded degree graphs for constant degree was determined to the bit by Alon and Nenadov [97]. In the remainder of this thesis we shall use the term "air-tight" to describe the quality of such labeling schemes.

The best known bounds on the size of induced universal graphs for important graph families are found in Table 6.1.

TABLE 6.1: Best known results for the size of induced universal graphs for particular graph families. The references concern the upper bound. Excluding [98], each upper bound $f(n)$ mentioned in this table has a labeling scheme of size $\log(f(n))$.

Graph family	Lower bound	Upper bound	Reference
General graphs	$2^{\frac{n-1}{2}}$	$(1+o(1))2^{\frac{n-1}{2}}$	Alon [98]
Tournaments	$2^{\frac{n-1}{2}}$	$(1+o(1))2^{\frac{n-1}{2}}$	Alon [98]
Bipartite graphs	$\Omega(2^{\frac{n}{4}})$	$(1+o(1))2^{\frac{n}{4}}$	Alon [98]
Graphs of at most cn edges	$\lceil\frac{\sqrt{cn}}{2}\rceil$	$\sqrt{2cn\log n}+O(\log n)$	Petersen et al. [10]
α-Power-law Graphs	$\Omega(\sqrt[\alpha]{n})$	$O(\sqrt[\alpha]{n}(logn)^{1-1/\alpha})$	Petersen et al. [10]
Graphs of max degree $\Delta(n)$[2]	$\binom{\lfloor n/2\rfloor}{\lfloor\Delta/2\rfloor}\cdot 2^{-O(R)}$	$\binom{\lfloor n/2\rfloor}{\lfloor\Delta/2\rfloor}\cdot 2^{O(R)}$	Abrahamsen et al. [96]
Graphs of max degree Δ	$\Omega(n^{\frac{\Delta}{2}})$	$c\Delta n^{\Delta/2}$	Alon and Nenadov [97]
Graphs of max degree 2	$\lceil\frac{11n}{6}\rceil$ [99]	$2n-1$	Abrahamsen et al. [96]
Graphs excluding a fixed minor	$\Omega(n)$	$n^2(\log n)^{O(1)}$	Gavoille and Labourel [6]
Planar graphs	$\Omega(n)$	$n^2\log n^{O(1)}$	Gavoille and Labourel [6]
Planar graphs of max degree Δ	$\Omega(n)$	$O(n^2)$	Bhatt et al. [62]
Planar graphs of max degree 4	$\Omega(n)$	$O(n^{\frac{3}{2}})$	Adjiashvili and Rotbart [7]
Outerplanar graphs	$\Omega(n)$	$n(\log n)^{O(1)}$	Gavoille and Labourel [6]
Outerplanar graphs of max degree Δ	$\Omega(n)$		Bhatt et al. [62]
Graphs of treewidth k	$n2^{\Omega(k)}$	$n(\log\frac{n}{k})^{O(k)}$	Gavoille and Labourel [6]
Graphs of arboricity k	$\frac{n^k}{2^{O(k^2)}}$	n^k	Alstrup et al. [8, 45]
Forests	$1.5n$	$O(n)$	Alstrup et al. [45]
Forests of bounded degree Δ	n	$O(n)$	Adjiashvili and Rotbart [7]
Trees of bounded depth δ	n	$O(n\delta^3)$	Fraigniaud and Korman [100]
Caterpillars	$1.5n$	$12n$	Alstrup et al. [45]

6.2 The implicit representation conjecture

A careful look at Table 6.1 will show a separation between graph families to some that have $O(\log n)$ labels, and some who have labels much larger than that. We say that a graph family has an *implicit representation* if it has an $O(\log n)$ *Adjacency* labeling scheme. Recall that the number of all graphs is bounded by $2^{\Theta(n^2)}$. The study of graph families is assisted by the terminology *speed* (introduced in [101]) to describe the number of graphs in an n vertex graph family as a function of n. A graph family of $f(n)$ graphs has constant speed if $\log f(n)=\Theta(1)$, polynomial if $\log f(n)=\Theta(\log n)$, exponential if $\log f(n)=\Theta(n)$, factorial if $\log f(n)=\Theta(n\log n)$ and super-factorial for larger speeds (up to $\log f(n)=\Theta(n^2)$). If a graph family has a super-factorial speed it can not have an implicit representation for the following reason. The union of all labels defines a unique graph from the graph family, and can be expressed using a bit string of $O(n\log n)$ bits. Since there are $2^{O(n\log n)}$ possible bit strings, and strictly more (distinct) graphs in such a family, it follows that it can not have an implicit representation. Is this restriction enough to guarantee that the graph family has an implicit representation? A brief look at the lower bound for the factorial speed family of sparse graphs

[2] We set $R=\sqrt{\Delta\log n}\log(n/\Delta)$.

in Table 6.1 (proved in Theorem 12) shows that this is not the case. Notice that a small subgraph of a sparse graph can have an arbitrary edge set without violating the graph being sparse, forcing labels in this small group to have relatively large labels. To stop this abuse, Kannan et al. [2] added the *hereditary* requirement. A graph family is *hereditary* if for every graph, all its induced subgraphs are also in this family. In common to all known implicitly representable graph families are these two properties, namely hereditary and of speed at most factorial. In the remainder of this section we denote such graph families as *suitable* graph families. Are all suitable graph families have an implicit representation? This interesting question was stated first by Kannan et al. [2] and posed as a conjecture by Spinrad [77], which we now write in a modified form:

Conjecture 1 (The implicit representation conjecture)**.** *Every hereditary family of unlabeled graphs of size* $2^{O(n \log n)}$ *has an* $O(\log n)$ Adjacency *labeling scheme.*

We modified the original conjecture statement in two ways:

- We added the requirement that the graphs are unlabeled. As a concrete example of the necessity of our addition of this detail we observe the speed of trees. The speed of labeled trees is n^{n-2} [102] (superfactorial) whereas unlabeled trees have factorial speed [103]. Moreover, the connection between labeling schemes and induced universal graphs from Section 6.1 applies for families of unlabeled graphs. While we have failed to find a version of the conjecture statement that contains explicitly the term unlabeled graph families, we regard it as such. See Section 6.5 for additional discussion on this difference.
- We omitted the requirement stated in the original statement [2] that the encoding and decoding of such labeling schemes is performed in polynomial time. We prove that this (minor) detail does not hold in Section 6.4.

6.3 Segment intersection graphs

Among the suitable graph families labeling scheme of which remains unresolved are several geometric graph families such as segment intersection graphs [104], unit-disk graphs [105] and dot-product graphs [106]. Unlike previously studied suited graph families, these families are not bounded by any familiar parameter[3]. Proving that these graph families are factorial is by itself non-trivial and relies on a reduction to the number theoretic *Warran's theroem* [107].

In this section we elaborate on the family of segment intersection graphs.

Definition 15. *Let* $\mathcal{S}$ *be a class of line segments in the Euclidean plane, then the family of intersection graphs of* $\mathcal{S}$ *is the class of all graphs isomorphic to graphs of the form* $G = (V, E)$ *where* $V \subseteq \mathcal{S}$ *and for* $u, v \in V$, $e = uv \in E$ *iff* $u \cup v \neq \emptyset$.

A restricted subclass of this family is the group of families $k - DIR$. For a fixed k real number $d_1, \ldots, d_k$, the family $k - DIR(d_1, \ldots, d_k)$ is defined

[3] See e.g. http://graphclasses.org/classes/gc_389.html.

as the family of intersection graph where all segments have slopes among $d_1, \ldots, d_k$. As a private example of such graph family, if we set $k = 1$ we get *interval graphs* [108], for which the following simple $2\log n + 2$ exists. The endpoints of the intervals representing the graph are assigned integer endpoints in the range $1, \ldots, 2n$, corresponding to their ordering. The label of each segment consists of the ordering of both its endpoints, using at most $2\log n + 2$ bits. *Adjacency* testing between two intervals is determined by range inclusion testing.

One may ask if geometric representation of a segment graph can be transformed directly into an implicit representation. Even though the segments may be defined over the reals, every segment graph can be realized by segments with rational endpoints, as the graphs discussed are finite. Assigning each vertex its segments rational endpoints (four integers) is a valid labeling scheme, operating in the same manner as the one discussed for interval graphs. If each such point can be described using a number bounded by a polynomial of n, then the resulting labels are of logarithmic size. This approach to implicit representation was proven to fail, as Matousek [104] proved that the precision needed to realize these graphs is double exponential in the number of segments. McDiarmid and Müller [109] proved in addition that this bound is tight. Interestingly, the lower bound is achieved using $3-DIR$ graphs, meaning that this approach fails already for this very restrictive family. In contrast we now prove the following:

Theorem 14. *There exist a* $2k\log n + \log k + 2k$ Adjacency *labeling scheme for the family* $k - DIR$.

Proof. A segment a is represented by two points a_1, a_2 ($2n$ points in total), and its label is a concatenation of the labels of each point, along with additional $\log k$ bits to describe the type of its slope. The label $\mathcal{L}(p)$ of a point p consists of k times $\log 2n$ size bit strings described as follows. We perform a scan for each slope $d_1, \ldots, d_k$, such that each scan begins with all points on one half plane. A point p gets a $\log(2n)$ sub-label $\mathcal{L}(p[d_i])$, which is the rank of p in the ordering defined by the scan d_i. A scan will be performed from the top left to the bottom right of the plane. The points of segments with similar slope as a scan will encounter it simultaneously and are labeled such that a point closer to the top has the lower label [4]. Note that this way, non endpoints on a segment receive labels according to their order on this segment.

Let $a = (a_1, a_2)$ and $b = (b_1, b_2)$ be two segments with slopes d_a and d_b respectively. If $d_a = d_b$, the decoder description is similar to the one from the labeling scheme for interval graphs. If $d_a \neq d_b$, we assume w.l.o.g that the scan d_b encounters a_1 before a_2 and b_1 before b_2 and that d_a encounters b_1 before b_2 and a_1 before a_2. We prove that the two segments intersect if and only if

$$\underbrace{\mathcal{L}(a_1[d_b]) < \mathcal{L}(b_1[d_b]) < \mathcal{L}(a_2[d_b])}_{\text{Condition 1}} \text{ and } \underbrace{\mathcal{L}(b_1[d_a]) < \mathcal{L}(a_1[d_a]) < \mathcal{L}(b_2[d_a])}_{\text{Condition 2}}.$$

[4] In the exception of a total horizontal line, the ordering will be such that the point closer to the left gets the lower number.

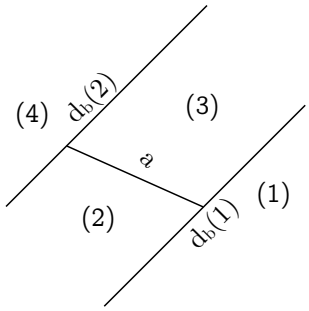


FIGURE 6.2: A division of the plane described in the proof of Theorem 14. Parts (1) and (4) are only labeled in the drawing as they are not essential to the proof.

To prove that it is necessary, suppose first that condition 1 is not met. In this case the scan d_b encountered all of segment a completely before or completely after meeting segment b, implying that the segments can not intersect. The argument is similar if we assume that condition 2 is not met.

To prove that it is sufficient, we divide the plane into four parts using two lines $d_b(1)$, and $d_b(2)$, both with slope d_b, touching segment a at its endpoints a_1 and a_2 respectively. We call the parts that divide the strip induced by the lines $d_b(1)$ and $d_b(2)$ as parts (2) and (3), where (2) is below segment a and (3) is above it. See Figure 6.2 for illustration. From condition 1 we know that segment b can only lie in parts (2) and (3) of the plane. Every point p in (2) must have $\mathcal{L}(p[d_a]) < \mathcal{L}(a_1[d_a])$ and every point p_2 in (3) must have $\mathcal{L}(a_1[d_a]) < \mathcal{L}(p_2[d_a])$. Combining this with condition 2 we get that b_1 must be in (2) and b_2 must be in (3), which proves that a and b intersect.

□

6.4 Non polynomially decodable implicit graph classes

The conjecture in its original statement required an encoding and decoding time that are polynomial in the size of the graph and label respectively. While this requirement does not seem crucial for the conjecture, in this section we prove the following theorem:

Theorem 15. *There is a family of finite graphs with implicit representation, if the decoder is allowed to run in time $O(2^{n^2})$, and has no implicit representation if the decoder has time strictly less than $O(2^n)$.*

This result in this section is mostly due to Jakob Grue Simonsen, and much to our surprise, was reinvented a year later and published in an extended form by Chandoo [110].

6.4.1 Preliminaries

We assume that Turing machines are described by a suitable binary coding. If $M \in \{0,1\}^+$ is a Turing machine, we let $\phi_M : \{0,1\}^+ \rightharpoonup \{0,1\}^+$ denote the partial function computed by M. We assume a Turing machine may take several arguments by suitable binary coding of pairs, e.g. writing $\phi_M(a.b)$ when M gets more than one argument. If M is run on empty input, we denote this by empty parentheses, e.g. $\phi_M()$.

We fix a universal Turing machine U that is able to simulate every Turing machine with at most linear overhead, that is,

- $\forall M.\forall x \in \{0,1\}^+.\phi_U(M.x) = \phi_M(x)$ (universality), and
- $\exists N \in \mathbb{N}.\forall M.\forall x \in \{0,1\}^+$, U simulates one step of M on input x using at most $N \cdot |M.x|$ steps (linear overhead)

If $n \in \mathbb{N}$ we denote by $\langle n \rangle$ some standard binary representation of n such that $|\langle n \rangle| = O(\log n)$.

The *Kolmogorov complexity* of $x \in \{0,1\}^+$ (wrt. U) is

$$K_U(x) = \min\{|M|.\phi_U(M) = x\}$$

that is, $K_U(x)$ is the length of a shortest (string encoding a) Turing machine that, when run on empty input, produces x. By standard results in Kolmogorov complexity (the so-called "Invariance Theorem"), if U' is a universal Turing machine, there exists a constant $C_{U'}$ such that, for all $x \in \{0,1\}^+$, $K_{U'}(x) \leq K_U(x) + C_{U'}$. Hence, changing universal machine incurs at most a bounded difference in Kolmogorov complexity which is independent of x. We shall therefore merely write $K(x)$ instead of $K_U(x)$.

6.4.2 The construction

The following lemma is a variation of standard results from resource-bounded Kolmogorov complexity [111].

Lemma 16. *There is a Turing machine Q such that:*

- *For each n, Q, on input $\langle n \rangle$, outputs a string, s_n, of length $n(n-1)/2$*
- *Q runs in time $O(2^{n^2})$.*
- *For each $n \in \mathbb{N}$, there is no Turing machine R such that (i) $\text{TIME}_R() < 2^n$, (ii) $|R| < n(n-1)/2$, and (iii) $\phi_R() = s_n$.*

Proof. On input $\langle n \rangle$, Q successively generates each binary string y of length at most $n(n-1)/2 - 1$ and then uses a copy of the universal machine U as a subroutine to simulate U running on input y for $2^n - 1$ steps. As U simulates each (the Turing machine encoded by) y with linear overhead, each such simulation takes at most $N \cdot |y|(2^n - 1)$ steps where N is a constant independent of y. Hence, the total time taken to simulate all strings is $O(2^{n(n-1)/2} \cdot (n(n-1)/2 - 1)(2^n - 1)) = O(2^{n^2})$; the overhead needed to generate each string and other housekeeping operations is negligible compared to this.

During the simulation, Q stores all strings of length exactly $n(n-1)/2$ as output by strings y during the simulation in a set S (i.e., if $|\phi_U(y)| = n(n-1)/2$ and y runs for at most $2^n - 1$ steps on empty input, Q stores $\phi_U(y)$. When all strings have been simulated, Q outputs the lexicographically smallest string of length $n(n-1)/2$ that is *not* in S.

By construction, Q always terminates with output a string of length $n(n-1)/2$ and runs in time $O(2^{n^2})$. Suppose, for contradiction, that there were an $n \in \mathbb{N}$ and a Turing machine R such that (i) $\text{TIME}_R() < 2^n$, (ii) $|R| < n(n-1)/2$, and (iii) $\phi_R() = s_n$. By construction, Q simulates R for exactly $2^n - 1$ steps, and thus $\phi_R() = \phi_U(R) = s_n \in S$. But by construction of Q it follows $s_n \notin S$, which contradicts the assumption. □

Note that $|\langle n\rangle| = O(\log n)$, so the time complexity of Q is $O\left(2^{2^{2\log n}}\right)$, that is, doubly exponential in the square of its input size.

Theorem 16. *There is a family $\mathcal{F}$ of graphs such that:*

- *$\mathcal{F}$ has a labeling scheme such that every element of $\mathcal{F}_n$ has maximum label size $O(\log n)$, the encoding is computable, and the decoder runs in time $O(2^{n^2})$.*
- *If a labeling scheme for $\mathcal{F}$ (i) has maximum label size $\leq n/2 - 2$ for infinitely many n, then no decoder for the labeling scheme can run in polynomial time in n.*

Proof. There is a computable bijection between the set of (labeled) simple, undirected graphs with n vertices (given by an $n \times n$ symmetric adjacency matrix with zero diagonal) and the set of binary strings of length $n(n-1)/2$: if s is such a string, the first $n-1$ bits is the first row above the diagonal of the adjacency matrix, the next $n-2$ bits the second row, and so forth.

Thus, the Turing machine Q of Lemma 16 produces, for each n, an adjacency matrix for a graph of n vertices. We set $\mathcal{F}_n = \{\phi_Q(\langle n\rangle) : n \in \mathbb{N}\} = \{s_n : n \in \mathbb{N}\}$. Note that $\mathcal{F}_n$ contains exactly one labeled graph[5].

We prove the claims of the theorem in turn:

- Each element $G \in \mathcal{F}$ can be labeled by a Turing machine as follows: Each vertex receives a label comprising (i) a representation of n (using $\log n$ bits), and (ii) an identifier of the vertex (using $\log n$ bits), that is, a total of $O(\log n)$ bits. Furthermore, a decoder running in time $O(2^{n^2})$ can be devised as follows: Given labels of two vertices, the decoder first decodes $\log n$ bits to obtain n, then runs a copy of Q as a subroutine to obtain $\phi_Q(\langle n\rangle)$ (i.e., the adjacency matrix of the unique graph in $\mathcal{F}_n$), then decoding the identifier part of the labels of the two vertices, and finally performing a lookup in the adjacency matrix using the two identifiers. The cost of all operations except for computing $\phi_Q(\langle n\rangle)$ can clearly be done in polynomial time in n. Thus, the total time use is dominated by the time for computing $\phi_Q(\langle n\rangle)$, namely $O(2^{n^2})$.

- Assume, for contradiction, that there is a labeling scheme for $\mathcal{F}$ that has maximum label size $\leq n/2 - 2$ for all n in some infinite set $I = \{n_1, n_2, \ldots\}$ *and* has a decoder running in time $P(n)$ where P is some polynomial.

  Then, there are constants c, C such that we can build a family of programs p_n each of size $|p_n| \leq c + n^2/2 - n - 1$ such that for each $n \in I$, $\phi_{p_n} = \phi_Q(\langle n\rangle)$ and p_n runs in time $Cn^2(P(n) + n^2)$. To see this, note that one can simply store a string consisting of a concatenation of all the labels of the vertices separated by a fresh symbol $\#$, and use the decoder as a subroutine to ascertain, for each pair of vertices, whether they are adjacent, and thus outputting the adjacency matrix $\phi_Q(\langle n\rangle)$. The string consisting of a concatenation of the vertex labels and separators can be stored using at most $n(n/2-2)+n-1 = n^2/2-n-1$ bits,

[5] If one wants the family $\mathcal{F}$ to be closed under isomorphism, one can merely add all graphs isomorphic to the ones in $\mathcal{F}_n$. This incurs the cost of re-coding the identifiers of the graph, adding $\log n$ to the label size.

and the remaining program logic can be stored using c bits for some c, independently of n. Hence, $|p_n| \leq c + n^2/2 - n - 1$. The running time of p_n is bounded above by the time to query all pairs of vertices using the decoder as well as quadratic time in the size of the string used to store all labels (i.e., $O(n^4)$) to move tape pointers into position. Hence, the total running time of p_n is bounded above by $Cn^2(P(n) + n^2)$ for some constant C, and is hence polynomial in n.

But then there is an N such that for all $n > N$ with $n \in I$, we have (i) $\text{TIME}_{p_n}() = Cn^2(P(n) + n^2 < 2^n$. In addition, for all $n \in I$, we have(ii) $|p_n| < n^2/2 - n - 1 < n(n-1)/2$ and (iii) $\phi_{p_n}() = s_n$. As I was infinite, there is thus at least one n that satisfies (i), (ii) and (iii), contradicting Lemma 16.

□

Corollary 4. *There is a family of finite graphs that has (i) a labeling scheme producing labels of size $O(\log n)$, with a decoder running in time $O(2^{n^2})$, but (ii) if a labeling scheme has a decoder running in time polynomial in the label size (or in n), the maximum label size produced by this scheme is $\Omega(n)$.*

Kannan, Naor and Rudich [2] consider families of finite graphs to have labeling scheme if there is a computable labeling encoder producing labels of size at most $O(\log n)$ and a decoder running in time polynomial in the label size. A fortiori, running in time polynomial in the label size means running in polynomial time in n. Theorem 16 arriving at Theorem 15.

6.5 The implicit representation conjecture holds for speeds $2^{O(n^{1/2})}$

We repeat the main labeling scheme conjecture: All hereditary families of n-vertex graphs of size at most $2^{O(n \log n)}$ have an implicit representation. In a follow up to the seminal paper by Scheinerman and Zito [112], Scheinerman [113] showed that for *labeled* graph families of size $2^{kn \log n}$ for any fixed $k < 1/2$ not only is the conjecture true, but the size of each label is constant[6].

As discussed in Section 6.2, we argue that the conjecture asks the question on unlabeled families rather than labeled ones. We denote the speed of a unlabeled graph family $\mathcal{P}^n$ as $|\mathcal{P}^n|$, and the speed of the same labeled [7] family $\mathcal{P}_n$ as $|\mathcal{P}_n|$. The relation between these sizes is:

$$|\mathcal{P}_n| \leq |\mathcal{P}^n| \leq n! \, |\mathcal{P}_n|.$$

As $n!$ has a growth rate of $2^{\Theta(n \log n)}$ one can not simply convert Scheinerman's result to have meaningful statement about unlabeled graphs. However, we can show a related result to Scheinerman's [113] for unlabeled graphs using a later result by Balogh et al. [114]. The remainder of this section is dedicated to this end.

We call $\mathcal{CL}$ the collection of the following six unlabeled n vertex graph families: Cliques, star forests and path forests, along with the complement

[6] These constant size labels are of course non-unique (Remark 1), and can be made unique by adding additional $\log n$ bits per label.

[7] The set of isomorphism classes of n-vertex graphs.

families of each of the three. Denote by $S(n)$ the number of partitions of a set with n indistinguishable elements into non-empty subsets. Each of the graph families in $\mathcal{CL}$ have speed $S(n)$, which has growth rate of $2^{\Theta(\sqrt{n})}$.

Recall that the list of neighbors of a vertex v in a graph is denoted $N(v)$. For a graph G, two vertices $\{x, y\} \in V(G)$ are twins if $N(x) \cup \{x, y\} = N(y) \cup \{x, y\}$. A homogeneous k-part graph is a graph with k-partition $(V_1, \ldots, V_k)$ where each pair of vertices in V_i are twins. It follows that every k-partition is either an independent set or a clique.

We now present an adaptation of the main theorem in [114].

Theorem 17. *There are constants k and t such that the following holds. For every unlabeled hereditary n-vertex graph family $\mathcal{G}_n$, if its speed $|\mathcal{G}_n| < S(n)$ for all sufficiently large $n_0 > n$, every graph in G_n is the symmetric difference of a homogeneous k-part graph and a graph in which every component has at most t vertices. If $|\mathcal{G}_n| = S(n)$ then $\mathcal{G}_n \in \mathcal{CL}$.*

At this point all we have to show is that the mentioned graph families have an implicit representation. As mentioned, each l-partition in an l-part graph consists of either an independent or a clique, which has a trivial $\log n + 1$ *Adjacency* labeling scheme. By the definition of l-part graph, the edge set between each two l-partitions in an l-part graph is either empty or complete. It follows that using only l bits per vertex, we can determine *Adjacency* relation between two nodes in different l-partitions. It is also not difficult to show a $\log n + t \log t$ bit labeling scheme for a graph where each component has at most t vertices, and a graph that is a symmetric difference of the two types can be labeled by extending each of the labels with a first bit to describe to which type the graph belongs to. When $|\mathcal{G}_n| = 2^{c(n^{1/2})}$, by Theorem 17 the graph belongs to $\mathcal{CL}$, each of which has a trivial labeling scheme of size $\log n + 1$.

Corollary 5. *For a constant c depending on k and t from Theorem 17 , all unlabeled hereditary families of n-vertex graphs of size at most $2^{cn^{1/2}}$ have an implicit representation.*

Chapter 7

Ancestry Labeling Schemes

We discuss labeling schemes for *Ancestry*. First, we describe in Section 7.1 the CLASSIC $2\log n$ labeling scheme for the function, followed by literature review in Section 7.2. We then present in Section 7.3 a generic *method* to assign intervals to tree vertices, which we will use twice, once in Section 7.4 to re-describe the CLASSIC labeling scheme, and finally in Section 7.5 for the best known $\log n + 2\log\log n$ labeling scheme. The bound is matched asymptotically by a lower bound in Section 7.6. Finally, in Section 7.7, we discuss dynamic *Ancestry* labeling schemes and present in detail a lower bound for a natural dynamic model.

7.1 The CLASSIC algorithm

The following $2\log n$ *Ancestry* labeling scheme was introduced by Kannan et al. [2]. Similarly to the one in Section 1.1, it is composed of two numbers in the set $\{1 \dots n\}$. Using a dfs traversal (Section 2.2), the encoder assigns vertex $v \in T$, with $\mathcal{L}(v) = (\text{dfs}(v), \text{dfs}(w))$ where w is the descendant of u with largest dfs number (if v is a leaf we set $w = v$).

Encoding each label is done in a single dfs traversal. The resulting label $\mathcal{L}(v) = (\text{dfs}(v), \text{dfs}(w))$ represents an interval $I(v)$, where $I(r) = \{1 \dots n\}$. Given the labels $\mathcal{L}(u)$ and $\mathcal{L}(v)$ the decoder returns true if $I(v) \subseteq I(u)$. See Figure 7.1 for a demonstration of the labeling scheme.

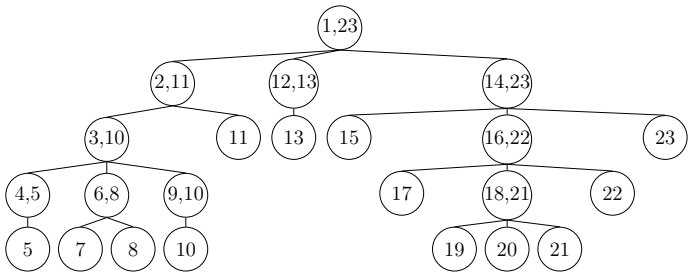


FIGURE 7.1: A tree with $n = 23$ vertices. Each vertex is assigned a label of size $2\log n$ supporting *Ancestry* queries.

7.2 Literature review

Ancestry labeling schemes are typically classified into two categories; *range-based* and *prefix-based*. *Range-based* labels, such as the CLASSIC labeling scheme, are decoded by comparing the ranges assigned to each vertex. An additional example of a range-based labeling scheme is found in Section 7.5.

Prefix-based labels are decoded by comparing the prefix of both labels such that u is an ancestor of v if and only if $\mathcal{L}(u)$ is a prefix of $\mathcal{L}(v)$. An example of prefix-based labeling scheme is found in Section 7.7.

The $2\log n$ labeling scheme presented above was improved gradually. Abiteboul, Kaplan and Milo [16] achieved an upper bound of $3/2\log n + O(\log\log n)$ bits, which was improved to $\log n + O(\sqrt{\log n})$ [115]. Shortly after, Alstrup and Bille [42] constructed a lower bound of $\log n + \log\log n$ for any labeling scheme supporting the function. Fraigniaud and Korman [47] showed that $Trees(n,\delta)$ enjoy a labeling scheme of $\log n + O(\log\delta)$. The result was generalised in a follow up paper [58] for $Trees(n)$ in a labeling scheme of size $\log n + 4\log\log n$, which is asymptotically optimal. Interestingly, the asymptotically optimal labeling scheme is based on the first one presented eighteen years prior [2].

In the case where *Ancestry* may be determined if the distance between the vertices is at most d, Alstrup et al. [42] constructed a labeling scheme of size $\log n + O(d\sqrt{\log n})$, for which the new upper bound performs better for any d.

Due to their great applicability in queries for XML documents, a staggering number of papers address this practical aspect [19,20,116–123]. Typically, those address the dynamic variant thereof. For a short survey on dynamic *Ancestry* labeling schemes, see [121].

7.2.1 Preliminaries

Recall that denote the subtree rooted in u as T_u, i.e. the tree consisting of all descendants of u, and stress that a vertex is both an ancestor and descendant of itself.

We denote the *interval* assigned to a vertex u by $I(u) = [a(u), b(u)]$, where $a(u)$ and $b(u)$ denote the lower and upper part of the interval, respectively. We also define $\overline{a}(u)$ and $\overline{b}(u)$ to be the maximum value of $a(v)$ respectively $b(v)$, where v is a descendant of u (note that this includes u itself). We will use the following notion:

Definition 16. *Let T be a rooted tree and I an interval assignment defined on $V(T)$. We say that the interval assignment I is* left-including *if for each $u, v \in T$ it holds that u is an ancestor of v iff $a(v) \in I(u)$.*

In contrast to Definition 16, the literature surveyed [2,16,58,124] considers intervals where u is an ancestor of v iff $I(v) \subseteq I(u)$, i.e. the interval of a descendant vertex is fully contained in the interval of the ancestor. This distinction is amongst the unused leverage points which we will use to arrive at our new labeling scheme.

7.3 A method for interval based labeling schemes

In this section we introduce a method for assigning intervals to tree vertices. We will see in Sections 7.4 and 7.5 how this method can be used to describe *Ancestry* labeling schemes. The method relies heavily on the values defined in Section 7.2.1, namely $a(u), b(u), \overline{a}(u), \overline{b}(u)$. An illustration of these values is found in Figure 7.2 below. The interval $[\overline{a}(u), \overline{b}(u)]$ can be seen as a slack interval from which $b(u)$ can be chosen. This will prove useful in Section 7.5.

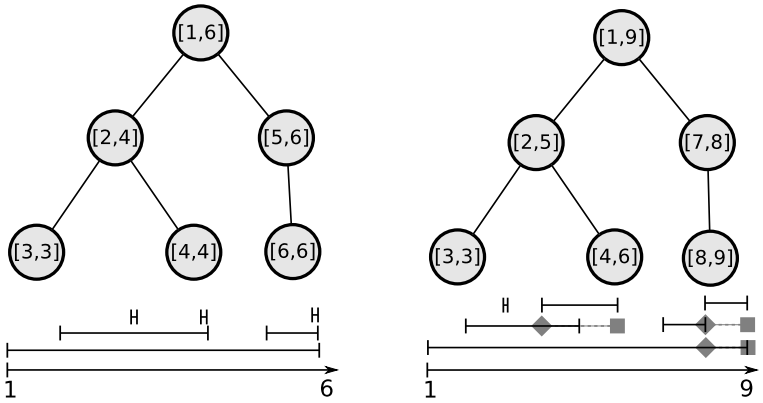


FIGURE 7.2: Two examples of left-including interval assignments to a tree. Left: a left-including assignment as used for CLASSIC in the introduction corresponding to $b(u) = \overline{a}(u)$. Right: a different left-including assignment for the same tree. For internal vertices where $\overline{a}(u)$ and $\overline{b}(u)$ do not coincide with $b(u)$, we have marked these by a grey diamond and square respectively.

The following lemmas contain necessary and sufficient conditions for interval assignments satisfying the left inclusion property.

Lemma 17. *Let T be a rooted tree and I a left-including interval assignment defined on $V(T)$. Then the following is true:*

1. *For each $u \in T$, $b(u) \geq \overline{a}(u)$.*
2. *For each $u \in T$ and $v \in T_u \backslash \{u\}$ a descendant of u, $a(v) > a(u)$.*
3. *For each $u \in T$, $[a(u), \overline{b}(u)] = \bigcup_{v \in T_u} I(v) = \bigcup_{v \in T_u} [a(v), b(v)]$*
4. *For any two distinct vertices $u, v \in T$ such that u is not an ancestor of v and v is not an ancestor of u the intervals $[a(u), \overline{b}(u)]$ and $[a(v), \overline{b}(v)]$ are disjoint.*

Lemma 18. *Let T be a rooted tree and I an interval assignment defined on $V(T)$. If the following conditions are satisfied, then I is a left-including interval assignment.*

i *For each $u \in T$, $b(u) \geq \overline{a}(u)$.*

ii *For each $u \in T$ and $v \in T$ a child of u, $a(v) > a(u)$.*

iii *For any two siblings $u, v \in T$ the intervals $[a(u), \overline{b}(u)]$ and $[a(v), \overline{b}(v)]$ are disjoint.*

We now consider a general approach for creating left-including interval assignments. For a vertex $u \in T$ and a positive integer t we define the procedure ASSIGN(u, t) that assigns intervals to T_u recursively and in particular, assigns $a(u) = t$. For pseudocode of the procedure see Algorithm 1.

Algorithm 1 provides a general method for assigning intervals using a depth-first traversal. We can use it to design an actual interval assignment by specifying: (1) the way we choose $b(u)$, and (2) the order in which the children are traversed. These specifications correspond to Line 6 and Line 3,

Algorithm 1 Assigning intervals to all vertices in the subtree T_u rooted at u ensuring $a(u) = t$.

procedure ASSIGN(u, t)
  $(a(u), \overline{a}(u), b(u), \overline{b}(u)) \leftarrow (t, t, t, t)$
  for $v \in children(u)$ **do**
    ASSIGN$(v, \overline{b}(u) + 1)$
    $\left(\overline{a}(u), \overline{b}(u)\right) \leftarrow \left(\overline{a}(v), \overline{b}(v)\right)$
  Assign $b(u)$ such that $b(u) \geq \overline{a}(u)$.
  $\overline{b}(u) \leftarrow \max\left\{b(u), \overline{b}(u)\right\}$

respectively, and determine entirely the way the intervals are assigned. It may seem counter-intuitive to pick $b(u) > \bar{a}(u)$, but we will show that doing so in a systematic way, we are able to describe the interval using fewer bits by limiting the choices for $b(u)$. In the remainder of this chapter, we will see how these two decisions impact also the label size, and produce our claimed labeling scheme.

We now show that any ordering of the children and any way of choosing $b(u)$ satisfying $b(u) \geq \overline{a}(u)$ generates a left-including interval assignment.

Lemma 19. *Let T be a tree rooted in r. After running Algorithm 1 with* ASSIGN$(r, 0)$ *the values of $\overline{a}(u), \overline{b}(u)$ are correct, i.e. for all $u \in T$:*

$$\overline{a}(u) = \max_{v \in T_u} \{a(v)\}, \quad \overline{b}(u) = \max_{v \in T_u} \{b(v)\}.$$

The following Lemma is useful for showing several properties in the method.

Lemma 20. *Let u be a vertex in a tree T with children $v_1 \dots v_k$. After running Algorithm 1 with parameters* ASSIGN$(r, 0)$ *where $v_1 \dots v_k$ are processed in that order, the following properties hold:*

1. $\overline{b}(u) - a(u) + 1 = \left(\sum_{i=1}^{k} \overline{b}(v_i) - a(v_i) + 1\right) + 1.$
2. $\overline{a}(u) - a(u) + 1 = \overline{a}(v_k) - a(v_k) + \left(\sum_{i=1}^{k-1} \overline{b}(v_i) - a(v_i) + 1\right) + 1.$

Proof. By the definition of ASSIGN we see that for all $i = 1, \dots, k-1$, $a(v_{i+1}) = 1 + \overline{b}(v_i)$. Furthermore $a(v_1) = a(u) + 1$ and $\overline{b}(v_k) = \overline{b}(u)$. Hence:

$$\begin{aligned}
\overline{b}(u) - a(u) + 1 &= \overline{b}(v_k) - a(v_1) + 2 \\
&= \left(\sum_{i=2}^{k} \overline{b}(v_i) - \overline{b}(v_{i-1})\right) + \overline{b}(v_1) - a(v_1) + 2 \\
&= \left(\sum_{i=1}^{k} \overline{b}(v_i) - a(v_i) + 1\right) + 1.
\end{aligned}$$

The second equality follows by the same line of argument. □

Theorem 18. *Let T be a tree rooted in r. After running Algorithm 1 with parameters* ASSIGN$(r, 0)$ *the set of intervals produced are left-including.*

Proof. Consider any vertex $u \in T$ and a call Assign(u, t). We will prove each of the conditions of Lemma 18, which implies the theorem.

i This condition is trivially satisfied by Line 6.

iii First, observe that any interval assigned to a vertex w by a call to Assign(v, t) has $a(w) \geq t$, and by i it has $\overline{b}(w) \geq b(w) \geq a(w)$. Let $v_1, \ldots, v_k$ be the children in the order of the for loop in Line 3. By Lines 2, 4 and 5 we have $a(v_1) = a(u) + 1, a(v_2) = \overline{b}(v_1) + 1, a(v_3) = \overline{b}(v_2) + 1, \ldots, a(v_k) = \overline{b}(v_{k-1}) + 1$, thus the condition is satisfied.

ii The first child v of u has $a(v) = t + 1 = a(u) + 1$. By the same line of argument as in *iii* we see that all other children w of u must have $a(w) > a(v) = a(u) + 1$.

□

7.4 Using the method to describe the Classic labeling

To get acquainted with the method of Section 7.3, we use it to redefine the $2 \log n$ labeling scheme introduced in Section 7.1.

Let T be a tree rooted in r. We first modify the function Assign to create Assign-Classic such that the intervals $I(u) = [a(u), b(u)]$ correspond to the intervals of the algorithm described in the introduction. To do this we set $b(u) = \overline{a}(u)$ in Line 6 and traverse the children in any order in Line 3. We note that there is a clear distinction between an algorithm such as Assign-Classic and an encoder. This distinction will be more clear in Section 7.5. We will need the following lemma to describe the encoder.

Lemma 21. *After* Assign-Classic(u, t) *is called the following invariant is true:*

$$\overline{b}(u) - a(u) + 1 = |T_u|$$

Proof. We prove the claim by induction on $|T_u|$. When $|T_u| = 1$ u is a leaf and hence $\overline{b}(u) = a(u) = t$ and the claim holds.

Let $|T_u| = m > 1$ and assume that the claim holds for all vertices with subtree size $< m$. Let $v_1, \ldots, v_k$ be the children of u. By Lemma 20 and the induction hypothesis we have:

$$\begin{aligned} \overline{b}(u) - a(u) + 1 &= \left(\sum_{i=1}^{k} \overline{b}(v_i) - a(v_i) + 1 \right) + 1 \\ &= \left(\sum_{i=1}^{k} |T_{v_i}| \right) + 1 = |T_u| \,. \end{aligned}$$

This completes the induction. □

Description of the encoder: Let T be an n-vertex tree rooted in r. We first invoke a call to Assign-Classic$(r, 0)$. By Lemma 21 we have $\overline{b}(r) - a(r) + 1 = n$ and this implies $0 \leq a(u), b(u) \leq n - 1$ for every $u \in T$. Let x_u and y_u be the encoding of $a(u)$ and $b(u)$ using exactly[1]

[1]This can be accomplished by padding with zeros if necessary.

$\lceil \log n \rceil$ bits respectively. We set the label of u to be the concatenation of the two bitstrings, i.e. $\ell(u) = x_u \circ y_u$.

Description of the decoder: Let $\ell(u)$ and $\ell(v)$ be the labels of the vertices u and v in a tree T. By the definition of the encoder, the labels have the same size and it is $2z$ for some integer $z \geq 1$. Let $\ell(u) = x_u \circ y_u$ where x_u and y_u are the first and last z bits of $\ell(u)$ respectively. Let a_u and b_u the integers from $[2^z]$ corresponding to the bit strings x_u and y_u respectively. We define a_v and b_v analogously. The decoder responds `True`, i.e. that u is the ancestor of v, iff $a_v \in [a_u, b_u]$.

The correctness of the labeling scheme follows from Theorem 18 and the description of the decoder.

7.5 An improved $\log n + 2 \log \log n$ *Ancestry* labeling scheme

In this section we use approximation-based approach which improves the previously known label size:

Theorem 19. *There exist an* Ancestry *labeling scheme of size* $\lceil \log n \rceil + 2 \lceil \log \log n \rceil + 3$.

To prove this theorem, we use the method introduced in Section 7.3. The barrier in reducing the size of the Classic labeling scheme is that the number of different intervals that can be assigned to a vertex is $\Theta(n^2)$. It is impossible to encode so many different intervals without using at least $2 \log n - O(1)$ bits. The challenge is therefore to find a smaller set of intervals $I(u) = [a(u), b(u)]$ to assign to the vertices. First, note that Lemma 17 points 2 and 4 imply that any two vertices u, v must have $a(u) \neq a(v)$. By considering the n vertex tree T rooted in r where r has $n-1$ children, we also see that there must be at least $n-1$ different values of $b(u)$ (by Lemma 17 point 4). One might think that this implies the need for $\Omega(n^2)$ different intervals. This is, however, not the case. We consider a family of intervals, such that $a(u) = O(n)$ and the size of each interval, $b(u) - a(u) + 1$, comes from a much smaller set, S. Since there are $O(n\,|S|)$ such intervals we are able to encode them using $\log n + \log |S| + O(1)$ bits.

We now present a modification of Assign called Assign-New. Calling Assign-New$(r, 0)$ on an n-vertex tree T with root r will result in each $a(u) \in [2n]$ and $b(u) \in S$, where S is given by:

$$S = \left\{ \left\lfloor (1+\varepsilon)^k \right\rfloor \mid k \in \left[4 \lceil \log n \rceil^2 \right] \right\} , \tag{7.1}$$

where ε is the unique solution to the equation $\log(1+\varepsilon) = (\lceil \log n \rceil)^{-1}$. First, we examine some properties of S:

Lemma 22. *Let S be defined as in Equation (7.1). For every $m \in \{1, 2, \ldots, 2n\}$ there exists $s \in S$ such that:*

$$m \leq s < m(1+\varepsilon) .$$

Furthermore, $s = \left\lfloor (1+\varepsilon)^k \right\rfloor$ for some $k \in \left[4 \lceil \log n \rceil^2 \right]$, and both s and k can be computed in $O(1)$ time.

Proof. Fix $m \in \{1, 2, \ldots, 2n\}$. Let k be the largest integer such that $(1+\varepsilon)^{k-1} < m$. Equivalently, k is the largest integer such that:

$$k - 1 < \frac{\log m}{\log(1+\varepsilon)} = (\log m) \cdot \lceil \log n \rceil .$$

In other words we choose k as $\lceil (\log m) \cdot \lceil \log n \rceil \rceil$ and note that k is computed in $O(1)$ time. Since $\log m \leq \log(2n) \leq 2\log n$:

$$k \leq \lceil 2(\log n) \cdot \lceil \log n \rceil \rceil \leq 2\lceil \log n \rceil^2 < 4\lceil \log n \rceil^2 .$$

By setting $s = \lfloor (1+\varepsilon)^k \rfloor$ we have $s \in S$. By the definition of k we see that $(1+\varepsilon)^k \geq m$ and thus also $s \geq m$. Similarly:

$$m(1+\varepsilon) > (1+\varepsilon)^{k-1} \cdot (1+\varepsilon) = (1+\varepsilon)^k \geq s.$$

This proves that $s \in S$ satisfies the desired requirement. Furthermore s can be computed in $O(1)$ time by noting that:

$$s = \left\lfloor (1+\varepsilon)^k \right\rfloor = \left\lfloor 2^{\log(1+\varepsilon)k} \right\rfloor = \left\lfloor 2^{\lceil \log n \rceil^{-1} \cdot k} \right\rfloor .$$

□

We now define Assign-New by modifying Assign in the following two ways. First, we specify the order in which the children are traversed in Line 3. This is done in non-decreasing order of their subtree size, i.e. we iterate $v_1, \ldots, v_k$, where $|T_{v_1}| \leq \ldots \leq |T_{v_k}|$. Second, we choose $b(u)$ in Line 6 as the smallest value, such that $b(u) \geq \overline{a}(u)$ and $b(u) - a(u) + 1 \in S$. This is done by using Lemma 22 with $m = \overline{a}(u) - a(u) + 1$ and setting $b(u) = a(u) + s - 1$. In order to do this we must have $m \leq 2n$. To do this, we show the following lemma corresponding to Lemma 21 in Section 7.4.

Lemma 23. *After* Assign-New(u, t) *is called the following invariants holds:*

$$\overline{a}(u) - a(u) + 1 \leq |T_u| \, (1+\varepsilon)^{\lfloor \log |T_u| \rfloor} \tag{7.2}$$

$$\overline{b}(u) - a(u) + 1 \leq |T_u| \, (1+\varepsilon)^{\lfloor \log |T_u| \rfloor + 1} \tag{7.3}$$

Proof. We prove the claim by induction on $|T_u|$. When $|T_u| = 1$, u is a leaf, so $\overline{b}(u) = \overline{a}(u) = a(u) = t$ and the claim holds.

Now let $|T_u| = m > 1$ and assume that the claim holds for all vertices with subtree size $< m$. Let $v_1, \ldots, v_k$ be the children of u such that $|T_{v_1}| \leq \ldots \leq |T_{v_k}|$. First, we show that Equation (7.2) holds. By Lemma 20 we have the following expression for $\overline{a}(u) - a(u) + 1$:

$$\overline{a}(u) - a(u) + 1 = (\overline{a}(v_k) - a(v_k) + 1) + \left(\sum_{i=1}^{k-1} \overline{b}(v_i) - a(v_i) + 1 \right) + 1. \tag{7.4}$$

It follows from the induction hypothesis that:

$$\overline{a}(v_k) - a(v_k) + 1 \leq |T_{v_k}| \, (1+\varepsilon)^{\lfloor \log |T_{v_k}| \rfloor} \leq |T_{v_k}| \, (1+\varepsilon)^{\lfloor \log |T_u| \rfloor}. \tag{7.5}$$

Furthermore, by the ordering of the children, we have $\log|T_{v_i}| \leq \log|T_u| - 1$ for every $i = 1, \ldots, k-1$. Hence:

$$\overline{b}(v_i) - a(v_i) + 1 \leq |T_{v_i}|\,(1+\varepsilon)^{\lfloor \log|T_{v_i}| \rfloor + 1} \leq |T_{v_k}|\,(1+\varepsilon)^{\lfloor \log|T_u| \rfloor}. \tag{7.6}$$

Inserting Equations (7.5) and (7.6) into Equation (7.4) proves invariant Equation (7.2).

Since $\overline{b}(u) = \max\{\overline{b}(v_k), b(u)\}$ we only need to upper bound $\overline{b}(v_k) - a(u) + 1$ and $b(u) - a(u) + 1$. First we note that since $b(u)$ is chosen smallest possible such that $b(u) \geq \overline{a}(u)$ and $b(u) - a(u) + 1 \in S$, it is guaranteed by Lemma 22 that:

$$b(u) - a(u) + 1 < (1+\varepsilon)\,(\overline{a}(u) - a(u) + 1) \leq |T_u|\,(1+\varepsilon)^{\lfloor \log|T_u| \rfloor + 1}$$

Hence we just need to upper bound $\overline{b}(v_k) - a(u) + 1$. First we note that just as in Equation (7.4):

$$\overline{b}(v_k) - a(u) + 1 = \left(\sum_{i=1}^{k} \overline{b}(v_i) - a(v_i) + 1\right) + 1 \tag{7.7}$$

By the induction hypothesis, for every $i = 1, \ldots, k$:

$$\overline{b}(v_i) - a(v_i) + 1 \leq |T_{v_i}|\,(1+\varepsilon)^{\lfloor \log|T_{v_i}| \rfloor + 1} \leq |T_{v_i}|\,(1+\varepsilon)^{\lfloor \log|T_u| \rfloor + 1} \tag{7.8}$$

Inserting Equation (7.8) into Equation (7.7) gives the desired:

$$\overline{b}(v_k) - a(u) + 1 \leq 1 + \sum_{i=1}^{k} |T_{v_i}|\,(1+\varepsilon)^{\lfloor \log|T_u| \rfloor + 1} \leq |T_u|\,(1+\varepsilon)^{\lfloor \log|T_u| \rfloor + 1}.$$

This completes the induction. □

By Lemma 23 we see that for a tree T with n vertices and $u \in T$:

$$\overline{a}(u) - a(u) + 1 \leq |T_u|\,(1+\varepsilon)^{\lfloor \log|T_u| \rfloor} \leq n \cdot 2^{\lfloor \log n \rfloor \log(1+\varepsilon)} \leq n \cdot 2^1 = 2n.$$

In particular, for any $u \in T$ we see that $a(u) \leq 2n$, and by Lemma 22 the function Assign-New is well-defined.

We are now ready to describe the labeling scheme:

Description of the encoder: Given an n-vertex tree T rooted in r, the encoding algorithm works by first invoking a call to Approx-New$(r, 0)$. Recall that by Lemma 22 we find $b(u)$ such that $b(u) - a(u) + 1 = \lfloor (1+\varepsilon)^k \rfloor$ as well as the value of k in $O(1)$ time. For a vertex u, denote the value of k by $k(u)$ and let x_u and y_u be the bit strings representing $a(u)$ and $k(u)$ respectively, consisting of exactly $\lceil \log(2n) \rceil$ and $\left\lceil \log(4\lceil \log n \rceil^2) \right\rceil$ bits (padding with zeroes if necessary). This is possible since $a(u) \in [2n]$ and $k(u) \in \left[4\lceil \log n \rceil^2\right]$.

For each vertex $u \in T$ we assign the label $\ell(u) = x_u \circ y_u$. Since

$$\lceil \log(2n) \rceil = 1 + \lceil \log n \rceil\,, \quad \left\lceil \log(4\lceil \log n \rceil^2) \right\rceil = 2 + \lceil 2\log(\lceil \log n \rceil) \rceil = 2 + \lceil 2 \log\log n \rceil\,,$$

the label size of this scheme is $\lceil \log n \rceil + \lceil 2 \log\log n \rceil + 3$.

Description of the decoder: Let $\ell(u)$ and $\ell(v)$ be the labels of the vertices u and v in a tree T. By the definition of the encoder the labels have the same size and it is $s = z + \lceil 2 \log z \rceil + 3$ for some integer $z \geq 1$. By using that $s - \lceil 2 \log s \rceil - 3 = z - O(1)$ we can compute z in $O(1)$ time. We know that the number of vertices n in T satisfies $\lceil \log n \rceil = z$. We can therefore define ε to be the unique solution to $\log(1+\varepsilon) = \lceil \log n \rceil^{-1} = z^{-1}$. Let x_u and y_u be the first $z+1$ bits and last $\lceil 2 \log z \rceil + 2$ bits of $\ell(u)$ respectively. We let a_u and k_u be the integers in $[2^{z+1}]$ and $[4z^2]$ corresponding to the bit strings x_u and y_u respectively. We define s_u as $\lfloor (1+\varepsilon)^{k_u} \rfloor$ and $b_u = s_u + a_u - 1$. We define a_v, b_v, k_v, s_v analogously. The decoder responds `True`, i.e. that u is the ancestor of v, iff $a_v \in [a_u, b_u]$.

Theorem 19 is now achieved by using the labeling scheme described above. Correctness follows from Theorem 18.

7.6 Lower bound

The following lower bound is an extension of the one by Alstrup et al. [42] using the same technique, namely, boxes and groups (Section 2.4).

Theorem 20. *For any n, δ with $n \geq \delta + 1 \geq 3$, any* Ancestry *labeling scheme for $Trees(n, \delta)$ has a worst-case label size of at least $\lceil \log n \rceil + \lfloor \log \lfloor \log \delta \rfloor \rfloor - 1$.*[2]

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 $k = \lfloor \log \delta \rfloor$. Note that $k \leq \log m$. Construct for $i = 1 \ldots k$ the tree T_i as a root vertex to which $m/2^i$ paths of length 2^i have been attached. Thus, T_i has $m+1 \leq n$ vertices, whereof m belong to disjoint paths. Further, T_i has depth $2^i \leq 2^k \leq \delta$, and hence T_i belongs to $Trees(n, \delta)$. For an illustration of such trees see Figure 7.3.

Each vertex in a tree must be uniquely labeled by an *Ancestry* labeling scheme. Further, if two vertices 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 vertices on the same path have an *Ancestry* relation whereas vertices on different paths do not. We can therefore apply Lemma 4, using the m vertices of the paths of each tree as a "box" and each path as a "group", and it follows that we need at least $\frac{1}{2}m(k+1) = \frac{1}{2}m(\lfloor \log \delta \rfloor + 1)$ labels. If the worst-case label size is L we can create $2^{L+1} - 1$ distinct labels, and we must therefore have $\frac{1}{2}m(\lfloor \log \delta \rfloor + 1) \leq 2^{L+1} - 1$ from which it follows that $L \geq \lceil \log n \rceil + \lfloor \log \lfloor \log \delta \rfloor \rfloor - 1$. □

7.7 Dynamic *Ancestry* labeling schemes

All results described thus far present a worst-case analysis of static scenarios. We describe the first dynamic result, in which rather than receiving a tree T, the decoder receives a sequence of n operations constructing T. Operations considered in the literature are insertions and deletions of leaves, or arbitrary vertices. For convenience it is assumed that the sequence constructs a rooted tree, and its root exist from the beginning and is never deleted.

[2] Observe that the assumption $n \geq \delta + 1$ is natural, since any tree with n vertices and depth δ satisfies $n \geq \delta + 1$.

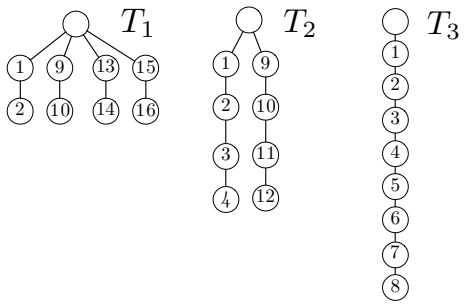


FIGURE 7.3: Illustration of T_1, T_2, T_3 for $n = 8$.

The section discusses a *persistent Ancestry* labeling scheme, i.e a dynamic labeling scheme that does not change the label given to a vertex. The result is described in a model in which the operation allowed is an insertion of leaves.

A trivial prefix-based labeling *Ancestry* labeling scheme for the model is built directly from the suffix free $code_0$ (Section 2.2.2). The root r receives the label 1. Suppose vertex v receives the label $\mathcal{L}(v)$. The i'th child of v receives the label $\mathcal{L}(v) \circ 10^{i-1}$. Decoding $\mathcal{L}(u), \mathcal{L}(v)$ is done in the standard way, and the label size is at most $O(n)$. The next section proves that this trivial labeling scheme is asymptotically optimal.

Lower bound for persistent *Ancestry* labeling scheme Cohen, Kaplan and Milo [17] proved a lower bound of $\Omega(n)$ on the label length for a sequence of $n+1$ insertions. They do so by presenting a family of insertion sequences of size 2^{n-1}, such that each sequence has at least one vertex with unique label.

Definition 17. *We define the family of insertion sequences $\mathcal{F}(n)$ recursively as follows. $\mathcal{F}(1)$ consists of a single sequence which inserts a root r and a child of r, w. $\mathcal{F}(n)$ extends each sequence s in $\mathcal{F}(n-1)$ rooted in r to two sequences s_1 and s_2. Both insertion sequences replace the insertion of the root r by the sequence r' and w', a child of r'. s_1 is defined by connecting vertices previously adjacent to r to be adjacent to w'. In the same manner, r' "replaces" r in s_2. For illustration, see Figure 7.4.*

For convenience, we denote the last vertex in a sequence s, as $last(s)$, and the set of all sequences s_2, respectively s_1 in $\mathcal{F}(n)$ created from $s \in \mathcal{F}(n-1)$ as s_2 type sequence respectively s_1 type sequence.

Since $|\mathcal{F}(1)| = 1$, and $|\mathcal{F}(i)| = 2 \cdot |\mathcal{F}(i-1)|$ it follows that the number of insertion sequences is $|\mathcal{F}(n)| = 2^{n-1}$. Each insertion sequence in $\mathcal{F}(n)$ has one more vertex than the sequence it was built upon from $\mathcal{F}(n-1)$. Since the size of the sequence in $\mathcal{F}(1)$ is 2, it follows that the size of each insertion sequence in $\mathcal{F}(n)$ is $n+1$.

We proceed to present the lower bound.

Lemma 24. *[16] Any persistent* Ancestry *labeling scheme with a deterministic encoder must give a unique label to the last insertion in each of the insertion sequences in $\mathcal{F}(n)$.*

Proof. The claim is proved by induction. Suppose the claim is true for $\mathcal{F}(n-1)$, and pick two insertion sequences $s, s' \in \mathcal{F}(n-1)$. Denote the sequences

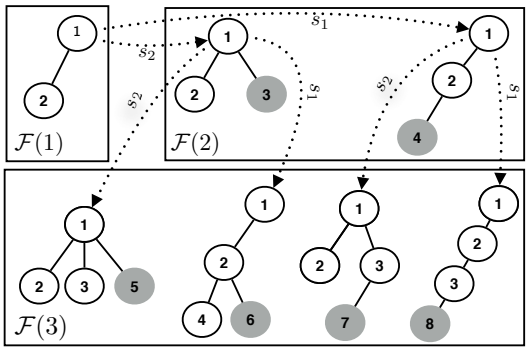


FIGURE 7.4: An illustration of the trees resulting by the sequences $\mathcal{F}(1)$, $\mathcal{F}(2)$ and $\mathcal{F}(3)$ from Definition 17. The grey vertices are the ones introduced by Lemma 24. The arrows denote the extension of a sequence, and marked s_1 or s_2 according to the type.

extending s and s' using the first extension type s_1 and s'_1 respectively. From the induction hypothesis it follows directly that $\mathcal{L}(last(s_1)) \neq \mathcal{L}(last(s'_1))$.

Denote now both derived sequences of s in $\mathcal{F}(n)$ as s_1 and s_2. Since the encoder is deterministic, and the first two elements of the insertions are identical, r' and w' receives the same label in both. The vertex $last(s_1)$ is not a decedent of w but $last(s_2)$ is. Since the decoder needs to determine an *Ancestry* relation, $d(\mathcal{L}(w), \mathcal{L}(last(s_1))) = false$ and $d(\mathcal{L}(w), \mathcal{L}(last(s_2))) = true$. Therefore, it is required that $\mathcal{L}(last(s_1)) \neq \mathcal{L}(last(s_2))$. □

From Lemma 24, and since there are 2^{n-1} insertion sequences of size n, it follows that 2^{n-1} distinct labels are needed for the last vertex in each sequence in $\mathcal{F}(n)$.

Corollary 6. *Every persistent* Ancestry *labeling scheme* $\langle e, d \rangle$ *is of size* $\Omega(n)$.

Using a similar argument, Cohen et al. [17] prove that the bound holds for a sequence corresponding to a tree from $Trees(n, \Delta)$, when $\Delta \geq 2$. Moreover, the bound is shown to hold even when the encoder is allowed to use randomisation.

To overcome this inherent difficulty, the authors introduce the notion of *sibling* and *direct clues*. Loosely speaking, sibling, respectively, direct clues provide every vertex v in the sequence a guarantee on the range of possible number of v's siblings, respectively size of v. Using this method, they show tight bounds on the label sizes for such labeling schemes with $\Theta(\log n)$ bits using sibling clues and $\Theta(\log^2 n)$ bits using direct clues. We show in Chapter 8 that this bound applies for the functions *NCA*, *Distance*, and *Routing* as well, and in contrast, that $2 \log n$ are sufficient and necessary in this model for the functions *Adjacency*, *Siblings* and *Connectivity*.

Chapter 8

Multifunctional and Dynamic Labeling Schemes

In this chapter we investigate labeling schemes supporting *Adjacency*, *Ancestry*, *Siblings*, and *Connectivity* queries in forests. In the course of more than 20 years, the existence of $\log n + O(\log \log n)$ labeling schemes supporting each of these functions was proven, last being ancestry, as seen in Chapter 7. Several multifunctional labeling schemes also enjoy lower or upper bounds of $\log n + \Omega(\log \log n)$ or $\log n + O(\log \log n)$ respectively. Notably an upper bound of $\log n + 2 \log \log n$ for *Adjacency* +*Siblings* and a lower bound of $\log n + \log \log n$ for each of the functions *Siblings*, *Ancestry*, and *Connectivity* [42]. We improve the constants hidden in the O-notation for several multifunctional labeling schemes. .

In the context of dynamic labeling schemes, we have seen in Section 7.7 that *Ancestry* requires $\Omega(n)$ bits. In contrast, we show upper and lower bounds on the label size for *Adjacency*, *Siblings*, and *Connectivity* of $2 \log n$ bits, and $3 \log n$ to support all three functions. We also show that there exist no efficient dynamic *Adjacency* labeling schemes for planar, bounded treewidth, bounded arboricity and bounded degree graphs.

8.1 Introduction

In their seminal paper, Kannan et al. [2] introduced labeling schemes using at most $2 \log n$ bits for each of the functions *Adjacency*, *Siblings* and *Ancestry*. Improving these results have been motivated heavily by the fact that a small improvement of the label size may contribute significantly to the performance of XML search engines. Alstrup, Bille and Rauhe [42] established a lower bound of $\log n + \log \log n$ for the functions *Siblings*, *Connectivity* and *Ancestry* along with a matching upper bound for the first two.

In most settings, it is the case that the structure of the graph to be labeled is not known in advance. In contrast to the *static* setting described above, a *dynamic* labeling scheme receives the tree as an online sequence of topological events. As described in Section 7.7, Cohen, Kaplan and Milo [121] considered *dynamic labeling schemes* where the encoder receives n leaf insertions and assigns unique labels that must remain unchanged throughout the labeling process. In this context, they showed a tight bound of $\Theta(n)$ bits for any dynamic *Ancestry* labeling scheme. We stress the importance of their lower bound by showing that it extends to routing, NCA, and distance as well. In light of this lower bound, Korman, Peleg and Rodeh [125] introduced dynamic labeling schemes where vertex re-label is permitted and performed by message passing. In this model they obtain a compact labeling scheme

Function	Static Label Size	Static Lower Bound	Dynamic
Adjacency	$\log n + O(\log^* n)$ [8]	$\log n + 1$	$2\log n$ (Theorem 21)
Connectivity	$\log n + \log\log n$ [42]	$\log n + \log\log n$ [42]	$2\log n$ (Theorem 21)
Siblings	$\log n + \log\log n$ [52]	$\log n + \log\log n$ [42]	$2\log n$ (Theorem 21)
Ancestry	$\log n + 2\log\log n$ [58]	$\log n + \log\log n$ [42]	n [121]
AD/S	$\log n + 2\log\log n$ [12])	$\log n + \log\log n$ [42]	$2\log n$ (Theorem 21)
C/S	$\log n + 2\log\log n$ (Theorem 25)	$\log n + 2\log\log n$ (Theorem 26)	$3\log n$ (Theorem 24)
C/AN	$\log n + 5\log\log n$ (Theorem 25)	$\log n + 2\log\log n$ (Theorem 27)	n [121]
C/AD/S	$\log n + 3\log\log n$ (Theorem 25)	$\log n + 2\log\log n$ (Theorem 26)	$3\log n$ (Theorem 24)
Routing	$(1 + o(1))\log n$ [9]	$\log n + \log\log n$ [42]	n (Section 8.2)
NCA	$2.772\log n$ [51]	$1.008\log n$ [51]	n (Section 8.2)
Distance	$1/4\log^2 n$ [48]	$1/4\log^2 n$ [48]	n (Section 8.2)
Siblings *	$\log n$	$\log n$	$\log n$
Connectivity *	$\log n$	$\log n$	$\log n$
C/S*	$\log n + \log\log n$ (Theorem 25)	$\log n + \log\log n$ (Theorem 28)	$2\log n$

TABLE 8.1: Upper and lower label sizes for labeling trees with n vertices (excluding additive constants). Routing is reported in the designer-port model [26] and NCA with no pre-existing labels [51]. Functions marked with * denote non-unique labeling schemes, and bounds without a reference are folklore. Dynamic labeling schemes are all tight.

for *Ancestry*, while keeping the number of messages small. Additional results in this setting include conversion methods for static labeling schemes [125, 126], as well as specialized distance [126] and routing [127] labeling schemes. See Chapter 9 for experimental evaluation of some of the aforementioned results.

Considering the static setting, a natural question is to determine the label size required to support some, or all, of the functions. Simply concatenating the labels mentioned yield an $O(\log n)$ label size, which is clearly undesired. Labeling schemes supporting multiple functions[1] were previously studied for *Adjacency* and *Siblings* queries. Alstrup et al. [42] proved a $\log n + 5\log\log n$ label size which was improved by Gavoille and Labourel [128] to $\log n + 2\log\log n$. See Table 8.1 for a summary of labeling schemes for forests including the results of this chapter.

8.1.1 Our contribution

We contribute several upper and lower bounds for both dynamic and multifunctional labeling schemes. First, we observe that the naïve $2\log n$ *Adjacency*, *Siblings* and *Connectivity* labeling schemes are suitable for the dynamic setting without the need of relabeling. We then present simple families of insertion sequences for which labels of size $2\log n$ are required, showing that in the dynamic setting the naïve labeling schemes are in fact optimal. The result is in contrast to the static case, where *Adjacency* labels requires strictly fewer bits than both *Siblings* and *Connectivity*. The labeling schemes also reveal an exponential gap between *Ancestry* and the functions mentioned for the dynamic setting. In Section 8.2.3 we show a construction of simple lower bounds of $\Omega(n)$ for *Adjacency* labeling schemes on various important graph families.

[1] We refer to such labeling schemes as multifunctional labeling schemes.

In the context of multifunctional labeling schemes, we show first that $3\log n$ bits are necessary and sufficient for any dynamic labeling scheme supporting *Adjacency* and *Connectivity*. Using a novel technique, we prove in Theorem 26 a lower bound of $\log n + 2\log\log n$ for any unique labeling scheme supporting both *Connectivity* and *Siblings*/*Ancestry*. This lower bound is preceded by a simple upper bound, proving that any labeling scheme of size $S(n)$ growing faster than $\log n$ can be altered to support *Connectivity* as well by adding at most $\log\log n$ bits. Note that in the case of *Connectivity* and *Siblings* the upper and lower bounds match. All omitted proofs appear in [13].

8.1.2 Preliminaries

We define some particular dynamic encoder and decoder functions. If the encoder receives G as a sequence of topological events[2] the labeling scheme is *dynamic*. Recall that If for all graphs $G \in \mathcal{G}$, the label assignment e_G is an injective mapping, i.e. for all distinct $u, v \in V(G)$, $e_G(u) \neq e_G(v)$, we say that the labeling scheme assigns *unique* labels. Unless stated otherwise, the labeling schemes presented are assumed to assign unique labels. Moreover, we allow the decoder to know the label size.

Let G be a graph in a family of graphs $\mathcal{H}$ and suppose that an f-labeling scheme assigns a vertex $v \in G$ the label $\mathcal{L}(v)$. If $\mathcal{L}(v)$ does not appear in any of the label assignments for the other graphs in $\mathcal{H}$, we say that the label is *distinct* for the labeling scheme over $\mathcal{H}$. This notion will be useful in proving the lower bounds. All labeling schemes constructed in this chapter require $O(n)$ encoding time and $O(1)$ decoding time under the assumption of a $\Omega(\log n)$ word size RAM model. See [9] for additional details.

8.2 Dynamic labeling schemes

We first note that the lower bound for *Ancestry* due to Cohen, et al. also holds for NCA, since the labels computed by an NCA labeling scheme can decide *Ancestry*: Given the labels $\mathcal{L}(u), \mathcal{L}(v)$ of two vertices u, v in the tree T, return true if $\mathcal{L}(u)$ is equal to the label returned by the original NCA decoder, and false otherwise. Similarly, suppose a labeling scheme for routing[3] assigns 0 as the port number on the path to the root. Given $\mathcal{L}(u), \mathcal{L}(v)$ as before, return true if $routing(\mathcal{L}(u), \mathcal{L}(v)) \neq 0$ and $routing(\mathcal{L}(v), \mathcal{L}(u)) = 0$. If there were to exist a dynamic labeling scheme for routing or NCA with size $o(n)$, the labels produced would be sufficient to determine *Ancestry*, in contrast to Cohen's bound. Peleg [25] proved that any $f(n)$ distance labeling scheme can be converted to $f(n) + \log(n)$ labeling scheme for NCA by attaching the depth of any vertex. Since the depth of a vertex inserted can not change in our dynamic setting, we conclude that the lower bound applies to distance up to additive $O(\log n)$ factor.

[2] Cohen et al. defines such a sequence as a set of insertion of vertices into an initially empty tree, where the root is inserted first, and all other insertions are of the form "insert vertex u as a child of vertex v". We extend it to support "remove leaf u", where the root may never be deleted.

[3] Routing in the designer port model [26], in which this assumption is standard.

8.2.1 Upper Bounds

The following naïve *Adjacency* labeling scheme was introduced by Kannan et al. [2]. Consider an arbitrary rooted tree T with n vertices. Enumerate the vertices in the tree with the numbers 0 through $n-1$, and let, for each vertex v, $Id(v)$ be the number associated with v. Let $parent(v)$ be the parent of a vertex v in the tree. The label of v is $\mathcal{L}(v) = (Id(v) \circ Id(parent(v)))$, and the root is labeled $(0,0)$. Given the labels $\mathcal{L}(v), \mathcal{L}(v')$ of two vertices v and v', two vertices are adjacent if and only if either $Id(parent(v)) = Id(v')$ or $Id(parent(v')) = Id(v)$ but not both, so that the root is not adjacent to itself.

This is also a dynamic labeling scheme for *Adjacency* with equal label size. Moreover, it is also both a static and dynamic labeling scheme for *Siblings*, in which case, the decoder must check if $Id(parent(v)) = Id(parent(v'))$. A labeling scheme for *Connectivity* can be constructed by storing the component number rather than the parent id. After n insertions, each label contains two parts, each in the range $[0, n-1]$. Therefore, the label size required is $2\log n$.

The labeling schemes suggested extend to larger families of graphs. In particular, the dynamic *Connectivity* labeling scheme holds for the family of all graphs. The family of k-bounded degree graphs enjoys a similar dynamic *Adjacency* labeling scheme of size $(k+1)\log n$.

8.2.2 Lower Bounds

We show that $2\log n$ is a tight bound for any dynamic *Adjacency* labeling scheme for trees. We denote by $\mathcal{F}_n(k)$ an insertion sequence of n vertices, creating an *initial path* of length $1 < k \leq n$, followed by $n-k$ *adjacent leaves* to vertex $k-1$ on the path. The family of all such insertions sequences is denoted $\mathcal{F}_n$. For illustration see Figure 8.1.

Lemma 25. *Fix some dynamic labeling scheme that supports* Adjacency. *For any* $1 < k < n$, $\mathcal{F}_n(k)$ *must contain at least* $n-k$ *distinct labels for this labeling scheme over* $\mathcal{F}_n$.

Proof. The labels of $\mathcal{F}_n(n)$ are set to $P_1 \dots P_n$ respectively. Since the encoder is deterministic, and since every insertion sequence $\mathcal{F}_n(k)$ first inserts vertices on the initial path, these vertices must be labeled $P_1 \dots P_k$. Let the labels of the adjacent leaves of such an insertion sequence be denoted by $L_1^k \dots L_{n-k}^k$.

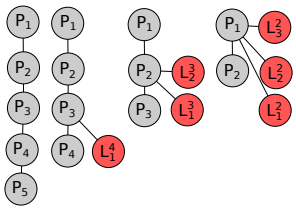


FIGURE 8.1: Illustration of $\mathcal{F}_5$.

Clearly, $L_1^k \dots L_{n-k}^k$ must be different from $P_1 \dots P_n$, as the only other labels adjacent to P_{k-1} are P_{k-2} and P_k, which have already been used on the initial path. Consider now any vertex labeled L_i^j of $\mathcal{F}_n(j)$ for $j \neq k$. Assume w.l.o.g that $j > k$. Such a vertex must be adjacent to P_{j-1} and *not*

to P_{k-1}, as P_{k-1} is contained in the path to P_{j-1}. Therefore we must have $L_i^j \notin \{L_1^k, \ldots, L_{n-k}^k\}$. □

Identical lower bounds are attained similarly for both *Siblings* and *Connectivity*.

Theorem 21. *Any dynamic labeling scheme supporting either* Adjacency*,* Connectivity*, or* Siblings *requires at least* $2\log n - 1$ *bits.*

Proof. According to Lemma 25, at least $n + \sum_{i=2}^{n-1} i = n^2/2 + O(n)$ distinct labels are required to label $\mathcal{F}_n$ if *Adjacency* or *Siblings* requests are supported, and the same applies for $\mathcal{F}_n^c$ if *Connectivity* is supported. □

A natural question is whether a randomized labeling scheme could provide labels of size less than $2\log n - O(1)$. The next theorem, based on Thm. 3.4 in [121] answer this question negatively.

Theorem 22. *For any randomized dynamic labeling scheme supporting either* Adjacency*,* Connectivity*, or* Siblings *queries there exists an insertion sequence such that the expected value of the maximal label size is at least* $2\log n - O(1)$ *bits.*

8.2.3 Other Graph Families

In this part, we expand our lower bound ideas to *Adjacency* labeling schemes for the following families with at most n vertices: bounded arboricity-k graphs[4] $\mathcal{A}_k$, bounded degree-k graphs Δ_k, planar graphs $\mathcal{P}$ and bounded treewidth-k graphs $\mathcal{T}_k$. In the context of (static) *Adjacency* labeling schemes, these families are well studied [2, 6–8]. In particular, $\mathcal{T}_k$, $\mathcal{P}$, Δ_k and $\mathcal{A}_k$ enjoy *Adjacency* labeling schemes of size $\log n + O(k \log\log(n/k))$ [6], $2\log n + O(\log\log n)$ [6], $\lfloor \frac{\Delta(n)}{2} \rfloor + 1$ [7], and $k \log n$ [7] respectively.

We consider a sequence of vertex insertions along with all edges adjacent to them, such that an edge (u, v) may be introduced along with vertex v if vertex u appeared prior in the sequence, and prove the following.

> **Theorem 23.** *Any dynamic* Adjacency *labeling scheme for each* $\mathcal{A}_2$*,* $\mathcal{P}$ *and* $\mathcal{T}_3$ *requires* $\Omega(n)$ *bits. Similarly, any dynamic* Adjacency *labeling scheme for* Δ_k *requires* $k \log n$ *bits.*

Proof. Let S be the collection of all nonempty subsets of the integers $1 \ldots n-1$. For every $s \in S$, we denote by $\mathcal{F}_n(s)$ an insertion sequence of n vertices, creating a path of length $n-1$, followed by a single vertex v connected to the vertices on the path whose number is a member of s. Such a graph has arboricity 2 since it can be decomposed into an initial path and a star rooted in v. For each of the $|S|$ insertion sequences, v's label must be distinct. We conclude that the number of bits required for any *Adjacency* labeling scheme is at least $\log(|S|) = n - 1$. See Figure 8.2 for illustration.

The construction of $\mathcal{F}_n(s)$ implies an identical lower bound for the family of planar graphs, as well as interval graphs. By considering all sets s of at most k elements instead, we get a bound of $k \log n$ label size for any *Adjacency* labeling scheme for Δ_k, where k is constant. □

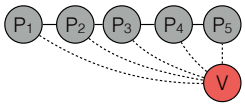


FIGURE 8.2: Illustration of $\mathcal{F}_5(s)$. The dotted lines may or may not appear in the insertion sequence depending on the element of S chosen.

In the following two sections we investigate labeling schemes incorporating two or more of the functions mentioned for both dynamic and static labeling schemes.

8.2.4 Dynamic multifunctional Labeling Schemes

A $3\log n$ dynamic labeling scheme for any combination of *Connectivity*, *Adjacency* and *Siblings* queries can be obtained by setting the label of a vertex v to be $(Id(v) \circ Id(parent(v)) \circ component(v))$, as described in Section 8.2.1.

We now show that this upper bound is in fact tight. More precisely, we show that $3\log n$ bits are required to answer the combination of *Connectivity* and *Adjacency*. Let $I_n(j,k)$ be an insertion sequence designed as follows: First j vertices are inserted creating an *initial forest* of single vertex trees. Then k vertices are added as a path with root in the jth tree. At last, $n-j-k$ adjacent *path leaves* are added to the second-to-last vertex on the path. For a given n we define I_n as the family of all such insertion sequences.

Lemma 26. *Fix some dynamic labeling scheme that supports* Adjacency *and* Connectivity *requests. For any* $1 < j + k < n$, $I_n(k)$ *must contain at least* $n - j - k$ *distinct labels for this labeling scheme over* I_n.

According to this Lemma, at least $\sum_{j=1}^{n-1}\sum_{k=1}^{n-j-1} n-j-k = \frac{1}{6}n^3 - O(n^2)$ distinct labels are required to label the family I_n. We can thus conclude.

Theorem 24. *Any dynamic labeling scheme supporting both* Adjacency *and* Connectivity *queries requires at least* $3\log n - O(1)$ *bits.*

The same family of insertion sequences can be used to show a $3\log n - O(1)$ lower bound for any dynamic labeling scheme supporting both *Siblings* and *Connectivity* queries. Furthermore, similarly to Theorem 22, the bound holds even without the assumption that the encoder is deterministic.

8.3 Static multifunctional Labeling Schemes

As seen in Theorem 24, the requirement to support both *Connectivity* and *Adjacency* forces an increased label size for any dynamic labeling scheme. In the remainder of the paper we prove lower and upper bounds for static labeling schemes that support those operations, both for the case where the labels are necessarily unique, and for the case that they are not. From hereon, all labeling schemes are on the family of rooted forests with at most n vertices. We show that most labeling schemes can be altered to support *Connectivity* as well.

[4]The *arboricity* of a graph G is the minimum number of edge-disjoint acyclic subgraphs whose union is G.

Theorem 25. *Consider any function f of two vertices in a single tree on n vertices. If there exists an f-labeling scheme of size $S(n)$, where $S(n)$ is non-decreasing and $S(a) - S(b) \geq \log a - \log b - O(1)$ for any $a \geq b$. Then there exists an f-labeling scheme, which also supports* Connectivity *queries of size at most $S(n) + \log\log n + O(1)$.*

Proof. We will consider the label $\mathcal{L}(v) = (C \circ L \circ sep)$ defined as follows. First, sort the trees of the forest according to their sizes. For the ith biggest tree we set $C = i$ using $\log i$ bits. Since the tree has at most n/i vertices, we can pick the label L internally in the tree using only $S(n/i)$ bits. Finally, we need a separator, *sep*, to separate C from L. We can represent this using $\log\log n$ bits, since i uses at most $\log n$ bits.

The total label size is $\log i + S(n/i) + \log\log n + O(1)$ bits, which is less than $S(n) + \log\log n + O(1)$ if $S(n) - S(n/i) \geq \log i - c$ for some constant c. Since f is a function of two vertices from the same tree, this altered labeling scheme can answer both queries for f as well as *Connectivity*. It is now required that any label assigned has size exactly $S(n) + \log\log n$ bits, so that the decoder may correctly identify *sep* in the bit string. For that purpose we pad labels with less bits with sufficiently many 0's. The decoder can identify C in $O(1)$ time. □

As a corollary, we get labeling schemes of the sizes reported in Table 8.1.

8.3.1 Lower Bounds

We now show that the upper bounds implied by Theorem 25 for labeling schemes supporting *Siblings* and *Connectivity* are indeed tight for both the unique and non-unique cases. To that end we consider the following forests: For any integers a, b, n such that $ab \mid n$ denote by $F_n(a, b)$ a forest consisting of a components (trees), each with b *sibling groups*, where each sibling group consist of $\frac{n}{a \cdot b}$ vertices. Note that $n \leq |F_n(a, b)| < 2n$ since we add one auxiliary root per component.

Our proofs work as follows: Firstly, for any two forests $F_n(a, b)$ and $F_n(c, d)$ as defined above, we establish an upper bound on the number of labels that can be assigned to both $F_n(a, b)$ and $F_n(c, d)$. Secondly, for a carefully chosen family of forests $F_n(a_1, b_1), \ldots, F_n(a_k, b_k)$, we show that when labeling $F_n(a_i, b_i)$ at least a constant fraction of the labels has to be distinct from the labels of $F_n(a_1, b_1), \ldots, F_n(a_{i-1}, b_{i-1})$. Finally, by summing over each $F_n(a_i, b_i)$ we show that a sufficiently large number of bits are required by any labeling scheme supporting the desired queries.

Our technique simplifies the boxes and groups argument of Alstrup et al. [42], and generalizes to the case of two nested equivalence classes[5], namely *Connectivity* and *Siblings*.

Lemma 27. *Let $F_n(a, b)$ and $F_n(c, d)$ be two forests such that $ab \geq cd$. Fix some unique labeling scheme supporting both* Connectivity *and* Siblings*, and denote the set of labels assigned to $F_n(a, b)$ and $F_n(c, d)$ as e_1 and e_2 respectively. Then*

$$|e_1 \cap e_2| \leq \min(a, c) \cdot \min(b, d) \cdot \frac{n}{a \cdot b} .$$

[5] See [52] for definitions and further discussion.

Proof. Consider label sets s_1 and s_2 of two sibling groups from $F_n(a,b)$ and $F_n(c,d)$ respectively for which $|s_1 \cap s_2| \geq 1$. Clearly, we must have $|s_1 \cap s_2| \leq \min(|s_1|,|s_2|) = \frac{n}{a \cdot b}$. Furthermore, no other sibling group of $F_n(a,b)$ or $F_n(c,d)$ can be assigned labels from $s_1 \cup s_2$, as the *Siblings* relationship must be maintained. We can thus create a one-to-one matching between the sibling groups of $F_n(a,b)$ and $F_n(c,d)$, that have labels in common (note that not all sibling groups will necessarily be mapped). Bounding the number of common labels thus becomes a problem of bounding the size of this matching. In order to maintain the *Connectivity* relation, sibling groups from one component cannot be matched to several components. Therefore at most $\min(b,d)$ sibling groups can be shared per component, and at most $\min(a,c)$ components can be shared. Combining this gives the final bound of $\min(a,c) \cdot \min(b,d) \cdot \frac{n}{a \cdot b}$. □

Lemma 28. *Let $F_n(a_1,b_1),\ldots,F_n(a_i,b_i)$ be a family of forests with $a_1 \cdot b_1 \leq \ldots \leq a_i \cdot b_i$. Assume there exists a unique labeling scheme supporting both* Connectivity *and* Siblings*, and let e_j be the set of labels assigned by this scheme to the forest $F_n(a_j,b_j)$. Assume that the sets $e_1,\ldots,e_{i-1}$ have been assigned. The number of distinct labels introduced by the encoder when assigning e_i is at least*

$$n - \sum_{j=1}^{i-1} \min(a_j,a_i) \cdot \min(b_j,b_i) \cdot \frac{n}{a_i \cdot b_i} \ .$$

We demonstrate the use of Lemma 28 by showing the following known result [42].

Warmup 1. *Any static labeling scheme for* Connectivity *queries requires at least $\log n + \log\log n - O(1)$ bits.*

Proof. Consider the family of $\log_3 n$ forests $F_n(3^0,1), F_n(3^1,1),\ldots,F_n(3^{\log_3 n},1)$. This family is demonstrated in Figure 8.3 for $n = 9$. Two vertices are *Siblings* if and only if they are connected in this family. Therefore we can use Lemma 28 even though we want to show a lower bound for only *Connectivity*. Note, that in Figure 8.3 the second forest can at most reuse 3 labels from the first, and the third can at most reuse 4 from the two previous.

Let e_j denote the label set assigned by an encoder for $F_n(3^j,1)$. We assume that the labels are assigned in the order $e_0,\ldots,e_{\log_3 n}$. By Lemma 28 the number of distinct labels introduced when assigning e_j is at least

$$n - n\sum_{i=0}^{j-1} 3^{i-j} > n/2 \ .$$

It follows that labeling the $\log_3 n$ forests in the family requires at least $\Omega(n \log n)$ distinct labels. □

We are now ready to prove the main theorem of this part.

Theorem 26. *Any unique static labeling scheme supporting both* Connectivity *and sibling queries requires labels of size at least $\log n + 2\log\log n - O(1)$.*

Proof. Fix some integer x, and assume that n is a power of x. We consider the family of forests $F_n(1,1), F_n(x,1), F_n(1,x), F_n(x^2,1), F_n(x,x), F_n(1,x^2),\ldots,$ $F_n(1,x^{\log_x n})$.

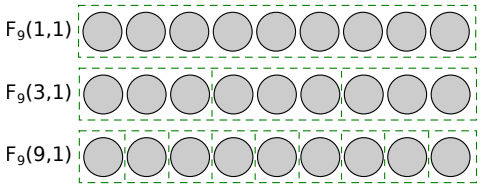


FIGURE 8.3: The family of forests $F_9(1,1), F_9(3,1), F_9(9,1)$. Vertices inside the same box are connected and *Siblings*. Note that component roots have been omitted.

Let e_a^b denote the label set assigned to $F_n(x^a, x^b)$ by an encoder. We assign the labels in the order $e_0^0, e_1^0, e_0^1, e_2^0, e_1^1, \ldots, e_0^{\log_x n}$. Thus, when assigning e_a^b we have already assigned all label sets e_c^d with $c+d < a+b$ or $c+d = a+b$ and $d < b$. By Lemma 28, the number of distinct labels introduced when assigning e_a^b is at least

$$n - \sum_{\substack{c+d<a+b \\ c,d\geq 0}} \frac{n}{x^{a+b}} \cdot x^{\min(a,c)+\min(b,d)} + \sum_{d=0}^{b-1} \frac{n}{x^{a+b}} \cdot x^{a+d}$$

This counting argument is better demonstrated in Figure 8.4. In the figure, we are concerned with assigning the labels in e_2^2. The grey boxes represent the label sets already assigned, and the right-side figure shows the fractions of n that each set e_c^d at most has in common with e_2^2. Observe that we can split the above sum into three cases as demonstrated in the figure: If $c \leq a$ and $d \leq b$ the bound supplied by Lemma 27 is $x^{c+d-a-b}$. Otherwise, either $c > a$ or $d > b$, but not both. If $c > a$, recall that $d < b$ so the bound is x^{d-b}. For $d > b$ the bound is x^{c-a} by the same argument. Applying these rules, we see that the number of distinct labels introduced is at least

$$\begin{aligned} &n - n \cdot \left(\sum_{c=0}^{a} \sum_{d=0}^{b} x^{c+d-a-b} + \sum_{d=0}^{b-1} (b-d) \cdot x^{d-b} + \sum_{c=0}^{a-2} (a-c) \cdot x^{c-a} \right) + n \\ &\geq n - n \cdot \left(\frac{x^2+x+2}{(x-1)^2} \right) + n = n - n \cdot \frac{3x+1}{(x-1)^2} \ . \end{aligned}$$

Note that we add n, as we have also subtracted n labels for the case $(c,d) = (a,b)$.

By setting $x = 6$ we get that the encoder must introduce $6n/25$ distinct labels for each e_a^b. Since we have $\Theta(\log^2 n)$ forests, a total of $\Omega(n \log^2 n)$ labels are required for labeling the family of forests. Each forest consists of no more than $2n$ vertices, which concludes the proof. □

The same proof technique is used to prove the following theorems.

Theorem 27. *Any unique static labeling scheme supporting both* Connectivity *and* Ancestry *queries requires labels of size at least* $\log n + 2\log\log n - O(1)$.

Theorem 28. *Any static labeling scheme supporting both* Connectivity *and sibling queries requires at least* $\log n + \log\log n - O(1)$ *bits if the labels need not be unique.*

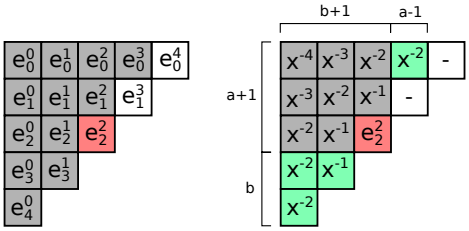


FIGURE 8.4: Demonstration of the label counting for e_2^2.

Proof. Assume w.l.o.g. that n is a power of 3. Consider the family of $\log_3 n$ forests $F_n(1,n), F_n(3,n/3), F_n(3^2,n/3^2), \ldots, F_n(3^{\log_3 n},1)$. Since each sibling group of the forest $F_n(3^i,n/3^i)$ has exactly one vertex, we note that no two vertices are *Siblings*. Thus each label of the forest has to be unique, since we have assumed that a vertex is sibling to itself. We can thus use Lemma 27 as if we were in the unique case for this family of forests.

Let e_j denote the label set assigned by an encoder for $F_n(3^j,n/3^j)$. We assume that the labels are assigned in the order $e_0,\ldots,e_{\log_3 n}$. By Lemma 28 the number of distinct labels introduced when assigning e_j is at least

$$n - n\sum_{i=0}^{j-1} 3^{i-j} > n/2.$$

It follows that when labeling each of the $\log_3 n$ forests in the family, any encoder must introduce at least $n/2$ distinct labels, i.e. $\Omega(n\log n)$ distinct labels in total. The family consist of forests with no more than $2n$ vertices, which concludes the proof. □

8.4 Concluding remarks

We have considered multifunctional labels for the functions *Adjacency*, *Siblings* and *Connectivity*. We also provided a lower bound for *Ancestry* and *Connectivity*. A major open question is whether it is possible to have a label of size $\log n + O(\log\log n)$ supporting all of the functions.

In the context of dynamic labeling schemes, if arbitrary insertion is permitted, neither *Adjacency* nor sibling labels are possible. All dynamic labeling schemes also operate when leaf removal is allowed, simply by erasing the removed label.

Chapter 9

An experimental analysis of dynamic labeling schemes

We present an implementation and evaluation based on simulation of dynamic labeling schemes for tree networks. Unlike the results from Chapter 8 and section 7.7, the model used in this body of work lifts the assumptions that labels may not be modified. Two algorithms are studied: a general scheme that converts static labeling schemes to dynamic, and a specialized dynamic distance labeling scheme. Our study shows that theoretical bounds only partially portray the performance of such dynamic labeling schemes in practice. First, we observe order-of-magnitude differences between the gains in label size when compared to the number of messages passed. Second, we demonstrate a significant bottleneck in the tree network, suggesting that the current practice of counting total messages passed in the whole network is insufficient to properly characterize performance of these distributed algorithms. Finally, our experiments provide intuition on the worst case scenarios for the stated algorithms, in particular path tree networks and fully dynamic schemes permitting both vertex additions and deletions.

9.1 Introduction

Korman, Peleg and Rodeh [125] studied labeling schemes in the context of distributed systems. In this context, information about labeling is exclusively communicated via messages, and vertices may join or leave the system dynamically. Since communication is required, maximum label size is not the sole criterion for the quality of such labeling schemes, but also metrics related to the number and total size of messages passing in the network. The authors suggested two algorithms that fit this context: a conversion of static labeling schemes, and a specialized distance labeling scheme. Additional work investigated specialized labeling schemes for routing [127, 129] and various trade-offs for distance [34, 126].

As seen in Section 7.7, Cohen, Kaplan and Milo [121] showed that if relabeling is not permitted, there is a a tight bound of $\Theta(m)$ bit for labels supporting m insertions, and it is thus not surprising that a large body of work for real-life dynamic systems for most functions [1] focused on a dynamic model with re-labeling. We focus on the function distance, as its labels are expressive enough to answer all the queries mentioned above.

[1] More specifically: *NCA*, *Routing*, *Distance*, *MaxFlow* and *Center*. Recall that in Chapter 8 we showed that *Siblings*, *Adjacency* and *Connectivity* enjoy simple dynamic labeling schemes of size $2 \log n$.

Previous experimental papers on the topic focused on the performance of static labeling schemes, emphasizing the label size as the main metric. Caminiti et al. [130] experimented on the influence that different tree decompositions have on label size for *NCA* labeling schemes. Fischer [131] evaluated the label size expected for a variant of *NCA*, using various coding algorithms. Kaplan et al. [116] presented experiments to support an ancestry labeling scheme trading slightly larger labels for much faster query time. Finally, Cohen et al. [132] performed experiments on a novel technique for labeling schemes for graphs.

These papers reveal that different families of trees present very different label sizes. A natural question is whether dynamic labeling schemes present the same behaviour. In addition, it remains unclear how the various communication and label size trade-offs suggested for dynamic labeling schemes behave in practice. Moreover, one may question whether it is sensible to develop dynamic labeling schemes for individual functions, or whether the trade-offs given by general methods are good enough [125]. Furthermore, the price of allowing for deletions, rather than just insertions, still needs to be characterized in representative scenarios. In short, many questions are relevant and interesting for dynamic labeling schemes, but we are unaware of any experimental papers on the topic.

In this chapter, we devise a simulation of a tree network and answer those questions for the dynamic labeling schemes proposed by Korman, Peleg and Rodeh [125]. These labeling schemes can be classified along two dimensions: whether the labeling scheme is specialized for distance or supports general static labelers, and whether the labeling scheme supports both insertion and deletions or only insertions. Our findings suggest three main observations:

1. Generally, the amortized complexity analysis of [125] is verified in our experiments. However, for labeling schemes supporting general static labelers, the observed behaviour indicates that the complexity of the static labeler is overshadowed by the overhead introduced by the dynamic scheme.

2. In order to achieve an asymptotically tight label size, the specialized distance labeling scheme uses significantly more communication.

3. The message distribution in labeling schemes that support both insertion and deletion is severely skewed. We show an experiment in which the majority of messages in the system is communicated by less than 0.4% of the vertices.

9.1.1 Preliminaries

We adopt the terminology in [125], which names dynamic encoding algorithms as *distributed online protocols* or simply *protocols*. From here on, all dynamic labeling schemes discussed are for tree networks. The following types of topology changes are considered: *a*) *Add-Leaf*, where a new degree-one vertex u is added as a child of an existing vertex v ; and *b*) *Remove-Leaf*, where a (non-root) leaf v is deleted. Dynamic labeling schemes that support only *Add-Leaf* are denoted *semi-dynamic*, and those that support both operations are denoted *fully-dynamic*.

Metrics for dynamic labeling schemes. In the literature surveyed, labeling schemes aim for smallest possible label. If communication is not accounted for, dynamic labeling schemes can trivially achieve the optimal, static bounds. Therefore, in order to account for the cost of communication, the following metrics are introduced [125]. Let M be a dynamic labeling scheme, with re-labeling allowed, and let $S(n)$ be a sequence of n topological changes.

1. *Label Size* $\mathcal{LS}(M, n)$: the maximum size of a label assigned by M on $S(n)$.
2. *Message Complexity*, $\mathcal{MC}(M, n)$: the maximum number of messages sent in total by M on $S(n)$. Messages are sent exclusively between adjacent vertices.
3. *Bit Complexity*, $\mathcal{BC}(M, n)$: the maximum number of bits sent in total by M on $S(n)$. Note that $\mathcal{BC}(M, n) \leq \mathcal{MC}(M, n) \cdot \mathcal{LS}(M, n)$.

9.2 Dynamic labeling schemes for tree networks

Korman et al. [125] presented *semi-dynamic* and *fully-dynamic* labeling schemes that support distance queries and a *semi-dynamic* and *fully-dynamic* labeling scheme that receives a static labeling scheme[2] as input and supports queries of its type. In this section, we give a brief and informal description of both semi-dynamic labeling schemes as well as of the conversion necessary to make these schemes fully-dynamic.

We denote the static distance labeling scheme as *Distance*, its extension to a specialized semi-dynamic mode *SemDL*, and its fully-dynamic specialized extension as *DL*. Korman et al. [125] define the general dynamic labeling schemes *SemGL* and *GL*, which maintain labels on each vertex of a dynamically changing tree network using a static labeling scheme as a subroutine. The performance of these schemes is tightly coupled to the performance of the static scheme used. Throughout the remainder, we denote the general labeling schemes operating on *Distance* simply as *SemGL* and *GL*, and explicitly mention other distance functions where appropriate. Table 9.1 reports the different complexities for the aforementioned schemes (the parameter d is explained in Section 9.2.1).

9.2.1 Brief overview

In this section, we provide an informal description for *Distance*, *SemDL*, *SemGL*, and the approach for semi-dynamic to dynamic conversion.

Encoding *Distance* directly from heavy-light decomposition. For every non-leaf v in a rooted tree T, we denote a child with the heaviest weight[3] as *heavy* and the rest as *light*; we mark the root of the tree as light and its leaves as heavy. The light vertices divide T into disjoint *heavy paths*. For any $v \in T$, the path from the root traverses at most $\log n$ light vertices,

[2] The static labeling scheme must respect a set of conditions as mentioned in [125], which to the best of our knowledge are respected by all labeling schemes in the literature.

[3] The weight of a vertex v is the number of its descendants in a tree.

Labeling Scheme	Label Size	$\mathcal{MC}$	$\mathcal{BC}$
Distance	$O(\log^2 n)$	$O(n)$	$O(n \log^2 n)$
SemDL	$O(\log^2 n)$	$O(n \log^2 n)$	$O(n \log^2 n \log\log n)$
DL	$O(\log^2 n)$	$O(n \log^2 n)$	$O(n \log^2 n \log\log n)$
SemGL	$O(\frac{d-1}{\log d} \log^3 n)$	$O(n \frac{\log n}{\log d})$	unreported
GL	$O(\log^3 n)$	$O(n \log^2 n)$	unreported

TABLE 9.1: Simplified complexity estimates for $Trees(n)$. Bounds for GL are reported with $d = 2$. See [125] for an elaborate complexity report.

and accordingly at most $\log n$ heavy paths. A label of a vertex can now be defined by interleaving *light sub-labels*, containing the index of a light child, with *heavy sub-labels*, containing only the length of the heavy path. Such labels clearly require $O(\log^2 n)$ bits, which is asymptotically optimal for labels supporting distance on trees. Computing the distance between two vertices can be done by comparing the prefixes of their corresponding labels. Finally, we note that the labels produced by *Distance* can be used to solve most queries in the literature, namely *Adjacency*, *Siblings*, *Ancestry*, *Routing* and *NCA*.

SemDL. It is not essential for the correctness of the decomposition that the heavy vertex selected be in fact the heaviest, or say, the 3rd heaviest. It is only essential for the bound on the label size. *semDL* maintains a dynamic version of the *Distance* labels using precisely this observation. Every vertex maintains an estimate of its weight, and transfers this estimate to its parent, using a previously introduced binning method [133]. When the estimate exceeds a threshold in a vertex, this implies that a vertex other than the heavy vertex is now significantly heavier than the heavy vertex. Thus, a re-labeling (shuffle, in [125]) is instantiated on the subtree to maintain the $O(\log^2 n)$ label size.

SemGL. The protocol is designed to transform any static labeling scheme *Static* to semi-dynamic, and therefore, does not utilize the heavy-light decomposition directly. Instead, the protocol operates *Static* separately on a cleverly constructed sets of so called bubbles described as follows.

Each vertex $v \in T$ is included in exactly one induced subtree of T, denoted *bubble* of *order* i ($0 \leq i \leq \log_d n$) that contains at least d^i vertices. The bubbles constitute a *bubble tree*, where the order of a bubble is always less or equal to the order of the parent bubble. In addition, there are no d consecutive bubbles of the same order in any path of the bubble tree. A vertex added to the tree is assigned the order 0. If this insertion yields d consecutive bubbles of order 0 in the bubble tree, they are merged to a single bubble of order 1, and the condition is checked again for bubbles of order 1 and so on. We denote the parent of the root of bubble b as b_p and the function that *Static* supports as f. Essentially, the label of a vertex v in a bubble b is a concatenation of the label of b_p with both a local *Static* label of v in b, and the result of $f(v, b_p)$.

Semi-dynamic to fully-dynamic conversion. Both labeling schemes are converted to their counterpart fully-dynamic labeling schemes, *DL* and GL, using an additional simple protocol. The protocol uses the binning method [133] to maintain local weight estimates for each vertex. The protocol simply ignores the topological changes up to the point in which their number is large, and then re-labels the entire tree.

9.3 Experimental framework

We have used simulation as our performance measurement methodology [134]. There are several reasons for this choice. First, distributed systems of realistic size in the order of millions of vertices are unavailable to us. Second, simulation allows us to remain isolated from effects such as network interference. Third, simulation is a sufficient tool to compare the metrics intended for our purposes. All implementations have been realized in C#. The full package is available over the Internet at the URL: `http://www.diku.dk/~noyro/dynamic-labeling.zip`.

Performance Indicators. In order to match the analysis, we use the metrics defined above, namely $\mathcal{MC}$, $\mathcal{BC}$, and $\mathcal{LS}$, denoted *total network messages*, *total network bandwidth*, and *maximum label length* resp. In particular, we are interested in the tradeoff between $\mathcal{LS}$ and $\mathcal{MC}$ in the different systems. Even though we report the bit complexity, message sizes in our setting are bounded by the label sizes, which tend to be very small. Since in networks the start up costs of sending small messages dominate messaging cost, it is sufficient to focus on message complexity alone. In addition, we study the distribution of messages passing through a single vertex. The latter provides insight into the congestion for worst-case network performance.

The algorithms present slightly improved bounds if the number of topological events n is known in advance. We simulate labeling schemes that are not aware of n. It is beyond the scope of our paper to account for neither the construction time, nor the performance of the decoder.

Selected static functions. To test *SemGL* and *GL*, we provide, in addition to *Distance*, the static labeling scheme *Ancestry* [2] with labels of size $2 \log n$.

Test Sets. We consider insertions sequences creating the following trees.

- **Complete K-ary Trees.** K-ary trees are trees that have a maximum number of children allowed per vertex. A complete K-ary tree is a K-ary tree in which all leaf vertices reside on the lowest level and all other vertices have K children. The leaves are inserted starting from the leftmost vertices of the tree. We denote a 2-ary tree as *full binary tree*.

- **Skewed Trees: star and path.** A star is a rooted tree with children adjacent to the root. In a path tree, all vertices have either one or no children.

- **Random Trees.** We implemented random insertion sequences creating trees of bounded depth δ as well as of unbounded depth. While generating trees that are truly random is challenging, we follow a simple iterative procedure. First, select a vertex with equal probability among the vertices with depth less than δ. Then, insert a new vertex under the selected vertex and iterate.

From [130, 131], we conclude that labels constructed using a heavy-light decomposition yield the largest label size when applied on full binary trees. Since we would like to focus on the performance of the dynamic protocols, we use full binary trees as our main test set.

Counted Bits. The labels constructed have polylogarithmic size, and each is composed of a number of components of variable sizes. Therefore, we must account for a realistic bit encoding. Bit accounting matters for both $\mathcal{LS}$ and $\mathcal{BC}$. All labels are accounted by twice their theoretical label size, so that the encoder could find the beginning and the end of each of the components. The same applies for messages. We use unary encoding to represent the number of non-empty elements in each row of the label matrix [125].

9.4 Experimental results

In this section, we summarize our main experimental findings. First, we test *semGL* with two static labeling schemes and outline the role d plays in the protocol (Section 9.4.1). We then proceed by directly comparing *SemGL* to *SemDL* (Section 9.4.2). We conclude by observing the performance of *GL* in comparison to *SemGL* and *DL*, as well as comparing the distribution of messages in *GL* and *SemGL* (Section 9.4.3). All figures presented but Figure 9.3 and Figure 9.7 have the number of vertex additions in their x-axes in a log-scale. Figures describing $\mathcal{MC}$ and $\mathcal{BC}$ have their y-axes in log-scale, and figures describing $\mathcal{LS}$ are in linear scale.

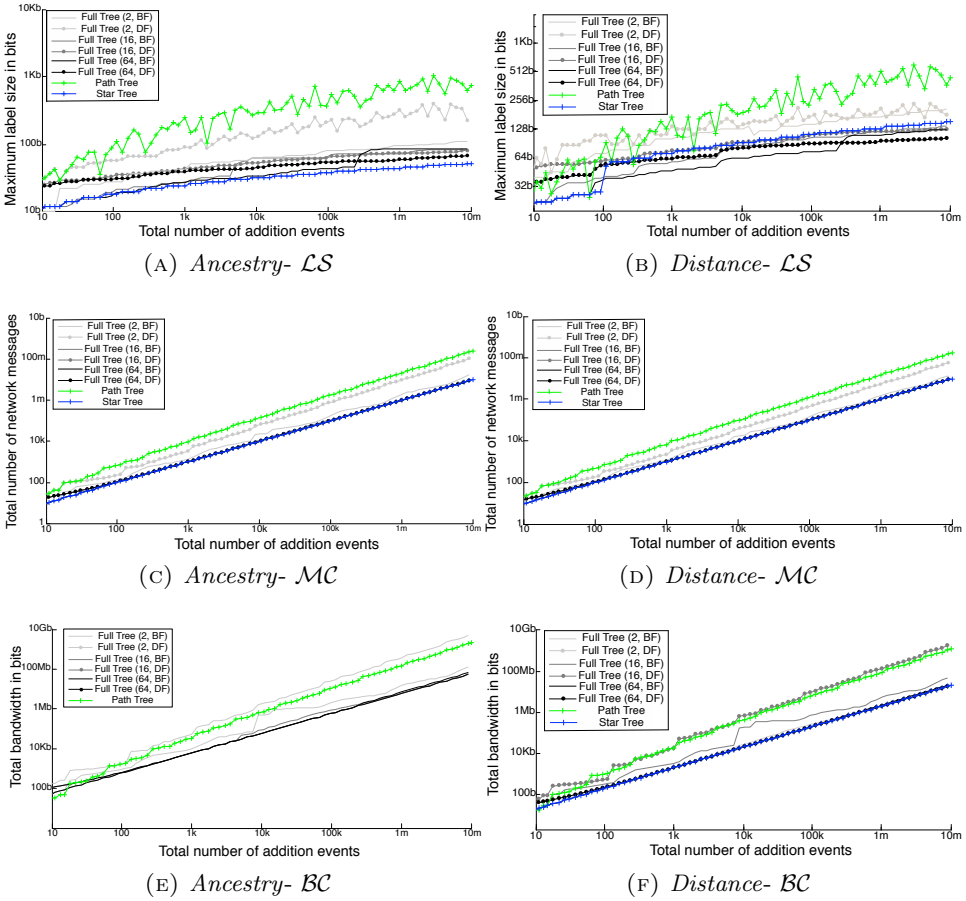
9.4.1 *SemGL*

We discuss two experiments dealing exclusively with the performance of *semGL*.

Varying tree type. For the purposes of this section, we compare *SemGL* under two static labeling schemes, namely *Distance* and *Ancestry*. The test is performed on eight different, deterministically constructed trees, where the full K-ary trees are expanded either in a *breadth-first* or a *depth-first* manner. We fix $d = 4$, as it best illustrates the differences observed.

Overall, we observe similar trends for both static labeling schemes (Figures 9.1a, 9.1c and 9.1e vs. Figures 9.1b, 9.1d and 9.1f). However, we observe an apparent anomaly on the largest label sizes produced in Figure 9.1b. While in general labels for *Ancestry* should be more compact than labels for *Distance*, the opposite effect is evident for paths. This effect can be explained when we recall that on a path, the label size of *Distance* is a straightforward $\log n$ bits, whereas the label size of *Ancestry* is fixed to $2 \log n$.

We observe that the maximum label size of *SemGL* is directly related to the depth of the tree, such that trees with higher depth yield larger labels.



(A) *Ancestry- $\mathcal{LS}$* (B) *Distance- $\mathcal{LS}$*

(C) *Ancestry- $\mathcal{MC}$* (D) *Distance- $\mathcal{MC}$*

(E) *Ancestry- $\mathcal{BC}$* (F) *Distance- $\mathcal{BC}$*

FIGURE 9.1: *SemGL* on *Distance* and *Ancestry* static labeling schemes, for various trees of different order and vertex insertion modes, with $d = 4$.

In addition, trees with higher depth have more variance in their maximum label size. Both of these properties can be attributed to the *bubble tree* structure. Since the bubble order directly relates to the maximum vertex depth, higher depth leads to larger *bubble order*, which yields a larger label size. The second property, namely higher variance of maximum label size in trees of higher depth, is attributed to more frequent high-level relabeling. At the noticeable decline points, the encoder relabels large parts of the tree to ensure the logarithmic label size.

A clear evidence for the correlation of label size to the depth of the tree is that for random trees of bounded and unbounded depth, the label size grows consistently with depth, as shown in Figure 9.2. Comparing Figure 9.2 to Figure 9.1, the values for random trees lie between the values for path and the remaining trees. We also note that while the total number of messages is essentially identical, the bit complexity is higher as the depth is higher. We deduce that the depth influences the size of each message, and not just the number of messages.

Given these results, in the rest of the experiments we focus on full binary trees traversed in depth-first manner. We have observed in separate experiments that these trees generate similar trends as random trees.

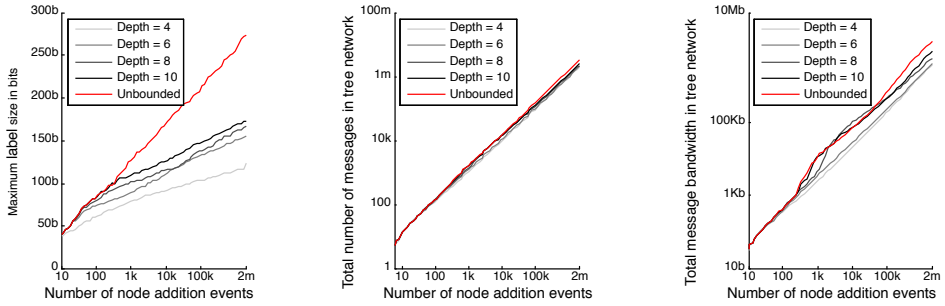


FIGURE 9.2: *SemGL* labels for random trees with variable bounded depth. Each curve represents the average of 10 executions, with $d = 4$.

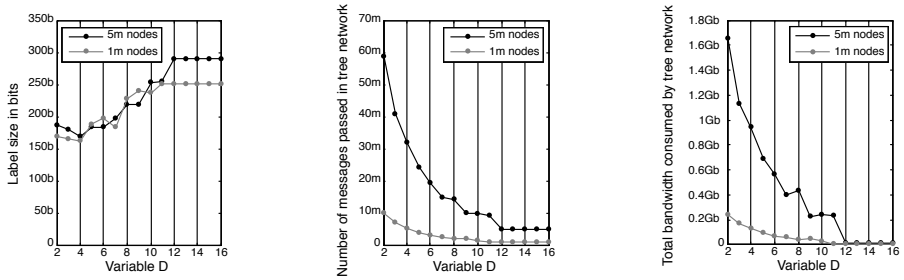


FIGURE 9.3: *SemGL* Variable d experiment for full binary trees expanded in a depth-first manner. Samples are taken for d in [2,16] in trees of size 1m and 5m vertices.

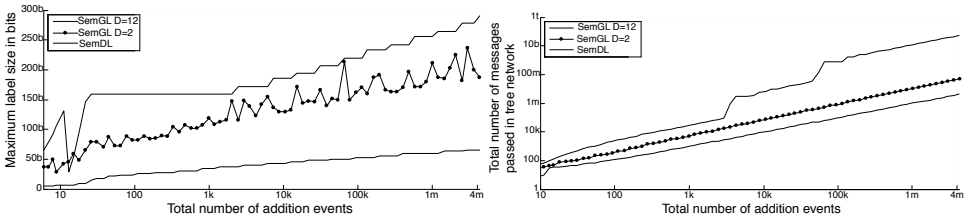
Varying the d variable. In Figure 9.3, we vary the value of d in values between 2 and 16, and analyze its influence on *SemGL*. We choose a set of low d values to trigger larger overhead on the message complexity of *SemGL*, such that performance changes are more noticeable. The results are similar for *SemGL* with the *Ancestry* static labeling scheme, and thus omitted from the figure.

We observe two trends in the figures. First, the number of messages passed in the network drops logarithmically as we increase d. The same drop can be observed for the total bandwidth. In addition, the label size increases as d increases. This trend is in line with the expected $\frac{d-1}{\log d}$ expression reported in Table 9.1.

In addition, we notice a decay in number of messages in Figure 9.3 which is also in line with the expected $O((\log d)^{-1})$ curve for $\mathcal{MC}(SemGL, n)$. The total bandwidth consumed by tree network reacts to d in the same manner that the total number of messages does. The values stabilize when $d = 12$ for approx. 5 m (million) vertices and 11 for 1 m vertices. Since both complexities are showing similar behavior throughout all experiments, we omit graphs describing the total bandwidth in the remainder for brevity.

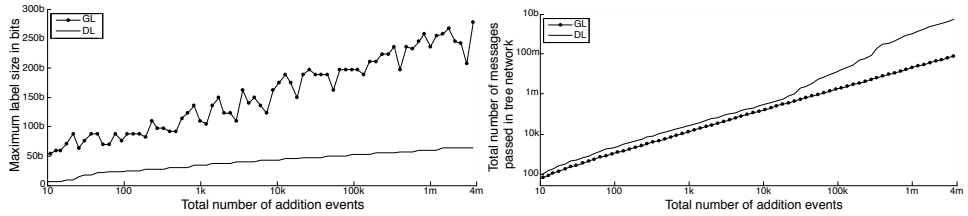
9.4.2 Comparison of *SemGL* and *SemDL*

Figure 9.4a shows as expected that *SemGL* produces larger labels than *semDL*. Recall that by Table 9.1 we expect a log factor in the label size between both methods. For trees with 4m vertices, the latter translates into a factor of 20, in contrast to the factor of four seen in the figure. On the



(A) (B)

FIGURE 9.4: *SemGL* versus *SemDL* on full binary trees expanded in a depth-first manner: (a) the maximum label size; (b) the number of messages sent.



(A) (B)

FIGURE 9.5: Comparison of *DL* and *GL* (a) is the maximum label size measured in bits and (b) is the total number of messages passed

other hand, Figure 9.4b shows that the distance-specialized *semDL* uses up to 270 times more messages in comparison to *SemGL*, when we would again by Table 9.1 expect a reverse multiplicative log factor[4]. This suggests that the trade-off between label size and communication expected by the reported complexities is in practice overshadowed by significant constants. The ratio of the factor differences amounts to two orders of magnitude.

Comparing Figure 9.4a and Figure 9.4b, *semDL* presents a stable increase in the label size in contrast to the two bumps in its message size. The latter is in contrast to the previously explained phenomena in *SemGL*. The two bumps mark a re-label operation done from the root, and contribute to almost an order-of-magnitude difference in the number of messages passed.

9.4.3 Fully-dynamic labeling schemes

In this section, we analyze the performance of fully-dynamic protocols, and the performance differences between semi-dynamic and dynamic labeling schemes. We use a full binary tree, generated in a depth-first manner.

Comparison of *DL* and *GL*. In this experiment, we compare *DL* and *GL* for tree expansion, i.e., additions of vertices to the tree. We remark that *SemGL* exhibited a definite advantage in $\mathcal{MC}$ over *semDL* (Section 9.4.2). Similarly, we note in Figures 9.5a and 9.5b that *GL* performs significantly better than *DL* in terms of network messages. For 4m vertices, *GL* uses roughly 100 times less messages compared to *DL*. At the same time, as expected *DL* yields labels that are up to five times smaller than *GL*.

[4]Note that d is a low constant in our experiment

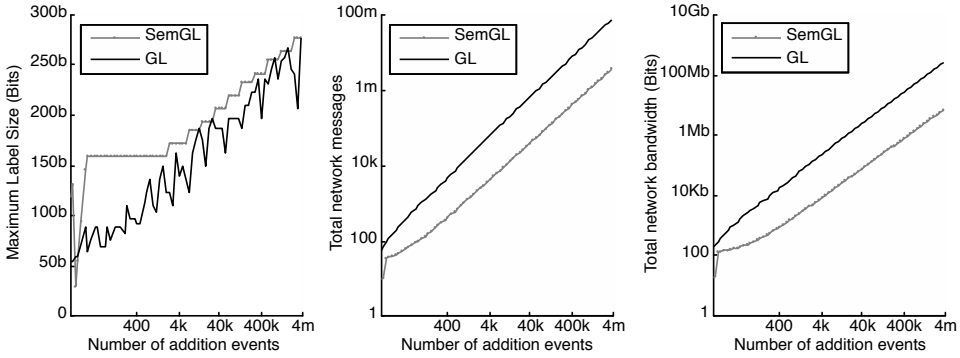


FIGURE 9.6: Comparison of *semGL* ($d = 12$) and *GL*

Comparison of *semGL* and *GL*. In this experiment, we compare *semGL* and *GL* again over a tree expansion. We emphasise that no deletion operations are performed. We aim to give an insight on the overhead caused by transforming the semi-dynamic labeling scheme to fully-dynamic.

In Figure 9.6, we view that the maximum label size is lower for *GL*. However, as we scale on the number of vertices, both curves tend to approach the same value. GL's number of messages is, however, 100 times larger than *semGL* for 4m vertices. This constitutes a significant overhead in the execution of the fully-dynamic labeling scheme. For perspective, 4.3m messages suffice to label 4m vertices using *semGL*, but only 0.23m vertices using *GL*.

messages (range)	vertices	Percent of vertices	Percent of messages
0 - 5	500018	50.00%	2.6%
5-50	468725	46.87%	23.4%
51-500	27366	2.74%	15.9%
501-5000	3646	0.36%	20.86%
5001-50000	215	0.02%	15.66%
50001-500000	30	0.00003%	21.55%

TABLE 9.2: The distribution of messages per vertex for *GL*, with $d = 14$.

Message distribution. We group the vertices according to the number of messages that pass through them throughout the *GL* protocol. Table 9.2 shows that the distribution of messages is remarkably skewed. Interestingly, 0.00003% of the vertices account for 21.55% of the total number of messages passed. Each of the top 30 vertices in the distribution have either sent or received between $50001 - 500000$ messages. The top vertex, in particular, has either sent or received about 0.5m out of a total of 19m messages. We have verified that the top vertex in the distribution corresponds to the root, and the top 30 vertices are the shallowest vertices in the tree.

In contrast, Figure 9.7 shows the distribution of messages for *semGL* on a worst case path network with 1m vertices. *semGL* does not incur similar bottlenecks as *GL*. Instead, the figure shows a distribution that resembles a bell curve: Most vertices either send or receive around 20 messages, and the differences in number of messages among vertices are not stark.

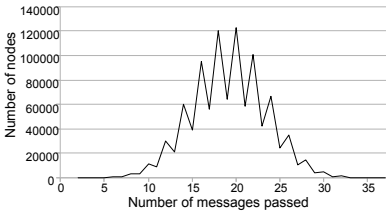


FIGURE 9.7: The distribution of messages per vertex for *semGL*, with $d = 2$ on a path.

This result suggests that *GL* creates a network bottleneck by concentrating the messages passed. This behavior is undesirable in a distributed system.

9.5 Conclusions

Overall, the algorithms perform as expected given the amortized complexity analysis. However, the experiments reveal constant factors that clearly affect performance. We observed that fully-dynamic protocols induce a severe performance overhead when compared to semi-dynamic variants. Moreover, as seen in the distribution of messages passed, fully-dynamic protocols also lead to severe bottlenecks in the tree network.

We affirm Korman et al.'s recommendations, namely to reset the d parameter of *GL* on a restart, and for *semGL*, to predict, if plausible, the target tree length n_f in order to set d to a value close to $\log n_f$. On the other hand, Korman et al. argue for trading network communication for improved label size in *DL* [125]. In light of our results, we suggest that in practice, the opposite trade-off is required, namely larger labeling schemes with decreased message complexity.

Korman et al. have developed *DL* with a metric that accumulates network messages, amortizing this value over a number of insertions. Our experiments show that such a metric allows for major bottlenecks and non-distributed system in practice. We suggest that a revised metric that accounts for the total number of messages passing through a single vertex be investigated as part of future work.

Chapter 10

Routing Labeling Schemes

We first describe general *Routing* schemes and two models for labeling schemes of *Routing* called *designer port model* and *fixed port model*. In Section 10.1.1 we provide a literature review. In Section 10.2 we describe in detail the current best upper bound for *Routing* in the designer port model due to [9]. We provide a corrected proof for this result. In Section 10.3 we present a proof omitted in [9], that shows an efficient construction of a *Routing* scheme for the *fixed port* model.

10.1 Introduction to *Routing* schemes

Labeling schemes are one of many methods to maintain a *Routing* scheme. A *Routing* scheme is a mechanism that can deliver packets of information between any two vertices in the network. Typically, such mechanism consist of *I.*) a *Routing* function *II.*) the format of the address; *III.*) a local labeling of the vertices; *IV.*) a message header format, and *V.*) local information to perform the computation of the *Routing* function [26].

Routing labeling schemes produce *direct Routing* schemes, which means that the message header is fixed once by the source host, and cannot be modified by intermediate vertices on its route to the destination. Rather than supplying the entire path, a vertex provides only the next vertex to visit in order for the packet to arrive at its destination. The local labelings of edges from every vertex are identified by so called *port-numbers*.

Definition 18. *Let $T = (V, E)$ be a tree with n vertices, and let $u \in V$ be a vertex with degree Δ. A* port numbering *is an injective function that assigns integers to the edges incident to v.*

Note that a port assignment is performed locally, and an edge can receive two different port numbers from each of its vertices. The two main variants considered in the literature are the *designer port* model and *fixed port* model. The former allows the encoder to freely enumerate the incident ports to all vertices, while the latter assumes that the port numbers are fixed by an adversary.

Routing labeling schemes are related to two functions discussed in the thesis, namely *Ancestry* and *NCA*. Unlike *Ancestry*, a *Routing* query is meaningful even for unrooted trees. In the literature surveyed, the tree is assumed to be rooted, and for the designer port model the port number of the parent is 0 and the port number of the heavy[1] child is 1. Under these assumptions, designer port *Routing* labeling schemes produce labels that are able to determine *Ancestry*, since for an ancestry query $\mathcal{L}(u), \mathcal{L}(v)$ we can run

[1] A child with maximal number of decedents.

the *Routing* decoder and return true if its value is not 0. Finally, we note that the *Label-NCA* labeling scheme presented in Theorem 37 can answer designer port *Routing* queries, since the labels can determine the first edge on the shortest path between the vertices queried.

10.1.1 Literature review

Labeling schemes for *Routing* were introduced by Peleg [135]. Fraigniaud and Gavoille [26] achieved a labeling scheme of size $3\log n$ for the function. Thorup and Zwick [9] presented, almost at the same time, an improvement of the label size to $(1+o(1))\log n$ for the designer port model. Fraigniaud and Gavoille [136] then showed a lower bound for any fixed port labeling scheme for *Routing* of $\Omega(\log^2 n/\log\log n)$. In an experimental paper Krioukov et al. [137] compared the performance of several labeling schemes for both models. Korman and Peleg [127,138], studied the function in a dynamic tree network settings, with permitted relabeling.

Unlike the situation in trees, there could be many paths between two vertices in arbitrary graphs. For this class of graphs, *Routing* schemes attempt to route the package along a shortest, or a close-to shortest path. The parameter measuring the quality of the path is called *stretch*[2]. [139] showed a lower bound implying that achieving stretch 3 or less requires $\Omega(n)$ bits. Abraham et al. [140] showed a stretch 3 *Routing* scheme with a $O(\sqrt{n})$ local information stored in each vertex. For surveys on *Routing* schemes see [141], and [28].

10.2 Designer port model

We present the proof for the following theorem.

Theorem 29. *[9] There exist a designer port model* Routing *labeling scheme for* $Trees(n)$*, denoted* $\langle e,d\rangle$*, with labels of size at most* $(1+o(1))\log n$*.*

We first describe the label, prove that its size is bounded, and finally describe the decoding process.

Label description Given a tree T, we first decompose it using the spines decomposition defined in Section 2.3.3. In this decomposition all vertices are on a $heavy_s$ path at some stage of the decomposition. In particular, a vertex v of size $\text{size}(v)$ is a member of a $heavy_s$ path at level $\mathcal{LD}(v)$ if $\text{size}(v) > n/b^{\mathcal{LD}(v)}$. The *level* number $\mathcal{LD}(v)$ is in the range $\{1\ldots\log_b n\}$ and stands for the number of iterations of the decomposition where v is a $light_s$ vertex.

Similarly to the definitions in Section 2.3.1, a vertex v in T with $light_s$ children $v_1\ldots v_d$ has *light size* $\text{lsize}_s(v)=\sum_{i=1}^{d} size(v_i)+1$. We also define the *light size* of a path $P=u(1)\rightsquigarrow u(k)$ as $\text{lsize}_s(P)=\sum_{i=1}^{k}\text{lsize}_s(u(i))$. The collection of paths in level i, in decreasing order of their light size, is denoted $P^i=P^i_1\ldots P^i_l$, and the light size of P^i is just the sum of their light sizes. The vertex with the k'th largest light size in the path with

[2]Formally the stretch of a *Routing* scheme is the worst case ratio between the length of the path obtained by the *Routing* scheme and the length of the shortest path between the source vertex and the destination vertex.

the j'th largest path size at level i is denoted $P_j^i[k]$. See Figure 10.1 for a demonstration.

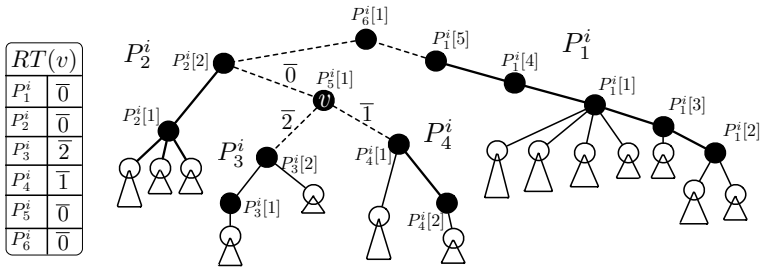


FIGURE 10.1: A demonstration of naming of the paths and $heavy_s$ vertices on level i of a spines decomposition of a subtree. The thick lines are $heavy_s$ paths. The $heavy_s$ vertices are full, the $light_s$ vertices are empty, and the triangles are represent subtrees of $light_s$ vertices. The routing table of vertex v to the left describes which one of the ports $\overline{0}, \overline{1}$ and $\overline{2}$ it needs to traverse to arrive at vertices at other $heavy_s$ paths on level i. The i'th triplet in $\mathcal{L}(v)$ is $\langle 5, 1, \epsilon \rangle$.

A vertex v is assigned a two part label $\mathcal{L}(v) = (\mathrm{Id}(v), RT(v))$, described below. The part $\mathrm{Id}(v)$ is, by itself, a concatenation of triplets of the form $\langle j_i, k_i, p_i \rangle$, for $1 \leq i \leq \mathcal{LD}(v) \leq \log_b n$, defined as follows: The parts j_i, and k_i specify $P_j^i[k]$, the $heavy_s$ vertex on level i on the path $r \rightsquigarrow v$ of maximal depth. We use Lemma 2 to number both j_i and k_i. More specifically, we choose j_i according to the relative contribution of $\mathrm{lsize}_s(P_j^i)$ to $\mathrm{lsize}_s(P^i)$, such that $|j_i| = \log(\mathrm{lsize}_s(P^i)/\mathrm{lsize}_s(P_j^i)) + 1$. Similarly, we choose k_i according to the relative contribution of $\mathrm{lsize}(P_j^i[k])$ to $\mathrm{lsize}_s(P_j^i)$. The part p_i is the port number used from $P_j^i[k]$ to the vertex v, which is also assigned using Lemma 2 with $\mathrm{size}(p_i)$ and $\mathrm{lsize}(P_j^i[k])$. Note that for the last triplet in $\mathrm{Id}(v)$ we use the empty string for $p_{\mathcal{LD}(v)}$. As in Section 10.1, the ports to the root and the $heavy_s$ child are marked 0 and 1, respectively. Each label contains up to $\log_b n$ triplets of three parts, where each part may be of different length. In order to distinguish between the parts of each triplet, we store each part in $\mathrm{Id}(v)$ using the suffix code $code_1$ (see Section 2.2). In this way a part of m bits requires $m + O(\log m)$ bits, and the different parts may be concatenated.

The second part of $\mathcal{L}(v)$, is the level routing table $RT(v)$ of v that contains the ports v must use to arrive at each of the $heavy_s$ paths $P_1^{\mathcal{LD}(v)} \ldots P_l^{\mathcal{LD}(v)}$ in v's level $\mathcal{LD}(v)$. If v is not the last vertex on its heavy path, then each description concludes in traversing the path in some direction, i.e., up or down. If v is the last vertex, we assign the associated ports with the numbers of the ports leading to the roots of other paths in level $\mathcal{LD}(v)$. This concludes the description of the label.

To compute the size of this labeling scheme we first prove the following Lemma.

Lemma 29. $\sum_{i=1}^{\log_b n} |j_i| + |k_i| + |p_i| = \log n + O(\log_b n)$.

Proof. Let $\langle j_1, k_1, p_1 \rangle$ be the first triplet in $Id(v)$ for a vertex v, and let u be the light child of $P_j^1[k]$ to which port p_1 is addressed to. We first prove that

$$|j_1| + |k_1| + |p_1| \leq \log n - \log(\mathrm{size}(u)) + O(1).$$

By definition:

$$|j_1| \leq \log(n/\text{lsize}_s(P_j^1)) + O(1) = \log n - \log(\text{lsize}_s(P_j^1)) + O(1),$$

and also

$$\begin{aligned} |k_1| &\leq \log(\text{lsize}_s(P_j^1)/\text{lsize}_s(P_j^1[k])) + O(1) \\ &= \log(\text{lsize}_s(P_j^1)) - \log(\text{lsize}_s(P_j^1[k])) + O(1). \end{aligned}$$

Similarly,

$$\begin{aligned} |p_1| &\leq \log(\text{lsize}_s(P_j^1[k])/\text{size}(u)) + O(1) \\ &= \log(\text{lsize}_s(P_j^1[k])) - \log(\text{size}(u)) + O(1). \end{aligned}$$

Summing those inequalities we get:

$$|j_1|+|k_1|+|p_1| \leq \log n - \log(\text{size}(u)) + O(1),$$

as requested.

Since $\text{lsize}_s(P^2) \leq \text{size}(u)$, we can repeat the argument so that for w, the light child of $P_j^2[k]$ addressed by port p_2 we have:

$$|j_2|+|k_2|+|p_2| \leq \log(\text{size}(u)) - \log(\text{size}(w)) + O(1).$$

Summing over all the triplets, it follows that:

$$\sum_{i=1}^{\mathcal{LD}(v)} |j_i|+|k_i|+|p_i| \leq \log n + O(1) \cdot \mathcal{LD}(v) \leq \log n + O(\log_b n).$$

□

Lemma 30. *The encoder described above produces labels of size at most* $\log n + O(\log n / \log\log n \cdot \log\log\log n)$.

Proof. Choosing $b = \lceil \sqrt{\log n} \rceil$ we can bound the number of the triplets in $\text{Id}(v)$ by $\log_b n = \frac{\log\log n}{2}$. The number of bits required to represent them is:

$$\sum_{i=1}^{\log_b n} |j_i|+|k_i|+|p_i| + O(\sum_{i=1}^{\log_b n} \log(|j_i|) + \log(|k_i|) + \log(|p_i|) + 3).$$

The first sum corresponds to the number of bits required to store the information, and the second sum is the additive number of bits required to encode each part in $code_1$. By Lemma 29 the first sum is bounded by $\log n + O(\log_b n)$ bits.

For brevity, we denote the number of parts by $k = 3\log_b n$ and the size of the k parts in $\text{Id}(v)$ by $c_1 \ldots c_{3k}$ Next we bound $\sum_{i=1}^{k} \log c_i$, where $\sum_{i=1}^{k} c_i \leq \log n + O(k)$ by Lemma 29. Since log is a concave function, we have:

$$\sum_{i=1}^{k} \log c_i \leq k \log(\frac{\sum_{i=1}^{k} c_i}{k}) = k \log(\frac{\log n + O(k)}{k}) \leq k \log(\frac{\log n}{k}) + O(1)k.$$

Since $\frac{\log n}{3\log_b n} = \frac{\log_2 b}{3}$ the expression can be bounded by $O(k \log\log b + 3\log_b n)$, which is bounded by $O(\frac{\log n}{\log\log n}\log\log\log n)$.

To account for the size of $RT(v)$, recall that at every level of the spines decomposition there can be at most $2b-1$ paths. Furthermore, since the ports stored lead to subtrees with at least n/b vertices, it is guaranteed that each port will be assigned an identifier with $O(\log b)$ bits. Thus, storing $RT(v)$ requires $O(b\log b)$ bits, and for $b = \lceil\sqrt{\log n}\rceil$, this can also be bounded by $O(\frac{\log n \cdot \log\log\log n}{\log\log n})$. □

Decoding We confirm that the information stored in the label is sufficient to determine the function *Routing*. A key observation is that while it is not possible to determine the rank of the a vertex in its $heavy_s$ path, the order between two vertices on the same $heavy_s$ path can be determined.

Let v,u be two vertices in T with labels $\mathcal{L}(v) = (\mathrm{Id}(v), RT(v))$, $\mathcal{L}(u) = (\mathrm{Id}(u), RT(u))$ with depth $\mathcal{LD}(v)$, $\mathcal{LD}(u)$, respectively in the recursive decomposition of T.

If $|\mathrm{Id}(v)| < |\mathrm{Id}(u)|$ then the decoder returns 0, since that implies that u is not an ancestor of v and the path $u \rightsquigarrow v$ must begin with the edge traversing upwards from u. Using the same argument, we return 0 if the first $\mathcal{LD}(u)-1$ triplets of both $\mathrm{Id}(v)$ and $\mathrm{Id}(u)$ are not equal.

Let $(path_v, vertex_v, edge_v)$ and $(path_u, vertex_u, \varepsilon)$ be triplets number $\mathcal{LD}(u)$ in $\mathrm{Id}(v)$ and $\mathrm{Id}(u)$, respectively. There are three possible scenarios:

- If $path_v = path_u$ and $vertex_v = vertex_u$ return $edge_v$.
- If $path_v = path_u$ and $vertex_v \neq vertex_u$, then determine which of $vertex_v$ and $vertex_u$ are first on the heavy path and return 0 if the former and 1 if the latter.
- If $path_v \neq path_u$ then return the edge corresponding to the traversal from $path_u$ to $path_v$ in $RT(u)$.

Lemma 29 is a correction to Lemma 2.4 in the original proof in [9]. Using similar definitions, the Lemma argues that every triplet requires at most $\log b + 2$ bits. The term $\mathrm{lsize}_s(\overline{v})$ represents the size of the subtree rooted in a $light_s$ vertex at any level including the first, and therefore $1 \leq \mathrm{size}(\overline{v}) \leq n/b$. From that we get that $\log_2 \frac{n}{\mathrm{lsize}_s(\overline{v})} + 2 > \log_2 b + 2$. In fact the claim $\log_2 \frac{n}{\mathrm{lsize}_s(\overline{v})} + 2 \leq \log_2 b + 2$ in [9] holds in the opposite direction.

Implications From this upper bound and the recent lower bound for *NCA* [51] it follows that any labeling scheme for *NCA* is of provably larger size than the size required for *Routing*. The question of whether the best possible lower order term is $O(\log n/\log\log n)$ or $O(\log\log n)$ remains open. The current lower bound for designer port *Routing* in trees stands at $\log n + O(\log\log n)$. This follows from its relationship to *Ancestry* labeling schemes reported earlier. Clearly, it is of great interest to prove that a larger label size is needed for *Routing* than needed for *Ancestry*/*Siblings*/*Connectivity* labeling schemes.

10.3 Fixed Port Model

We now consider the fixed port model, i.e the model where the port numbers are chosen by an adversary. We provide a proof of the following result of [9] that was omitted in their paper.

Theorem 30. *There exist a fixed port model* Routing *labeling scheme for* $Trees(n)$*, denoted* $\langle e, d \rangle$*, with labels of size at most* $O(\log^2 n / \log\log n)$*.*

Proof. The encoder operates on the tree T similarly to the one for the designer port problem. A vertex $v \in T$ with label $(\mathrm{Id}(v), RT(v))$ is transformed in the following manner: We store additional $\log n$ bits to denote the port number assigned by the adversary to the edge in each triplet in $\mathrm{Id}(v)$. Recall that the *Routing* table $RT(v)$ contains at most b port numbers. We store the edge numbering assigned by the adversary for each entry in the table. The new $\mathrm{Id}(v)$ requires additional $\log n$ bits for each of the $\log_b n$ parts, and the new $RT(v)$ requires $b \log n$ additional bits. Setting $b = \lceil \sqrt{\log n} \rceil$ as before, the total new label length is bounded by $O(\log^2 n / \log\log n)$, as required. □

The labeling scheme in Section 10.3 is asymptotically optimal in light of the lower bound in [136]. An alternative proof for the existence of such a labeling scheme is given in [26].

Chapter 11

The Future

We conclude the thesis as one should, looking to the future. First, in Section 11.1 we present an unpublished idea where we discuss labeling schemes as the beginning of a study of information dissemination. We end the thesis with a comprehensive list of open questions for labeling schemes in Section 11.2.

11.1 Cluster labeling

With ever increasing size of graphs in real-world applications (e.g. web graphs, social networks) many distributed graph systems attempt to store, preprocess and analyze large-scale graphs [142]. These systems disseminate the structure of the graph among multiple clusters such that certain queries can be performed quickly while minimizing both communication and storage. The method researched in thesis thus far, that of labeling scheme, can be seen as an extreme case of such dissemination.

We propose a novel method called cluster labeling. This method extends the concept of labeling scheme to study the overhead incurred by the distribution of graph structure across multiple machines. The encoder of a cluster labeling receives a graph from a particular graph family, a desired number of clusters $\mathcal{C}$, and a two variable query to be supported and returns a strategy to assign $\mathcal{C}$ bit strings to each graph in the family such that any query can be determined by at most two of them.

Labeling schemes for a combination of important graph families and several key functions are well studied. Compared to (centralized) data structures, this body of work can be interpreted as the study of the overhead incurred by labeling on the space required for a total dissemination of the data structure, e.g.:

- Distance queries for planar graphs can be performed in a centralized data structure using linear space [143]. In contrast, labeling schemes for the same function require labels of size at least $\Omega(n^{1/3})$ each [22], summing up to a total of $\Omega(n^{4/3})$ bits.

- It was recently shown that the current best centralised data structures to support constant time distance queries for general graphs and the current best known labeling schemes for this functions bare the same total size [144, 145].

We view centralized data structures and labeling schemes as two extremes in a *spectrum* of information dissemination. Between the two, we may instead divide an input graph into a number of smaller graphs which we call *clusters*. A cluster labeling assigns labels to each of the clusters that contains sufficient information to answer a requested query for vertices associated to

this cluster. When two vertices belong to two different clusters, the union of the information in both cluster labels is sufficient to answer the query. This design is done to maintain three important design factors implied by labeling schemes: locality of a query, function specific data structure and minimal communication. We argue that this concept may assist both in the design of large-scale distributed graph algorithms, and in the theoretical understanding of the limits these design must adhere.

We first define this notion formally in Section 11.1.1, and then present two motivating examples in Section 11.1.2. The examples are (i) upper and lower bounds for *Adjacency* cluster labeling for general graphs, and (ii) a conversion from any labeling scheme to a cluster labeling in trees. Note that the presented bounds are asymptotically optimal for the cases $\mathcal{C} = 1$ and $\mathcal{C} = n$.

11.1.1 Definition

As defined in Section 1.8.3, a vertex v in the n-vertex graph has a unique identifier of $\log n$ bits $\mathrm{Id}(v)$ an encoder may assign. Given a graph G, a cluster is a subgraph thereof, and the set of clusters $C_1 \dots C_{\mathcal{C}}$ is a *clustering* of graph G if $C_1 \cup \dots \cup C_{\mathcal{C}} = G$. We define *Cluster labeling schemes*, abbreviated cluster labeling, as follows:

Definition 19. *Let $\mathcal{C}$ be an integer, S be a nonempty set, $\mathcal{G}$ be a family of graphs and let, for each $G \in \mathcal{G}$, $F_G : V(G) \times V(G) \longrightarrow S$ be a function. A cluster labeling for a graph G is a pair of encoder and decoder (E, D) such that:*

- *The* encoder *E receives a graph $G \in \mathcal{G}$ and computes a clustering $C_1 \dots C_{\mathcal{C}}$ of G, along with their corresponding* cluster labels*, the bit-strings $\mathcal{L}(C_1), \dots, \mathcal{L}(C_{\mathcal{C}}) \in \{0,1\}^+$.*
- *Let C_i and C_j be clusters in graph G and vertices $u \in C_i$ and $v \in C_j$. The decoder receives a quadruple $(\mathrm{Id}(u), \mathrm{Id}(v), \mathcal{L}(C_i), \mathcal{L}(C_j))$ and returns $F_G(u, v)$.*

The quality of a cluster labeling is measured by the maximum size of its cluster labels. The goal is therefore to minimize $\max(|\mathcal{L}(C_1)|, \dots, |\mathcal{L}(C_{\mathcal{C}})|)$. We refer to this as the *size* of the cluster labeling and give it as a function of $n = |V(G)|$ and $\mathcal{C}$. The cases $\mathcal{C} = 1$ or $\mathcal{C} = n$ correspond to succinct data structures [146] and labeling schemes [2], respectively. Finally, note that the definition allows a vertex to belong to more than one cluster.

11.1.2 Motivating examples

In the following we derive simple upper and lower bounds for cluster labeling for adjacency in general graphs, as well as a general conversion scheme from (regular) labeling schemes on trees. In both examples we will use so-called *local labels* explained as follows. The encoder assigns the nodes in every cluster vertex identifiers from a range of consecutive integeres, and we set $\log |C_i|$ local labels of the vertices in the cluster C_i from the set $\{1, \dots, |C_i|\}$ according to their vertex identifier ordering. The smallest unique identifier

in the cluster is saved as a $\log n$ bit offset to allow for a conversion between the two identifier types[1].

Adjacency in general graphs.

Theorem 31. *There is a cluster labeling for adjacency for the family of n vertex graphs $\mathcal{G}_n$ of size $\frac{n^2}{2\mathcal{C}} + O(\frac{n}{\mathcal{C}}\log\mathcal{C}) + \log n$.*

Proof. We assign the vertices of a graph $G = (V, E)$ labels of size $n/2 + O(1)$ using the scheme in [45], denoted *original*. A vertex v now has a tuple of two labels, its unique identifier $\mathrm{Id}(v)$ and an *original* label $\mathcal{L}(v)$. We now split the graph into $\mathcal{C}$ parts of $n/\mathcal{C}$ vertices each, with consecutive unique identifiers. The cluster label $\mathcal{L}(C_i)$ of a cluster C_i is simply a concatenation of the original labels of the vertices it consists of, ordered by their vertex identifier. The length of each such cluster label is $\frac{n}{\mathcal{C}}(\frac{n}{2} + \log\mathcal{C} + O(1)) + \log n$ bits. Given a quadruple $(\mathrm{Id}(u), \mathrm{Id}(v), \mathcal{L}(C_i), \mathcal{L}(C_j))$ the decoder computes *Adjacency* in the following manner. First, it uses $\mathrm{Id}(u)$ and $\mathrm{Id}(v)$ to extract the *original* labels of u and v contained in $\mathcal{L}(C_i)$ and $\mathcal{L}(C_j)$ respectively. The decoder then computes the *original* decoder on the extracted labels $\mathcal{L}(u)$ and $\mathcal{L}(v)$ and returns its result. □

We proceed to showing a lower bound for this case.

Theorem 32. *For every clustering of $\mathcal{C}$ clusters and for all ϵ, any* Adjacency *cluster labeling for $\mathcal{G}_n$ must have label size at least $\frac{n^2}{(2+\epsilon)\mathcal{C}}$ bits.*

Proof. Consider any graph G with $n/\mathcal{C}$ vertices. Construct $\mathcal{C}$ identical copies of G to obtain a graph of size n. The union of all cluster labels and vertex identifiers forms a description D of G. Suppose the label size of each cluster is strictly less than $\frac{n^2}{(2+\epsilon)\mathcal{C}}$ then:

$$|D| < \mathcal{C}\frac{n^2}{(2+\epsilon)\mathcal{C}} = \frac{1}{2+\epsilon}n^2.$$

For all sufficiently large n, $|D| < \frac{n(n-1)}{2}$ bits. Since the labeling scheme works for all graphs, we can obtain, for any string s of length $n(n-1)/2$, a strictly shorter description of s by constructing the graph whose incidence matrix it represents. This contradicts the fact that there is at least one string of length $n(n-1)/2$ that is incompressible. □

Cluster labeling for trees. As seen in the thesis, there are numerous functions well understood in the context of labeling schemes for trees. Korman [147] proposed a method[2] to convert each such labeling scheme to a fully dynamic one. In this spirit, we propose a method to convert each labeling scheme for trees to a corresponding cluster labeling. Note that in [147], some natural restrictions on the supported functions are assumed. Since all functions discussed in the thesis adhere to these restrictions, we use the term *function* to mean a function among those described in Section 1.2. The following assumption suffices for our purposes. Let $u, w \in C_i$, $v, x \in C_j$ be

[1] While this seems to be a good practice, we omit it from the basic definition since it might be sub-optimal.

[2] For further notes on this topic, see Chapter 9.

nodes in a tree T in two distinct clusters C_i and C_j, and w and x are the boundary nodes closest to the root of C_i and C_j respectively. The result of the function $F_G(u, v)$ can be deduced from $F_G(u, w)$, $F_G(w, x)$ and $F_G(x, v)$. As a concrete example, it is easy to see that the relation *Distance* in this case is simply $F_G(u, w)+F_G(w, x)+F_G(x, v)$. Finally, we stress that information on the boundary nodes appear in all the clusters containing them. The following proof hinges on the cluster partition in Section 2.3.4, and uses concepts and terminology that can be found in this section.

Theorem 33. *Let $\mathcal{L}$ be a labeling scheme of size $f(n)$ for the family of rooted trees. Then there exists a matching $O(n/C)+O(f(C))+\log n$ cluster labeling.*

Proof. We assign every vertex in the tree its identifier $\mathrm{Id}(v)$ by a $\mathrm{dfs_i}$ traversal on the tree, as defined in Section 2.2.1. For every $1 \leq \mathcal{C} \leq n$, we partition the tree to at most $\mathcal{C}$ trees of size at most $O(n/\mathcal{C})$ rooted in one boundary vertex, with at most one more additional boundary vertex. Each cluster contains vertices with consecutive Id numbers. We then form the vertex macro tree M whose $|2C|$ vertices correspond to the boundary vertices and its edges correspond to pairs (u, v) of boundary vertices belonging to the same cluster.

The encoder assigns the label $\mathcal{L}(C_i)$ for a cluster C_i with boundary nodes $u(C_i)$ and $v(C_i)$ as the concatenation of the following sub-labels: (i) an $f(\mathcal{C})$ bits label of the cluster C_i in the macro tree M, as assigned by the labeling scheme $\mathcal{L}$, (ii) an $O(n/\mathcal{C})$ (centralized) bits data structure (e.g. [148]) for the tree C_i with $O(n/\mathcal{C})$ vertices supporting the function f, and (iii) a concatenation of the local vertex identifiers of the vertices in C_i.

Given a quadruple $(\mathrm{Id}(u), \mathrm{Id}(v), \mathcal{L}(C_i), \mathcal{L}(C_j))$ the decoder deduces the function f in the following manner. First, it uses $\mathrm{Id}(u)$ and $\mathrm{Id}(v)$ and extracts the local labels of u and v contained in sub-labels (ii) of $\mathcal{L}(C_i)$ and $\mathcal{L}(C_j)$ respectively. If $\mathcal{L}(C_i) = \mathcal{L}(C_j)$, the decoder uses the local labels to identify the vertices in the centralized data structure and returns the function f. If $\mathcal{L}(C_i) \neq \mathcal{L}(C_j)$ the decoder first computes the result of $\mathcal{L}$ on part (i) of both cluster labels. It then computes the function F_G for vertex v and the root of C_i, and computes the desired function F_G for u and the root of C_j. The decoder may now compute $F_G(u, v)$ using the results of these three computations. □

11.2 Open Questions

In this section we enumerate the open problems we leave behind. To rank the hardness and importance of these problems we will use a ranking[3] of conferences such results have a chance to appear in, which we estimate based on the venues similar results appeared in. The ordering is STOC/FOCS, SODA, ICALP, ESA[4]. If the results are in flavour of distributed computing conferences we also suggest PODC. If the results are in flavour of graph theory, we suggest Journal of Combinatorial Theory Series B.

11.2.1 Open questions on trees

1. Labeling schemes for designer port *Routing* on trees remain the single biggest open question for labeling schemes on trees. We have seen in Chapter 10 a $\log n + o(\log n)$[5] labeling scheme and a $\log n + \log\log n$ lower bound. A proof that *NCA* is separated from both *Routing* and *Ancestry* was SODA worthy [51]. We'd argue that separating *Routing* from *Ancestry* stands at least on the same pedestal. A completely tight result with non-trivial technique is a STOC/FOCS candidate.

2. Given the recent "air-tight" results by Alon [98] and Alon and Nenadov [97], one might ask what is the real size of the induced universal graph for trees? Is it bounded by $200n$ as implied in [45] or closer to $2n$? A simple argument shows that it is at least $1.5n$ using a path graph and a star graph. Proving an air tight result is clearly a STOC/FOCS worthy. Simplifying [45] and additional improvement may well be a SODA paper.

3. We have shown in Theorem 26 a *tight* $\log n + 2\log\log n$ multifunctional labeling scheme for *Siblings* and *Connectivity*, and in Theorem 19 an upper bound $\log n + 2\log\log n$ labeling scheme for *Ancestry*. We also know now a $\log n + O(1)$ labeling scheme for adjacency, and that asymptotically all these results are tight. It would be interesting to show what would be the size of a labeling scheme capable of doing all of the aforementioned functions. Our current best result is $\log n + 5\log\log n$. We suspect $\log n + 2\log\log n$ suffice, and if non-trivial in technique this is ESA worthy.

4. With respect to the previously mentioned *Ancestry* labels, we are nearly convinced that it requires $\log n + 2\log\log n$ bits but were never able to prove it. If this indeed is the case, such lower bound result is ESA worthy. If the label can be decreased further, it might very well be ICALP worthy.

5. The exact label size for *Label-NCA* function is somewhere between 1.008 and 2.75. We will not be surprised if it is 2. Similar rewards to the previous clause are expected.

[3] This is also under the implicit assumption that the techniques used to prove these questions will be non-trivial, a sad truth of our profession.

[4] This ranking is based on the one by the Computing Research and Education Association of Australasia, available online in http://lipn.univ-paris13.fr/ bennani/CSRank.html.

[5] Precisely: $\log n + O(\frac{\log n \log\log\log n}{\log\log n})$.

6. Our re-labelable dynamic labeling schemes analysis (Chapter 9) showed that Korman's methods for re-labeling [125] are very centralized in practice. Decentralizing these methods seems PODC worthy.

Figure 11.1 describes the current state of affairs in this study.

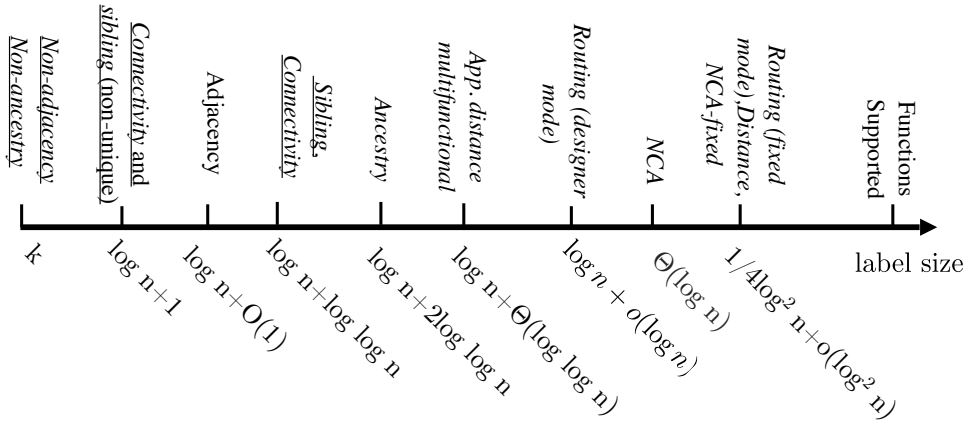


FIGURE 11.1: Illustration of the current study on labeling schemes for various functions on trees. Functions underlined have "air-tight" label sizes. The remaining are tight up to the additive factor.

11.2.2 Open questions on other graph families

1. Proving the implicit graph conjecture is the most important problem discussed in the thesis. In case it is false, it might be a nearly trivial lower bound on some geometric graph class as reported in Section 6.3. In which case, we would rank it is as STOC/FOCS worthy. If the conjecture is in fact correct, its proof should discover something remarkable about the structure of hereditary graph properties. As numerous Journal of Combinatorial Theory Series B papers were written on this topic [112, 114, 149–151], it is at least that worthy.

2. As we have seen in Section 6.5, the implicit representation conjecture is indeed true up to speed of $2^{\sqrt{n}}$. Can this bound be improved based on similar techniques? A strong progress forward in this direction may also be Journal of Combinatorial Theory Series B worthy.

3. *Distance* labeling for unweighted planar graphs is not very well known. The current lower and upper bounds are $\sqrt[3]{n}$ and $\sqrt{n}\log n$ [22] respectively. In the same paper weighted planar graphs were shown to have nearly matching $\sqrt{n}$ and $\sqrt{n}\log n$, and Abboud and Dahlgaard [152] shwed a lower bound of $\sqrt{n}\log n$, sealing the weighted question and making the gap in the unweighted version even more odd. Closing this gap is STOC/FOCS worthy.

4. Still in planar graphs, there is still a large gap for *Adjacency* between $\log n + 1.5$ and $2\log n + O(\log\log n)$ [6]. A reduction of the $O(\log\log n)$ factor is ICALP worthy, a $\log n + o(\log n)$ is SODA worthy, and a tight result is STOC/FOCS worthy. We mention that the current best bound stems from a decomposition of planar graph to two outer-planar graphs,

which in turn have treewidth 1. Even a smarter union of two such labels is not yet available in the literature. We conjecture that the right label size here is $1.5 \log n$.

5. As seen in Chapter 5 we have only just begun the study of *Adjacency* labeling schemes for sparse and power-law graphs. We suspect that the whole multiplicative $O(\sqrt{\log n})$ is redundant. A tight result for sparse graphs is SODA worthy.

6. Distance for sparse graphs seems to be notoriously difficult. We were the first to spot that Alstrup et al. [86] preliminary results effectively produce a sub-linear distance labeling scheme for this family[6]. The difference here is between a follow up result [87] of $O(\frac{n}{R} \log^2 R)$ where $R = \frac{\log n}{\log \frac{m+n}{n}}$ and a lower bound of $\sqrt{n}$ from *Adjacency*. Closing this gap tight is at least SODA worthy.

7. Unlike on trees and planar graphs, we have not yet been able to separate between *Adjacency* and *Distance* for neither bounded degree, sparse and general graphs. This fundamental question is STOC/FOCS worthy for non-trivial technique.

8. The recent result by Alon [98] is carried out using the probabilistic method. This means that while induced universal graphs of this size were proven to exist, no such explicit construction was given. Labeling schemes are not the only method to do so, but showing a matching bound using an *Adjacency* labeling scheme seems both likely and important. This would be a clear STOC/FOCS paper.

[6] They indeed thanked me for "useful discussions" in the text.

Bibliography

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

[2] S. Kannan, M. Naor, S. Rudich, Implicit representation of graphs, in: SIAM Journal On Discrete Mathematics, 1992, pp. 334–343.

[3] J. H. Muller, Local structure in graph classes, Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, USA, order No: GAX88-11342 (1988).

[4] C. Gavoille, D. Peleg, Compact and localized distributed data structures, Distrib. Comput. 16 (2-3) (2003) 111–120.

[5] R. Rado, Universal graphs and universal functions, Acta Arithmetica 9 (4) (1964) 331–340.

[6] C. Gavoille, A. Labourel, Shorter implicit representation for planar graphs and bounded treewidth graphs, in: Algorithms–ESA 2007, Springer, 2007, pp. 582–593.

[7] D. Adjiashvili, N. Rotbart, Labeling schemes for bounded degree graphs, in: International Colloquium on Automata, Languages, and Programming, Springer, 2014, pp. 375–386.

[8] S. Alstrup, T. Rauhe, Small induced-universal graphs and compact implicit graph representations, in: Proceedings of the 43rd Symposium on Foundations of Computer Science, FOCS '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 53–62.

[9] M. Thorup, U. Zwick, Compact routing schemes, in: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures, SPAA '01, ACM, New York, NY, USA, 2001, pp. 1–10.

[10] C. Petersen, N. Rotbart, J. G. Simonsen, C. Wulff-Nilsen, Near Optimal Adjacency Labeling Schemes for Power-Law Graphs, in: 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), Vol. 55, 2016, pp. 127:1–127:15.

[11] C. Petersen, N. Rotbart, J. Grue Simonsen, C. Wulff-Nilsen, Brief announcement: Labeling schemes for power-law graphs, in: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, ACM, 2016, pp. 39–41.

[12] S. Dahlgaard, M. B. T. Knudsen, N. Rotbart, A simple and optimal ancestry labeling scheme for trees, in: International Colloquium on Automata, Languages, and Programming, Springer Berlin Heidelberg, 2015, pp. 564–574.

[13] S. Dahlgaard, M. B. T. Knudsen, N. Rotbart, Dynamic and multi-functional labeling schemes, in: International Symposium on Algorithms and Computation, Springer, 2014, pp. 141–153.

[14] S. Dahlgaard, M. B. T. Knudsen, N. Rotbart, Brief announcement: On dynamic and multi-functional labeling schemes, Proceedings of the 2014 International symposium on Distributed Computing (DISC) (2014) 544.

[15] N. Rotbart, M. V. S. Salles, I. Zotos, An evaluation of dynamic labeling schemes for tree networks, in: International Symposium on Experimental Algorithms, Springer International Publishing, 2014, pp. 199–210.

[16] S. Abiteboul, H. Kaplan, T. Milo, Compact labeling schemes for ancestor queries, in: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, SODA '01, 2001, pp. 547–556.

[17] E. Cohen, H. Kaplan, T. Milo, Labeling dynamic xml trees, in: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '02, ACM, New York, NY, USA, 2002, pp. 271–281.

[18] J. Lu, Xml labeling scheme, in: An Introduction to XML Query Processing and Keyword Search, Springer, 2013, pp. 9–32.

[19] T. Härder, M. Haustein, C. Mathis, M. Wagner, Node labeling schemes for dynamic xml documents reconsidered, Data & Knowledge Engineering 60 (1) (2007) 126–149.

[20] X. Wu, M. L. Lee, W. Hsu, A prime number labeling scheme for dynamic ordered xml trees, in: Data Engineering, 2004. Proceedings. 20th International Conference on, IEEE, 2004, pp. 66–78.

[21] E. Cohen, H. Kaplan, T. Milo, Labeling dynamic xml trees, SIAM Journal on Computing 39 (5) (2010) 2048–2074.

[22] C. Gavoille, D. Peleg, S. Pérennes, R. Raz, Distance labeling in graphs, in: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, SODA '01, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001, pp. 210–219.

[23] I. Abraham, D. Delling, A. V. Goldberg, R. F. Werneck, A hub-based labeling algorithm for shortest paths on road networks.

[24] I. Abraham, D. Delling, A. V. Goldberg, R. F. Werneck, Hierarchical hub labelings for shortest paths, in: Algorithms–ESA 2012, Springer, 2012, pp. 24–35.

[25] D. Peleg, Informative labeling schemes for graphs, Theor. Comput. Sci. 340 (3) (2005) 577–593.

[26] P. Fraigniaud, C. Gavoille, Routing in trees, in: IN 28 TH INTERNATIONAL COLLOQUIUM ON AUTOMATA, LANGUAGES AND PROGRAMMING (ICALP, Springer, 2001, pp. 757–772.

[27] C. Gavoille, S. Pérennès, Memory requirement for routing in distributed networks, in: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, ACM, 1996, pp. 125–133.

[28] M. Dom, in: D. Wagner, R. Wattenhofer (Eds.), Algorithms for Sensor and Ad Hoc Networks, Vol. 4621 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 187–202.

[29] A. Korman, S. Kutten, Labeling schemes with queries, in: Proceedings of the 14th international conference on Structural information and communication complexity, SIROCCO'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 109–123.

[30] D. Krioukov, K. Fall, X. Yang, Compact routing on internet-like graphs, in: INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, Vol. 1, IEEE, 2004.

[31] I. Abraham, C. Gavoille, A. V. Goldberg, D. Malkhi, Routing in networks with low doubling dimension, in: Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on, IEEE, 2006, pp. 75–75.

[32] I. Abraham, D. Malkhi, Name independent routing for growth bounded networks, in: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures, ACM, 2005, pp. 49–55.

[33] M.-Y. Kao, X.-Y. Li, W. Wang, Average case analysis for tree labelling schemes, Theoretical Computer Science 378 (3) (2007) 271 – 291, <ce:title>Algorithms and Computation</ce:title>.

[34] A. Korman, D. Peleg, Labeling schemes for weighted dynamic trees, Information and Computation 205 (12) (2007) 1721–1740.

[35] P. Fraigniaud, A. Korman, On randomized representations of graphs using short labels, in: Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures, ACM, 2009, pp. 131–137.

[36] B. Courcelle, C. Gavoille, M. Kanté, A. Twigg, Forbidden-set labeling on graphs, in: 2nd Workshop on Locality Preserving Distributed Computing Methods (LOCALITY)", Co-located with PODC, 2007.

[37] B. Courcelle, A. Twigg, Compact forbidden-set routing, in: Annual Symposium on Theoretical Aspects of Computer Science, Springer, 2007, pp. 37–48.

[38] I. Abraham, S. Chechik, C. Gavoille, D. Peleg, Forbidden-set distance labels for graphs of bounded doubling dimension, in: Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing, ACM, 2010, pp. 192–200.

[39] B. Courcelle, A. Twigg, Constrained-path labellings on graphs of bounded clique-width, Theory of Computing Systems 47 (2) (2010) 531–567.

[40] S. Chechik, M. Langberg, D. Peleg, L. Roditty, f-sensitivity distance oracles and routing schemes, in: European Symposium on Algorithms, Springer, 2010, pp. 84–96.

[41] C. Gavoille, A. Labourel, Distributed relationship schemes for trees, Algorithms and Computation (2007) 728–738.

[42] S. Alstrup, P. Bille, T. Rauhe, Labeling schemes for small distances in trees, SIAM J. Discret. Math. 19 (2) (2005) 448–462.

[43] B. H. Esben, Graph labeling schemes, Master's thesis, University of Copenhagen, Universitetsparken 5, Copenhagen (2013).

[44] S. Alstrup, C. Gavoille, H. Kaplan, T. Rauhe, Nearest common ancestors: A survey and a new distributed algorithm, in: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures, ACM PRESS, 2002, pp. 258–264.

[45] S. Alstrup, S. Dahlgaard, M. B. T. Knudsen, Optimal induced universal graphs and adjacency labeling for trees, in: Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on, IEEE, 2015, pp. 1311–1326.

[46] N. Bonichon, C. Gavoille, A. Labourel, Short labels by traversal and jumping, in: Proceedings of the 13th international conference on Structural Information and Communication Complexity, SIROCCO'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 143–156.

[47] P. Fraigniaud, A. Korman, Compact ancestry labeling schemes for xml trees, in: In Proc. 21st ACM-SIAM Symp. on Discrete Algorithms (SODA, 2010.

[48] O. Freedman, P. Gawrychowski, P. K. Nicholson, O. Weimann, Optimal distance labeling schemes for trees, arXiv preprint arXiv:1608.00212.

[49] C. Gavoille, M. Katz, N. A. Katz, C. Paul, D. Peleg, Approximate distance labeling schemes (extended abstract) (2000).

[50] M. Katz, N. A. Katz, A. Korman, D. Peleg, Labeling schemes for flow and connectivity, SIAM Journal on Computing 34 (1) (2004) 23–40.

[51] S. Alstrup, E. Bistrup Halvorsen, K. Green Larsen, Near-optimal labeling schemes for nearest common ancestors, in: Proceedings of the thirty first annual ACM-SIAM symposium on Discrete algorithms, SODA '14, 2014.

[52] M. Lewenstein, J. I. Munro, V. Raman, Succinct data structures for representing equivalence classes, in: International Symposium on Algorithms and Computation, Springer, 2013, pp. 502–512.

[53] T. M. Cover, J. A. Thomas, Elements of information theory, John Wiley & Sons, 2012.

[54] D. Harel, R. E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM Journal on Computing 13 (2) (1984) 338–355.

[55] F. R. Chung, Separator theorems and their applications, Forschungsinst. für Diskrete Mathematik, 1989.

[56] C. Jordan, Sur les assemblages de lignes., Journal für die reine und angewandte Mathematik 70 (1869) 185–190.

[57] A. Korman, D. Peleg, Compact separator decompositions in dynamic trees and applications to labeling schemes, in: Distributed Computing, Springer, 2007, pp. 313–327.

[58] P. Fraigniaud, A. Korman, An optimal ancestry labeling scheme with applications to xml trees and universal posets, Journal of the ACM (JACM) 63 (1) (2016) 6.

[59] S. Alstrup, P. Lauridsen, P. Sommerlund, M. Thorup, Finding cores of limited length, Lecture notes in computer science (1997) 45–54.

[60] C. Nash-Williams, Edge-disjoint spanning trees of finite graphs, Journal of the London Mathematical Society 1 (1) (1961) 445–450.

[61] B. Chen, M. Matsumoto, J. Wang, Z. Zhang, J. Zhang, A short proof of nash-williams' theorem for the arboricity of a graph, Graphs and Combinatorics 10 (1) (1994) 27–28.

[62] S. Bhatt, F. R. Graham Chung, T. Leighton, A. Rosenberg, Universal Graphs for Bounded-Degree Trees and Planar Graphs, SIAM Journal on Discrete Mathematics 2 (2) (1989) 145–155.

[63] D. Gonçalves, Edge partition of planar sraphs into two outerplanar graphs, in: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, ACM, 2005, pp. 504–512.

[64] S. Alstrup, H. Kaplan, M. Thorup, U. Zwick, Adjacency labeling schemes and induced-universal graphs, in: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, ACM, 2015, pp. 625–634.

[65] D. Knuth, Combinatorial algorithms: Part 1, the art of computer programming, vol. 4a (2011).

[66] O. Weimann, D. Peleg, A note on exact distance labeling, Inf. Process. Lett. 111 (14) (2011) 671–673.

[67] G. Siganos, M. Faloutsos, P. Faloutsos, C. Faloutsos, Power laws and the as-level internet topology, IEEE/ACM Transactions on Networking (TON) 11 (4) (2003) 514–524.

[68] A. Akella, S. Chawla, A. Kannan, S. Seshan, Scaling properties of the internet graph, in: Proceedings of the twenty-second annual symposium on Principles of distributed computing, ACM, 2003, pp. 337–346.

[69] A. Clauset, C. R. Shalizi, M. E. Newman, Power-law distributions in empirical data, SIAM review 51 (4) (2009) 661–703.

[70] M. Mitzenmacher, A brief history of generative models for power law and lognormal distributions, Internet mathematics 1 (2) (2004) 226–251.

[71] D. Achlioptas, A. Clauset, D. Kempe, C. Moore, On the bias of traceroute sampling, in: STOC, ACM, Vol. 1581139608, 2005, p. 0005.

[72] P. Brach, M. Cygan, J. Łącki, P. Sankowski, Algorithmic complexity of power law networks, in: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2016, pp. 1306–1325.

[73] F. Chung, L. Lu, The average distance in a random graph with given expected degrees, Internet Mathematics 1 (1) (2004) 91–113.

[74] Y. Eom, S. Fortunato, M. Perc, Characterizing and modeling citation dynamics, PLoS ONE 6 (9) (2011) e24926.

[75] F. R. Chung, L. Lu, Complex graphs and networks, Vol. 107, American mathematical society Providence, 2006.

[76] J. Moon, On minimal n-universal graphs, in: Proceedings of the Glasgow Mathematical Association, Vol. 7, Cambridge Univ Press, 1965, pp. 32–33.

[77] J. P. Spinrad, Efficient graph representations, American mathematical society, 2003.

[78] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, science 286 (5439) (1999) 509–512.

[79] B. Bollobás, O. Riordan, J. Spencer, G. Tusnády, et al., The degree sequence of a scale-free random graph process, Random Structures & Algorithms 18 (3) (2001) 279–290.

[80] G. Goel, J. Gustedt, Bounded arboricity to determine the local structure of sparse graphs, in: International Workshop on Graph-Theoretic Concepts in Computer Science, Springer, 2006, pp. 159–167.

[81] Ł. Kowalik, Approximation scheme for lowest outdegree orientation and graph density measures, in: International Symposium on Algorithms and Computation, Springer, 2006, pp. 557–566.

[82] S. R. Arikati, A. Maheshwari, C. D. Zaroliagis, Efficient computation of implicit representations of sparse graphs, Discrete Applied Mathematics 78 (1) (1997) 1–16.

[83] B. M. Waxman, Routing of multipoint connections, IEEE journal on selected areas in communications 6 (9) (1988) 1617–1622.

[84] K. L. Calvert, M. B. Doar, E. W. Zegura, Modeling internet topology, IEEE Communications magazine 35 (6) (1997) 160–163.

[85] T. H. Cormen, C. Stein, R. L. Rivest, C. E. Leiserson, Introduction to Algorithms, 2nd Edition, McGraw-Hill Higher Education, 2001.

[86] S. Alstrup, S. Dahlgaard, M. B. T. Knudsen, E. Porat, Sublinear Distance Labeling, in: 24th Annual European Symposium on Algorithms (ESA 2016), Vol. 57, 2016, pp. 5:1–5:15.

[87] P. Gawrychowski, A. Kosowski, P. Uznanski, Sublinear-space distance labeling using hubs, in: International Symposium on Distributed Computing, Springer, 2016.

[88] S. L. Hakimi, On realizability of a set of integers as degrees of the vertices of a linear graph. i, Journal of the Society for Industrial & Applied Mathematics 10 (3) (1962) 496–506.

[89] R. Albert, H. Jeong, A.-L. Barabási, Diameter of the world-wide web, Nature 401 (6749) (1999) 130–131.

[90] J. Leskovec, K. J. Lang, A. Dasgupta, M. W. Mahoney, Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters, Internet Mathematics 6 (1) (2009) 29–123.

[91] M. Newman, Network data, http://www-personal.umich.edu/~mejn/netdata/ (2013).

[92] M. Gol'dberg, E. Livshits, On minimal universal trees, Mathematical Notes of the Academy of Sciences of the USSR 4 (3) (1968) 713–717.

[93] F. C. R. GRAHAM, D. COPPERSMITH, On trees containing all small trees, in: The Theory and applications of graphs: Fourth International Conference, May 6-9, 1980, Western Michigan University, Kalamazoo, Michigan, John Wiley & Sons, 1981, p. 265.

[94] A. Korman, D. Peleg, Y. Rodeh, Constructing labeling schemes through universal matrices, Algorithmica 57 (4) (2010) 641–652.

[95] C. Gavoille, C. Paul, Split decomposition and distance labelling: an optimal scheme for distance hereditary graphs, Electronic Notes in Discrete Mathematics 10 (2001) 117–120.

[96] M. Abrahamsen, S. Alstrup, J. Holm, M. B. T. Knudsen, M. Stöckel, Near-optimal induced universal graphs for bounded degree graphs, arXiv preprint arXiv:1607.04911.

[97] N. Alon, R. Nenadov, Optimal induced universal graphs for bounded-degree graphs, arXiv preprint arXiv:1607.03234.

[98] N. Alon, Asymptotically optimal induced universal graphs.

[99] L. Esperet, A. Labourel, P. Ochem, On induced-universal graphs for the class of bounded-degree graphs, Information Processing Letters 108 (5) (2008) 255–260.

[100] P. Fraigniaud, A. Korman, An optimal ancestry labeling scheme with applications to xml trees and universal posets, Journal of the ACM (JACM) 63 (1) (2016) 6.

[101] B. R. P. Erdős, D.J. Kleitman, Asymptotic enumeration of kn-free graphs, Colloquio Internazionale sulle Teorie Combinatorie 17 (1976) 19–27.

[102] A. Cayley, A theorem on trees, Quart. J. Math. 23 (1889) 376–378.

[103] R. Otter, The number of trees, Annals of Mathematics (1948) 583–599.

[104] J. Kratochvíl, J. Matousek, Intersection graphs of segments, Journal of Combinatorial Theory, Series B 62 (2) (1994) 289–315.

[105] B. N. Clark, C. J. Colbourn, D. S. Johnson, Unit disk graphs, Discrete mathematics 86 (1-3) (1990) 165–177.

[106] E. R. Scheinerman, K. Tucker, Modeling graphs using dot product representations, Computational Statistics 25 (1) (2010) 1–16.

[107] H. E. Warren, Lower bounds for approximation by nonlinear manifolds, Transactions of the American Mathematical Society 133 (1) (1968) 167–178.

[108] D. Fulkerson, O. Gross, Incidence matrices and interval graphs, Pacific journal of mathematics 15 (3) (1965) 835–855.

[109] C. McDiarmid, T. Müller, Integer realizations of disk and segment graphs, Journal of Combinatorial Theory, Series B 103 (1) (2013) 114–143.

[110] M. Chandoo, On the Implicit Graph Conjecture, in: 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016), Vol. 58, 2016, pp. 23:1–23:13.

[111] L. Longpré, Ph.D. thesis.

[112] E. R. Scheinerman, J. Zito, On the size of hereditary classes of graphs, Journal of Combinatorial Theory, Series B 61 (1) (1994) 16–39.

[113] E. R. Scheinerman, Local representations using very short labels, Discrete mathematics 203 (1) (1999) 287–290.

[114] J. Balogh, B. Bollobás, M. Saks, V. T. Sós, The unlabelled speed of a hereditary graph property, Journal of Combinatorial Theory, Series B 99 (1) (2009) 9–19.

[115] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo, T. Rauhe, Compact labeling scheme for ancestor queries, SIAM J. Comput. 35 (6) (2006) 1295–1309.

[116] H. Kaplan, T. Milo, R. Shabo, A comparison of labeling schemes for ancestor queries, in: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2002, pp. 954–963.

[117] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, N. Westbury, Ordpaths: insert-friendly xml node labels, in: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, ACM, 2004, pp. 903–908.

[118] C. Li, T. W. Ling, Qed: a novel quaternary encoding to completely avoid re-labeling in xml updates, in: Proceedings of the 14th ACM international conference on Information and knowledge management, ACM, 2005, pp. 501–508.

[119] J. Lu, T. W. Ling, C.-Y. Chan, T. Chen, From region encoding to extended dewey: on efficient processing of xml twig pattern matching, in: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005, pp. 193–204.

[120] L. Xu, T. W. Ling, H. Wu, Z. Bao, Dde: from dewey to a fully dynamic xml labeling scheme, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, ACM, 2009, pp. 719–730.

[121] E. Cohen, H. Kaplan, T. Milo, Labeling dynamic xml trees, SIAM Journal on Computing 39 (5) (2010) 2048–2074.

[122] M. F. OConnor, M. Roantree, Scooter: A compact and scalable dynamic labeling scheme for xml updates, in: Database and Expert Systems Applications, Springer, 2012, pp. 26–40.

[123] T. A. Ghaleb, S. Mohammed, Novel scheme for labeling xml trees based on bits-masking and logical matching, in: Computer and Information Technology (WCCIT), 2013 World Congress on, IEEE, 2013, pp. 1–5.

[124] S. Alstrup, T. Rauhe, Improved labeling scheme for ancestor queries, in: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2002, pp. 947–953.

[125] A. Korman, D. Peleg, Y. Rodeh, Labeling schemes for dynamic tree networks, Theory of Computing Systems 37 (1) (2004) 49–75.

[126] A. Korman, General compact labeling schemes for dynamic trees, Distributed Computing 20 (3) (2007) 179–193.

[127] A. Korman, Improved compact routing schemes for dynamic trees, in: PODC '08, ACM, 2008, pp. 185–194.

[128] C. Gavoille, A. Labourel, Distributed relationship schemes for trees, in: International Symposium on Algorithms and Computation, Springer Berlin Heidelberg, 2007, pp. 728–738.

[129] A. Korman, Compact routing schemes for dynamic trees in the fixed port model, in: International Conference on Distributed Computing and Networking, Springer, 2009, pp. 218–229.

[130] S. Caminiti, I. Finocchi, R. Petreschi, Engineering tree labeling schemes: A case study on least common ancestors, in: European Symposium on Algorithms, Springer, 2008, pp. 234–245.

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

[132] E. Cohen, E. Halperin, H. Kaplan, U. Zwick, Reachability and distance queries via 2-hop labels, SIAM J. Comp. 32 (5) (2003) 1338–1355.

[133] Y. Afek, B. Awerbuch, S. Plotkin, M. Saks, Local management of a global resource in a communication network, JACM 43 (1) (1996) 1–19.

[134] R. Jain, The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling (1990).

[135] D. Peleg, Proximity-preserving labeling schemes and their applications, in: Graph-Theoretic Concepts in Computer Science, Springer, 1999, pp. 30–41.

[136] P. Fraigniaud, C. Gavoille, A space lower bound for routing in trees, in: STACS 2002, Springer, 2002, pp. 65–75.

[137] D. Krioukov, K. Fall, A. Brady, et al., On compact routing for the internet, ACM SIGCOMM Computer Communication Review 37 (3) (2007) 41–52.

[138] A. Korman, D. Peleg, Dynamic routing schemes for general graphs, Automata, Languages and Programming (2006) 619–630.

[139] C. Gavoille, M. Gengler, Space-efficiency for routing schemes of stretch factor three, Journal of Parallel and Distributed Computing 61 (5) (2001) 679–687.

[140] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, M. Thorup, Compact name-independent routing with minimum stretch, in: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, ACM, 2004, pp. 20–24.

[141] C. Gavoille, Routing in distributed networks: Overview and open problems, ACM SIGACT News 32 (1) (2001) 36–52.

[142] Y. Lu, J. Cheng, D. Yan, H. Wu, Large-scale distributed graph computing systems: An experimental evaluation, Proceedings of the VLDB Endowment 8 (3).

[143] S. Mozes, C. Sommer, Exact distance oracles for planar graphs, in: Proceedings of the 23rd SODA, SIAM, 2012, pp. 209–222.

[144] P. Ferragina, I. Nitto, R. Venturini, On compact representations of all-pairs-shortest-path-distance matrices, Theoretical Computer Science 411 (34) (2010) 3293–3300.

[145] S. Alstrup, C. Gavoille, E. B. Halvorsen, H. Petersen, Simpler, faster and shorter labels for distances in graphs, in: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2016, pp. 338–350.

[146] D. K. Blandford, G. E. Blelloch, I. A. Kash, Compact representations of separable graphs, in: Proceedings of the 14th SODA, 2003, pp. 679–688.

[147] A. Korman, General compact labeling schemes for dynamic trees, Distributed Computing (2005) 457–471.

[148] G. J. Jacobson, Succinct static data structures.

[149] N. Alon, J. Balogh, B. Bollobás, R. Morris, The structure of almost all graphs in a hereditary property, Journal of Combinatorial Theory, Series B 101 (2) (2011) 85–110.

[150] J. Balogh, B. Bollobás, D. Weinreich, The speed of hereditary properties of graphs, Journal of Combinatorial Theory, Series B 79 (2) (2000) 131–156.

[151] J. Balogh, B. Bollobás, D. Weinreich, A jump to the bell number for hereditary graph properties, Journal of Combinatorial Theory, Series B 95 (1) (2005) 29–48.

[152] A. Abboud, S. Dahlgaard, Popular conjectures as a barrier for dynamic planar graph algorithms, in: Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on, IEEE, 2016.

[153] A. V. Aho, J. E. Hopcroft, J. D. Ullman, On finding lowest common ancestors in trees, in: Proceedings of the fifth annual ACM symposium on Theory of computing, STOC '73, ACM, New York, NY, USA, 1973, pp. 253–265.

[154] D. Peleg, Informative labeling schemes for graphs, in: Mathematical Foundations of Computer Science 2000, Springer, 2000, pp. 579–588.

[155] L. Blin, S. Dolev, M. G. Potop-Butucaru, S. Rovedakis, Fast self-stabilizing minimum spanning tree construction, in: Distributed Computing, Springer, 2010, pp. 480–494.

[156] S. Caminiti, I. Finocchi, R. Petreschi, Informative labeling schemes for the least common ancestor problem., in: ICTCS, 2009, pp. 66–70.

[157] H. N. Gabow, J. L. Bentley, R. E. Tarjan, Scaling and related techniques for geometry problems, in: Proceedings of the sixteenth annual ACM symposium on Theory of computing, STOC '84, ACM, New York, NY, USA, 1984, pp. 135–143.

[158] C. Gavoillea, D. Peleg, S. Pérennesc, R. Razb, Distance labeling in graphs, Journal of Algorithms 53 (2004) 85–112.

[159] D. Peleg, Proximity-preserving labeling schemes, Journal of Graph Theory 33 (3) (2000) 167–176.

Appendix A

Labeling Schemes for Nearest Common Ancestor

In this appendix we discuss labeling schemes for the function *NCA*. Two variants of labeling schemes that support the function appear in the literature, *Label-NCA* and *Id-NCA* explained hereafter. Suppose a tree $T = (V, E)$ has a predefined label assignment of $\log n$ bits from a preset name domain, and denote the product of such an assignment for a vertex $v \in V$ as the *vertex identifier* of v, or simply $\mathrm{Id}(v)$. We can extend all labeling schemes presented so far to support vertex identifiers by modifying their encoder to concatenate the vertex identifier to each label. In the case of *NCA* such an extension is not as straightforward since it should return, for two vertices in the tree, the vertex identifier of a third vertex. We treat both variants, and denote labeling schemes for *NCA* specifically designed to support vertex identifiers as *Id-NCA*, and those that do not as *Label-NCA*.

In Section A.1 we provide a literature overview and describe connections to related problems. In Section A.2 we survey a connection between *Id-NCA* and the functions *Distance*, *SepLevel* and *Center*. The connection leads to a lower bound, for which an asymptotically identical upper bound is presented. We dedicate Section A.3 to the construction of a *Label-NCA* labeling scheme, of the same asymptotic size. We then show how to improve this labeling scheme to achieve labels of size $O(\log n)$. Finally, we discuss an extension of labeling schemes that provides a natural bridge between the two variants.

A.1 Literature review

The problem of finding nearest (occasionally referred, least) common ancestors (NCAs) is non-trivial already in the non-distributed setting. Its importance is derived by its role as a subroutine of common algorithms for minimum spanning trees in a graph, finding maximum weighted matching in a graph, and bounded tree-width algorithms. Aho, Hopcroft and Ullman [153] were among the first to consider the problem of finding NCAs, and Harel and Tarjan [54] were the first to describe an algorithm that uses only linear time and space for pre-processing 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 generalise the results to arbitrary trees. A more recent, simpler, and distributed algorithm was presented by Alstrup et. al [44] in the form of a *Label-NCA* labeling scheme of size $10 \log n + O(1)$. The labeling scheme was improved recently

by Green et. al [51] to $3\log n + O(1)$, along with a lower bound of $1.008n$, and a non-constructive proof of a labeling scheme of size $2.772\log n + O(1)$.

Peleg [154] showed that for *Id-NCA* labels of size $\Theta(log^2 n)$ are sufficient and required. Blin et. al [155] extend Alstrup et al.'s labeling scheme for labels of length bound by constant k. Experimental studies of labeling schemes for the function are considered in [131,156]. Korman [29] studied *Id-NCA* in a query labeling scheme variant (Section 1.8.1). Recall that In this model, the decoder has access to a *query* function, that gets two labels and returns a vertex identifier. In this setting *Id-NCA* queries can be answered using labels of size $O(\log n)$.

Connection to other problems The function *NCA* is tightly related to two problems. First, finding a nearest common ancestor labeling scheme is equivalent to the *discrete range searching problem* [157]. Second, for both *Id-NCA* and *Label-NCA*, the labels produced can determine ancestry relation directly. Put formally, any labeling scheme $\langle e, d\rangle$ supporting the *NCA* function computes a label assignment e_T with following property: For $u, v \in T$, we can construct a decoder that receives $\mathcal{L}(u), \mathcal{L}(v)$ determines ancestry. The decoder may use the *NCA* decoder d and compare the result to $\mathcal{L}(u)$. If u is a parent of v, the two are equivalent and our ancestry decoder may safely return *True*. As a result, any lower bounds that apply to ancestry labeling schemes apply to NCA schemes as well. A last connection between the function *NCA* and routing is discussed in Section A.3.

A.2 *Id-NCA*

In this section, we present lower and upper bounds for *Id-NCA*.

A.2.1 *NCA*, *SepLevel*, and their connection to *Distance*

Peleg [25] proved that $\Omega(\log^2 n)$ bits are necessary for any *Id-NCA* labeling scheme, as well as the functions *SepLevel*, and *Center*. Recall that the function *SepLevel* returns the length of the path $r \rightsquigarrow w$ in T where r is the root and w is $NCA_T(u, v)$.

Theorem 34. *[25] For labeling schemes on $Trees(n)$, we have the following claims:*

1. *Given an $f(n)$* Distance *labeling scheme, we can construct a $f(n) + \log n$-*SepLevel *labeling scheme.*
2. *Given an $f(n)$* SepLevel *labeling scheme, we can construct a $f(n) + \log n$-*Distance *labeling scheme.*
3. *Let $g(n)$ denote the maximum size of an identifier of any vertex in $Trees(n)$. If an* Id-NCA *labeling scheme has labels of size at most $l(n) \cdot g(n)$ then there exist a* SepLevel *labeling scheme of size $l(n) \cdot (g(n) + \log n)$.*

Proof. To prove Claim 1, assume there exist a distance labeling scheme $\langle e_{dist}, d_{dist}\rangle$. The encoder e_{dist} computes a label assignment for a tree T

rooted in r with vertices u, v and w such that $w = NCA(u, v)$. We transform the encoder e_{dist} by concatenating a prefix depth(v) [1] to the label $\mathcal{L}(v)$ assigned by e_{dist} to every $v \in V$, using additional $\log n$ extra bits. The decoder can now compute *SepLevel* (u, v), by observing that $Distance(u, v) = Distance(u, NCA(u, v)) + Distance(v, NCA(u, v))$, and depth($u$) = depth($NCA(u, v)$)+ $Distance(u, NCA(u, v))$ (the same formula holds by replacing u with v). It follows that

$$\text{depth}(NCA(u, v)) = \frac{\text{depth}(u) + \text{depth}(v) - dist(u, v)}{2} = SepLevel(u, v).$$

We prove Claim 2 using the same transformation, i.e. adding depth(v) to all labels produced by the encoder of any *SepLevel* labeling scheme. We prove Claim 3 by the same transformation, with the exception that the new encoder adds depth(v) to the vertex identifier, adding exactly $\log n$ for each of the vertex identifiers. □

Distance labeling scheme has a lower bound of $\Omega(\log^2 n)$ on the size of the label [158]. Assume that there exist a labeling scheme for *Id-NCA* of size $\phi(n) = o(\log^2(n))$. Since $g(n) = \log n$, it implies that $l(n) = o(\log n)$. By Claim 3, there also exist a corresponding labeling scheme for *SepLevel* of size $o(\log^2 n)$. That labeling scheme, by Claim 2, leads to a distance labeling scheme for distance of size $o(\log^2 n)$, in contrast with the lower bound mentioned [22].

Corollary 7. *Any labeling scheme supporting* Id-NCA *for Trees(n) must use labels of size* $\Omega(\log^2(n))$.

A.2.2 Upper bound for *Id-NCA*

It remains to prove that there exist an *Id-NCA* matching (asymptotically) the lower bound. For brevity, we repeat the definitions related to heavy-light decomposition (Section 2.3.1):

hchild(v) is the (unique) heavy child of v, lchildren(v) is the set of light children of v, lsize(v) is the weight of v not including the tree rooted in its heavy child, lpath(v) is the list of all light vertices in $r \rightsquigarrow v$, ldepth(v) = |lpath(v)|, hpath(v) is the set of vertices on the same heavy path as v.

Theorem 35. *[25] There exist an* Id-NCA *labeling scheme with labels of size at most* $O(\log^2 n)$.

Proof. Let T be a tree rooted in r, where every vertex $v \in V$ with light depth ldepth(v) has a unique predefined identifier, Id(v). The label of vertex $v \in T$ is defined as a concatenation of two parts, namely, $\mathcal{C}_a(v)$ and $\mathcal{C}_b(v)$ defined below.

$\mathcal{C}_a(v)$ contains Id(v), and an ancestry label as defined in Section 7.1, using at most $2\log n$ bits, and in total at most $3\log n$ bits. $\mathcal{C}_b(v)$ contains a concatenation (Section 2.1) of triplets of the form $\langle \text{Id}(l_i), \text{Id}(n_i), d_i \rangle$, for $1 \leq i \leq ldepth(v)$, where l_i is the i'th light vertex in lpath(v), d_i is the depth of l_i, and n_i is the vertex adjacent to l_i on $r \rightsquigarrow l_i$. Each triplet requires at most $3\log n$ bits, and thus $\mathcal{C}_b$ contains at most $3\log^2 n$ bits. In conclusion, for every vertex $v \in T$, $|\mathcal{L}(v)| = |\mathcal{C}_b(v)| + |\mathcal{C}_a(v)| \leq 3(\log^2 n + \log n)$.

[1] Section 1.7): A vertex v in a tree $T = (V, E)$ rooted in r has depth(v) edges on the path $r \rightsquigarrow v$.

We describe the operation on two vertices $u, v \in T$ with NCA w. The decoder uses $\mathcal{C}_a$ to determine if one is an ancestor of the other, and if so returns its vertex identifier. At this point we can safely claim that the path $u \rightsquigarrow v$ traverses exactly two of w's edges, where at least one of those must be a light edge (see Figure A.1 for a demonstration). Thus, the decoder computes the common prefix of $\mathcal{C}_b(u)$ and $\mathcal{C}_b(v)$, and determines the first k for which the k'th triplet in $\mathcal{C}_b(u)$ is not equal to the k'th triplet in $\mathcal{C}_b(v)$. Since each triplet contains the depth of the light vertex it represents, we can deduce which of them is closer to w, and if the depth is equal we choose one arbitrarily. We return the parent of the light vertex, stored in the selected triplet. □

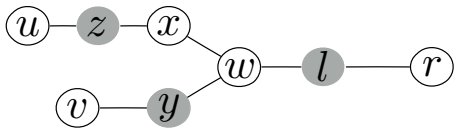


FIGURE A.1: The nearest common ancestor w=*NCA* (u,v) in a root rooted in r. The vertex l is the last light vertex in common in the paths $r \rightsquigarrow u$ and $r \rightsquigarrow v$, and y and z are the first light vertices immediately after l on those paths, where $depth(z) > depth(y)$. Straight lines represent edges, dashed lines are paths, and grey vertices are light vertices.

A.3 *Label-NCA*

We prove that there exist a *Label-NCA* labeling scheme of size $O(\log n)$ in two steps. First, we present a simple (inefficient) labeling scheme with $O(\log^2 n)$ and then we prove that by a slight improvement, the label size decreases to $O(\log n)$. The first labeling scheme is by itself a modified version of the one presented in Theorem 35. Rather than storing the light path for every vertex along with the vertex above it, we store the light path along with the distance of the heavy paths between each two consecutive light vertices.

Theorem 36. *There exist a* Label-NCA *labeling scheme* $\langle e, d \rangle$ *with label size bounded by* $O(\log^2 n)$.

Proof. Let $T = (V, E)$ be a tree rooted in r. For convenience, r is a heavy vertex with the label 0. We first compute a heavy-light decomposition of T (Section 2.3.1).

The new label comprises the concatenation of the tuples $\langle h_i, l_i \rangle$, where l_i is the index of i'th light vertex in $\text{lpath}(v)$, and h_i is the (possibly null) length of the heavy path $\text{hpath}(l_i)$, from l_{i-1} to l_i or to v when $i = \text{ldepth}(v)$. See Figure A.2 for an illustration.

Since $1 \leq h_i, l_i \leq n$, each tuple requires at most $2 \log n$ bits. Since $|\text{lpath}(v)| \leq \log n$ (Section 2.3.1), the total label size is bounded by $2 \log^2 n$. To complete the proof we show how to compute the label of the NCA of two vertices u and v with labels $\mathcal{L}(u) = h^u_1, l^u_1 \ldots h^u_k, l^u_k$ and $\mathcal{L}(v) = h^v_1, l^v_1 \ldots h^v_{k'}, l^v_{k'}$ respectively. Without loss of generality assume that $k \leq k'$. We find the first i for which $h^u_i, l^u_i \neq h^v_i, l^v_i$. If there is no such i, then u is the ancestor of v and we return its label. If $h^u_i = h^v_i$ we return the label $h^u_1, l^u_1 \ldots h^u_i$. Otherwise, we return $h^u_1, l^u_1 \ldots min\{h^u_i, h^v_i\}$.

□

Note that this labeling scheme can return the depth of the NCA, in other words the *SepLevel*. By Theorem 34, any labeling scheme supporting *SepLevel* must have labels of size $\Omega(\log^2 n)$. Moreover, using the formulas from Theorem 34 we observe that the function *Distance* may also be computed using these labels. The first labeling scheme for the function *Distance* [159] uses separator decomposition. The label created in this labeling scheme can not be used to determine the functions *NCA*, *Adjacency* and *Ancestry*. In contrast, our labeling scheme stores enough information on the topology of the tree to determine all(!) the functions surveyed on trees.

The labeling scheme presented is based on the ability to choose a vertex's name. The one presented next utilises the ability further, and reduces the bound on the label size from $O(\log^2 n)$ to $O(\log n)$.

A.3.1 *Label-NCA* with $O(\log n)$ bits

The maximum label size in Theorem 36 is the longest description of a sequence of at most $\log n$ tuples of the form $\langle h_i, l_i \rangle$. We do not consider assigning short labels to vertices with big size, nor do we account for the total length possible for the heavy paths. However, even with those improvements, we will still be able to determine *SepLevel*, which implies a label of size $\Omega(\log^2 n)$ as mentioned.

The key observation is that the function *NCA* can be determined even without knowing the exact length of the heavy paths. We only require that each label of the vertices on a heavy path $h_1 \dots h_k$ has a total order on that path, i.e., given two labels, determine which of them is first on the path. Alstrup, Gavoille, Kaplan and Rauhe [44] use those two observations as well as Lemma 2 to construct a $O(\log n)$ *Label-NCA* labeling scheme presented below.

Theorem 37. *[44] There exist a* Label-NCA *labeling scheme of size* $O(\log n)$.

Sketch.
Consider a vertex v where $parent(v) = u$ in a tree T rooted in r with $\text{ldepth}(v) = k$ and $\text{lpath} = \{lp_1 \dots lp_k\}$. The label comprises the concatenation of the tuples $\langle h'_i, l'_i \rangle$. We first construct the labels given by Theorem 36 as auxiliary labels where $\langle h_i, l_i \rangle$ defined as in Theorem 36.

To compute each l'_i, we use Lemma 2 with $\text{size}(lp_i)$ and the lsize of each of its *siblings* in T. The lemma provides l'_i a label appropriate to $\text{size}(lp_i)$, and more precisely:

$$|l'_i| \leq \log \text{size}(v) - \log \text{lsize}(p(lp_i)) + 1,$$

where $p(lp_i)$ is the parent of lp_i.

To compute each h'_i we use Lemma 2 with the size of all vertices on the heavy path rooted in lp_i, $\text{hpath}(v)$, ordered by their depth. That is, since we want to have labels as small as possible the closer we are to lp_i. The lemma provides h'_i a label appropriate to the light size of the h_i'th vertex on $\text{hpath}(v)$. More precisely:

$$|h'_i| \leq \log \text{size}(lp_i) - \log \text{lsize}(v) + 1.$$

In contrast to the previous labeling scheme, both l'_i and h'_i have variable size. In order to distinguish the different parts in $\mathcal{L}(v) = h'_1, l'_1, \ldots h'_k, l'_k$ we use a separating string (Section 2.1), which doubles the size of the label. By induction, it can be shown that $|h'_1|, |l'_1|, \ldots |h'_k|, |l'_k| \leq \log n + 4k + 2$. The proof is similar to that of Lemma 29.

The decoder is computed similarly to the one defined in Theorem 36 with the exception that in the last case we use $\min_{<_{\text{lex}}}\{h_i, h'_i\}$ instead of $\min\{h_i, h'_i\}$.

For proof of the induction, and the correctness of the decoder, see [43]. □

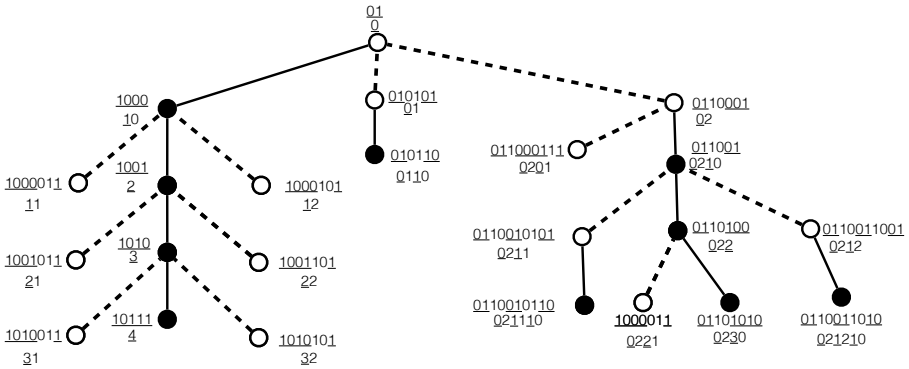


FIGURE A.2: A tree with heavy (black) and light (white) vertices marked in the labeling scheme from Theorem 37 in binary on top and Theorem 35 in decimal at the bottom. Both labels have their heavy sub-labels underlined.

The labels constructed in Theorem 37 can determine the first edge on the shortest path between the vertices queried. Therefore, using a slightly different decoder, we can construct a *Routing* labeling scheme (Chapter 10).

The label size achieved in this method is at most $10 \log n$. Green et. al [51] recently achieved an improved labeling scheme of size $3 \log n$ by replacing Lemma 2 with a more compact total order which allows for empty strings.

Labeling schemes with a query Korman [29] extend the definition of labeling scheme such that alongside the encoder and decoder, they define a *query* function. Formally, given the labels $\mathcal{L}(u)$ and $\mathcal{L}(v)$ of $u, v \in V$ outputs $Q(\mathcal{L}(u), \mathcal{L}(v))$ which is a vertex $c \in V$. The decoder is free to use c to compute the query. In this context, the authors proved that both *Label-NCA* and *Id-NCA* have a similar label size of $O(\log n)$. The authors achieve similar label sizes for the functions *Distance*, *Routing* and *MaxFlow*.

Appendix B

Proofs Omitted

B.1 Proof of Lemma 5

Consider the non-leaf node v with parent $p(v)$ and heavy child $h(v)$ in a rooted tree T. Recall that the encoder of $\langle e_\alpha, d_\alpha \rangle$ labels v as a concatenation of the bit strings: *I.)* $\mathrm{dfs_i}(v)$; *II.)* $\mathrm{ldepth}(v)$; *III.)* a 1/2-approximation of $\mathrm{dfs_i}(v) - \mathrm{dfs_i}(p(v))$; and *IV.)* a 1/2-approximation of $\mathrm{dfs_i}(h(v)) - \mathrm{dfs_i}(v)$.

We denote the 1/2-approximation of $\mathrm{dfs_i}(v) - \mathrm{dfs_i}(p(v))$ as $\lfloor P \rfloor_2(v)$ and and the 1/2-approximation of $\mathrm{dfs_i}(h(v)) - \mathrm{dfs_i}(v)$ as $\lfloor C \rfloor_2(v)$. The decoder receives both labels $\mathcal{L}(u)$ and $\mathcal{L}(v)$ and computes the value $\mathrm{recomp}(u, v)$ which takes a 1/2 approximation of the values $\mathrm{dfs_i}(u) - \mathrm{dfs_i}(v)$ given by their labels.

Suppose u is the parent of the node v. The decoder d_α returns true for $\mathcal{L}(u), \mathcal{L}(v)$ by verifying the following predicates hold:

- $\mathrm{dfs_i}(u) < \mathrm{dfs_i}(v)$.
- $\mathrm{ldepth}(v) = \mathrm{ldepth}(u)$ if v is heavy, and $\mathrm{ldepth}(v) = \mathrm{ldepth}(u) + 1$ if v is light.
- $\lfloor C \rfloor_2(u) = \lfloor P \rfloor_2(v)$ if v is heavy and $\lfloor C \rfloor_2(u) \geq \lfloor P \rfloor_2(v)$ if v is light.
- $\mathrm{recomp}(u, v) = \lfloor P \rfloor_2(v)$.

The above conditions are clearly sufficient if u is the parent of v. We now denote the heavy child of u as $h(u)$. Assume in contradiction that the conditions hold, but u is not the parent of the node v. First, assume that v is light. The first two predicates assure that u can not be a descendant of v. Since v has $\mathrm{ldepth}(u) + 1$ it must be the child of either of $h(u)$ or not a decedent of v at all. In both cases the path $u \rightsquigarrow v$ must contain at least one more node $y = p(v)$ where $\mathrm{dfs_i}(y) \geq \mathrm{dfs_i}(h(u))$. It follows that :

$$\mathrm{dfs_i}(v) - \mathrm{dfs_i}(u) \geq \mathrm{dfs_i}(v) - \mathrm{dfs_i}(y) + \mathrm{dfs_i}(h(u)) - \mathrm{dfs_i}(u) \geq 2min(\lfloor C \rfloor_2(u), \lfloor P \rfloor_2(v)). \tag{B.1}$$

Since v is light we know that:

$$\lfloor C \rfloor_2(u) \geq \lfloor P \rfloor_2(v) = \mathrm{recomp}(u, v).$$

It follows that $\mathrm{recomp}(u, v) = min(\lfloor C \rfloor_2(u), \lfloor P \rfloor_2(v))$, and since recomp is a 1/2-approximation:

$$2\,\mathrm{recomp}(u, v) = 2min(\lfloor C \rfloor_2(u), \lfloor P \rfloor_2(v)) > \mathrm{dfs_i}(v) - \mathrm{dfs_i}(u),$$

which is a contradiction.

If v is heavy, it may no longer be a decedent of u, and there exist a $y = p(v)$ for which equation B.1 holds as before, and the contradiction holds similarly.

B.2 Proof of the claim in Section 3.3

The proof follows directly from the following lemma.

Lemma 31. *The following property holds for every $2 \leq m \leq p$ for some constant c':*

$$\sum_{i=m}^{p} 2^{t_i} \leq c' \sum_{i=m}^{p} \lceil l(i) \rceil_2 - 2^{t_m}.$$

Proof. We prove this claim by induction over m from p down to 2. The claim holds trivially for $m = p$. Assume that the property holds for m, for $m-1$ by the induction hypothesis:

$$\sum_{i=m-1}^{p} 2^{t_i} \leq c' \sum_{i=m}^{p} \lceil l(i) \rceil_2 - 2^{t_m} + 2^{t_{m-1}}.$$

If $\lceil l(m-1) \rceil_2 \geq 2^{t_m}$ then $2^{t_{m-1}} = \lceil l(m-1) \rceil_2$ by definition, and thus for $c' > 2$:

$$\sum_{i=m-1}^{p} 2^{t_i} \leq c' \sum_{i=m}^{p} \lceil l(i) \rceil_2 - 2^{t_m} + 2^{t_{m-1}} = c' \sum_{i=m-1}^{p} \lceil l(i) \rceil_2 - 2^{t_m} - (c'-1)\lceil l(m-1) \rceil_2$$
$$\leq c' \sum_{i=m-1}^{p} \lceil l(i) \rceil_2 - 2^{t_{m-1}}.$$

If $\lceil l(m-1) \rceil_2 < 2^{t_m}$ then $2^{t_{m-1}} \leq 2t_m$ by definition. We want to prove that in this case

$$2^{t_{m-1}} \leq \frac{1}{2} 2^{t_m} = 2^{t_m - 1}$$

Which holds so long as $t_i \geq 4$ for all i, which can be maintained as long as $c' > 16$. We can now replace the terms such that:

$$\sum_{i=m-1}^{p} 2^{t_i} \leq c' \sum_{i=m}^{p} \lceil l(i) \rceil_2 - 2^{t_m} + 2^{t_{m-1}} = c' \sum_{i=m-1}^{p} \lceil l(i) \rceil_2 - 2^{t_m}/2 \leq c' \sum_{i=m-1}^{p} \lceil l(i) \rceil_2 - 2^{t_{m-1}}.$$

□

Choosing $c = c' + 1 \geq 5$ we conclude:

$$\sum_{i=1}^{p} 2^{t_i} \leq c \sum_{i=m-1}^{p} l(i) - 2^{t_{m-1}}.$$