Labeling Schemes for Dynamic Tree Networks

Amos Korman * David Peleg * † Yoav Rodeh *

September 5, 2006

Abstract

Distance labeling schemes are composed of a *marker* algorithm for labeling the vertices of a graph with short labels, coupled with a *decoder* algorithm allowing one to compute the distance between any two vertices directly from their labels (without using any additional information). As applications for distance labeling schemes concern mainly large and dynamically changing networks, it is of interest to study *distributed dynamic* labeling schemes. The current paper considers the problem on dynamic trees, and proposes efficient distributed schemes for it.

The paper first presents a labeling scheme for distances in the dynamic tree model, with amortized message complexity $O(\log^2 n)$ per operation, where n is the size of the tree at the time the operation takes place. The protocol maintains $O(\log^2 n)$ bit labels. This label size is known to be optimal even in the static scenario.

A more general labeling scheme is then introduced for the dynamic tree model, based on extending an existing *static* tree labeling scheme to the dynamic setting. The approach fits a number of natural tree functions, such as distance, separation level and flow. The main resulting scheme incurs an overhead of a $O(\log n)$ multiplicative factor in both the label size and amortized message complexity in the case of *dynamically growing* trees (with no vertex deletions). If an upper bound on n is known in advance, this method yields a different tradeoff, with an $O(\log^2 n/\log\log n)$ multiplicative overhead on the label size but only an $O(\log n/\log\log n)$ overhead on the amortized message complexity. In the fully-dynamic model the scheme incurs also an increased *additive* overhead in amortized communication, of $O(\log^2 n)$ messages per operation.

*Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: {pandit,peleg,yrodeh}@wisdom.weizmann.ac.il.

†Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: peleg@wisdom.weizmann.ac.il. Supported in part by a grant from the Israel Science Foundation.

1 Introduction

Network representations play an extensive role in the areas of distributed computing and communication networks. Their goal is to cheaply store useful information about the network and make it readily and conveniently accessible. This is particularly significant when the network is large and geographically dispersed, and information about its structure must be accessed from various local points in it.

The current paper deals with a network representation method based on assigning *informative labels* to the vertices of the network. Whereas most traditional network representations rely on maintaining a *global* description of the network, the informative localized labeling schemes studied here are based on associating a label with each vertex, allowing us to infer information about any two vertices *directly* from their labels, without using *any* additional information sources. Hence the entire representation is based on the set of labels alone. Naturally, our focus is on labeling schemes using relatively *short* labels (say, of length poly-logarithmic in n). Labeling schemes of this type have been developed for a variety of information types, including vertex adjacency [Bre66, BF67, KNR88], distance [Pel99, KKP00, GPPR01, GKKPP01, GP01a, KM01, T01, CHKZ02], tree routing [FG01, TZ01], flow and connectivity [KKKP02], tree ancestry [AKM01, KM01a, AGKR01, AR02, KMS02], and various other tree functions, such as center, least common ancestor, separation level, and Steiner weight of a given subset of vertices [Pel00]. See [GP01b] for a survey.

These studies provide a reasonable understanding of informative localized labeling schemes and their basic combinatorial properties for *static* (fixed topology) networks. However, they are somewhat limited when it comes to handling applications such as distributed systems and communication networks. First, in these application areas the typical setting is that of a dynamic network, whose topology undergoes repeated changes. Therefore, for a representation scheme to be practically useful, it should be capable of reflecting online the current up-to-date picture in a dynamic setting. Moreover, the schemes described in the above cited papers are *centralized*, in the sense that they are based on a sequential algorithm which given a description of the entire graph outputs the entire set of vertex labels. Hence while the resulting labels reflect local knowledge and can be *used* locally, their *generation* process is still centralized and global.

Consequently, our interest in the current paper is in the development of localized informative labeling schemes in the *dynamic distributed* setting, using distributed online protocols. The current paper makes a first step in that direction, by studying informative labeling schemes for a *dynamic tree network*. The model is that of a network with a tree topology,

where the leaves of the tree can be removed from the network and new leaves can be added at any time. This model has been studied, for instance, in [AAPS89].

Our main results are as follows. Throughout the paper, denote by n the current number of vertices in the dynamic tree, at any given time. We first present a labeling scheme for distances in the dynamic tree model, with amortized message complexity $O(\log^2 n)$ per operation. The protocol maintains $O(\log^2 n)$ bit labels. This label size is optimal even in the static scenario [GPPR01].

It is worth noting an interesting point concerning the amortization method. Most of the literature on dynamic trees usually assumes a static graph spanned by the dynamic tree, and n is taken to be the size of the underlying graph. In contrast, our approach allows for dispensing with the underlying graph and basing the analysis on the size of the tree alone, at the expense of slightly more involved definitions for the amortized complexity measures.

We then introduce a rather general labeling scheme for the dynamic tree model, based on extending an existing *static* tree labeling scheme to the dynamic setting. The approach fits a number of natural tree functions, such as distance [Pel99], separation level [Pel00] and flow [KKKP02]. Intuitively, the resulting scheme incurs an overhead of an $O(\log n)$ multiplicative factor in both the label size and amortized message complexity in the case of *dynamically growing* trees (where no vertex deletions are allowed). In the special case where an upper bound on n is known in advance, this method can yield a different tradeoff, with an $O(\log^2 n/\log\log n)$ multiplicative overhead factor on the label size but an overhead factor of only $O(\log n/\log\log n)$ on the amortized message complexity. In the general model where vertex deletions are allowed as well, the scheme incurs also an increased *additive* overhead in amortized communication, of $O(\log^2 n)$ messages per operation.

As the current paper does not deal with the time complexity of the problem, we need not concern ourselves with the particulars of the communication model (e.g., synchronous or asynchronous communication, telephone or all-neighbors model etc.) In any case, even in the most restricted model, and even assuming messages are sent serially, taking no advantage of parallelism, the time complexity cannot exceed the communication complexity (in bits).

In a follow-up paper [KP03] we study approximate distance labeling schemes on dynamic weighted trees and cycles where the vertices are fixed but the (positive integral) weights of the edges may change, and present labeling schemes and lower bounds in the fully dynamic model and the "increasing" model in which edge weights can only grow.

2 Preliminaries

Our communication network model is restricted to tree topologies. The network is assumed to dynamically change via vertex additions and deletions. It is assumed that the *root* of the tree is never deleted. The following events may occur:

1. A new vertex u is *added* as a child of an existing vertex v. Subsequently, v is informed of this event, and assigns a unique *child-number* to u, in the sense that no currently existing child of v has the same child-number.

2. A leaf of the tree is *deleted*. Subsequently, the leaf's parent is informed of this event.

To implement child-numbers, each vertex keeps a counter that is increased by one for every new child. If vertex deletions are allowed, this counter may become quite large, although the size of the tree is small. This problem will be handled in detail later.

We use the following notation. For a vertex u, denote by $\omega(u)$ the *weight* of u, representing the number of vertices in u's subtree. For a non-root vertex u, denote by $p(u)$ its parent in the tree. The *ancestry* relation in the tree is defined as the transitive closure of the parenthood relation. In particular, a vertex is its own ancestor.

A *labeling scheme* $\pi = \langle \mathcal{M}_\pi, \mathcal{D}_\pi \rangle$ for a function F on pairs of vertices of a tree is composed of the following components:

1. A *marker* algorithm $\mathcal{M}_\pi$ that given a tree, assigns labels to its vertices.

2. A polynomial time *decoder* algorithm $\mathcal{D}_\pi$ that given the labels $L(u)$ and $L(v)$ of two vertices u and v, outputs $F(u, v)$.

In this paper we are interested in distributed networks where each vertex in the tree is a processor. This does not affect the definition of the decoder algorithm of the labeling scheme, but the marker algorithm changes into a distributed marker protocol.

Let us first consider static networks, where no changes in the topology of the network are allowed. For these networks we define *static* labeling schemes, where the marker protocol $\mathcal{M}$ is initiated at the root of a tree network and assigns static labels to all the vertices once and for all.

We use the following complexity measures to evaluate a static labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$.

1. *Label Size*, $\mathcal{LS}(\mathcal{M}, n)$: the maximum size of a label assigned by $\mathcal{M}$ to a vertex on any n-vertex tree.

2. *Message Complexity*, $\mathcal{MC}(\mathcal{M}, n)$: the maximum number of messages sent by $\mathcal{M}$ during the labeling process on any n-vertex tree. Note that messages can only be sent between neighboring vertices.

3. *Bit Complexity*, $\mathcal{BC}(\mathcal{M}, n)$: the maximum number of bits sent by $\mathcal{M}$ during the labeling process on any n-vertex tree.

Example 2.1 *The following is a possible static labeling scheme* StatDFS *for the ancestry relation on trees based on the notion of interval schemes ([SK85], cf. [Pel00a]). Given a rooted tree, simply perform a depth-first search starting at the root, assigning each vertex v the interval $I(v) = [a, b]$ where a is its DFS number and b is the largest DFS number given to any of its descendents. The corresponding decoder decides that v is an ancestor of w if their corresponding intervals, $I(v)$ and $I(w)$, satisfy $I(v) \subseteq I(w)$. It is easy to verify that this is a correct labeling scheme for the ancestry relation. Clearly, $\mathcal{MC}(\text{StatDFS}, n) = O(n)$, $\mathcal{BC}(\text{StatDFS}, n) = O(n \log n)$ and $\mathcal{LS}(\text{StatDFS}, n) = O(\log n)$.*

A labeling scheme for routing is presented in [FG01]. The scheme of [FG01] is designed as a sequential algorithm, but examining the details reveals that this algorithm can be easily transformed into a distributed protocol, and so we get a static labeling scheme for routing with label size and communication complexity similar to those of the StatDFS *static labeling scheme.*

The second setting considered is that of dynamically growing tree networks, where only vertex additions are allowed. For these networks we define *semi-dynamic* labeling schemes which involve a marker protocol $\mathcal{M}$ which is activated after every change in the network topology. The protocol $\mathcal{M}$ maintains the labels of all vertices in the tree so that the corresponding decoder algorithm will work correctly. We assume that the topological changes occur sequentially. Moreover, we assume that the changes are sufficiently spaced, so that the protocol has enough time to complete its operation in response to a given topological change before the occurrence of the next change.

For a semi-dynamic labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$, we denote $n_f = n_0 + n_+$. The definition of $\mathcal{LS}(\mathcal{M}, n)$ remains as for static schemes, and we are interested in the following complexity measures.

1. $\mathcal{MC}(\mathcal{M}, n_0, n_+)$: the maximum number of messages sent by $\mathcal{M}$, where n_0 is the initial tree size (when $\mathcal{M}$ is invoked) and n_+ is the number of additions made to the tree.

2. $\mathcal{BC}(\mathcal{M}, n_0, n_+)$: the maximum number of bits sent by $\mathcal{M}$, for n_0 and n_+ as above.

Finally, we consider fully dynamic networks as defined above, where both vertex additions and deletions are allowed. For more explicit time references, we use the notation $\bar{n} = (n_1, n_2, \ldots, n_t)$, where n_i is the size of the tree after the ith topological change. For simplicity, we assume $n_1 = 1$ unless stated otherwise. The definition of $\mathcal{LS}(\mathcal{M}, n)$ remains as before, and our complexity measures are modified as follows.

1. $\mathcal{MC}(\mathcal{M}, \bar{n})$: the maximum number of messages sent by $\mathcal{M}$.
2. $\mathcal{BC}(\mathcal{M}, \bar{n})$: the maximum number of bits sent by $\mathcal{M}$.

3 A dynamic labeling scheme for distance

In this section, we introduce a dynamic distance labeling scheme, called DL, with label size $\mathcal{LS}(\mathrm{DL}, n) = O(\log^2 n)$, message complexity $\mathcal{MC}(\mathrm{DL}, \bar{n}) = O(\sum_i \log^2 n_i)$ and bit complexity $\mathcal{BC}(\mathrm{DL}, \bar{n}) = O(\sum_i \log^2 n_i \log\log n_i)$. Thus the amortized message complexity and bit complexity of a topological change are $O(\log^2 n)$ and $O(\log^2 n \log\log n)$ respectively.

To this aim, we first introduce (in Subsect. 3.1) a static distance labeling scheme, called StatDL, with label size $\mathcal{LS}(\textsc{StatDL}, n) = O(\log^2 n)$, message complexity $\mathcal{MC}(\textsc{StatDL}, n) = O(n \log n)$ and bit complexity $\mathcal{BC}(\textsc{StatDL}, n) = O(n \log^2 n)$. We then introduce (in Subsection 3.3) a semi-dynamic distance labeling scheme based on StatDL, called SemDL, with label size $\mathcal{LS}(\textsc{SemDL}, n) = O(\log^2 n)$, message complexity $\mathcal{MC}(\textsc{SemDL}, n_0, n_+) = O(n_f \log^2 n_f)$ and bit complexity $\mathcal{BC}(\textsc{SemDL}, n_0, n_+) = O(n_f \log^2 n_f \log\log n_f)$, where $n_f = n_0 + n_+$ as defined earlier. Finally, we show (in Subsect. 3.4) how to use SemDL for constructing the dynamic labeling scheme DL.

3.1 The StatDL static labeling scheme

A basic ingredient in both Protocols StatDL and SemDL, is that each vertex u holds a pointer $\mu(u)$ to one of its children, henceforth referred to as the *marked child* of u. We refer to a child v of u such that $v \neq \mu(u)$ as a *common child* of u, and denote the set of common children of u by $\texttt{Common}(u)$.

Mark all edges $(u, \mu(u))$ in the tree. At any given time, the pointers are required to satisfy the following.

Balance property: For every vertex u in the tree, the path from the root to u contains at most $O(\log n)$ unmarked edges.

This property holds in both StatDL and SemDL.

3.1.1 The label structure

A simple representation of the path from the root to some vertex u is based on consecutively listing all the child-numbers along the path. Towards compacting this representation, we now do the following: for every marked edge along the path, replace the corresponding child-number in the list by a $\star$. The resulting list is called the *full label* of u, and is denoted by $L^{\text{full}}(u)$. For example, in Figure 1,

$$L^{\text{full}}(u) \;=\; (2,*,*,*,1,*) \qquad \text{and} \qquad L^{\text{full}}(v) \;=\; (2,*,3,*,*,2)\ .$$

Figure 1: A typical tree

As the full label $L^{\text{full}}(u)$ is of length $depth(u)$, we now compact it by replacing any maximal sublist of d consecutive stars with the condensed notation $\star^d$. The resulting list is called the *compact label* of u, denoted $L^{\text{comp}}(u)$. For example, in Figure 1,

$$L^{\text{comp}}(u) \;=\; (2,*^3,1,*), \qquad \text{and} \qquad L^{\text{comp}}(v) \;=\; (2,*,3,*^2,2)\ .$$

By the balance property of the pointers in the tree, the compact label of any vertex in an n-vertex tree is of length at most $O(\log n)$.

Since we do not allow deletions in our network, every child-number used is in the range $\{1\ldots,n\}$, and therefore can be encoded using at most $O(\log n)$ bits. Clearly, elements of the form $\star^d$ in the list can also be encoded using $O(\log n)$ bits, since $d \leq n$. Therefore, since the list consists of at most $O(\log n)$ elements, the total label size is $O(\log^2 n)$.

Note that from the compact label we can easily extract its full form. Therefore we will always assume the vertices store their compact labels, but may freely use their full label.

3.1.2 The decoder algorithm

Given two full labels, $L^{\text{full}}(u)$ and $L^{\text{full}}(v)$, describing the path from the root to two vertices u and v, we can calculate the distance between u and v by finding the first place where the

lists differ and adding up the lengths of their suffixes. For example, in Figure 1, u and v agree in the first two positions, and therefore the distance $d(u, v)$ is 8.

Formally, given $L^{\text{full}}(u)$ and $L^{\text{full}}(v)$, the decoder algorithm calculates the distance between u and v as follows.

1. Find the largest prefix z that is common to both $L^{\text{full}}(u)$ and $L^{\text{full}}(v)$.
2. Return $|L^{\text{full}}(v)| + |L^{\text{full}}(u)| - 2 \cdot |z|$.

To see that the decoder operates correctly, note that the prefix z is the full label of u's and v's least common ancestor w, i.e., $z = L^{\text{full}}(w)$. The sought distance satisfies $d(u, v) = d(u, w) + d(w, v)$. But $d(u, w)$ is simply the length of $L^{\text{full}}(u)$ minus that of $L^{\text{full}}(w)$. The same method is used to calculate $d(w, v)$, and the claim follows.

3.1.3 The marker protocol

The marker protocol of the static scheme StatDL is initiated at the root of a tree of size n, where the vertices may or may not already have child-numbers. The protocol operates as follows.

1. The root broadcasts a message requesting each vertex to assign its children new child-numbers. Each vertex orders its children arbitrarily, and assigns them the child-numbers $1, 2, \ldots$, and so on.
2. The root broadcasts a message requesting all vertices to (recursively) calculate their weight. This is done by a simple convergecast protocol (cf. [Pel00a]), whereby each vertex collects the weights of all of its children, sums their weights and adds one, thereby obtaining a weight equal to the number of vertices in its subtree.
3. Each vertex u sets its pointer $\mu(u)$ to point at its heaviest child v.
4. The distance labels are now assigned recursively starting at the root r. $L^{\text{full}}(r)$ is set to the empty list Λ. Once a vertex u computes its own compact label $L^{\text{comp}}(u)$, it sends it to each of its children. For a child v of u with child-number s, the full label of v is defined as u's full label plus s appended at the end, or $\star$ if u is pointing at v, i.e.,

$$L^{\text{full}}(v) \leftarrow \begin{cases} L^{\text{full}}(u) \cdot \langle \star \rangle, & \text{if } \mu(u) = v, \\ L^{\text{full}}(u) \cdot \langle s \rangle, & \text{otherwise.} \end{cases}$$

Clearly, the labels assigned by this protocol are of the form defined in Subsection 3.1.1. Also, it is easy to verify that the new child-numbers are of size at most $\log n$ bits. Since every common child v of a vertex u has $\omega(v) \leq \omega(u)/2$, the balance property is also clearly satisfied. Therefore we have the following.

Proposition 3.1

1. $\mathcal{LS}(\text{StatDL}, n) = O(\log^2 n)$,
2. $\mathcal{MC}(\text{StatDL}, n) = O(n)$,
3. $\mathcal{BC}(\text{StatDL}, n) = O(n \log^2 n)$.

3.2 Dynamic distributed tools

Protocol SemDL attempts to mimic Protocol StatDL and assigns the same labels. Since the vertices cannot maintain their exact weight in a dynamic network without incurring a considerable overhead in communication, we will only keep an estimate of the weight, using Protocol WeightWatch presented in Subsection 3.2.1. As a result, a vertex does not exactly know which of its children is the heaviest. Therefore, we require only that the common children are small. Note that it is necessary to minimize the number of times a vertex pointer changes, since each such change results in extra communication. Pointer changes are handled by Protocol HeavyChild, described in Subsection 3.2.3.

3.2.1 Protocol WeightWatch

The method of [AAPS89] is applied with technical changes to fit our setting. We give an overview of the method and our changes. Our goal is to give a protocol that operates in our dynamic tree model and, for a constant parameter $\delta > 1$, enables every vertex v to maintain a weight approximation $\tilde{\omega}(v)$ satisfying the following invariant:

The δ-estimation property: At any time,

$$\tilde{\omega}(v) \;\leq\; \omega(v) \;\leq\; \delta \cdot \tilde{\omega}(v).$$

This is achieved in the following way. We maintain two bins per each node v, a "local" bin and a "global" bin, denoted by $b_l(v)$ and $b_g(v)$, respectively. Each bin is assigned a *level*, a *supervisor bin* and a *capacity* (depending on its level). For each non-root node v at height $H(v)$, the levels of its bins are defined as follows:

1. $Level(b_g(v)) = \max\{i \mid 2^i \text{ divides } H(v)\}$, and
2. $Level(b_l(v)) = -1$.

Note that the level of the bin determines whether it is of type b_l or b_g. Therefore, in the following discussion we omit the subscripts g and l unless it might cause confusion. For each bin b at node v, the closest bin b' on the path from the root to v such that $Level(b') = Level(b) + 1$ is set to be the *supervisor* of b. If there is no such bin then the root is set to be b's supervisor. Note that the supervisor of a local bin $b_l(v)$ is either the global bin $b_g(v)$ at the same node v or the global bin of v's parent in the tree. This defines a *bin hierarchy* with the following easy-to-prove properties.

1. The depth of the bin hierarchy is at most $\log n + 1$ where n is the current number of vertices.
2. If $Level(b(v)) = l$ then any path from v to a node that holds one of b's supervisors has at most $O(2^l)$ nodes.
3. On any subtree T_v, the number of bins at level l that are supervisors is at most $|T_v|/2^{l-1}$. To see this, observe the following. For any level l supervisor bin v, let A_v be the closest 2^{l-1} nodes below v. Then A_v and A_u are disjoint for any two level l supervisor bins v and u.

Let $\alpha = \frac{\delta}{\delta - 1}$ for some constant $\delta > 1$. We define the *capacity* of a bin b at $Level(b) = l$ to be $Cap(b) = \max\{1\ ,\ 2^l/(\alpha(\log n' + 1))\}$ where initially $n' = cn_0$ for some constant $c > \delta$. The value of n' is changed from time to time to ensure that it satisfied the following invariant:

The n'-validity property: At all times $\frac{n'}{c} \leq n \leq n'$ where n is the current number of vertices in the tree.

This technicality is handled in Sub-protocol AdjustBins. Informally a bin b can store at most $Cap(b)$ tokens. Protocol WeightWatch works as follows.

1. Initially all bins are empty.
2. Each time a node v learns that it has a new child w, it sends it a token to fill its local bin $b_l(w)$.
3. Whenever a bin $b(v)$ gets filled with tokens, v immediately empties the bin and broadcasts a signal to its supervisor bin b'. Whenever the node holding b' gets this signal, it adds $Cap(b)$ tokens to b'. This signal can be simply $Level(b)$.

In addition, each node v monitors the signals passing through it and estimates $\omega(v)$ in the following way.

1. Each node v keeps a counter $\tilde{\omega}(v)$, initially set to v's initial weight.
2. When a signal of level l reaches v or passes through it, v adds $Cap(b)$ to $\tilde{\omega}(v)$, where b is the bin initiating this broadcast. (Note that v knows $Cap(b)$ since this value depends only on $Level(b) = l$).

Lemma 3.2 *For any constant $\delta > 1$, Protocol* WeightWatch *enables every vertex v to maintain a weight approximation $\tilde{\omega}(v)$ satisfying the δ-estimation property.*

Proof: We need to show that at any time, $\tilde{\omega}(v) \leq \omega(v) \leq \delta \cdot \tilde{\omega}(v)$. Initially all bins are empty. If the capacity of a bin equals 1, the bin always remains empty since it serves only as a relay between the node it supervises and its supervisor. The only bins that might not be empty are global bins (b_g's) that act as supervisors with capacity larger than 1. Whenever such a bin is not empty, it is half-full.

Fix a node v and let T_v be the subtree rooted at v. The *total waste* at T_v, denoted $Waste(v)$, is the number of tokens in the nonempty bins in T_v. Note that this value is exactly $Waste(v) = \omega(v) - \tilde{\omega}(v)$. In the subtree T_v, for each level l, there are at most $\omega(v)/2^{l-1}$ bins at level l that are supervisors. Even if all of them are half-full, we get that the total number of wasted tokens in bins of level l is at most $\omega(v)/(\alpha(\log n' + 1))$. Since the number of levels is at most $\log n$ where n is the current tree size and since $n \leq n'$, we get that $\omega(v) - \tilde{\omega}(v) = Waste(v) \leq \frac{\omega(v)}{\alpha}$ and the proof follows. ∎

Lemma 3.3

1. $\mathcal{MC}(\text{WeightWatch}, n_0, n_+) = O(n_f \log^2 n_f)$,
2. $\mathcal{BC}(\text{WeightWatch}, n_0, n_+) = O(n_f \log^2 n_f \log\log n_f)$.

Proof: Fix a level l. The number of times a signal is initiated at some bin of level l is equal to the number of times some level l bin got filled, which is a most $n_f/Cap(l)$ where $Cap(l) = \max\{1\ ,\ 2^l/(\alpha(\log n_f + 1))\}$. A signal initiated at some level l bin is broadcast to a distance $O(2^l)$ along the path from the bin to the root. The first part of the lemma follows since the number of levels is at most $\log n_f + 1$. The second part follows from the first part combined with the fact that each signal is of size $O(\log\log n_f)$. ∎

3.2.2 Subprotocol AdjustBins

This subprotocol has to ensure that the bin capacities are defined according to a valid n' value, i.e, n' satisfying $\frac{n'}{c} \leq n \leq n'$. Initially $n' = cn_0$ for some constant $c > 1$, so the n'-validity property is initially valid. In fact, as long as $\tilde{\omega}(root) < n'/\delta$ the n'-validity property is guaranteed to hold since Protocol WeightWatch is correct as long as n' is valid. The idea is that each time $\tilde{\omega}(root)$ exceeds $\frac{n'}{\delta}$ the root initiates the following operation.

1. The root broadcasts a signal instructing all the tree vertices to calculate their exact weight through a convergecast process.
2. The root informs the vertices about n, the correct number of vertices in the entire tree.
3. The vertices change their bin capacities according to $n' = cn$

This subprotocol ensures that at all times n' is valid. It remains to show that the additional communication costs incurred by this subprotocol do not damage the asymptotic complexity bounds established for Protocol WeightWatch. However, this is clear since the massage complexity of Subprotocol AdjustBins is linear in the current number of vertices and the subprotocol is performed at most $\log_c n_f = O(\log n_f)$ times using massages of size at most $O(\log n_f)$ bits.

3.2.3 Protocol HeavyChild

Protocol HeavyChild uses Protocol WeightWatch with $\delta = 5/4$. We further assume that each vertex knows not only its own estimated weight $\tilde{\omega}(v)$ but also all of its children's estimated weights. This is done in the following manner. Whenever a vertex v updates its estimated weight $\tilde{\omega}(v)$ it informs the new value of $\tilde{\omega}$ to $p(v)$, its parent in the tree. The parent $p(v)$ does not use this new information to update its estimated weight and therefore the requirement that each vertex knows also all of its children's estimated weights will at most double the communication complexity of Protocol WeightWatch. For each vertex u, define the *weight ratios*

$$\varrho(u) \;=\; \frac{\omega(u)}{\omega(p(u))} \qquad \text{and} \qquad \tilde{\varrho}(u) = \frac{\tilde{\omega}(u)}{\tilde{\omega}(p(u))}\ .$$

We have

$$\frac{1}{\delta} \cdot \varrho(u) \;\leq\; \tilde{\varrho}(u) \;\leq\; \delta \cdot \varrho(u). \tag{1}$$

Each vertex can calculate $\tilde{\varrho}(u)$ for all of its children. Protocol HeavyChild is initiated on a tree where vertices already have pointers, and operates as follows. Whenever a vertex v

observes that one of its common children u satisfies $\tilde{\varrho}(u) > 3/4$, it changes its pointer $\mu(v)$ to u. This event is called a *shuffle* γ of the vertex v. We associate with each shuffle event γ a *weight* defined as $\omega(\gamma) = \omega(v)$.

Our choice of δ and Eq. (1) imply the following.

Observation 3.4

1. *If a vertex u has $\tilde{\varrho}(u) > 3/4$ then $\varrho(u) > \frac{3}{4} \cdot \frac{1}{\delta} = 3/5$. Hence at most one child u of a vertex v can have $\tilde{\varrho}(u) > 3/4$.*

2. *For every vertex v, and every common child u of v, $\varrho(u) < \frac{3}{4} \cdot \delta < 15/16$.*

By this observation, we get the following.

Claim 3.5 [Balance Property] *For every vertex v in an n-vertex tree, the path from the root to v has at most $O(\log n)$ unmarked edges.*

As mentioned, the communication complexity of Protocol HeavyChild is at most twice that of Protocol WeightWatch.

3.3 Protocol SemDL

Protocol SemDL deals with the semi-dynamic model allowing only vertex additions.

3.3.1 The protocol

The first step of Protocol SemDL is to run Protocol StatDL. It then initiates Protocol HeavyChild and monitors its behavior.

1. If a new vertex v is added as a child of a vertex u with child-number s, then v's label is defined as
$$L^{\text{full}}(v) \leftarrow \begin{cases} L^{\text{full}}(u) \cdot \langle \star \rangle, & \text{if } \mu(u) = v \\ L^{\text{full}}(u) \cdot \langle s \rangle, & \text{otherwise.} \end{cases}$$

2. If a shuffle γ of a vertex v occurs in Protocol HeavyChild, and v shifts its pointer from u^{old} to u^{new}, whose child numbers are s^{old} and s^{new} respectively, then v initiates the following changes:

   (a) It instructs all vertices in the subtree of u^{new} to change the entry in position $depth(v)$ of their full label from s^{new} to $\star$.

(b) It instructs all vertices in the subtree of u^{old} to change the entry in position $depth(v)$ of their full label from $\star$ to s^{old}.

As these changes are applied to the full label, each vertex needs to recompact its label after performing the change.

3.3.2 Complexity

Clearly, Protocol SemDL maintains the labels as defined in Subsection 3.1.1, and therefore since Protocol HeavyChild maintains the balance property, we have that the resulting labels are of size $\mathcal{LS}(\text{SemDL}, n) = O(\log^2 n)$.

The operation of step 1 of the protocol involves sending one message of size $O(\log^2 n)$, and the operation of step 2 requires $O(\omega(v)) = O(\omega(\gamma))$ messages of size $O(\log n)$. Since the current tree size satisfies $n < n_f$ at all times, the communication complexity of Protocol SemDL is bounded above by

$$\begin{aligned}
\mathcal{MC}(\text{SemDL}, n_0, n_+) &= \mathcal{MC}(\text{HeavyChild}, n_0, n_+) + O(n_+) + O(\textstyle\sum_{\gamma\in\Gamma} \omega(\gamma)) \\
&= O(n_f \log^2 n_f) + O(\textstyle\sum_{\gamma\in\Gamma} \omega(\gamma)), \\
\mathcal{BC}(\text{SemDL}, n_0, n_+) &= \mathcal{BC}(\text{HeavyChild}, n_0, n_+) + O(n_+ \log^2 n_f) + O(\textstyle\sum_{\gamma\in\Gamma} \omega(\gamma) \log n_f) \\
&= O(n_f \log^2 n_f \log\log n_f) + O(\textstyle\sum_{\gamma\in\Gamma} \omega(\gamma) \log n_f).
\end{aligned}$$

Lemma 3.6 *If Protocol* HeavyChild *is initiated on a tree of size n_0 with pointers satisfying the balance property, then*

$$\sum_{\gamma\in\Gamma} \omega(\gamma) \;=\; O(n_f \log n_f) \ .$$

Proof: For every vertex v, let $\psi(v) = \sum_{u\in\texttt{Common}(v)} \omega(u)$. Examine the behavior of the value $\Psi = \sum_v \psi(v)$ during the execution.

First suppose a vertex v is added to the tree. For every vertex u, $\psi(u)$ increases by 1 iff v is a descendent of one of u's common children. On the path from the root to v, there are at most $O(\log n) \le O(\log n_f)$ unmarked edges and the start points of these edges are exactly the vertices that increase their ψ by 1. Therefore Ψ increases by at most $O(\log n_f)$.

By an argument similar to the above we have that initially $\Psi \le O(n_0 \log n_0)$.

Now suppose a shuffle γ of a vertex v causes v to change its pointer from u^{old} to u^{new}. Then $\varrho(u^{new}) > 3/5$, and therefore $\varrho(u^{old}) \le 2/5$. Hence $\psi(v)$ after the change equals $\psi(v)$ before the change plus $\omega(u^{old}) - \omega(u^{new}) < \frac{1}{5}\omega(v)$. Therefore, after this change, Ψ decreases by $\Omega(\omega(v)) = \Omega(\omega(\gamma))$.

To sum up, at any point in time,

$$\Psi \;\leq\; O\left(n_0 \log n_0\right) + O\left(n_+ \log n_f\right) - \Omega\left(\sum_{\gamma\in\Gamma} \omega(\gamma)\right).$$

Since always $\Psi \geq 0$, the lemma follows. ∎

Subsequently we have the following.

Proposition 3.7

1. $\mathcal{LS}(\textsc{SemDL}, n) = O(\log^2 n)$,
2. $\mathcal{MC}(\textsc{SemDL}, n_0, n_+) = O(n_f \log^2 n_f)$,
3. $\mathcal{BC}(\textsc{SemDL}, n_0, n_+) = O(n_f \log^2 n_f \log\log n_f)$.

3.4 The DL dynamic distance labeling scheme

We now present our algorithm for the fully dynamic case (allowing vertex deletions). The general idea is to run Protocol SemDL while ignoring deletions, i.e., treating deleted vertices as if they were never deleted. However, at some point or another in the execution, deletions must be accounted for, since if the tree size falls significantly, then the old labels (and child numbers) which were small with respect to the original tree size, might become too large. We therefore run, in parallel to Protocol SemDL, a protocol for estimating the number of topological changes in the tree. Every $\Theta(n)$ topological changes, we restart Protocol SemDL again. Clearly, the labels will be as defined in Subsection 3.1.1, and the only remaining question concerns the resulting communication complexity.

3.4.1 Protocol ChangeWatch

Denote by τ the number of topological changes made to the tree during the execution. Fix $\delta = 9/8$. We use Protocol ChangeWatch, which is an instance of the protocol of [AAPS89] in which the root maintains an estimate $\tilde{\tau}$ of τ. This is done by applying the same mechanism as in Protocol WeightWatch separately for the additions of vertices and for the deletions. I.e, we run two protocols in parallel. The first is designed to count the additions. In order to do that, we ignore the deletions and perform the same steps as in Protocol ChangeWatch. The second protocol is designed to count the deletions. For this we ignore the additions, and carry the same steps as in Protocol ChangeWatch, except for deletions rather than for additions.

Let n_0 be the number of vertices in the tree when Protocol ChangeWatch was initiated. Let n_+ and n_- be the number of additions and deletions respectively and let $\tilde{n}_+$ and $\tilde{n}_-$ be the root's estimated number of additions and deletions respectively. By the same reasoning as in Protocol ChangeWatch at any time the following inequalities hold.

$$n_0 + \tilde{n}_+ \ \leq \ n_0 + n_+ \ \leq \ \frac{9}{8}(n_0 + \tilde{n}_+)$$

and

$$n_0 + \tilde{n}_- \ \leq \ n_0 + n_- \ \leq \ \frac{9}{8}(n_0 + \tilde{n}_-).$$

Therefore, as long as $\tilde{n}_+ \leq \frac{n_0}{9}$ and $\tilde{n}_- \leq \frac{n_0}{9}$ we know $\tau = n_+ + n_- \leq \frac{n_0}{2}$. Moreover, if $\tilde{n}_+ > \frac{n_0}{9}$ or $\tilde{n}^- > \frac{n_0}{9}$ then $\tau > \frac{n_0}{9}$. As in Protocol WeightWatch,

1. $\mathcal{MC}(\text{ChangeWatch}, \bar{n}) = O((n_1 + \tau)\log^2(n_1 + \tau))$,
2. $\mathcal{BC}(\text{ChangeWatch}, \bar{n}) = O((n_1 + \tau)\log^2(n_1 + \tau)\log\log(n_1 + \tau))$.

Here n_1 is not necessarily 1, since the protocol is initiated on trees of larger size.

3.4.2 Protocol DL

1. The root broadcasts a massage to perform a convergecast in order to calculate the initial number of vertices in the tree, n_0.
2. Protocols SemDL and ChangeWatch are started. If a vertex v is deleted during this run, then its parent u simulates its behavior as if v and all its descendents were never deleted. In particular:
   (a) For Protocols WeightWatch and HeavyChild, this simulation is trivial, and requires no extra messages.
   (b) For Protocol SemDL, u will simply not pass to v any label-changing messages resulting from a shuffle.
   (c) Protocol ChangeWatch estimates the number of additions and deletions.
   For estimating the number of additions, Protocol ChangeWatch will ignore the deletion of v;
   For estimating the number of deletions, Protocol ChangeWatch will act as described in Subsect. 3.4.1.
3. When one of the estimates $\tilde{n}_+$ or $\tilde{n}_-$ becomes at least $n_0/9$, return to Step 1.

3.4.3 Complexity

Theorem 3.8

1. $\mathcal{LS}(\mathrm{DL}, n) = O(\log^2 n)$,
2. $\mathcal{MC}(\mathrm{DL}, \bar{n}) = O(\sum_i \log^2 n_i)$,
3. $\mathcal{BC}(\mathrm{DL}, \bar{n}) = O(\sum_i \log^2 n_i \log\log n_i)$.

Proof: The protocol is restarted by returning to Step 1 after τ topological changes, for $\tau = \Theta(n_0)$, where n_0 is the last recorded tree size at Step 1. Consequently, the current tree size satisfies $n = \Theta(n_0)$ and the child-numbers are always of size $O(\log n)$.

Moreover, since deleted vertices are treated as if they are still in the tree, every vertex holds an estimate $\tilde{\omega}$ that is not related to its true weight but to its weight including the deleted vertices. Therefore, the decisions made by Protocol HeavyChild concerning the marked child are done according to the weight including deleted vertices. Denoting by n_- the number of deletions, the balance property of pointers holds with respect to $O(\log(n_0 + n_+))$ rather than $O(\log(n_0 + n_+ - n_-))$. Since $n_+ + n_- = \Theta(n_0)$, the balance property holds w.r.t. $O(\log n_0) = O(\log n)$. Altogether, we get that the labels are of size $O(\log^2 n)$, proving Claim 1.

Let $i_1, \ldots, i_k$ be the indices of the topological changes on which the protocol returns to Step 1. Denote by M_l the message complexity starting at initialization l and before the next initialization occurs. Clearly

$$\mathcal{MC}(\mathrm{DL}, \bar{n}) \;=\; \sum_{l=1}^{k} M_l \ .$$

M_l is the sum of the message complexities of Step 1 and Protocols SemDL and ChangeWatch. Therefore, since the number of changes relevant to M_l is $\Theta(n_{i_l})$, we get

$$M_l \;\leq\; O(n_{i_l}) + O(n_{i_l} \log^2 n_{i_l}) \;=\; O(n_{i_l} \log^2 n_{i_l}) \ .$$

Since $n_{i_l} = \Theta(n_{i_{l+1}})$, we actually have $M_{l+1} \leq O(n_{i_l} \log^2 n_{i_l})$, and therefore

$$\mathcal{MC}(\mathrm{DL}, \bar{n}) \;=\; \sum_{l=1}^{k} M_l \;=\; M_1 + \sum_{l=1}^{k-1} M_{l+1} \;=\; O(1) + O(\sum_{l=1}^{k-1} n_{i_l} \log^2 n_{i_l}) \ .$$

Finally, observe that for every $l < k$,

$$\sum_{j=i_l}^{i_{l+1}-1} \log^2 n_j \;=\; \Omega(n_{i_l} \log^2 n_{i_l})$$

since all the relevant n_j's are of size $\Theta(n_{i_l})$, and there are at least $\Omega(n_{i_l})$ such j's. Claim 2 follows.

The proof of Claim 3 is carried out in exactly the same manner. ∎

4 A general dynamic labeling scheme

In this section we present a dynamic labeling scheme for a wide variety of binary functions on trees. This scheme takes an existing *static* labeling scheme for the same function and uses it as a subroutine. Specifically, we proceed as follows. Given a static labeling scheme $\pi = \langle \mathcal{M}_\pi, \mathcal{D}_\pi \rangle$ for the function F, we first describe a semi-dynamic labeling scheme SemGL for F, and later show how it can be transformed into a dynamic labeling scheme GL. Assuming the message complexity of the static labeling scheme π is polynomial[1] in the size of the tree, we show that $\mathcal{LS}(\text{SemGL}, n) = O(\log n \cdot \mathcal{LS}(\pi, n))$ and $\mathcal{MC}(\text{SemGL}, n_0, n_+) = O(\log n_f \cdot \mathcal{MC}(\pi, n_f))$. In case n_f is known in advance, we get $\mathcal{LS}(\text{SemGL}, n) = O(\frac{\log n_f \log n}{\log \log n_f} \mathcal{LS}(\pi, n))$ and $\mathcal{MC}(\text{SemGL}, n_0, n_+) = O(\frac{\log n}{\log \log n_f} \mathcal{MC}(\pi, n_f))$. For the fully-dynamic scheme we have $\mathcal{LS}(\text{GL}, n) = O(\log n \cdot \mathcal{LS}(\pi, n))$ and $\mathcal{MC}(\text{GL}, \bar{n}) = O(\sum_i \log n_i \frac{\mathcal{MC}(\pi, n_i)}{n_i}) + O(\sum_i \log^2 n_i)$, or alternatively $\mathcal{LS}(\text{GL}, n) = O(\frac{\log^2 n}{\log \log n} \mathcal{LS}(\pi, n))$ and $\mathcal{MC}(\text{GL}, \bar{n}) = O(\sum_i \frac{\log n_i}{\log \log n_i} \frac{\mathcal{MC}(\pi, n_i)}{n_i}) + O(\sum_i \log^2 n_i)$.

Example 4.1 *Consider the static labeling scheme* StatDFS *for the ancestry relation, presented in Example 2.1. If n is not known (Part 1), we can get a semi-dynamic labeling scheme* SemDFS *with $\mathcal{LS}(\text{SemDFS}, n) = \log^2 n$, and $\mathcal{MC}(\text{SemDFS}, 1, n_+) = n_+ \log n_+$. We can also get a dynamic labeling scheme* DFS *(using Part 3), with $\mathcal{LS}(\text{DFS}, n) = \log^2 n$, and $\mathcal{MC}(\text{DFS}, \bar{n}) = O(\sum_i \log n_i) + O(\sum_i \log^2 n_i) = O(\sum_i \log^2 n_i)$.*

As mentioned in Example 2.1, there is static labeling scheme for routing with the same label size and communication complexity as StatDFS*. Consequently, we can create semi-dynamic and dynamic routing labeling schemes with same label size and communication complexity as the* SemDFS *and* DFS *schemes respectively.*

The family of functions to which our method is applicable can be characterized as all the functions satisfying the following two conditions:

[1] We actually require that the message complexity is bounded above by some function f which satisfies $f(\alpha \cdot n) = \Theta(f(n))$ for any constant $\alpha > 0$. We also require $f(a+b) \geq f(a) + f(b)$. These two requirements are satisfied by most natural relevant functions, such as $c \cdot n^\alpha \log^\beta n$, where $c > 0$, $\alpha \geq 1$ and $\beta > 0$. For simplicity, we assume $\mathcal{MC}(\cdot, n)$ itself satisfies these requirements.

(C1) For any two vertices u and v in the tree, $F(u, v)$ depends only on the path between them (including the child-numbers of the edges in the path).

(C2) For any three vertices u, w and v in the tree, such that w is on the path between u and v, $F(u, v)$ can be calculated in polynomial time from $F(u, w)$ and $F(w, v)$.

In particular, our method can be applied to the ancestry relation and the labeling for routing, thereby extending static routing labeling schemes such as those of [FG01, TZ01, FG02] to the dynamic setting. We further assume, for simplicity of presentation, that F is symmetric, i.e., $F(u, v) = F(v, u)$. A slight change to the suggested protocols handles the more general case, without affecting the asymptotic complexity results.

We assume that the labels given by the static labeling scheme are unique; i.e., given a tree, no two vertices get the same static label. Given any static labeling scheme, this additional requirement can be ensured at an extra cost of at most n to $\mathcal{MC}(n)$ and $\log n$ to $\mathcal{LS}(n)$, and therefore it will not affect the asymptotic complexities of our scheme.

4.1 Protocol SemGL

In this description, we assume the initial tree size is 1, i.e., it consists of just the root. To handle initial tree sizes different than 1, we simply simulate Protocol SemGL, starting from the root of the initial tree, and as if vertices of the initial tree were added one after the other. We therefore get

$$\mathcal{MC}(\text{SemGL}, n_0, n_+) \;=\; \mathcal{MC}(\text{SemGL}, 1, n_0 + n_+ - 1) \;.$$

For simplicity of notation, we denote $\mathcal{MC}(\text{SemGL}, n) = \mathcal{MC}(\text{SemGL}, 1, n - 1)$ throughout this section.

4.1.1 Informal description of protocol SemGL

Protocol SemGL uses an existing static labeling scheme as a subroutine. We refer to a run of this static labeling scheme as a *shuffle*. A shuffle is always performed on a subtree of the current tree, and it has a nonnegative integral *order*, i, whose meaning is to be clarified later. At any given time in the execution of SemGL and for every vertex v existing at that time, denote by $\gamma(v)$ the last shuffle in which v has participated. A *bubble* is a maximal set of vertices sharing their last shuffle, and the bubble's order is the order of that shuffle. For a bubble b, denote by $order(b)$ the order of b. These bubbles form a tree, called the *bubble tree*.

Protocol SemGL uses a positive integer d as a parameter fixed in advance. Protocol SemGL maintains the following invariants on the bubble tree.

1. If bubble b_1 is the parent of b_2 in the bubble tree then $order(b_1) \geq order(b_2)$.
2. There is no set of d bubbles $b_1, b_2, \ldots, b_d$ such that b_i is the parent of b_{i+1} for $1 \leq i \leq d-1$ and $order(b_1) = \ldots = order(b_d)$.

Whenever a new vertex is added to hte tree, a shuffle of order 0 is performed on the new vertex alone. Now, if this operation creates a sequence of d consecutive order 0 bubbles in the bubble tree, then the root vertex of the highest bubble in the sequence performs a level 1 shuffle on its subtree, which consists of bubbles of order 0 only (by invariant 1). Note that there could be more bubbles here than those in the original sequence. This operation combines all these order 0 bubbles into an order 1 bubble. In turn, if this new bubble now creates a sequence of d consecutive order 1 bubbles in the bubble tree then we proceed as before, and perform an order 2 shuffle rooted at the highest bubble in this sequence. We proceed in the same manner, initiating a new shuffle of order i whenever a sequence of d consecutive order $i-1$ bubbles is created. This clearly maintains both invariants. We also get that the size of an order i bubble (namely, the number of vertices it contains) is at least d^i. See Figure 2 for an illustration of a shuffle in the bubble tree.

Figure 2: An example of a bubble tree where d is set to be 4. The numbers mark the bubbles' order. If a new vertex is added as a child of vertex u then the last three level 0 bubbles plus u undergo an order 1 shuffle which turns them into an order 1 bubble. This, in turn, creates a sequence of 4 order 1 bubbles which causes the circled subtree to undergo a shuffle and become a bubble of order 2.

For a vertex v in bubble b, denote by `p-root`(v) the vertex that is the parent of the bubble's root. (For example, in Figure 2 we have $a =$ `p-root`(b).) Note that `p-root`(v) is undefined for vertices in the bubble at the root of the bubble tree.

The label of a vertex v contains three elements.

1. The static label of v in its last shuffle (recall that this shuffle is a run of the static labeling scheme on v's bubble).
2. $F(v,$ `p-root`$(v))$.
3. The label of `p-root`(v) whenever `p-root`(v) is defined. Note that this is defined recursively.

Given two labels of vertices u and v in the same bubble, the value $F(u, v)$ can be calculated using the decoder of the static labeling scheme by running it on the first elements of the labels. From u's and v's labels, it is easy to identify which bubble is the least common ancestor of u's bubble and v's bubble in the bubble tree. If this bubble is v's bubble then $\texttt{p-root}(u)$ is on the path from u to v. Recursively we now calculate $F(\texttt{p-root}(u), v))$, and since the second element of u's label is $F(u, \texttt{p-root}(u))$, we can calculate $F(u, v)$ using Condition $(C2)$ on F. Otherwise, $\texttt{p-root}(v)$ is on the path between u and v, and so we can proceed in exactly the same fashion and calculate $F(u, v)$ using $F(v, \texttt{p-root}(v))$ from v's label, and $F(\texttt{p-root}(v), u)$ calculated recursively.

4.1.2 The label structure

The label $L(v)$ of each vertex v can be represented as a 2-dimensional unbounded matrix, where each element in the matrix is either empty or a triplet $\langle l, R, s\rangle$, for l, R, s to be defined later on. In each row $i \geq 0$, the nonempty elements are aligned to the left, followed by empty entries on the right. We say that a row is *empty* if all its entries are empty, and that a row is *full* if the number of non-empty elements in it is at least $d - 1$, where $d > 1$ is an integer fixed in advance (we show in Lemma 4.10 that in fact each row will contain at most $d - 1$ non-empty elements). For a label L we use the following notation.

1. $\mathrm{NF}(L)$ is the first *nonfull* row of L.
2. $\mathrm{NE}(L)$ is the first *nonempty* row of L. We define $\mathrm{NE}(L) = \infty$ for the empty matrix.
3. $\mathrm{Least}(L)$ the last nonempty element on row $\mathrm{NE}(L)$ (undefined for the empty matrix). The l field of this element corresponds to v's static label in v's bubble.

See Figure 3 for an example of a dynamic label.

4.1.3 The shuffle operation

Our protocol repeatedly engages the marker protocol of the static labeling scheme, and uses the labels it outputs to construct our dynamic label. In doing so, the protocol occasionally applies to the already labeled portion of the tree a *shuffle* operation defined as follows. A *level i shuffle* of a subtree rooted at w is the following protocol:

1. The vertex w invokes the static labeling scheme on its subtree T_w, and gives a new static label $L^{\mathrm{stat}}(v)$ to each vertex v in the subtree T_w.

Figure 3: A typical label $L(v)$. The elements in row i correspond to order i ancestor bubbles of v.

2. Let $z = p(w)$, and denote its label by $L = L(z)$. Let s be w's child-number.

3. For every vertex v in the subtree T_w do:

   (a) Let $l = L^{\text{stat}}(v)$ and $R = F(v, z)$.

   (b) Give v a label $L(v)$ composed of the label L modified by inserting the element $\langle l, R, s\rangle$ instead of the first empty element of row i in L.

We define the *root* and *p-root* of this shuffle operation to be w and z, respectively.

4.1.4 Adding a vertex

The label of the root is the empty matrix. Suppose a vertex v is added as a child of the vertex u. Then the marker protocol acts as follows.

1. Let $i = \text{NF}(L(u))$.

2. Search up in the tree from u until reaching the first vertex z (including u) such that $\text{NE}(L(z)) \geq i$. Let w be z's child on the path to v. (In particular, if $z = u$ then $w = v$.)

3. Perform a level i-shuffle rooted at w. We say u is the *initiator* of this shuffle.

See Figure 4 for an illustration of a shuffle resulting from a vertex addition.

We extensively use the following lemma.

Figure 4: In this example, the parameter d is set to be 4. The new vertex was added as a child of u, where $\mathrm{NF}(u) = 2$. The depicted label of u is the label before the new vertex was added. This initiated a level i shuffle, where z is the p-root. The label of z does not change as a result of this shuffle. The vertex v is in the subtree affected by the shuffle, and the depicted label is v's label after the shuffle. Note that this shuffle is exactly the shuffle simulated in Figure 2.

Lemma 4.2 *If u is a parent of v then $L(u)$ and $L(v)$ agree on all elements except* $\mathrm{Least}(L(v))$.

Proof: By induction on the addition of vertices to the tree. When a vertex is added, one of its ancestors w (possibly the new vertex itself) performs a shuffle. As a result, the vertices in w's subtree T_w (including the new vertex) are all assigned new labels.

If $v \notin T_w$, then its label and u's label do not change, and therefore by the induction hypothesis we are done. So now suppose $v \in T_w$. Then its label is the same as $L(p(w))$, except for one element, which is $\mathrm{Least}(L(v))$. There are two subcases to consider.

1. If also $u \in T_w$, then its label is also exactly as the p-root's label except for one new element which is $\mathrm{Least}(L(u))$, and so v and u differ only on $\mathrm{Least}(L(u))$ and $\mathrm{Least}(L(v))$.

2. If $v = w$, then $u = p(w)$, and so v's new label is u's label plus one new element which is $\mathrm{Least}(L(v))$. ∎

Corollary 4.3 *If row i is full in $L(v)$, then it is full in all of v's descendents.*

Corollary 4.4 *Every ancestor w of v satisfies* $\mathrm{NE}(L(w)) \geq \mathrm{NE}(L(v))$.

Corollary 4.5 *If u is an initiator of a level i shuffle, and z is its p-root, then just before the shuffle, $L(u)$ and $L(z)$ are the same from row i on. Also, rows $1, \ldots, i-1$ are full in $L(u)$ and empty in $L(z)$.*

4.1.5 The decoder algorithm

For a label L, denote by $L{\uparrow}$ the label obtained from L by removing the element in position $\mathrm{Least}(L)$, leaving it empty. For two labels L_1 and L_2, we say that $L_1 \subseteq L_2$ if $L_2 \underbrace{{\uparrow}{\uparrow} \ldots {\uparrow}}_{i} = L_1$ for some $i \geq 0$. The label L, as discussed in section 4.1.2, is in fact a matrix. Let e be the element at position (i, j) in the matrix L, denote (i, j) by *the label position* of e.

Procedure $\mathcal{D}(L(v), L(u))$

The decoder procedure receives as input the labels $L(u)$ and $L(v)$ of two vertices u and v. It proceeds in one of two ways, either calculating $F(u, v)$ directly using the decoder $\mathcal{D}_\pi$ of the static scheme π or finding a label $L(y)$ of an intermediate vertex y such that y is on the path between u and v, and recursively invoking procedures $\mathcal{D}(L(v), L(y))$ and $\mathcal{D}(L(y), L(u))$ calculating $F(u, y)$ and $F(y, v)$ respectively, and then calculating $F(u, v)$ using Condition (C2) on F. The procedure operates as follows.

1. If $L(u)\uparrow = L(v)$ then return the R field of Least$(L(u))$ as $F(u, v)$. Analogously for the case $L(v)\uparrow = L(u)$.
2. Otherwise, if $L(v) \subseteq L(u)$, then let $L = L(u)\uparrow$, recursively call $\mathcal{D}(L(v), L)$ and $\mathcal{D}(L, L(u))$ calculating $F(u, y)$ and $F(y, v)$ respectively, and then calculate $F(u, v)$ using Condition (C2) on F. Analogously for the case $L(u) \subseteq L(v)$.
3. Otherwise, if $L(u)\uparrow \not\subseteq L(v)$, then let $L = L(u)\uparrow$, recursively call $\mathcal{D}(L(v), L)$ and $\mathcal{D}(L, L(u))$ calculating $F(u, y)$ and $F(y, v)$ respectively, and then calculate $F(u, v)$ using Condition (C2) on F. Analogously for the case $L(v)\uparrow \not\subseteq L(u)$.
4. Otherwise $L(u)\uparrow \subseteq L(v)$ and $L(v)\uparrow \subseteq L(u)$, and therefore $L(v)\uparrow = L(u)\uparrow$. Then,
   (a) If the label position of element Least$(L(u))$ is different than that of Least$(L(v))$, then set $L = L(v)\uparrow$ and proceed recursively calling $\mathcal{D}(L(v), L)$ and $\mathcal{D}(L, L(u))$ as before.
   (b) If the label position of element Least$(L(u))$ is the same as that of Least$(L(v))$, and the s field of Least$(L(u))$ and Least$(L(v))$ is different, then set $L = L(v)\uparrow$ and proceed recursively calling $\mathcal{D}(L(v), L)$ and $\mathcal{D}(L, L(u))$ as before.
   (c) If the label position of element Least$(L(u))$ is the same as that of Least$(L(v))$ and the s field is also the same, then denote by l_v and l_u the l fields of Least$(L(v))$ and Least$(L(u))$ respectively, and return $F(u, v) = \mathcal{D}_\pi(l_u, l_v)$.

To prove the correctness of the decoder, we need the following notation. For a vertex v, denote by $\gamma(v)$ the last shuffle v participated in. For a shuffle γ, denote by `p-root`(γ) the p-root of γ and let `p-root`(v) = `p-root`$(\gamma(v))$.

We also need the following three preliminary lemmas.

Lemma 4.6 *For every vertex other than the root,* $L($`p-root`$(v)) = L(v)\uparrow$.

Proof: The label of $\texttt{p-root}(v)$ has not changed since the last shuffle v participated in (because any shuffle that includes $\texttt{p-root}(v)$ must also include v). Therefore, v's label is $L(\texttt{p-root}(v))$ plus one new element, which is Least$(L(v))$. ∎

Lemma 4.7 *If $L(u) = L(v)$ then $u = v$.*

Proof: By induction on the number of nonempty elements in $L(u)$. If $L(u) = L(v)$, then by Lemma 4.6 $L(\texttt{p-root}(u)) = L(u)\uparrow = L(v)\uparrow = L(\texttt{p-root}(v))$. Therefore, by the induction hypothesis, $\texttt{p-root}(u) = \texttt{p-root}(v)$. Since the s field of Least$(L(u))$ is the same in Least$(L(v))$, the vertices v and u share their last shuffle, where $\texttt{p-root}(u) = \texttt{p-root}(v)$. In this shuffle, by our assumption on the static marker protocol, vertices get unique labels, and therefore, since the l field of Least$(L(u))$ is the same as in Least$(L(v))$, we get that $u = v$. ∎

Lemma 4.8 *For vertices u and v, if $L(u)\uparrow \not\subseteq L(v)$ then $\texttt{p-root}(u)$ is on the path from u to v.*

Proof: Let w be the root of u's last shuffle. The vertices w and u share their last shuffle, and therefore $L(w)\uparrow = L(\texttt{p-root}(u))$. By Lemma 4.2, every vertex x in w's subtree T_w has label $L(w)\uparrow$ plus some other elements. Therefore $L(\texttt{p-root}(u)) \subseteq L(x)$. However, $L(\texttt{p-root}(u)) \not\subseteq L(v)$ and therefore v is not in T_w. The lemma follows. ∎

We can now prove the correctness of the scheme.

Proposition 4.9 *The decoder algorithm is correct.*

Proof: Let S be the sum of the number of non-empty elements in the label $L(u)$ and the number of non-empty elements in the label $L(v)$. The proof is by induction on S. If $S = 0$ then both $u = v = root$ and we are done. Otherwise, consider the following cases.

If $L(u)\uparrow = L(v)$, then by Lemmas 4.6 and 4.7, $\texttt{p-root}(v) = u$ and $F(u, v)$ is then the R field of Least$(L(v))$, so the algorithm is correct. Similarly for the other way around.

If $L(v) \subseteq L(u)$, then let $y = \texttt{p-root}(u)$. If $y = v$ then by the above case we are done. Otherwise, $L(v) \subseteq L(y)$, and therefore v is an ancestor of y which is an ancestor of u and so y is on the path between u and v. $F(u, y)$ is the R field of Least$(L(u))$. $F(y, v)$ is calculated recursively using the induction hypothesis. (note the the number of non-empty elements in $L(y)$ is less than the number of non-empty element in $L(u)$. $F(u, v)$ is now calculated using Condition (C2) on F. Similarly for the analogous case.

If $L(u)\uparrow \not\subseteq L(v)$, then by Lemma 4.8, $y = \texttt{p-root}(u)$ is on the path between u and v, and again the recursive solution will produce a correct result. Similarly for the analogous case.

In the last case, $L(v)\uparrow = L(u)\uparrow$, let $y = \texttt{p-root}(u) = \texttt{p-root}(v)$ (the last inequality follows by Lemmas 4.6 and 4.7). There are three subcases to consider.

1. If the placing of element Least($L(u)$) is different than that of Least($L(v)$), then y is the least common ancestor of u and v by Lemmas 4.2 and 4.7, and therefore is on the path between u and v, so the recursive solution is correct.

2. If the placing is the same, and the s field of Least($L(u)$) and Least($L(v)$) is different, then again y is the least common ancestor of u and v and we are done.

3. Finally, if the placing is the same, and the s field is the same, then u and v share their last shuffle. Therefore, the l field in these elements was given by the static marker protocol π, and then the static-decoder on these labels will indeed result in $F(u, v)$. ∎

4.1.6 Complexity

Let us now analyze the label size and the communication complexity of Protocol StatGL.

Lemma 4.10 *For every vertex v, every row i of $L(v)$ contains at most $d - 1$ non-empty elements.*

Proof: By induction on the number of vertices. The base case is after the initial assignment of the empty label to the root, where the claim holds trivially.

Now assume a new vertex is added as a child of u. Let $i = \mathrm{NF}(L(u))$. A level i shuffle γ is now performed. Let z be the ancestor of u such that $z = \texttt{p-root}(\gamma)$. By Corollary 4.3, row i of $L(z)$ is not full, since otherwise it would be full also in u. Therefore, the new labels assigned to the vertices in the subtree (which are $L(z)$ plus one new element in row i) satisfy the requirement. ∎

For a label L, denote by $c_i(L)$ the number of elements in row i of L, and let $c(L) = \sum_i c_i(L)d^i$.

Lemma 4.11 *For every vertex v in the n-vertex tree, $c(L(v)) \leq n$.*

Proof: By induction on the number of vertices. The base case is after the initial assignment of the empty label to the root, where the claim holds trivially. Now assume a new vertex is added as a child of u. Let γ be the level i shuffle resulting from this new addition. Let z be the ancestor of the new child such that $z = \texttt{p-root}(\gamma)$.

If v does not participate in this shuffle then v's label does not change, and we are done by the induction hypothesis. So suppose v participates in the shuffle. By Corollary 4.5, before the shuffle $c_j(L(u)) = d-1$ for every $0 \leq j \leq i-1$, so

$$c(L(u)) \;=\; c(L(z)) + \sum_{j=0}^{i-1}(d-1)d^j \;=\; c(L(z)) + d^i - 1.$$

Let $L(v)$ be v's new label after the shuffle. Since v participates in the shuffle, $L(v)$ is $L(z)$ plus one new element in row i, i.e., $c(L(v)) = c(L(z)) + d^i = c(L(u)) + 1$. By the induction hypothesis, the tree size before the new addition, $n-1$, was at least $c(L(u))$. Therefore the current tree size, n, is at least $c(L(v))$. ∎

By Lemma 4.11, for every vertex in an n-vertex tree, all rows from $\log_d n$ and on are empty. By Lemma 4.10, every row contains at most $d-1$ elements, and therefore the number of elements in any label in the tree contains at most $(d-1)\log_d n$ elements.

Let us now bound the size of an element $\langle l, R, s\rangle$ in a label. The field l is the result of a static marker protocol on at most n vertices, and therefore its representation is bounded above by $\mathcal{LS}(\pi, n)$. The field R equals $F(u, v)$ for some two vertices in the tree. The existence of a static labeling scheme π with labels of size at most $\mathcal{LS}(\pi, n)$ for F implies that R can be encoded using at most $2 \cdot \mathcal{LS}(\pi, n)$, by simply writing the labels of the two vertices. Finally, the child-number s of any vertex is of size at most $\log n$. Since we assumed the labels given by Protocol π are unique, $\log n \leq \mathcal{LS}(\pi, n)$. Altogether, the size of an element is $O(\mathcal{LS}(\pi, n))$.

We have the following bound on the size of a label.

Lemma 4.12

$$\mathcal{LS}(\textsc{SemGL}, n) \;=\; O\left(\frac{d-1}{\log d}\log n \cdot \mathcal{LS}(\pi, n)\right) .$$

Lemma 4.13 *Every vertex goes through at most one level i shuffle.*

Proof: Suppose v goes through a shuffle of level i. As a result, $\mathrm{NE}(L(v)) \geq i$. Consider a level i-shuffle initiated afterwards. The root of this shuffle, w, satisfies $\mathrm{NE}(L(w)) < i$. By Corollary 4.4, w is not an ancestor of v, and therefore v does not participate in the shuffle. ∎

Let us now bound the amount of communication resulting from level i shuffles. By Lemma 4.13, each vertex goes through at most one level i shuffle. Therefore the different level i shuffles act on disjoint sets A^i_j where $\bigcup_j A^i_j \subseteq V$. We assume $\mathcal{MC}(\pi, a) + \mathcal{MC}(\pi, b) \leq \mathcal{MC}(\pi, a+b)$ (see footnote 1). Therefore the communication resulting from level i shuffles is bounded above by

$$\sum_j \mathcal{MC}(\pi, |A^i_j|) \;\leq\; \mathcal{MC}(\pi, n).$$

For every level i shuffle, there is also extra communication that was used to start the shuffle, but this extra communication is always bounded by the size of the shuffle, and therefore the whole communication related to level i shuffles is always bounded above by $O(\mathcal{MC}(\pi, n))$

By Lemma 4.11 there are no shuffles of level greater than $\log_d n$. We therefore have

Lemma 4.14

$$\mathcal{MC}(\textsc{SemGL}, n) \;=\; O\left(\frac{\log n}{\log d}\mathcal{MC}(\pi, n)\right) \;.$$

Setting $d = 2$ we obtain the following (setting d to be any other constant yields the same asymptotic results).

Theorem 4.15 *1.* $\mathcal{LS}(\textsc{SemGL}, n) = O(\log n \cdot \mathcal{LS}(\pi, n))$,

2. $\mathcal{MC}(\textsc{SemGL}, n_0, n_+) = O(\log n_f \cdot \mathcal{MC}(\pi, n_f))$.

If n_f is known in advance, then setting $d = \log n_f$ we achieve a different tradeoff between the communication complexity and label size.

Theorem 4.16 *If n_f is known in advance, then*

1. $\mathcal{LS}(\textsc{SemGL}, n) = O(\frac{\log n_f \log n}{\log\log n_f}\mathcal{LS}(\pi, n))$,

2. $\mathcal{MC}(\textsc{SemGL}, n_0, n_+) = O(\frac{\log n}{\log\log n_f}\mathcal{MC}(\pi, n_f))$.

Note that if we want to keep the multiplicative overhead factor of the label size polylogarithmic we must set d to be polylogarithmic in n_f. In this case the asymptotic overhead factor in the communication is the same as setting $d = \log n_f$.

4.2 Protocol GL

Protocol GL uses Protocol SemGL in much the same way as Protocol DL uses Protocol SemDL. Again, the idea is to ignore deletions completely, while counting topological changes, until the number of topological changes becomes $\Theta(n)$, and then start Protocol SemGL from scratch. In this case, the size of the tree during each subexecution of Protocol SemGL, is $\Theta(n_0)$, where n_0 is the size of the tree at the beginning of the subexecution. We can therefore either set $d = 2$ for all subexecutions, or set $d = \log n_0$ with the appropriate n_0 for each subexecution.

Protocol GL operates as follows.

1. The root calculates and records the number of vertices in the tree n_0.

2. Protocols SemGL and ChangeWatch are started, where in Protocol SemGL, d is set to either 2 or $\log n_0$. The fact that a vertex v gets deleted does not change anything in the execution of Protocol SemGL, including the subexecutions of Protocol π.

3. When the estimated number of topological changes becomes at least $n_0/4$, return to Step 1.

Theorem 4.17 *Let* GL^1 *be* GL *setting* $d = 2$ *and* GL^2 *be* GL *setting* $d = \log n_0$. *Then*

1. $\mathcal{LS}(\mathrm{GL}^1, n) = O(\log n \cdot \mathcal{LS}(\pi, n))$,
2. $\mathcal{MC}(\mathrm{GL}^1, \bar{n}) = O\left(\sum_i \log n_i \frac{\mathcal{MC}(\pi, n_i)}{n_i}\right) + O(\sum_i \log^2 n_i)$,
3. $\mathcal{LS}(\mathrm{GL}^2, n) = O(\frac{\log^2 n}{\log\log n}\mathcal{LS}(\pi, n))$,
4. $\mathcal{MC}(\mathrm{GL}^2, \bar{n}) = O\left(\sum_i \frac{\log n_i}{\log\log n_i} \frac{\mathcal{MC}(\pi, n_i)}{n_i}\right) + O(\sum_i \log^2 n_i)$.

Proof: The proof proceeds in a similar fashion to that of Theorem 3.8. As before, after at least $n_0/5$ and at most $n_0/4$ topological changes, the protocol restarts by returning to Step 1. Consequently, in each subexecution the current tree size n is always of size $\Theta(n_0)$, the child-numbers of size $O(\log n)$, and consequently the label size is at most $O(2\log n \cdot \mathcal{LS}(\pi, n))$ in the case $d = 2$ and at most $O(\log^2 n/\log\log n \cdot \mathcal{LS}(\pi, n))$ in the second case, as desired.

Denote by $i_1, \ldots, i_k$ the times the protocol returns to Step 1. Denote by M_l the message complexity starting at initialization l and before the next initialization occurs. Clearly

$$\mathcal{MC}(\mathrm{GL}, \bar{n}) \;=\; \sum_{l=1}^{k} M_l \ .$$

M_l is the sum of the message complexities of Step 1 and Protocols SemGL and ChangeWatch during the relevant time period. Denote by d_l the value of d at the lth initialization (either $d_l = 2$ or $d_l = \log n_{i_l}$). Since the number of changes relevant to M_l is at most $n_{i_l}/4$, we get

$$M_l \;=\; O(n_{i_l}) + O\left(\frac{\log n_{i_l}}{\log d_l}\mathcal{MC}(\pi, n_{i_l})\right) + O\left(n_{i_l}\log^2 n_{i_l}\right) \ .$$

Since $n_{i_l} = \Theta(n_{i_{l+1}})$, and $\mathcal{MC}(\pi, \alpha k) = \Theta(\mathcal{MC}(\pi, k))$ (see footnote 1), actually

$$M_{l+1} \;=\; O\left(\frac{\log n_{i_l}}{\log d_l}\mathcal{MC}(\pi, n_{i_l})\right) + O(n_{i_l}\log^2 n_{i_l}),$$

and therefore

$$\begin{aligned}\mathcal{MC}(\mathrm{DL}, \bar{n}) \;&=\; \sum_{l=1}^{k} M_l \;=\; M_1 + \sum_{l=1}^{k-1} M_{l+1} \\ &=\; O(\sum_{l=1}^{k-1} \frac{\log n_{i_l}}{\log d_l}\mathcal{MC}(\pi, n_{i_l})) + O(\sum_{l=1}^{k-1} n_{i_l}\log^2 n_{i_l}) \ .\end{aligned}$$

It remains to show that for $l < k$,

$$\begin{aligned} \textstyle\sum_{j=i_l}^{i_{l+1}-1} \log^2 n_j &> \Omega(n_{i_l} \log^2 n_{i_l}) \\ \textstyle\sum_{j=i_l}^{i_{l+1}-1} \frac{\mathcal{MC}(\pi,n_j)}{n_j} &> \Omega(\mathcal{MC}(\pi, n_{i_l})) \ . \end{aligned}$$

This follows since all the relevant n_j's are of size $\Theta(n_{i_l})$ and there are at least $\Omega(n_{i_l})$ such j's, and therefore, $\mathcal{MC}(\pi, n_j) = \Theta(\mathcal{MC}(\pi, n_{i_l}))$. ∎

References

[AKM01] S. Abiteboul, H. Kaplan and T. Milo. Compact labeling schemes for ancestor queries. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2001.

[AAPS89] Y. Afek, B. Awerbuch, S.A. Plotkin and M. Saks. Local management of a global resource in a communication. *J. of the ACM*, pages 1–19, 1989.

[AGKR01] S. Alstrup, C. Gavoille, H. Kaplan and T. Rauhe. Identifying nearest common ancestors in a distributed environment. IT-C Technical Report 2001-6, The IT University, Copenhagen, Denmark, Aug. 2001.

[AR02] S. Alstrup and T. Rauhe. Improved Labeling Scheme for Ancestor Queries. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.

[BF67] M.A. Breuer and J. Folkman. An unexpected result on coding the vertices of a graph. *J. of Mathematical Analysis and Applications*, 20:583–600, 1967.

[Bre66] M.A. Breuer. Coding the vertexes of a graph. *IEEE Trans. on Information Theory*, IT-12:148–153, 1966.

[CHKZ02] E. Cohen, E. Halperin, H. Kaplan and U. Zwick. Reachability and Distance Queries via 2-hop Labels. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.

[FG01] P. Fraigniaud and C. Gavoille. Routing in trees. In *Proc. 28th Int. Colloq. on Automata, Languages & Prog.*, LNCS 2076, pages 757–772, July 2001.

[FG02] P. Fraigniaud and C. Gavoille. A space lower bound for routing in trees. In *Proc.* 19^{th} *Symp. on Theoretical Aspects of Computer Science*, Mar. 2002.

[GP01a] C. Gavoille and C. Paul. Split decomposition and distance labelling: an optimal scheme for distance hereditary graphs. In *Proc. European Conf. on Combinatorics, Graph Theory and Applications*, Sept. 2001.

[GP01b] C. Gavoille and D. Peleg. Compact and Localized Distributed Data Structures. Research Report RR-1261-01, LaBRI, Univ. of Bordeaux, France, Aug. 2001.

[GKKPP01] C. Gavoille, M. Katz, N.A. Katz, C. Paul and D. Peleg. Approximate Distance Labeling Schemes. In *9th European Symp. on Algorithms*, Aug. 2001, Aarhus, Denmark, SV-LNCS 2161, 476–488.

[GPPR01] C. Gavoille, D. Peleg, S. Pérennes and R. Raz. Distance labeling in graphs. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, pages 210–219, Jan. 2001.

[KNR88] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 334–343, May 1988.

[KM01] H. Kaplan and T. Milo. Short and simple labels for small distances and other functions. In *Workshop on Algorithms and Data Structures*, Aug. 2001.

[KM01a] H. Kaplan and T. Milo. Parent and ancestor queries using a compact index. In *Proc. 20th ACM Symp. on Principles of Database Systems*, May 2001.

[KMS02] H. Kaplan, T. Milo and R. Shabo. A Comparison of Labeling Schemes for Ancestor Queries. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.

[KKKP02] M. Katz, N.A. Katz, A. Korman and D. Peleg. Labeling schemes for flow and connectivity. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.

[KKP00] M. Katz, N.A. Katz, and D. Peleg. Distance labeling schemes for well-separated graph classes. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science*, pages 516–528, February 2000.

[KP03] A. Korman and D. Peleg. Labeling Schemes for Weighted Dynamic Trees. In *Proc. 30th Int. Colloq. on Automata, Languages & Prog.*, Eindhoven, The Netherlands, July 2003, Springer LNCS.

[Pel99] D. Peleg. Proximity-preserving labeling schemes and their applications. In *Proc. 25th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 30–41, June 1999.

[Pel00] D. Peleg. Informative labeling schemes for graphs. In *Proc. 25th Symp. on Mathematical Foundations of Computer Science*, volume LNCS-1893, pages 579–588. Springer-Verlag, Aug. 2000.

[Pel00a] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[SK85] N. Santoro and R. Khatib. Labelling and implicit routing in networks. *The Computer Journal*, 28:5–8, 1985.

[T01] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science*, Oct. 2001.

[TZ01] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architecture*, pages 1–10, Hersonissos, Crete, Greece, July 2001.