A

A Survey on Labeling Schemes for Trees

Stephen Alstrup, Esben Bistrup Halvorsen and Noy Rotbart

A labeling scheme is a method of labeling the nodes of a graph so that queries concerning nodes can be answered efficiently directly from their labels. For the last 25 years, labeling schemes have been investigated, and they play an important role in the theory of distributed data structures. The two main characteristics of a labeling scheme are: the type of queries that can be answered from the labels; and the family of graphs which the labeling scheme can accept as input. The primary indicator of quality of a labeling scheme is the size of the labels it produces. We survey the literature on lower and upper bounds for the worst-case sizes of labels produced by labeling schemes for trees. In addition, we present selected techniques and results that achieve the best known lower and upper bounds as well as selected variants thereof. The survey presents the results systematically on a standard selection of families of trees.

Additional Key Words and Phrases: Labeling schemes, Induced universal graphs

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 its scalability and performance.

Labeling schemes allow the structure of a graph to be distributed among its nodes 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 nodes themselves. On the one hand, storing the entire data structure at each node 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.

The papers surveyed are built for specific types of queries. *Adjacency* labeling schemes were the first to be investigated in the literature. The nodes in a graph are labeled in such a way that the *Adjacency* between two nodes can be determined directly from their labels. A restricted form of *Adjacency* labeling schemes for graphs was studied almost 50 years ago by Breuer and Folkman [1967]. The more general concept was defined 25 years later by Kannan, Naor and Rudich [1992] as well as by Muller [1988]. 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 detail on the historical development of labeling schemes can be found in [Gavoille and Peleg 2003].

I don't think "adjacency" should be written like this all over the place (but it's fine to have it slanted here).

This survey considers labeling schemes for the family of trees with at most n nodes. We focus on this particular family for the following reasons: When graphs are considered, labeling schemes for all of the functions we are surveying require a label size that is in the order of the size of the graph itself. Even for the most basic query, *Adjacency*, labeling schemes require at least $n/2$ bits to support the family of graphs [Rado 1964]. It is thus not surprising that the focus of the body of work surveyed, and the early definitions of labeling schemes, require labels of a size that is 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 [Gavoille and Labourel 2007b], bounded degree [Adjiashvili and Rotbart 2014], bounded arboricity [Alstrup and Rauhe 2002], and planar graphs [Thorup and Zwick 2001] 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 node a *label*, which is just some data, such that, given the labels of two nodes, one can determine if the nodes are adjacent or not.

Consider an arbitrary rooted tree with n nodes. Enumerate the nodes in the tree with the numbers 0 through $n-1$ as binary strings, and let, for each node v, $\mathrm{Id}(v)$ be the number associated with v and, when v is not the root, $p(v)$ be the parent of v in T. Consider the labeling $\mathcal{L}(v) = (\mathrm{Id}(v), \mathrm{Id}(p(v)))$ of all non-root nodes v and $\mathcal{L}(\mathrm{Id}(r), \mathrm{Id}(r))$ for the root node r. The encoding of the labels is demonstrated in Figure 1. Note that two nodes u and v 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. Thus it is possible to determine adjacency of u and v directly from their labels $\mathcal{L}(u)$ and $\mathcal{L}(v)$.

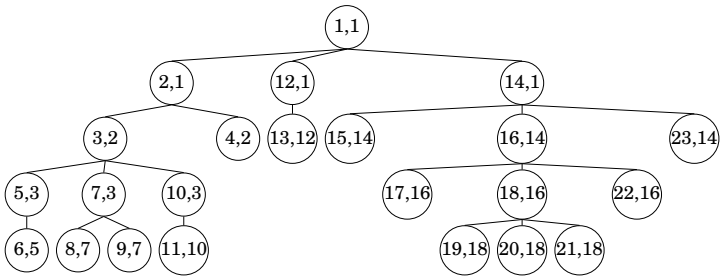

Fig. 1. A tree with $n = 23$ nodes. Each node is assigned a label of size $2 \log n$ supporting *Adjacency* queries. A node v with parent u is labeled $(\mathrm{Id}(v), \mathrm{Id}(u))$ and the root r is labeled $(\mathrm{Id}(r), \mathrm{Id}(r))$.

This is a labeling scheme. It consists of an *encoder* algorithm that labels the nodes of a tree and a *decoder* algorithm that can answer *Adjacency* queries using only labels as input. Note that the encoding algorithm relies on knowing the entire tree, whereas the decoding algorithm only knows the labels it receives as input and nothing else.

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.

Do we really want to assume that n is a power of 2? In principle, you cannot know that such an algorithm works for non-powers of 2 (although it most often will).

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 node an ancestor of the other?) or "*Distance*" (what is the distance between two nodes?). Second, there is the family of graphs under consideration. In our example, the labeling scheme is for the family of all trees, but both smaller and larger families would have been possible. A different choice of query or a different choice of graph family may lead to an entirely different labeling scheme with different properties and a different optimal worst-case label size.

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

1.2. Outline

This survey provides an 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 $\mathcal{G}$*. We mainly survey results on the graph family of trees with at most n nodes due to great interest in the literature, with occasional emphasis on several bounded cases (i.e bounded degree, bounded depth, caterpillars and binary trees). Tables I and II provide a concise description of the results. An overview of applications for labeling scheme is provided in Section 1.3, and formal definitions are given in Section 1.4.

Let $k \in \mathbb{N}$, and let $T = (V, E) \in Trees(n)$ be a tree rooted at the node r. We survey labeling schemes supporting the following functions for any $u, v, w \in V$:

(1) *Adjacency* : $(V \times V) \to \{\text{false}, \text{true}\}$ (Section 4)
Adjacency$_T(u, v) = \text{true}$ if and only if u and v are adjacent in T.
(2) *Ancestry* : $(V \times V) \to \{\text{false}, \text{true}\}$ (Section 3)
Ancestry$_T(u, v) = \text{true}$ if and only if u is an ancestor of v in T.
(3) *NCA* : $(V \times V) \to \{0, 1\}^+$ (Section 5)
NCA$_T(u, v)$ returns the label of the first node in common for the paths $v \rightsquigarrow r$ and $u \rightsquigarrow r$ in T.
(4) *Routing* : $(V \times V) \to \mathbb{N}$ (Section 6)
Routing$_T(u, v)$ returns the port number (Definition 6.1) leading to the next node on the path $u \rightsquigarrow v$ in T.

We also discuss the functions *Non-Adjacency* and *Non-Ancestry* which are the negated functions of *Adjacency* and *Ancestry*. Each section contains a detailed literature overview, as well as a report on one or more central results presented in some detail. Common for the algorithms presented is that, in most cases, they are the best known results.

Labeling schemes for the following functions are also reported in Table I and occasionally mentioned. Some of these functions are also defined for the family of edge weighted n node trees, i.e. trees where each edge is assigned an integer at most 2^M for some integer M. We denote this family as $WeightedTrees(n, 2^M)$.

(1) *Center* : $(V \times V \times V) \to \{0, 1\}^+$
Center$_T(u, v, w)$ returns the unique node z such that the three paths $z \rightsquigarrow u$, $z \rightsquigarrow v$, and $z \rightsquigarrow w$ in T are edge-disjoint.
(2) *Distance* : $(V \times V) \to \mathbb{N}$
Distance$_T(u, v)$ returns the length of the path $u \rightsquigarrow v$ in T, which is equal to the number of edges in $u \rightsquigarrow v$ if T is unweighted.
(3) *MaxFlow* : $(V \times V) \to \mathbb{N}$
MaxFlow$_T(u, v)$ returns the smallest edge weight on $u \rightsquigarrow v$ in T.
(4) *Siblings* : $(V \times V) \to \{false, true\}$
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 node is its own sibling.
(5) *SepLevel* : $(V \times V) \to \{0, 1\}^+$ (Section 5.2)
SepLevel$_T(u, v)$ returns the length of the path $r \rightsquigarrow w$ in T where r is the root and w is $NCA_T(u, v)$.
(6) *Small-Distance* : $(V \times V \times \mathbb{N}) \to \mathbb{N}$
Small-Distance$_T(u, v, k) = true$ if and only if *Distance*$_T(u, v) \leq k$ in T, otherwise, it returns ∞.

See Figure 2 for an illustration of the number of bits required for labeling schemes for each of these functions. In the course of the survey, we provide missing details and

Why are tables numerated with Roman numerals? It seems a bit inconsistent.

The "Trees" should be a LaTeX command - all over the document...

It's totally ugly, in my opinion, that all these functions are slanted. Use \operatorname or something.

Do we need to define and point out the uniquess of "$\rightsquigarrow$"?

I think this table should be properly introduced. Otherwise it's a bit weird to write "also".

"WeightedTrees" should be a LaTeX command

Do we need to define "length of the path" in the weighted case?

I'm not sure that is the standard definition. The labeling problem is different in the two cases—uniqueness is not required!

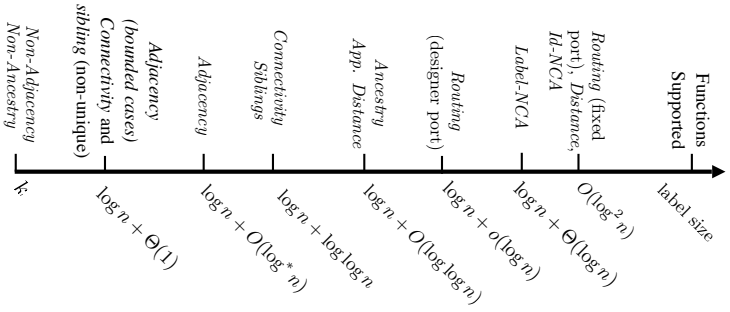

Fig. 2. Given a certain label size, the illustration describes which operations may be supported.

correct minor mistakes found in the literature surveyed. We also present an improved labeling scheme for the function *Ancestry* in Section 3.2.

Comment to Table I: There is also a less efficient NCA labeling scheme with smaller labels

Bounded trees. In Table II we complete the description of the following cases: $Caterpillars(n)$, $Trees(n, \delta)$, $Trees(n, \Delta)$, and $BinaryTrees(n)$, where Δ refers to maximum degree, M to 2^M edge weights, and δ to maximum depth. Note that the table is transposed compared to Table I. We chose to present the results in this fashion since the focus is on the restricted families.

Comment to Table II: It's not the encoder that is "unique"

1.3. Practical and theoretical applications of labeling schemes

Comment to non-existing table: What about all the other variations of labeling schemes, e.g. approximation, probabilistic etc.?

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 [Abiteboul et al. 2001]. An XML document can be viewed as a rooted tree in which each node 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 [Cohen et al. 2002; Lu 2013; Härder et al. 2007; Abiteboul et al. 2001; Wu et al. 2004; Cohen et al. 2010a].

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

Table I. Known upper and lower bounds for efficient 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 formally defined in Section 1.4. Results implied, but not specified in the literature are marked with †.

Reference	Variant	Upper bound	Lower bound	Encoder	Decoder
		Adjacency			
[Alstrup et al. 2015]	$Trees(n)$	$\log n + O(1)$	$\log n + 1$	$O(n)$	$O(1)$
[Bonichon et al. 2006]	$BinaryTrees(n)$	$\log n + O(1)$	$\log n + 1$	$O(n)$	$O(1)$
[Fraigniaud and Korman 2010a]	$Trees(n,\delta)$	$\log n + 3\log\delta + O(1)$	$\log n + 1$	$O(n)$	$O(1)$
[Adjiashvili and Rotbart 2014]	$Trees(n,\Delta)$	$\log n + O(\log\Delta)$	$\log n + 1$	$O(n\log n)$	$O(\log\log n)$
		Non-Adjacency			
[Fraigniaud and Korman 2009]	One sided error*	$2k+1$ with $p = 1 - \frac{1}{2^k}$	$---$	$O(n)$	$O(1)$
		Ancestry			
[Dahlgaard et al. 2014c]	$Trees(n)$	$\log n + 2\log\log n + 3$	$\log n + \log\log n$	$O(n)$	$O(1)$
[Fraigniaud and Korman 2010a]	$Trees(n,\delta)$	$\log n + 2\log\delta + O(1)$		$O(n)$	$O(1)$
[Fraigniaud and Korman 2009]	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			
[Fraigniaud and Korman 2009]	One sided error*	$\lceil \log n \rceil$ with $p = \frac{1}{2}$	$\log n + \log p - O(1)$	$O(n)$	$O(1)$
		Center			
[Peleg 2005]	$Trees(n)$, fixed model	$\Theta(\log^2 n)$	$\Theta(\log^2 n)$	$O(n\log n)$	$O(1)^\dagger$
		Connectivity			
[Alstrup et al. 2005]	$Forests(n)$	$\log n + \log\log n$	$\log n + \log\log n$	$O(n)$	$O(1)$
[Alstrup et al. 2005]	$Forests(n)$*	$\log n$	$\log n$	$O(n)$	$O(1)$
		Distance			
[Freedman et al. 2016]	$Trees(n)$	$\frac{1}{4}\log^2 n + o(\log^2 n)$	$\frac{1}{4}\log^2 n - O(\log n)$	$O(n\log n)$	$O(1)^\dagger$
[Gavoille et al. 2001]	$WeightedTrees(n, 2^M)$	$\Theta(M\log n + \log^2 n)$	$\Theta(M\log n + \log^2 n)$	$O(n\log n)$	$O(\log n)$
		Distance, approx. $1+1/n$			
[Gavoille et al. 2000]	2^M weighted diameter	$\Theta(\log n \cdot \log M)$	$\Theta(\log n \cdot \log M)$	$O(nm)$	$O(1)$
[Gavoille et al. 2000]	$Trees(n)$	$\Theta(\log n \cdot \log\log n)$	$\Theta(\log n \log\log n)$	$O(n\log n)$	$O(1)$
		MaxFlow			
[Katz et al. 2004]	$WeightedTrees(n, 2^M)$	$\Theta(M\log n + \log^2 n)$	$\Theta(M\log n + \log^2 n)$	$O(n)$	$O(1)$
		NCA			
[Alstrup et al. 2014]	$Trees(n)$, *Label-NCA*	$3\log n$	$1.008\log n$	$O(n)$	$O(1)$
[Peleg 2005]	$Trees(n)$, *Id-NCA*	$O(\log^2 n)$	$\Omega(\log^2 n)$	$O(n\log n)$	$O(1)^\dagger$
		Routing			
[Thorup and Zwick 2001]	$Trees(n)$, designer port	$(1+o(1))\log n$	$\log n + \log\log n$	$O(n\log n)$	$O(1)$
[Fraigniaud and Gavoille 2001]	$Trees(n)$, fixed port	$\Theta(\log^2 n/\log\log n)$	$\Theta(\log^2 n/\log\log n)$	$O(n)$	$O(1)$
		Siblings			
[Lewenstein et al. 2013]	$Trees(n)$	$\log n + \log\log n$	$\log n + \log\log n$	$O(n)$	$O(1)$
[Alstrup et al. 2005]	$Trees(n)$*	$\log n$	$\log n$	$O(n)$	$O(1)$
		Small-Distance			
[Freedman et al. 2016]	$Trees(n)$,distance k	$\min \log n + O(k\log(\log n/k))$ $O(\log n \cdot \log(k/\log n))$	$\log n + \Omega(k\log(\log n/(k\log k)))$	?	$O(1)$

Table II. Known upper and lower bounds for labeling schemes on trees with unique encoder.

	Adjacency	*Ancestry*	*Distance*	*NCA*	*Siblings*
$Caterpillars(n)$:					
Upper bound	$\log n + \Theta(1)$ [Bonichon et al. 2006]	$\log n + O(1)$	$2(\log n + M)$	$\log n + \log\log\Delta + O(1)$	$\log n + O(\log\log n)$ [Alstrup et al. 2005]
Lower bound	$\log n + \Theta(1)$	$\log n + \Theta(1)$	$\log n + \Theta(1)$	$\log n + \Theta(1)$	$\log n + \Omega(\log\log n)$
$BinaryTrees(n)$:					
Upper bound	$\log n + O(1)$ [Bonichon et al. 2006]	$\log n + \Theta(\log\log n)$ [Fraigniaud et al. 2010b]	$\frac{1}{2}\log^2 n + O(\log n)$ [Gavoille et al. 2001]	$O(\log n)$[Alstrup et al. 2002] [Alstrup et al. 2002]	$\log n + \Theta(1)$
Lower bound	$\log n + \Theta(1)$	$\log n + \Theta(\log\log n)$	$\frac{1}{8}\log^2 n - O(\log n)$ [Gavoille et al. 2001]	$\log n + \Theta(1)$	$\log n + \Theta(1)$
$Trees(n, \Delta)$:					
Upper bound	$\log n + O(1)$ [Adjiashvili et al. 2014]	$\log n + \Theta(\log\log n)$ [Fraigniaud et al. 2010b]	$\frac{1}{2}\log^2 n + O(\log n)$ [Gavoille et al. 2001]	$O(\log n)$ [Alstrup et al. 2002]	$\log n + \Theta(\log\log\Delta)$ [Alstrup et al. 2005]
Lower bound	$\log n + \Theta(1)$	$\log n + \Theta(1)$	$\frac{1}{8}\log^2 n - O(\log n)$ [Gavoille et al. 2001]	$\Omega(\log n)$ [Alstrup et al. 2002]	$\log n$
$Trees(n, \delta)$:					
Upper bound	$\log n + 3\log d + O(1)$ [Fraigniaud et al. 2010b]	$\log n + 2\log\log d + O(1)$ [Fraigniaud et al. 2010b]	$\frac{1}{2}\log^2 n + O(\log n)$	$\log n + O(1)$	$\log n + \Theta(\log\log n)$ [Alstrup et al. 2005]
Lower bound	$\log n + \Theta(1)$	$\log n + \Theta(1)$	$\log n + \Theta(1)$	$\log n + \Theta(1)$	$\log n + \Theta(\log\log n)$

particular direction focused on seemingly small benefits. For example, both relations used by practitioners, namely, adjacency (illustrated in Figure 1) and ancestry had strikingly simple labeling schemes with labels of at most $2\log n$ bits [Kannan et al. 1992]. In order to identify the XML element uniquely, the label of an element in an XML file with at most n elements requires at least $\log n$ bits. 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.

You just argued that it cannot be reduced. I guess you mean everything in addition to the $\log n$.

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

Kannan et al. [1992] noticed the tight correlation between adjacency labeling schemes and induced-universal graphs. Induced-universal graphs is a mathematical subfield of implicit graph representations and has been researched extensively since the 1960's [Rado 1964]. For various numbers of families, the goal is to find the smallest number of nodes required for a graph that contains each of the n-node graphs of the family as induced subgraphs. A subsequent connection between universal matrices and *Distance* labeling schemes was studied by Korman et al. [2010] and Gavoille and Paul [2001]. We defer further discussion on the nature of the connection to Section 4.

Shortest paths in road networks. Another practical aspect of labeling schemes concerns distributed shortest paths on road networks. Gavoille et al. [2001] first investi-

gated *Distance* labeling schemes. Abraham, Deiling, Goldberg and Werneck [2010; 2012] 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 [2005], 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 [Gavoille and Peleg 2003; Thorup and Zwick 2001; Fraigniaud and Gavoille 2001; Gavoille and Pérennès 1996; Dom 2007; Korman and Kutten 2007; Krioukov et al. 2004; Abraham et al. 2006; Abraham and Malkhi 2005].

1.4. Preliminaries

In the remainder of the paper we use the following terms: k is a positive integer. A binary string is a member of the set $\{0,1\}^*$. We denote $\lceil \log_2 n \rceil$ by $\log n$. 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 $\mathrm{bin}(x)$ its standard binary representation, and $|x|$ as the number of bits in $\mathrm{bin}(x)$. The concatenation of two bit strings a and b is denoted $a \circ b$.

Really? Maybe you mean that we *define* $\log n$ to mean $\lceil \log_2 n \rceil$?

1.4.1. Graph families related definitions. For a graph G we denote the set of nodes 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 nodes. Unless stated otherwise, the number of nodes 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 nodes 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 3 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.

The latter seems like a really dangerous and weird definition. Then we have e.g. $|4| = 3$.

Again, I think this is somewhat restrictive

Make a LaTeX command for "Forests"

If trees are rooted, this may not be the correct definition

1.4.2. 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 nodes of degree 1 other than the root are called *leaves*, and all other nodes are called *internal* nodes.

Let u and v be nodes in tree T. If $(v, u) \in E$ we say that v and u are *neighbours*. We denote the set of all neighbours of v as $N(v)$, and the degree $|N(v)|$ of v as $\deg(v)$. A non-root node with degree at most 1 is called a *leaf*, and an *inner node* is a non-leaf node. 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 nodes in a tree are connected by a unique

Since edges have no direction, I think the correct definition would be $\{v, u\}$

Can there be duplicates? If yes, then paths are not 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 nodes. If u is a node on the path from the root of a rooted tree to a node v, then u is an *ancestor* of v, and v is a *descendant* of u. Note, in particular, that a node is its own ancestor, descendant and sibling, but not its own child or parent. A *common ancestor* of two nodes is a node that is an ancestor of both nodes, and their *nearest common ancestor* (NCA) is the unique common ancestor with maximum depth. Given a node v, the descendants of v form an induced subtree T_v with v as root. Finally, the *size* of v, denoted $\text{size}(v)$, is the number of nodes in T_v.

Additional standard definitions are provided in Appendix A.

1.4.3. Labeling schemes and variations. We present definitions of labeling schemes, approximation labeling schemes and one-sided error labeling schemes.

Definition 1.1. Let $f : V(G)^k \to S$ be a k-ary function over nodes 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.

Why do you need two notations for the same thing?

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.

Why these strange brackets?

(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 1.2.

i. If, for all $G \in \mathcal{G}$, 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 node depends on the graph to which the node belongs, whereas the decoding of a k-tuple of labels is oblivious to the graph from which the labels come.

iii. By Definition 1.1 labels of unrestricted size can be used to encode the entire graph structure.

Definition 1.3. 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 $t(n)$.

Do we actually measure time for the whole graph and not just individual nodes?

(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 [Kao et al. 2007]).

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

Definition 1.4. Let $f : V(G)^k \to \mathbb{N}$ be a k-ary function over nodes 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 nodes $v_1 \dots v_k \in V$ with labels $\mathcal{L}(v_1) \dots \mathcal{L}(v_k)$ the value computed by d satisfies

Is this the standard definition? I would have thought that the lower bound did not divide by R. Also, what about additive errors, which we will need for distances?

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

We also define the forbidden-set variant [Courcelle et al. 2007].

Definition 1.5. Let f be a boolean function over node 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 1.1) 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 $G \setminus X$.

I simply don't understand this definition. What is this used for? And did we even define "subgraph" anywhere (should it be node-induced etc.?

Finally we define the one-sided error variant [Fraigniaud and Korman 2009].

Definition 1.6. Let f be a boolean function over node 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 1.1) 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 nodes $v_1 \dots v_k \in V$ with labels $\mathcal{L}(v_1)\dots\mathcal{L}(v_k)$

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

From hereon, unless stated otherwise, encoder and decoder stand for exact decoder and deterministic encoder.

I don't think we have defined non-deterministic encoders anywhere?

Denote the family of trees with at most n nodes $total(n)$ and the family of trees with exactly n nodes $exact(n)$. In the literature reviewed, some results [Abiteboul et al. 2001; Kannan et al. 1992] are defined for $total(n)$, while other [Alstrup and Rauhe 2002; Korman and Kutten 2007; Gavoille and Labourel 2007a; Alstrup et al. 2005] 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 [Bistrup Halvorsen 2013]. 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 $total(n)$.

HUH? I can trivially do that!

2. COMMON ALGORITHMIC 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 survey. 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 $|\text{bin}(x)|$.

It's not completely clear to me what "words" are, nor is it clear how a label can be seen as an integer?

A method used in *all* papers surveyed 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

Would it not be better to consider $\{0, \dots, n-1\}$ and use $\lfloor \log x \rfloor$ bits?

This requires that we know the length of the string. Can we always assume that? (In some situations with a "data stream", we know when the string starts but not necessarily when it ends.)

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 $|\mathrm{bin}(x)|< \log n$, then we can use *exactly* $\log n + 1$ bits to describe x using the bit string $\mathrm{bin}(x) \circ 1 \circ 0^i$ where $i = n + 1 - |\mathrm{bin}(x)|$, 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 $\mathrm{bin}(k)$ 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 $\mathrm{bin}(k)$. Now $\mathrm{bin}(k')$ 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 $\left\lfloor \frac{\log n}{\log n+1} k \right\rfloor < k^* \leq k$.

This definition of "approximation" does not comply with the previous one

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 $\mathrm{bin}(j) \circ \mathrm{bin}(k)$ is $j \cdot 2^{|\mathrm{bin}(k)|} + k$, and in addition $j < 2^{|\mathrm{bin}(j)|}$.

LEMMA 2.1. *Let w and z be two integers. One can compute an integer $x \in [z, z + 2^{|\mathrm{bin}(w)|})$ such that $\mathrm{bin}(w)$ is a suffix of $\mathrm{bin}(x)$.*

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

I think something is wrong here. Why is $x = z - m + w$?

2.2. Efficient encodings

2.2.1. Depth-first traversal. We denote a depth-first traversal of a tree by dfs. A substantial number of the results surveyed use a node numbering by a particular depth first traversal of a tree. A depth-first traversal of the tree is denoted $\mathrm{dfs_i}$ if children of small size[2] are visited before children of larger size. Given a tree $T = (V, E)$ every node $v \in V$ receives the number $\mathrm{dfs_i}(v)$ from $\mathrm{dfs_i}$, and we occasionally refer to it as the $\mathrm{dfs_i}$ identifier of v.

This seems like an unfortunate notation. The "i" is normally used for indexing.

2.2.2. Suffix-free codes. A *code* is a set of words,and a code is *suffix-free*[3], 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 $2^{|\mathrm{bin}(x)|} + 1$. We can extended this code to an efficient recursively constructed suffix-free code by $code_{i+1}(x) = \mathrm{bin}(x) \circ code_i(|bin(x)|-1)$ 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

....and what is a word?

[2]The *size* of a node in a tree is the number of its descendants.
[3]suffix-free codes are also known as suffix codes.

can be applied recursively up to $\log^* x$ times such that the length of $code_i(x)$ is at most $\lfloor \log x \rfloor + \lfloor \log \log x \rfloor + \cdots + O(\log^* x)$[4].

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 [Cover and Thomas 2012], 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 nodes of a rooted path that achieves the following two properties: First, nodes of a large size are assigned a small label, and second, the labels maintain a total order among the path's nodes.

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 2.2. *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 are 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 $\lfloor \log w_j \rfloor$ 0s in its least significant bits that can be discarded. As an example, the sequence $(w_i) = (2, 8, 1, 1, 4)$ with $w = 16$, may be represented by $0000, 1000, 1010, 1100, 1110$ and, accordingly, $(a_i) = (0, 1, 101, 11, 111)$. 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.

What does this last sentence have to do with all the rest?

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 has 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 node $v \in T$ is a member of exactly one path.

That seems a bit subjective. Do we want to niclude it?

2.3.1. Heavy-light decomposition. Harel and Tarjan [1984] show how to create a path-decomposition of a rooted tree where each node of a path, except its top-node[5], has maximum size among its siblings. The decomposition allows the location of a node in

[4] $\log^*$ is the number of times $\log$ should be iterated before the result is at most 1.
[5] The top-node of a path in a tree T rooted in r is the node closest to r.

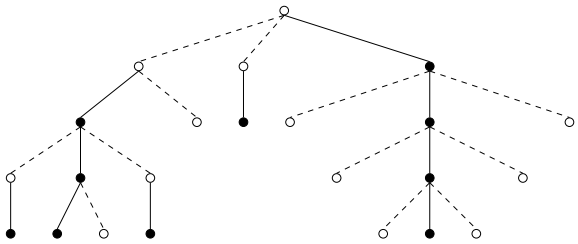

Fig. 3. A tree in which light and heavy nodes have been marked with "$\circ$" and "$\bullet$", respectively, and heavy and light edges have been drawn with solid and dashed lines, respectively.

the tree to be described by the sequence of paths that must be traversed in order to reach the node from the root.

Let T be a tree with root r. The nodes of T are classified as either *heavy* or *light* as follows. The root is light. For each internal node $v \in T$, pick one child w whose size is maximal among the children of v and classify it as heavy; classify the other children of v as light. We denote the unique heavy child of v by $\mathrm{hchild}(v)$ and the set of light children of v by $\mathrm{lchildren}(v)$. The *light size* of a node v is the number

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

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

An edge connecting a light node to its parent is a *light edge*, and an edge connecting a heavy node to its parent is a *heavy edge*. By removing the light edges, T is divided into a collection of *heavy paths*. The set of nodes on the same heavy path as v is denoted $\mathrm{hpath}(v)$. See Figure 3 for an example.

Given a node v in T, consider the list of light nodes $u_0 \dots u_k$ encountered on the path from the root r to v. The first of these light nodes is the root, $r = u_0$, and we denote the list $u_0 \dots u_k$ as $\mathrm{lpath}(v)$. The number k is the *light depth* of v, denoted $\mathrm{ldepth}(v)$. The light depth of T, $\mathrm{ldepth}(T)$, is the maximum light depth among the nodes in T. Since the size of every light node is bounded by the size of its heavy sibling, the size of u_{i+1} is at most half the size of u_i. From this, it follows that $\mathrm{ldepth}(v) \leq \lfloor \log n \rfloor$ for all nodes v, where n is the number of nodes in T.

Note that the results surveyed which use this path-decomposition perform also a $\mathrm{dfs_i}$ traversal of the tree (see Section 2.2.1).

2.3.2. Separator. Two sets of nodes in a graph are separated if no node in one is adjacent to any node in the other. A separator is typically defined as a subset of nodes in a graph G whose removal from G separates the graph into two subsets U and V such that $|U| \leq |V| < 2|U|$ [Chung 1989]. For all trees there exist a separator that consist of a single node.

THEOREM 2.3. *[Jordan 1869] Given a tree T rooted in node r, we can find, in linear time, a single node whose removal separates T into subtrees of at most $n/2$ nodes each.*

PROOF. Consider any node $v \in T$. 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 also holds naturally for forests. Korman and Peleg [2007] use Theorem 2.3 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 2.3 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 node $v \in V$ 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 node in all of the subtrees associated with v's ancestors in T^{sep}. For a demonstration, see Figure 4.

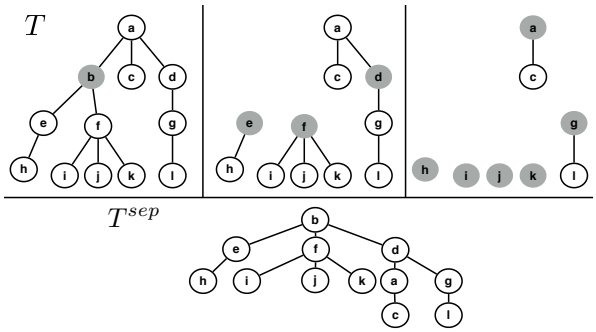

Fig. 4. Separator tree for the tree T, with separator nodes marked in grey. Top row: T rooted at a (left). The forest resulted by removing the separator b from T (center). The remaining forest after removing the separators e, f, d (right). Bottom row: the resulting T^{sep}.

2.3.3. Spines decomposition. This path-decomposition was invented by Thorup and Zwick [2001], and used by Fraigniaud and Korman [2010b]. Similarly to heavy-light decomposition, a *spines decomposition* decomposes an n node tree T into a collection of paths, using an additional parameter, an integer $1 \leq b \leq n$. A node v is $heavy_s$ if $size(v) \geq n/b$ and $light_s$ otherwise. Note that the heavy nodes in T induce a subtree rooted by the root of T, which is always a $heavy_s$ node. T_h is the subtree of T which spans the heavy nodes, 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 $P_1 \ldots P_l$, ($1 \leq l \leq 2b - 1$) which we denote as $heavy_s$ paths (some of which may consist of a single node). When $b = 2$, the single $heavy_s$ path is referred to as the *spine* of T.

Is this standard terminology? It seems a bit weird to me...

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 $light_s$ node v in T adjacent to a node in T_h is a root of a tree $T_v \in F$ of size $\text{size}(v) \leq n/b$. The decomposition stops when all nodes in T are $heavy_s$ nodes at some recursive step, and both node 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 nodes in T are $heavy_s$ nodes. See Figure 5 for a demonstration.

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 $O(x)$ nodes each, such that every cluster has at most two nodes in common with other clusters.

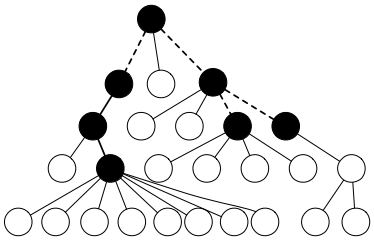

Fig. 5. The first step of a spines decomposition of T with $b = 6$ for a tree with $n = 24$ nodes. Black nodes are $heavy_s$ nodes, and the rest are $light_s$ nodes for the first step. Hatched lines are removed from T_h and the emphasised edges mark the $heavy_s$ paths.

Definition 2.4. Let T be a tree of size $n = |V(T)| > 1$. For a connected subtree C of T, we call a node in $V(C)$ incident with a node in $V(T) \setminus V(C)$ a *boundary node*. The boundary nodes 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 nodes u and v, where $depth(u) < depth(v)$. If a cluster has only one boundary node, we associate its rightmost leaf[6] as the second boundary node. 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 $r \in \delta C$.

The latter is unclear to me: doesn't this always hold for all nodes by definintion?

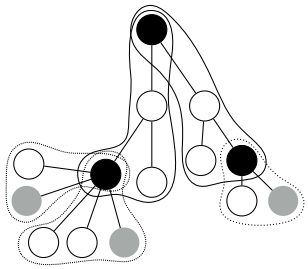

Fig. 6. A demonstration of a nice cluster partition with $n = 14$ and $x = 4$. The dotted clusters are leaf clusters and the rest are internal clusters. The black nodes are boundary nodes, and the grey nodes are the second boundary nodes added.

Definition 2.5. Given a tree T with $n > 1$ nodes rooted at r 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.

Shouldn't this definition depend on x? As in "x-nice"?

LEMMA 2.6. *For any tree T with n nodes 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 [Alstrup et al. 1997] (Lemma 12, Appendix A), and an illustration of the decomposition in Figure 6.

I think it is more than fair that we do not prove things here but just refer. But why do we then prove SOME things?

2.4. Boxes and groups

This section presents a single lemma, which is used throughout the survey to prove lower bounds on label sizes. Introduced by Alstrup, Bille and Rauhe [2005] and refined

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

by Dahlgaard et al. [2014b]. The technique uses a division into "boxes and groups" of the set that is to be labeled.

LEMMA 2.7. *Let X be a set with $|X|= nk$, where n is a power of 3 and $k \leq \log_3 n$. Further, let $e\colon X \to S$ be a function that labels the elements from X with labels from some set S. Assume that we have partitioned X into $k+1$ disjoint subsets of the same size:*

Should X have size $(k+1)n$ then?

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

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

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

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 $X_{prior} \cup X_b$. □

What is b? Don't you mean k?

3. *ANCESTRY*

We discuss labeling schemes for *Ancestry*. First, we describe a simple $2\log n$ labeling scheme for the function, followed by literature review in Section 3.1. We then present a recent labeling scheme of size $\log n + 2\log\log n$ in Section 3.2. The bound is matched asymptotically by a lower bound, presented in Section 3.3. Finally, in Section 3.4, we discuss dynamic *Ancestry* labeling schemes and present in detail a lower bound for a natural dynamic model.

Naïve algorithm. The following $2\log n$ *Ancestry* labeling scheme was introduced by Kannan et al. [1992]. 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 node $v \in T$, with $\mathcal{L}(v) = (\mathrm{dfs}(v), \mathrm{dfs}(w))$ 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 $\mathcal{L}(v) = (\mathrm{dfs}(v), \mathrm{dfs}(w))$ 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 for a demonstration of the labeling scheme.

3.1. Literature review

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

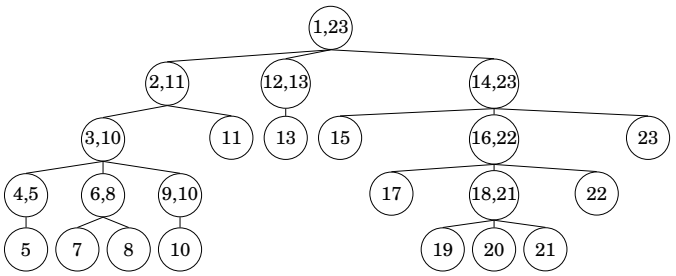

Fig. 7. A tree with $n = 23$ nodes. Each node is assigned a label of size $2 \log n$ supporting *Ancestry* queries.

labeling scheme is found in Section 3.2. *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 3.4.

The $2 \log n$ labeling scheme presented above was improved gradually. Abiteboul, Kaplan and Milo [2001] achieved an upper bound of $3/2 \log n + O(\log \log n)$ bits, which was improved to $\log n + O(\sqrt{\log n})$ [Abiteboul et al. 2006]. Shortly after, Alstrup and Bille [2005] constructed a lower bound of $\log n + \log \log n$ for any labeling scheme supporting the function. Fraigniaud and Korman [2010a] showed that $Trees(n, \delta)$ enjoy a labeling scheme of $\log n + O(\log \delta)$. The result was generalised in a follow up paper [Fraigniaud and Korman 2010b] for $Trees(n)$ in a labeling scheme of size $\log n + 4 \log \log n$, which is asymptotically optimal. The bound was improved last by Dahlgaard et al. [2014a] to $\log n + 2 \log \log n$. Interestingly, the asymptotically optimal labeling scheme is based on the first one presented eighteen years prior [Kannan et al. 1992].

In the case where *Ancestry* may be determined if the distance between the nodes is at most d, Alstrup et al. [2005] 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 [Kaplan et al. 2002; Wu et al. 2004; O'Neil et al. 2004; Li and Ling 2005; Lu et al. 2005; Härder et al. 2007; 2007; Xu et al. 2009; Cohen et al. 2010b; OConnor and Roantree 2012; Ghaleb and Mohammed 2013]. Typically, those address the dynamic variant thereof. For a short survey on dynamic *Ancestry* labeling schemes, see [Cohen et al. 2010b].

3.2. Upper bound

In this section we prove the following.

THEOREM 3.1. *There exist a* $\log n + 2 \log \log n + O(1)$ Ancestry *labeling scheme for* $Trees(n)$.

The naïve labeling scheme presented above stores a two part label (x_v, y_v) for every node v in a tree T that represent the interval $\{x_v \dots y_v\}$ of size $y_v - x_v + 1$. Both x_v and y_v may be any number in $\{1 \dots n\}$ and therefore the label size is at most $2 \log n$. The label can also be stored in the form $(x_v, y_v - x_v + 1)$ as demonstrated by the following pseudo code. The function computes a value a for every node $v \in T$ specifying the largest $x_{v'}$ for any v' in the subtree rooted in v. The function is initially called on the root r of T and $\mathrm{Id} = 1$.

```
1: function NAÏVE(node v, integer Id)
2:     a ← Id
3:     for all l ∈ children(v) do
4:         a ← NAÏVE(l, a + 1)
```

5: $v.label \leftarrow (\mathrm{Id}, (a - \mathrm{Id}))$
6: **return** a

To reduce the label size further, we do not store $y_v - x_v + 1$ explicitly. Instead, we use an approximation of that value that requires only $O(\log \log n)$ bits. The value x_v is now computed using an updated variant of the $\mathrm{dfs_i}$ traversal (Section 2.2). In particular, the order of the traversal remains the same. While $x_v \geq \mathrm{dfs_i}(v)$, it will still hold that $x_v = O(n)$.

As described in Section 2.1, we can store a $1 + \frac{1}{2^k}$ approximation of any number in the range $\{1 \dots n\}$ using $\log \log n + k$ bits. The following labeling scheme uses $1 + \frac{1}{2^k}$ approximation for carefully chosen k. For brevity, we now denote $\frac{1}{2^k}$ by ϵ, and call $\mathrm{Approx}(x)$ the function that returns a $1 + \epsilon$ approximation of an integer x. Notice that $\log \log x + \log \frac{1}{\epsilon}$ bits are required to store the result of the function.

The label of a node v in a rooted tree T consists of the following two parts $\mathcal{L}(v) = (\mathrm{Id}(v), \mathrm{Ap}(v))$. For every node $v \in T$ the function IMPROVED computes two values, a and b: in this a stands for the largest number assigned and b the largest number reserved in the subtree rooted in v. We initially call the recursive function using the root of T and $\mathrm{Id} = 1$.

1: **function** IMPROVED(node v, integer Id)
2: $(a, b) \leftarrow (\mathrm{Id}, \mathrm{Id})$
3: **for all** $l \in \mathrm{children}(v)$ in increasing ordered size **do**
4: $(a, b) \leftarrow$ IMPROVED$(l, b + 1)$
5: $v.label \leftarrow (\mathrm{Id}, \mathrm{Approx}(a - \mathrm{Id}))$
6: **return** $(a, \max\{b, \mathrm{Approx}(a - \mathrm{Id}) + \mathrm{Id}\})$

To prove that the encoder assigns labels of the requested size we first use lemma 3.2. Recall that $\mathrm{ldepth}(v)$ is the number of light nodes encountered on the path $r \rightsquigarrow v$, and that the $\mathrm{ldepth}(T)$ is the maximum light depth among the nodes in the tree T (Section 2.3.1). Recall also that a node v in a tree T roots the subtree $T(v)$ of size $\mathrm{size}(v)$.

LEMMA 3.2. *Let v be a node in a tree T of size $S = \mathrm{size}(v)$, x be an integer, and $l = \lfloor \log S \rfloor$. If $(a, b) =$ IMPROVED(v, x) then:*

(1) $a \leq x + S(1 + \epsilon)^l - 1$.
(2) $b \leq x + S(1 + \epsilon)^{l+1} - 1$.

PROOF. We prove the claims by induction over l, where the claims hold trivially for $l = 1$. Assume that both claims hold for $l - 1$ and we show that it holds for l. The children of v are denoted $w_1 \dots w_k$ according to their size, where w_k is the heavy node. The a and b values of node w_i are called a_i and b_i respectively. Since $w_1 \dots w_{k-1}$ are light nodes, $\mathrm{size}(w_1) \dots \mathrm{size}(w_{k-1}) < \frac{1}{2}S$, which implies that $\lfloor \log \mathrm{size}(w_i) \rfloor \leq l - 1$. Thus, by the induction hypothesis $b_1 \leq x + \mathrm{size}(w_1)(1 + \epsilon)^l - 1$. For both manners of assigning b_2 (line 6) it now holds that,

$$b_2 \leq b_1 + 1 + \mathrm{size}(w_2)(1 + \epsilon)^l - 1 \leq X + (\mathrm{size}(w_1) + \mathrm{size}(w_2))(1 + \epsilon)^l - 1.$$

Applying the argument to all b_i $3 \leq i \leq k - 1$ we get:

$$b_{k-1} \leq x + \sum_{i=1}^{k-1} \mathrm{size}(w_i)(1 + \epsilon)^l - 1$$

At this point we can use the first hypothesis to get:

$$a_k \leq x + \sum_{i=1}^{k-1} \text{size}(w_i)(1+\epsilon)^l + \text{size}(w_k)(1+\epsilon)^l$$

Which is simply $S(1+\epsilon)^l - 1$ As requested. The proof of the second argument is done similarly. □

Since $l = \lfloor \log S \rfloor \leq \log n$, by Lemma 3.2 the total interval required is of size at most $\lfloor n \cdot (1+\epsilon)^{\log n} \rfloor$. It follows that the size of the first label part is at most $\log n + \log n \cdot \log(1+\epsilon) + O(1)$ and the size of the second part is at most $\log\log n + \log \frac{1}{\epsilon} + O(1)$. Choosing $\epsilon = 1/\log n$, these are bounded by $\log n + O(1)$ and $2\log\log n + O(1)$, respectively.

The encoding is done similarly to the naïve decoder. Given labels $(\text{Id}(v), \text{Ap}(v))$ and $(\text{Id}(u), \text{Ap}(u))$ of nodes v and u, respectively, the node v is an ancestor of u if and only if $\text{Id}(v) \leq \text{Id}(u) \leq \text{Id}(v) + (1+\epsilon)^{\text{Ap}(v)}$. In this labeling scheme the interval of a node's decedent may be of equal size to the one used by the node. Therefore, unlike the naïve labeling scheme, the intervals are not necessarily fully contained.

3.3. Lower bound

The following lower bound is an extension of the one by Alstrup et al. [2005] using the same technique, namely, boxes and groups (Section 2.4).

THEOREM 3.3. *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 $\lceil \log n \rceil + \lfloor \log \lfloor \log \delta \rfloor \rfloor - 1$.*[7]

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 node to which $m/2^i$ paths of length 2^i have been attached. Thus, T_i has $m + 1 \leq n$ nodes, whereof m belong to disjoint paths. Further, T_i has depth $2^i \leq 2^k \leq \delta$, and hence T_i belongs to $Trees(n, \delta)$. For an illustration of such trees see Figure 8.

Each node in a tree must be uniquely labeled by an *Ancestry* labeling scheme. Further, if two nodes in T_i lie on distinct paths, then their labels cannot be used on the same path in T_j for any $j \neq i$, because nodes on the same path have an *Ancestry* relation whereas nodes on different paths do not. We can therefore apply Lemma 2.7, using the m nodes of the paths of each tree as a "box" and each path as a "group", and it follows that we need at least $\frac{1}{2}m(k+1) = \frac{1}{2}m(\lfloor \log \delta \rfloor + 1)$ 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 $L \geq \lceil \log n \rceil + \lfloor \log \lfloor \log \delta \rfloor \rfloor - 1$. □

3.4. Dynamic *Ancestry* labeling schemes

All results described thus far in the survey present a worst-case analysis of static trees. 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 nodes. For convenience it is assumed that the sequence constructs a rooted tree, and its root exist from the beginning and is never deleted.

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

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

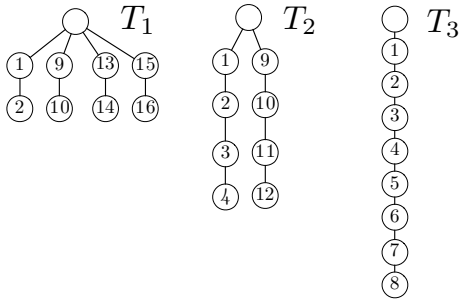

Fig. 8. Illustration of T_1, T_2, T_3 for $n = 8$.

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 node 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 [2002] 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 node with unique label.

Definition 3.4. 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 nodes previously adjacent to r to be adjacent to w'. In the same manner, r' "replaces" r in s_2. For illustration, see Figure 9.

For convenience, we denote the last node 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 node 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 3.5. *[Abiteboul et al. 2001] 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 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 node $last(s_1)$ 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 $\mathcal{L}(last(s_1)) \neq \mathcal{L}(last(s_2))$. □

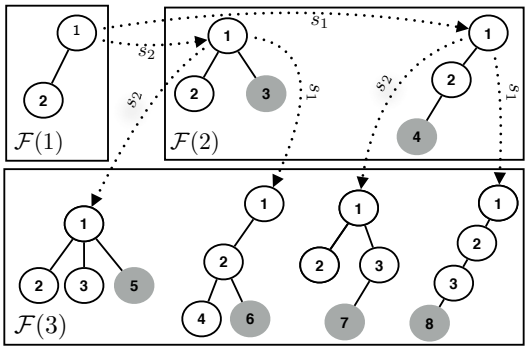

Fig. 9. An illustration of the trees resulting by the sequences $\mathcal{F}(1)$, $\mathcal{F}(2)$ and $\mathcal{F}(3)$ from Definition 3.4. The grey nodes are the ones introduced by Lemma 3.5. The arrows denote the extension of a sequence, and marked s_1 or s_2 according to the type.

From Lemma 3.5, 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 node in each sequence in $\mathcal{F}(n)$.

COROLLARY 3.6. *Every persistent* Ancestry *labeling scheme* $\langle e, d\rangle$ *is of size* $\Omega(n)$.

Using a similar argument, Cohen et al. [2002] 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 node 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. Dahlgaard et al. [2014b] showed that this bound applies for the functions *NCA*, *Distance*, and *Routing* as well. In contrast, they showed that $2\log n$ are sufficient and necessary in this model for the functions *Adjacency*, *Siblings* and *Connectivity*.

4. ADJACENCY

In this section we describe results relevant to the *Adjacency* function. We begin the section by a literature overview. Section 4.2 is dedicated to the current best bound. We begin it by showing two $\log n + O(\log\log n)$ *Adjacency* labeling schemes. Those are a stepping stone to an improved $\log n + O(\log^* n)$ labeling scheme, which is described in detail. Section 4.3 surveys the technique, "traversal and jumping", for *Adjacency* in caterpillars. In addition, we sketch a new optimal labeling scheme for bounded degree trees. Finally, in Section 4.4 we discuss the one-sided error variant.

4.1. Literature overview

Adjacency labeling schemes were introduced by Breuer and Folkman [1967], and revisited by Kannan et al. [1992] for trees and graphs. They were also independently defined by Muller [1988] 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 [2001] improved the label size to $1.5\log n + O(\log\log n)$. Alstrup and Rauhe [2002] proved that both $Forests(n)$ and $Trees(n)$ have an *Adjacency* labeling scheme of $\log n + O(\log^* n)$. Recently, Fraigniaud and Korman [2010a] showed that trees with bounded depth δ have a labeling scheme of size $\log n + 3\log\delta + O(1)$. Bonichon et al. [2006] proved that caterpillars and binary trees enjoy a labeling scheme

of size $\log(n) + O(1)$ using a method called "Traversal and jumping". Adjiashvili and Rotbart [2014] showed that trees with bounded degree Δ have a labeling scheme of size $\log n + \log \Delta + O(1)$. Fraigniaud and Korman [2009] showed that in a one-sided error labeling scheme, *Non-Adjacency* labeling schemes require $k+1$ bits to have a guarantee of $1 - \frac{1}{2^k}$.

Adjacency labeling schemes for trees are useful for general graphs due to an observation by Nash-Williams [1961], which states that a graph of arboricity[8] d can be decomposed into at most d forests[9]. Kannan et al. [1992] proved that graphs with arboricity d can be labeled using $(d+1)\log n$ bit labels. Therefore, the labeling scheme of Alstrup and Rauhe [2002] yields a label size of $d \log n + O(log^* n)$ for graphs with arboricity d.Graphs in $\mathcal{G}(n, \Delta)$ have arboricity of $\lceil \Delta/2 \rceil$ [Kannan et al. 1992]. It follows that graphs in $\mathcal{G}(n, \Delta)$ have a labeling scheme of $(\lceil \Delta/2 \rceil) \log n + O(log^* n)$ bits if Δ is constant.

4.2. Upper bound for $Trees(n)$

For general trees, Alstrup and Rauhe [2002] 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.

4.2.1. An Adjacency *labeling scheme of* $\log n + O(\log \log n)$ *bits.* We first introduce two labeling schemes for $Trees(n)$. The first favours equal size labeling scheme for all nodes, and the other achieves a smaller label size for leaves, at the expanse of bigger label size for internal nodes.

LEMMA 4.1. *[Alstrup and Rauhe 2002] 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 nodes, 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 nodes.*

Both labeling schemes are based on a heavy-light decomposition (Section 2.3.1) as well as a $\mathrm{dfs_i}$ 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 node 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.)*** $\mathrm{dfs_i}(v)$; ***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 $\mathrm{dfs_i}(h) - \mathrm{dfs_i}(v)$. Leaves are labeled similarly, with the last part set to 0, and the root is marked especially.

Alstrup and Rauhe [2002] prove that the information is sufficient to determine adjacency. For convenience, it is provided in App. B.

We create $\langle e_\beta, d_\beta \rangle$ by extending $\langle e_\alpha, d_\alpha \rangle$ such that each internal node 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 node 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 $\mathcal{L}(v) = (\mathrm{dfs_i}(v), 0)$ and $\mathcal{L}(u) = (\mathrm{dfs_i}(u) - \lfloor l \rfloor_2, 1)$. See Figure 10 for a demonstration. In conclusion, $\langle e_\beta, d_\beta \rangle$ produces labels of size $\log n + 4 \log \log n$ for internal nodes and $\log n + O(1)$ for leaves.

4.2.2. A $\log n + O(\log^* n)$ *labeling scheme.* 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

[8]The arboricity of a graph G is the minimum number of edge-disjoint acyclic subgraphs whose union is G.
[9]See also [Chen et al. 1994] for a simplified proof of the claim.

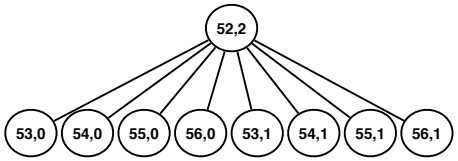

Fig. 10. An internal node with (simplified) label (52,2). The second number indicate that its $1/2$-approximation is $2^2 = 4$. The children are given labels in $\{53 \dots 56\}$ and an additional bit.

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 $O(x)$ nodes.

Definition 4.2. A cluster $C(u, v)$ (Def. 2.4) is a *single child cluster* if *I.*) it is a leaf cluster; or *II.*) it contains at most two nodes; or *III.*) $u \rightsquigarrow v$ contains at least 5 nodes, v has no children in $C(u, v)$, and u has only one, i.e., the node on the path to v. A nice cluster partition (Definition 2.5) is a *single child cluster partition* if all its clusters are single child clusters.

We complete an omitted proof of Lemma 7 in [Alstrup and Rauhe 2002].

LEMMA 4.3. *Let T be a rooted tree with n nodes. 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 $O(x)$ nodes.*

PROOF. Let $\mathcal{CS}$ be a nice cluster partition for T with $|\mathcal{CS}| \leq n/7x$ and each cluster contains $O(x)$ nodes. 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 nodes, 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 nodes. 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 nodes, and $u \rightsquigarrow v$ contains less than five, we decompose the cluster in the following manner (illustrated in Figure 11). An internal 2 node cluster $C(i, j)$ is created for every two adjacent nodes i and j on the path $u \rightsquigarrow v$. In addition, a leaf cluster $C(a, b)$ is created for every node 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 nodes in the tree rooted in a from $\mathcal{F}_d$, and $b \neq a$, an arbitrary node 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. □

The cluster partitioning guaranteed in Lemma 4.3 is used to create a correlating *macro tree*.

Definition 4.4. A cluster partition $\mathcal{CS}$ has a *macro tree* T_m defined as follows. The nodes $V(T_m)$ are the set of boundary nodes 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 node of some cluster in $\mathcal{CS}$.

THEOREM 4.5. *[Alstrup and Rauhe 2002] There exist an* Adjacency *labeling scheme for $Trees(n)$ whose worst-case label size is at most* $\log n + O(\log^* n)$.

PROOF SKETCH.
We begin by labeling the macro tree T_m (Def. 4.4) of a single child cluster partition of T,

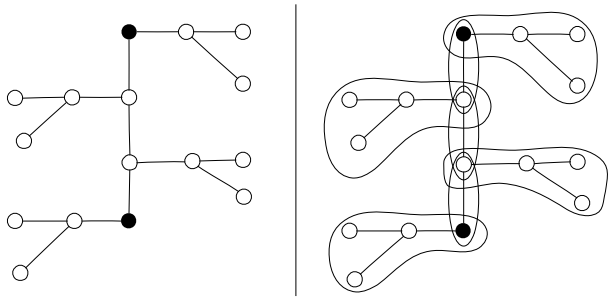

Fig. 11. Left: A nice cluster, Right: the cluster decomposed to 7 single child nice clusters. Black nodes are boundary nodes.

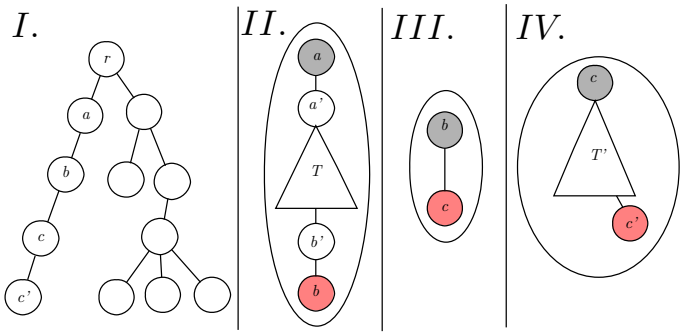

Fig. 12. From left to right: *I.*) A macro tree T_m with boundary nodes a, b, c, c'; *II.*) the internal cluster $C(a, b)$; *III.*) the internal cluster $C(b, c)$; and *IV.*) the leaf cluster $C(c, c')$. According to the labeling scheme the nodes in T rooted by a' store only the unique identifier of node b in T_m, rather than the entire $\mathcal{L}(b)$ assigned to the node in T_m. The last node in the $\mathrm{dfs_i}$ traversal of T is b'.

using the leaf favouring encoder e_β (defined in Lemma 4.1). Every edge $(u, v) \in E(T_m)$ represents a *micro tree*, a cluster of at most cx nodes, for some c, not including its boundary nodes. In addition, T_m has at most $n/x + 1$ nodes. We can use e_β to encode the leaves of T_m using $\log(n/x) + O(1)$ bits, and the inner node 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 $\mathrm{dfs_i}(u) < \mathrm{dfs_i}(v)$ is labeled in v's label, $\mathcal{L}(v)$.

Consider the nodes in the internal cluster $C(a, b) \setminus \{a, b\}$. These nodes maintain adjacency relation only between themselves and possibly the two boundary nodes 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 nodes from different clusters. Such a unique label is already given to every micro-tree labeled $\mathcal{L}(b)$ by $\mathrm{dfs_i}(b)$, which is just the first $\log(n/x)$ bits of the label given by e_β. For illustration, see Figure 12. We stress that the non-boundary nodes 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 now mark the boundary nodes, 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 nodes are the one exempt from storing this information. The decoder can detect if two nodes are in the same micro tree using the

first part, and check for *Adjacency* if the two nodes are in the same micro part, using the second part of the labels.

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

At this point the authors 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 node 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 nodes 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:

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

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

4.3. Traversal and jumping

Caterpillars were shown to have a labeling scheme of $\log n + O(1)$ by Bonichon et al. [2006], 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 nodes* $P = x_1 \dots x_p$ 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 nodes are adjacent in a caterpillar is either when one is a leaf child of the other or if they are adjacent internal nodes. 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 nodes in $L(i)$ before x_{i+1} for all $1 \le i \le p$. As with the labeling scheme in Section 4.2, 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 node's type.

We can construct a $2\log n$ *Adjacency* labeling scheme by assigning $\mathrm{dfs_i}(v)$ to every node v, and concatenate $l(i)$ to every internal node. An internal node 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 nodes 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

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

where $t_p = \log(\lceil l_p \rceil_2)$. An internal node $x_i \in P$ 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:

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

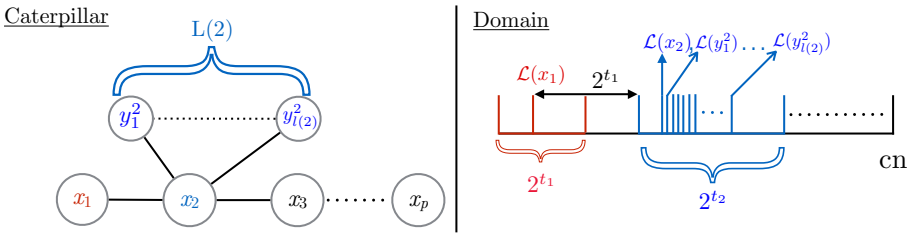

Fig. 13. Label assignment of the inner nodes and leaves of the caterpillar depicted on the left using traversal and jumping. The label $\mathcal{L}(x_1)$ is chosen from $\{1 \dots 2^{t_1}\}$, $\mathcal{L}(x_2)$ from $\{\mathcal{L}(x_1) + 2^{t_1} \dots \mathcal{L}(x_1) + 2^{t_1} + 2^{t_2}\}$ and the leaves $y_1^2 \dots y_{l(2)}^2 \in L_2$ are labeled s.t. $\mathcal{L}(y_1^2) = \mathcal{L}(x_2) + 1 \dots \mathcal{L}(y_{l(2)}^2) = \mathcal{L}(x_2) + l_2$.

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 2.1. 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': $\sum_{i=1}^{p} 2^{t_i} \leq c' \cdot \sum_{i=1}^{p} l(i)$. Since $\sum_{i=1}^{p} l(i) < n$, it follows that the maximum label size used by any internal node 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 13.

Optimal bounded degree. Recently, Adjiashvili and Rotbart [2014] proved the existence of a labeling scheme for $Trees(n, \Delta)$ with label size of $\log n + O(\log \Delta)$. The labeling scheme is based on an embedding technique of Bhatt, Chung, Leighton and Rosenberg [1989]. Bhatt et al. showed the existence of an induced universal graph with $O(\Delta \cdot n)$ nodes that contains the family $Trees(\Delta, n)$. Although the construction was explicit, it is unknown how to produce efficient decoder to match the possible encoding suggested by the construction.

Adjiashvili and Rotbart [2014] noticed that the simple construction in [Bhatt et al. 1989] (Lemma 3) can produce a $\log n + O(\Delta)$ labeling schemes for the family, where the label is a concatenation of a node and edge identifiers. More precisely, the label $\mathcal{L}(u)$ of node u with parent w, has two parts: u's unique number and the local number of the edge (u, w) in the universal graph. The major problem is to determine at which point in the binary string the first part stops and the second part starts. To that end the authors used the label size as an additional input for the decoder.

4.4. One-sided error *Adjacency* labeling scheme

Consider the function *Non-Adjacency* for any graph $G = (V, E)$ that returns true if and only if nodes $u, v \in V$ are *not* adjacent in G. Suppose we demand our query to be precise only when the nodes are in fact adjacent, and allow two non adjacent nodes to be declared adjacent. It is of course helpful to be able to predict and control how often such a "false positive" would occur. [Fraigniaud and Korman 2009] demonstrate the usefulness of one-sided error labeling scheme (Definition 1.6) for *Adjacency* in the following.

THEOREM 4.6. *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 node $v \in V$ 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 node 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 nodes 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 [2009] prove that any one-sided error *Adjacency* labeling scheme for $Trees(n)$ with guarantee p requires labels of size at least $\log n + \log p - O(1)$.

5. *NCA*

We now 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 node $v \in V$ as the *node identifier* of v, or simply $\mathrm{Id}(v)$. We can extend all labeling schemes presented so far to support node identifiers by modifying their encoder to concatenate the node identifier to each label. In the case of *NCA* such an extension is not as straightforward since it should return, for two nodes in the tree, the node identifier of a third node. We treat both variants, and denote labeling schemes for *NCA* specifically designed to support node identifiers as *Id-NCA*, and those that do not as *Label-NCA*.

In Section 5.1 we provide a literature overview and describe connections to related problems. In Section 5.2 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 5.3 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.

5.1. 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 [1973] were among the first to consider the problem of finding NCAs, and Harel and Tarjan [1984] 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. [2002] in the form of a *Label-NCA* labeling scheme of size $10 \log n + O(1)$. The labeling scheme was improved recently by Alstrup et al. [2014] 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 [2000a] showed that for *Id-NCA* labels of size $\Theta(log^2 n)$ are sufficient and required. Blin et al. [2010] 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 [Caminiti et al. 2009; Fischer 2009]. Korman and Kutten [2007] studied *Id-NCA* on a different model called *1-query*. In this model, the decoder has access to a *query* function, that gets two labels and returns a node 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* [Gabow et al. 1984]. 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 5.3.

5.2. *Id-NCA*

In this section, we present lower and upper bounds for *Id-NCA*.

5.2.1. NCA, SepLevel, *and their connection to* Distance. Peleg [2005] 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 5.1. *[Peleg 2005] 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 node 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 nodes u, v and w such that $w = \textit{NCA}(u, v)$. We transform the encoder e_{dist} by concatenating a prefix $\text{depth}(v)$ [10] 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 $\textit{Distance}(u, v) = \textit{Distance}(u, \textit{NCA}(u, v)) + \textit{Distance}(v, \textit{NCA}(u, v))$, and $\text{depth}(u) = \text{depth}(\textit{NCA}(u, v)) + \textit{Distance}(u, \textit{NCA}(u, v))$ (the same formula holds by replacing u with v). It follows that

$$\text{depth}(\textit{NCA}(u, v)) = \frac{\text{depth}(u) + \text{depth}(v) - \textit{dist}(u, v)}{2} = \textit{SepLevel}(u, v).$$

[10] Section 1.4.2): A node v in a tree $T = (V, E)$ rooted in r has $\text{depth}(v)$ edges on the path $r \rightsquigarrow v$.

We prove Claim 2 using the same transformation, i.e. adding $\mathrm{depth}(v)$ 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 $\mathrm{depth}(v)$ to the node identifier, adding exactly $\log n$ for each of the node identifiers. □

Distance labeling scheme has a lower bound of $\Omega(\log^2 n)$ on the size of the label [Gavoille et al. 2004]. 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 [Gavoille et al. 2004].

COROLLARY 5.2. *Any labeling scheme supporting* Id-NCA *for* $Trees(n)$ *must use labels of size* $\Omega(\log^2(n))$.

5.2.2. 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):
$\mathrm{hchild}(v)$ is the (unique) heavy child of v, $\mathrm{lchildren}(v)$ is the set of light children of v, $\mathrm{lsize}(v)$ is the weight of v not including the tree rooted in its heavy child, $\mathrm{lpath}(v)$ is the list of all light nodes in $r \rightsquigarrow v$, $\mathrm{ldepth}(v) = |\mathrm{lpath}(v)|$, $\mathrm{hpath}(v)$ is the set of nodes on the same heavy path as v.

THEOREM 5.3. *[Peleg 2005] 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 node $v \in V$ with light depth $\mathrm{ldepth}(v)$ has a unique predefined identifier, $\mathrm{Id}(v)$. The label of node $v \in T$ 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 $\mathrm{Id}(v)$, and an ancestry label as defined in Section 3, 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 $\langle \mathrm{Id}(l_i), \mathrm{Id}(n_i), d_i \rangle$, for $1 \leq i \leq ldepth(v)$, where l_i is the i'th light node in $\mathrm{lpath}(v)$, d_i is the depth of l_i, and n_i is the node 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 node $v \in T$, $|\mathcal{L}(v)| = |\mathcal{C}_b(v)| + |\mathcal{C}_a(v)| \leq 3(\log^2 n + \log n)$.
We describe the operation on two nodes $u, v \in T$ with NCA w. The decoder uses $\mathcal{C}_a$ to determine if one is an ancestor of the other, and if so returns its node 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 14 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 node 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 node, stored in the selected triplet. □

5.3. *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 5.3. Rather than storing the light path for every node along with the node above it, we store the light

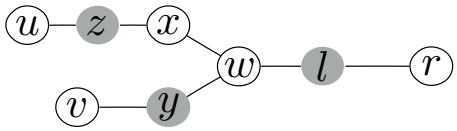

Fig. 14. The nearest common ancestor w=*NCA* (u,v) in a root rooted in r. The node l is the last light node in common in the paths $r \rightsquigarrow u$ and $r \rightsquigarrow v$, and y and z are the first light nodes immediately after l on those paths, where $depth(z) > depth(y)$. Straight lines represent edges, dashed lines are paths, and grey nodes are light nodes.

path along with the distance of the heavy paths between each two consecutive light nodes.

THEOREM 5.4. *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 node 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 node in $\mathrm{lpath}(v)$, and h_i is the (possibly null) length of the heavy path $\mathrm{hpath}(l_i)$, from l_{i-1} to l_i or to v when $i = \mathrm{ldepth}(v)$. See Figure 15 for an illustration.

Since $1 \leq h_i, l_i \leq n$, each tuple requires at most $2\log n$ bits. Since $|\mathrm{lpath}(v)| \leq \log n$ (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 nodes u and v with labels $\mathcal{L}(u) = h_1^u, l_1^u \ldots h_k^u, l_k^u$ and $\mathcal{L}(v) = h_1^v, l_1^v \ldots h_{k'}^v, l_{k'}^v$ respectively. Without loss of generality assume that $k \leq k'$. We find the first i for which $h_i^u, l_i^u \neq h_i^v, l_i^v$. If there is no such i, then u is the ancestor of v and we return its label. If $h_i^u = h_i^v$ we return the label $h_1^u, l_1^u \ldots h_i^u$. Otherwise, we return $h_1^u, l_1^u \ldots min\{h_i^u, h_i^v\}$.

□

Note that this labeling scheme can return the depth of the NCA, in other words the *SepLevel*. By Theorem 5.1, any labeling scheme supporting *SepLevel* must have labels of size $\Omega(\log^2 n)$. Moreover, using the formulas from Theorem 5.1 we observe that the function *Distance* may also be computed using these labels. The first labeling scheme for the function *Distance* [Peleg 2000b] 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 node's name. The one presented next utilises the ability further, and reduces the bound on the label size from $O(\log^2 n)$ to $O(\log n)$.

5.3.1. Label-NCA *with* $O(\log n)$ *bits.* The maximum label size in Theorem 5.4 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 nodes 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 nodes on a heavy path $h_1 \ldots h_k$ 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 [2002] use those two observations as well as Lemma 2.2 to construct a $O(\log n)$ *Label-NCA* labeling scheme presented below.

THEOREM 5.5. *[Alstrup et al. 2002] There exist a* Label-NCA *labeling scheme of size* $O(\log n)$.

PROOF SKETCH.
Consider a node v where $parent(v) = u$ in a tree T rooted in r with $\mathrm{ldepth}(v) = k$ and $\mathrm{lpath} = \{lp_1 \dots lp_k\}$. The label comprises the concatenation of the tuples $\langle h'_i, l'_i \rangle$. We first construct the labels given by Theorem 5.4 as auxiliary labels where $\langle h_i, l_i \rangle$ defined as in Theorem 5.4.

To compute each l'_i, we use Lemma 2.2 with $\mathrm{size}(lp_i)$ and the lsize of each of its *siblings* in T. The lemma provides l'_i a label appropriate to $\mathrm{size}(lp_i)$, and more precisely:

$$|l'_i| \leq \log \mathrm{size}(v) - \log \mathrm{lsize}(p(lp_i)) + 1,$$

where $p(lp_i)$ is the parent of lp_i.

To compute each h'_i we use Lemma 2.2 with the size of all nodes on the heavy path rooted in lp_i, $\mathrm{hpath}(v)$, 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 node on $\mathrm{hpath}(v)$. More precisely:

$$|h'_i| \leq \log \mathrm{size}(lp_i) - \log \mathrm{lsize}(v) + 1.$$

In contrast to the previous labeling scheme, both l'_i and h'_i have variable size. In order to distinguish the different parts in $\mathcal{L}(v) = h'_1, l'_1, \dots h'_k, l'_k$ we use a separating string (Section 2.1), which doubles the size of the label. By induction, it can be shown that $|h'_1|, |l'_1|, \dots |h'_k|, |l'_k| \leq \log n + 4k + 2$. The proof is similar to that of Lemma 6.3.

The decoder is computed similarly to the one defined in Theorem 5.4 with the exception that in the last case we use $\min_{<_{\mathrm{lex}}}\{h_i, h'_i\}$ instead of $\min\{h_i, h'_i\}$.

For proof of the induction, and the correctness of the decoder, see [Bistrup Halvorsen 2013]. □

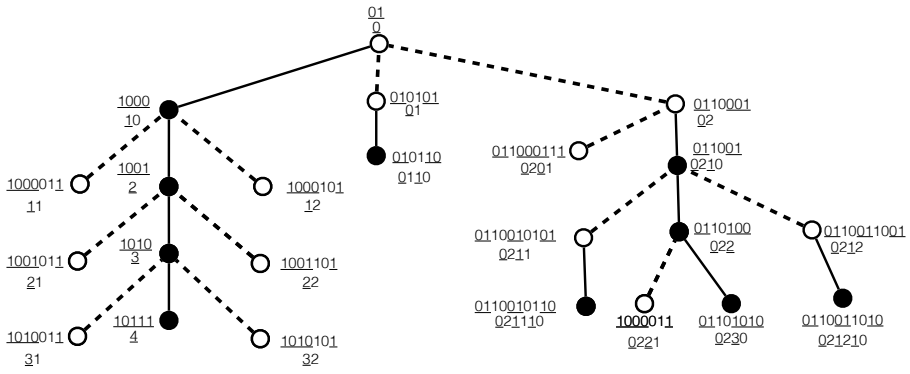

Fig. 15. A tree with heavy (black) and light (white) nodes marked in the labeling scheme from Theorem 5.5 in binary on top and Theorem 5.3 in decimal at the bottom. Both labels have their heavy sub-labels underlined.

The labels constructed in Theorem 5.5 can determine the first edge on the shortest path between the nodes queried. Therefore, using a slightly different decoder, we can construct a *Routing* labeling scheme (Section 6).

The label size achieved in this method is at most $10 \log n$. Alstrup et al. [2014] recently achieved an improved labeling scheme of size $3 \log n$ by replacing Lemma 2.2 with a more compact total order which allows for empty strings.

Labeling schemes with a query. Korman and Kutten [2007] 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 node $c \in V$. 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*.

6. *ROUTING*

We first describe general *Routing* schemes and two models for labeling schemes of *Routing* called *designer port model* and *fixed port model*. In Section 6.2 we provide a literature review. In Section 6.3 we describe in detail the current best upper bound for routing in the designer port model due to Thorup and Zwick [2001]. We provide a corrected proof for this result. In Section 6.4 we present a proof omitted in Thorup and Zwick [2001], that shows an efficient construction of a *Routing* scheme for the *fixed port* model.

6.1. 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 nodes in the network. Typically, such mechanism consist of *I.*) a *Routing* function *II.*) the format of the address; *III.*) a local labeling of the nodes; *IV.*) a message header format, and *V.*) local information to perform the computation of the *Routing* function [Fraigniaud and Gavoille 2001].

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 nodes on its route to the destination. Rather than supplying the entire path, a node provides only the next node to visit in order for the packet to arrive at its destination. The local labelings of edges from every node are identified by so called *port-numbers*.

Definition 6.1. Let $T = (V, E)$ be a tree with n nodes, and let $u \in V$ be a node 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 nodes. 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 nodes, while the latter assumes that the port numbers are fixed by an adversary.

Routing labeling schemes are related to two functions previously discussed in the survey, 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[11] 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 the *Routing* decoder and return true if its value is not 0. Finally, we note that the *Label-NCA* labeling scheme presented in Theroem 5.5 can answer designer port *Routing* queries, since the labels can determine the first edge on the shortest path between the nodes queried.

[11] A child with maximal number of decedents.

6.2. Literature review

Labeling schemes for *Routing* were introduced by Peleg [1999]. Fraigniaud and Gavoille [2001] achieved a labeling scheme of size $3\log n$ for the function. Thorup and Zwick [2001] 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 [2002] 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. [2007] compared the performance of several labeling schemes for both models. Korman and Peleg [2006; 2008], studied the function in a dynamic tree network settings, with permitted relabeling.

Unlike the situation in trees, there could be many paths between two nodes 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*[12]. Gavoille and Gengler [2001] showed a lower bound implying that achieving stretch 3 or less requires $\Omega(n)$ bits. Abraham et al. [2004] showed a stretch 3 *Routing* scheme with a $O(\sqrt{n})$ local information stored in each node. For surveys on *Routing* schemes see [Gavoille 2001], and [Dom 2007].

6.3. Designer port model

We present the proof for the following theorem.

THEOREM 6.2. *[Thorup and Zwick 2001] 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 nodes are on a $heavy_s$ path at some stage of the decomposition. In particular, a node v of size $\mathrm{size}(v)$ is a member of a $heavy_s$ path at level $\mathcal{LD}(v)$ if $\mathrm{size}(v) > n/b^{\mathcal{LD}(v)}$. The *level* number $\mathcal{LD}(v)$ is in the range $\{1 \dots \log_b n\}$ and stands for the number of iterations of the decomposition where v is a $light_s$ node.

Similarly to the definitions in Section 2.3.1, a node v in T with $light_s$ children $v_1 \dots v_d$ has ***light size*** $\mathrm{lsize}_s(v) = \sum_{i=1}^{d} size(v_i) + 1$. We also define the ***light size*** of a path $P = u(1) \rightsquigarrow u(k)$ as $\mathrm{lsize}_s(P) = \sum_{i=1}^{k} \mathrm{lsize}_s(u(i))$. The collection of paths in level i, in decreasing order of their light size, is denoted $P^i = P^i_1 \dots P^i_l$, and the light size of P^i is just the sum of their light sizes. The node with the k'th largest light size in the path with the j'th largest path size at level i is denoted $P^i_j[k]$. See Figure 16 for a demonstration.

A node 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 $1 \le i \le \mathcal{LD}(v) \le \log_b n$, defined as follows: The parts j_i, and k_i specify $P^i_j[k]$, the $heavy_s$ node on level i on the path $r \rightsquigarrow v$ of maximal depth. We use Lemma 2.2 to number both j_i and k_i. More specifically, we choose j_i according to the relative contribution of $\mathrm{lsize}_s(P^i_j)$ to $\mathrm{lsize}_s(P^i)$, such that $|j_i| = \log(\mathrm{lsize}_s(P^i)/\mathrm{lsize}_s(P^i_j)) + 1$. Similarly, we choose k_i according to the relative contribution of $\mathrm{lsize}(P^i_j[k])$ to $\mathrm{lsize}_s(P^i_j)$. The part p_i is the port number used from $P^i_j[k]$ to the node v, which is also assigned using Lemma 2.2 with $\mathrm{size}(p_i)$ and $\mathrm{lsize}(P^i_j[k])$. Note that for the last triplet in $\mathrm{Id}(v)$ we use the empty string for $p_{\mathcal{LD}(v)}$. As in Section 6.1, the ports to the root and the $heavy_s$ child are marked 0 and

[12]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 node and the destination node.

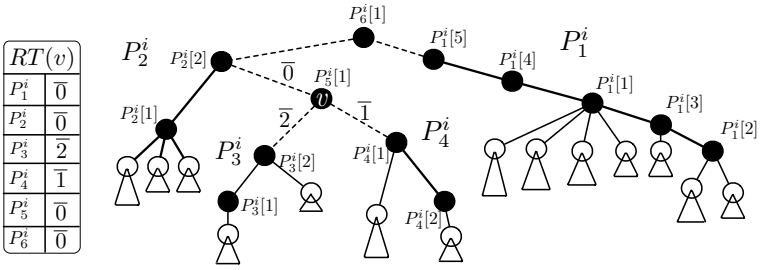

Fig. 16. A demonstration of naming of the paths and $heavy_s$ nodes on level i of a spines decomposition of a subtree. The thick lines are $heavy_s$ paths. The $heavy_s$ nodes are full, the $light_s$ nodes are empty, and the triangles are represent subtrees of $light_s$ nodes. The routing table of node v to the left describes which one of the ports $\bar{0}, \bar{1}$ and $\bar{2}$ it needs to traverse to arrive at nodes at other $heavy_s$ paths on level i. The i'th triplet in $\mathcal{L}(v)$ is $\langle 5, 1, \epsilon \rangle$.

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 $P_1^{\mathcal{LD}(v)} \dots P_l^{\mathcal{LD}(v)}$ in v's level $\mathcal{LD}(v)$. If v is not the last node 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 node, 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 6.3. $\sum_{i=1}^{\log_b n} |j_i| + |k_i| + |p_i| = \log n + O(\log_b n)$.

PROOF. Let $\langle j_1, k_1, p_1 \rangle$ be the first triplet in $Id(v)$ for a node 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:

$$|j_1| \leq \log(n/\mathrm{lsize}_s(P_j^1)) + O(1) = \log n - \log(\mathrm{lsize}_s(P_j^1)) + O(1),$$

and also

$$|k_1| \leq \log(\mathrm{lsize}_s(P_j^1)/\mathrm{lsize}_s(P_j^1[k])) + O(1) = \log(\mathrm{lsize}_s(P_j^1)) - \log(\mathrm{lsize}_s(P_j^1[k])) + O(1).$$

Similarly,

$$|p_1| \leq \log(\mathrm{lsize}_s(P_j^1[k])/\mathrm{size}(u)) + O(1) = \log(\mathrm{lsize}_s(P_j^1[k])) - \log(\mathrm{size}(u)) + O(1).$$

Summing those inequalities we get:

$$|j_1| + |k_1| + |p_1| \leq \log n - \log(\mathrm{size}(u)) + O(1),$$

as requested.

Since $\mathrm{lsize}_s(P^2) \leq \mathrm{size}(u)$, we can repeat the argument so that for w, the light child of $P_j^2[k]$ addressed by port p_2 we have:

$$|j_2| + |k_2| + |p_2| \leq \log(\mathrm{size}(u)) - \log(\mathrm{size}(w)) + O(1).$$

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 6.4. *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 $\mathrm{Id}(v)$ 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 6.3 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 $\mathrm{Id}(v)$ by $c_1 \dots c_{3k}$ Next we bound $\sum_{i=1}^{k} \log c_i$, where $\sum_{i=1}^{k} c_i \leq \log n + O(k)$ by Lemma 6.3. 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 nodes, 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 node in its $heavy_s$ path, the order between two nodes on the same $heavy_s$ path can be determined.

Let v,u be two nodes 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 $(path_v, node_v, edge_v)$ and $(path_u, node_u, \varepsilon)$ 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 $node_v = node_u$ return $edge_v$.
— If $path_v = path_u$ and $node_v \neq node_u$, then determine which of $node_v$ and $node_u$ 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 6.3 is a correction to Lemma 2.4 in the original proof in [Thorup and Zwick 2001]. Using similar definitions, the Lemma argues that every triplet requires at most $\log b + 2$ bits. The term $\text{lsize}_s(\overline{v})$ represents the size of the subtree rooted in a $light_s$ node at any level including the first, and therefore $1 \leq \text{size}(\overline{v}) \leq n/b$. From that we get that $\log_2 \frac{n}{\text{lsize}_s(\overline{v})} + 2 > \log_2 b + 2$. In fact the claim $\log_2 \frac{n}{\text{lsize}_s(\overline{v})} + 2 \leq \log_2 b + 2$ in [Thorup and Zwick 2001] holds in the opposite direction.

Implications. From this upper bound and the recent lower bound for *NCA* [Alstrup et al. 2014] 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.

6.4. 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 [Thorup and Zwick 2001] that was omitted in their paper.

THEOREM 6.5. *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 node $v \in T$ with label $(\text{Id}(v), RT(v))$ 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 $\text{Id}(v)$. 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 $\text{Id}(v)$ 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 Theorem 6.4 is asymptotically optimal in light of the lower bound in [Fraigniaud and Gavoille 2002]. An alternative proof for the existence of such a labeling scheme is given in [Fraigniaud and Gavoille 2001].

REFERENCES

Serge Abiteboul, Stephen Alstrup, Haim Kaplan, Tova Milo, and Theis Rauhe. 2006. Compact Labeling Scheme for Ancestor Queries. *SIAM J. Comput.* 35, 6 (June 2006), 1295–1309. DOI:http://dx.doi.org/10.1137/S0097539703437211

Serge Abiteboul, Haim Kaplan, and Tova Milo. 2001. Compact labeling schemes for ancestor queries. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms (SODA '01)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 547–556. http://dl.acm.org/citation.cfm?id=365411.365529

Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. 2010. A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks. (2010).

Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. 2012. Hierarchical hub labelings for shortest paths. In *Algorithms–ESA 2012*. Springer, 24–35.

Ittai Abraham, Cyril Gavoille, Andrew V Goldberg, and Dahlia Malkhi. 2006. Routing in networks with low doubling dimension. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*. IEEE, 75–75.

Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. 2004. Compact name-independent routing with minimum stretch. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 20–24.

Ittai Abraham and Dahlia Malkhi. 2005. Name independent routing for growth bounded networks. In *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 49–55.

David Adjiashvili and Noy Rotbart. 2014. Labeling schemes for bounded degree graphs. In *Automata, Languages, and Programming*. Springer, 375–386.

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. 1973. 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, 253–265. DOI:http://dx.doi.org/10.1145/800125.804056

Stephen Alstrup, Philip Bille, and Theis Rauhe. 2005. Labeling Schemes for Small Distances in Trees. *SIAM J. Discret. Math.* 19, 2 (June 2005), 448–462. DOI:http://dx.doi.org/10.1137/S0895480103433409

Stephen Alstrup, Esben Bistrup Halvorsen, and Kasper Green Larsen. 2014. Near-optimal labeling schemes for nearest common ancestors. In *Proceedings of the thirty first annual ACM-SIAM symposium on Discrete algorithms (SODA '14)*.

Stephen Alstrup, Søren Dahlgaard, and Mathias Bæk Tejs Knudsen. 2015. Optimal induced universal graphs and adjacency labeling for trees. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*. IEEE, 1311–1326.

Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe. 2002. 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, 258–264.

Stephen Alstrup, Peter Laurdisen, Peter Sommerland, and Mikkel Thorup. 1997. Finding cores of limited length. *Lecture notes in computer science* (1997), 45–54.

Stephen Alstrup and Theis Rauhe. 2002. 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, 53–62. http://dl.acm.org/citation.cfm?id=645413.652151

Sandeep Bhatt, Fan Rong Graham Chung, Thomson Leighton, and Arnold Rosenberg. 1989. Universal Graphs for Bounded-Degree Trees and Planar Graphs. *SIAM Journal on Discrete Mathematics* 2, 2 (1989), 145–155.

Esben Bistrup Halvorsen. 2013. *Graph labeling schemes*. Master's thesis. University of Copenhagen, Universitetsparken 5, Copenhagen.

Lélia Blin, Shlomi Dolev, Maria Gradinariu Potop-Butucaru, and Stephane Rovedakis. 2010. Fast Self-Stabilizing Minimum Spanning Tree Construction. In *Distributed Computing*. Springer, 480–494.

Nicolas Bonichon, Cyril Gavoille, and Arnaud Labourel. 2006. 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, 143–156. DOI:http://dx.doi.org/10.1007/11780823_12

Melvin A. Breuer and John Folkman. 1967. An unexpected result in coding the vertices of a graph. *J. Math. Anal. Appl.* 20 (1967), 583–600. Issue 3. DOI:http://dx.doi.org/10.1016/0022-247X(67)90082-0

Saverio Caminiti, Irene Finocchi, and Rossella Petreschi. 2009. Informative Labeling Schemes for the Least Common Ancestor Problem.. In *ICTCS*. 66–70.

Boliong Chen, Makoto Matsumoto, Jianfang Wang, Zhongfu Zhang, and Jianxun Zhang. 1994. A short proof of Nash-Williams' theorem for the arboricity of a graph. *Graphs and Combinatorics* 10, 1 (1994), 27–28.

Fan RK Chung. 1989. *Separator theorems and their applications*. Forschungsinst. für Diskrete Mathematik.

Edith Cohen, Haim Kaplan, and Tova Milo. 2002. 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, 271–281. DOI:http://dx.doi.org/10.1145/543613.543648

Edith Cohen, Haim Kaplan, and Tova Milo. 2010a. Labeling dynamic XML trees. *SIAM J. Comput.* 39, 5 (2010), 2048–2074.

Edith Cohen, Haim Kaplan, and Tova Milo. 2010b. Labeling dynamic XML trees. *SIAM J. Comput.* 39, 5 (2010), 2048–2074.

Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. 2001. *Introduction to Algorithms* (2nd ed.). McGraw-Hill Higher Education.

Bruno Courcelle, Cyril Gavoille, M Kanté, and Andrew Twigg. 2007. Forbidden-set labeling on graphs. In *2nd Workshop on Locality Preserving Distributed Computing Methods (LOCALITY)", Co-located with PODC*.

Thomas M Cover and Joy A Thomas. 2012. *Elements of information theory*. John Wiley & Sons.

Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Noy Rotbart. 2014a. private communication. (2014).

Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Noy Rotbart. 2014b. Dynamic and Multi-functional Labeling Schemes. *arXiv preprint arXiv:1404.4982* (2014).

Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Noy Rotbart. 2014c. Improved ancestry labeling scheme for trees. *arXiv preprint arXiv:1407.5011* (2014).

Michael Dom. 2007. Compact Routing. In *Algorithms for Sensor and Ad Hoc Networks*, Dorothea Wagner and Roger Wattenhofer (Eds.). Lecture Notes in Computer Science, Vol. 4621. Springer Berlin Heidelberg, 187–202. DOI:http://dx.doi.org/10.1007/978-3-540-74991-2_10

Johannes Fischer. 2009. Short Labels for Lowest Common Ancestors in Trees. In *ESA*. 752–763.

Pierre Fraigniaud and Cyril Gavoille. 2001. Routing in Trees. In *IN 28 TH INTERNATIONAL COLLOQUIUM ON AUTOMATA, LANGUAGES AND PROGRAMMING (ICALP*. Springer, 757–772.

Pierre Fraigniaud and Cyril Gavoille. 2002. A space lower bound for routing in trees. In *STACS 2002*. Springer, 65–75.

Pierre Fraigniaud and Amos Korman. 2009. On randomized representations of graphs using short labels. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*. ACM, 131–137.

Pierre Fraigniaud and Amos Korman. 2010a. Compact Ancestry Labeling Schemes for XML Trees. In *In Proc. 21st ACM-SIAM Symp. on Discrete Algorithms (SODA*.

Pierre Fraigniaud and Amos Korman. 2010b. An optimal ancestry scheme and small universal posets. In *Proceedings of the 42nd ACM symposium on Theory of computing (STOC '10)*. ACM, New York, NY, USA, 611–620. DOI:http://dx.doi.org/10.1145/1806689.1806773

Ofer Freedman, Paweł Gawrychowski, Patrick K Nicholson, and Oren Weimann. 2016. Optimal Distance Labeling Schemes for Trees. *arXiv preprint arXiv:1608.00212* (2016).

Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. 1984. 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, 135–143. DOI:http://dx.doi.org/10.1145/800057.808675

Cyril Gavoille. 2001. Routing in distributed networks: Overview and open problems. *ACM SIGACT News* 32, 1 (2001), 36–52.

Cyril Gavoille and Marc Gengler. 2001. Space-efficiency for routing schemes of stretch factor three. *J. Parallel and Distrib. Comput.* 61, 5 (2001), 679–687.

Cyril Gavoille, Michal Katz, Nir A. Katz, Christophe Paul, and David Peleg. 2000. Approximate Distance Labeling Schemes (Extended Abstract). (2000).

Cyril Gavoille and Arnaud Labourel. 2007a. Distributed relationship schemes for trees. *Algorithms and Computation* (2007), 728–738.

Cyril Gavoille and Arnaud Labourel. 2007b. Shorter implicit representation for planar graphs and bounded treewidth graphs. In *Algorithms–ESA 2007*. Springer, 582–593.

Cyril Gavoille and Christophe Paul. 2001. Split decomposition and distance labelling: an optimal scheme for distance hereditary graphs. *Electronic Notes in Discrete Mathematics* 10 (2001), 117–120.

Cyril Gavoille and David Peleg. 2003. Compact and localized distributed data structures. *Distrib. Comput.* 16, 2-3 (Sept. 2003), 111–120. DOI:http://dx.doi.org/10.1007/s00446-002-0073-5

Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. 2001. 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, 210–219. http://dl.acm.org/citation.cfm?id=365411.365447

Cyril Gavoille, David Peleg, Stéphane Pérennesc, and Ran Razb. 2004. Distance labeling in graphs. *Journal of Algorithms* 53 (2004), 85–112.

Cyril Gavoille and Stéphane Pérennès. 1996. Memory requirement for routing in distributed networks. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*. ACM, 125–133.

Taher Ahmed Ghaleb and Salahadin Mohammed. 2013. Novel scheme for labeling XML trees based on bits-masking and logical matching. In *Computer and Information Technology (WCCIT), 2013 World Congress on*. IEEE, 1–5.

Theo Härder, Michael Haustein, Christian Mathis, and Markus Wagner. 2007. Node labeling schemes for dynamic XML documents reconsidered. *Data & Knowledge Engineering* 60, 1 (2007), 126–149.

Dov Harel and Robert Endre Tarjan. 1984. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM J. Comput.* 13, 2 (1984), 338–355.

Camille Jordan. 1869. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik* 70 (1869), 185–190.

Sampath Kannan, Moni Naor, and Steven Rudich. 1992. Implicit Representation of Graphs. In *SIAM Journal On Discrete Mathematics*. 334–343.

Ming-Yang Kao, Xiang-Yang Li, and Weizhao Wang. 2007. Average case analysis for tree labelling schemes. *Theoretical Computer Science* 378, 3 (2007), 271 – 291. DOI:http://dx.doi.org/10.1016/j.tcs.2007.02.066 Algorithms and Computation.

Haim Kaplan, Tova Milo, and Ronen Shabo. 2002. 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, 954–963.

Michal Katz, Nir A Katz, Amos Korman, and David Peleg. 2004. Labeling schemes for flow and connectivity. *SIAM J. Comput.* 34, 1 (2004), 23–40.

Amos Korman. 2008. Improved compact routing schemes for dynamic trees. In *PODC '08*. ACM, 185–194.

Amos Korman and Shay Kutten. 2007. Labeling schemes with queries. In *Proceedings of the 14th international conference on Structural information and communication complexity (SIROCCO'07)*. Springer-Verlag, Berlin, Heidelberg, 109–123. http://dl.acm.org/citation.cfm?id=1760631.1760645

Amos Korman and David Peleg. 2006. Dynamic routing schemes for general graphs. *Automata, Languages and Programming* (2006), 619–630.

Amos Korman and David Peleg. 2007. Compact separator decompositions in dynamic trees and applications to labeling schemes. In *Distributed Computing*. Springer, 313–327.

Amos Korman, David Peleg, and Yoav Rodeh. 2010. Constructing Labeling Schemes through Universal Matrices. *Algorithmica* 57, 4 (Aug. 2010), 641–652. DOI:http://dx.doi.org/10.1007/s00453-008-9226-7

Dmitri Krioukov, Kevin Fall, Arthur Brady, and others. 2007. On compact routing for the Internet. *ACM SIGCOMM Computer Communication Review* 37, 3 (2007), 41–52.

Dmitri Krioukov, Kevin Fall, and Xiaowei Yang. 2004. Compact routing on Internet-like graphs. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, Vol. 1. IEEE.

Moshe Lewenstein, J Ian Munro, and Venkatesh Raman. 2013. Succinct data structures for representing equivalence classes. In *Algorithms and Computation*. Springer, 502–512.

Changqing Li and Tok Wang Ling. 2005. 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, 501–508.

Jiaheng Lu. 2013. XML Labeling Scheme. In *An Introduction to XML Query Processing and Keyword Search*. Springer, 9–32.

Jiaheng Lu, Tok Wang Ling, Chee-Yong Chan, and Ting Chen. 2005. 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, 193–204.

John Harold Muller. 1988. *Local structure in graph classes*. Ph.D. Dissertation. Georgia Institute of Technology, Atlanta, GA, USA. Order No: GAX88-11342.

Crispin Nash-Williams. 1961. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society* 1, 1 (1961), 445–450.

Martin F OConnor and Mark Roantree. 2012. SCOOTER: A Compact and Scalable Dynamic Labeling Scheme for XML Updates. In *Database and Expert Systems Applications*. Springer, 26–40.

Patrick O'Neil, Elizabeth O'Neil, Shankar Pal, Istvan Cseri, Gideon Schaller, and Nigel Westbury. 2004. ORDPATHs: insert-friendly XML node labels. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 903–908.

David Peleg. 1999. Proximity-preserving labeling schemes and their applications. In *Graph-Theoretic Concepts in Computer Science*. Springer, 30–41.

David Peleg. 2000a. Informative labeling schemes for graphs. In *Mathematical Foundations of Computer Science 2000*. Springer, 579–588.

David Peleg. 2000b. Proximity-preserving labeling schemes. *Journal of Graph Theory* 33, 3 (2000), 167–176. DOI:http://dx.doi.org/10.1002/(SICI)1097-0118(200003)33:3⟨167::AID-JGT7⟩3.0.CO;2-5

David Peleg. 2005. Informative labeling schemes for graphs. *Theor. Comput. Sci.* 340, 3 (Aug. 2005), 577–593. DOI:http://dx.doi.org/10.1016/j.tcs.2005.03.015

Richard Rado. 1964. Universal graphs and universal functions. *Acta Arithmetica* 9, 4 (1964), 331–340.

Mikkel Thorup and Uri Zwick. 2001. Compact routing schemes. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures (SPAA '01)*. ACM, New York, NY, USA, 1–10. DOI:http://dx.doi.org/10.1145/378580.378581

Xiaodong Wu, Mong Li Lee, and Wynne Hsu. 2004. A prime number labeling scheme for dynamic ordered XML trees. In *Data Engineering, 2004. Proceedings. 20th International Conference on*. IEEE, 66–78.

Liang Xu, Tok Wang Ling, Huayu Wu, and Zhifeng Bao. 2009. DDE: from dewey to a fully dynamic XML labeling scheme. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 719–730.

Online Appendix to: A Survey on Labeling Schemes for Trees

Stephen Alstrup, Esben Bistrup Halvorsen and Noy Rotbart

A. BASIC DEFINITIONS

A.1. Sets and Operations

Definition A.1. Let A and B be languages. We define the regular operations *union*, *intersection*, and *star* as follows:

(1) Union: $A \cup B = \{x | x \in A \text{ or } x \in B\}$.
(2) Intersection: $A \cap B = \{x | x \in A \text{ and } x \in B\}$.
(3) Star: $A^* = \{x_1 x_2 \dots x_k | k \geq 0 \text{ and each } x_i \in A\}$.

A.2. Universal graphs

The definitions are from Kannan et al. [1992].

Definition A.2. A node induced subgraph or simply an induced subgraph G' of G is a node set $V' \subseteq V$ together with the edge set $E' = E \cap (V' \times V')$.

Definition A.3. Given finite set of graphs $\mathcal{G}$, a graph G is node induced universal for $\mathcal{G}$ if every graph in $\mathcal{G}$ is a node induced subgraph of G. We say that a family $\mathcal{F}$ has universal graphs of size $g(n)$ if for every n there is a graph of size less than or equal to $g(n)$ which is node induced universal for the set of all graphs in $\mathcal{F}$ with n or fewer vertices.

A.3. Probability

Definition A.4. The definitions are cited from Cormen et al. [2001].

A sample space S is a set of *elementary events*. An *event* is a subset of the sample space S. Events A and B are mutually exclusive if $A \cap B = \emptyset$ A *Probability distribution* $prob()$ on a sample space S is a mapping from events of S to real numbers satisfying the following *probability axioms*:

(1) $prob(A) \geq 0$ for any event A.
(2) $prob(S) = 1$
(3) $prob(A \cup B) = prob(A) + prob(B)$ for any two mutually exclusive events A and B.

A random variable is a function from S into the real numbers.

For a random variable X and a real number x, we define the event $X = x$ to be $\{s \in S : X(s) = x\}$; thus,

$$prob(X = x) = \sum_{s \in S : X(s) = x} prob(s)$$

The expected value (or, synonymously, expectation or mean) of a discrete random variable X is:

$$EX = \sum_{x} x \cdot prob(X = x)$$

DOI: http://dx.doi.org/10.1145/0000000.0000000

B. PROOF OF THE CLAIM IN LEMMA 4.1

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 :

$$\mathrm{dfs_i}(v) - \mathrm{dfs_i}(u) \geq \mathrm{dfs_i}(v) - \mathrm{dfs_i}(y) + \mathrm{dfs_i}(h(u)) - \mathrm{dfs_i}(u) \geq 2min(\lfloor C \rfloor_2(u), \lfloor P \rfloor_2(v)). \quad (2)$$

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 2 holds as before, and the contradiction holds similarly.

C. PROOF OF THE CLAIM IN SECTION 4.3

The proof follows directly from the following lemma.

LEMMA C.1. *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:

$$\sum_{i=1}^{p} 2^{t_i} \leq c \sum_{i=m-1}^{p} l(i) - 2^{t_{m-1}}.$$