Labeling Schemes for Tree Representation*

Reuven Cohen † Pierre Fraigniaud ‡ David Ilcinkas ‡
Amos Korman § David Peleg ¶

Corresponding author:
David Ilcinkas
LRI, Université Paris-Sud, Bât 490
91405 Orsay Cedex FRANCE
Phone: +33 169 153 106
Fax: +33 169 156 586

Abstract

This paper deals with compact label-based representations for trees. Consider an n-node undirected connected graph G with a predefined numbering on the ports of each node. The *all-ports* tree labeling $\mathcal{L}_{all}$ gives each node v of G a label containing the port numbers of all the *tree* edges incident to v. The *upward* tree labeling $\mathcal{L}_{up}$ labels each node v by the number of the port leading from v to its parent in the tree. Our measure of interest is the worst case and total length of the labels used by the scheme, denoted $M_{up}(T)$ and $S_{up}(T)$ for $\mathcal{L}_{up}$ and $M_{all}(T)$ and $S_{all}(T)$ for $\mathcal{L}_{all}$. The problem studied in this paper is the following: Given a graph G and a predefined port labeling for it, with the ports of each node v numbered by $0, \ldots, \deg(v) - 1$, select a rooted spanning tree for G minimizing (one of) these measures. We show that the problem is polynomial for $M_{up}(T)$, $S_{up}(T)$ and $S_{all}(T)$ but NP-hard for $M_{all}(T)$ (even for 3-regular planar graphs). We show that for every graph G and port labeling there exists a spanning tree T for which $S_{up}(T) = O(n \log \log n)$. We give a tight bound of $O(n)$ in the cases of complete graphs with arbitrary labeling and arbitrary graphs with symmetric port labeling. We conclude by discussing some applications for our tree representation schemes.

Keywords: Labeling scheme, spanning tree, tree representation.

---

†Dept. of Electrical and Computer Eng., Boston University, Boston, MA, USA. E-mail: cohenr@bu.edu. Supported by the Pacific Theaters Foundation.

‡CNRS, LRI, Université Paris-Sud, France. E-mail: {pierre,ilcinkas}@lri.fr. Supported by the project "PairAPair" of the ACI Masses de Données, the project "Fragile" of the ACI Sécurité et Informatique, and by the project "Grand Large" of INRIA.

§Information Systems Group, Faculty of IE&M, The Technion, Haifa, Israel. E-mail: pandit@tx.technion.ac.il. Supported in part by an Aly Kaufman fellowship.

¶Dept. of Computer Science, The Weizmann Institute, Rehovot, Israel. E-mail: david.peleg@weizmann.ac.il. Supported in part by a grant from the Israel Science Foundation.

*A preliminary version of this paper has appeared in the proceedings of the 7th International Workshop on Distributed Computing (IWDC), Kharagpur, India, December 27-30, 2005, as part of [5].

1 Introduction

This paper deals with compact label-based representations for trees. It aims at providing, during a pre-processing stage, compact encodings of various representations of spanning trees. The encoded spanning trees can later be used for efficient distributed computations such as broadcast, convergecast, exploration, etc. We care specifically of the size of the encoding as we wish to store spanning trees while using a minimum amount of local memory resources. In particular, we seek encodings in which every node stores a constant number of bits, or at least in which the average number of bits stored at each node is constant. Some of our encodings can be computed in a distributed way, during a small number of computation steps. We stress however our desire to design compact labels. Whether or not these labels can be computed distributively is an important issue, but, as far as this paper is concerned, it is a secondary issue. Our objective is, up to some extent, similar to the one in compact routing schemes (e.g [23, 1, 11]), and in other types of informative labeling schemes (e.g [17, 18, 19, 4]), that is the design of compact data structures (labels) in a preprocessing stage, to be used (later) for solving network problems efficiently. Ideally, the pre-computed data structures are obtained in a distributed manner, but the main point is that they must be of small size.

Consider an n-node undirected connected graph G. Assume that we are given also a predefined numbering on the ports of each node, i.e., every edge e incident to a node u is given an integer label $l_u(e)$ in $\{0, \ldots, \deg(u) - 1\}$ so that $l_u(e) \neq l_u(e')$ for any two distinct edges e and e' incident to u. In general, one may consider two types of schemes for representing a spanning tree in a given graph. An *all-ports* tree representation has to ensure that each node in the graph knows the port labels of all its incident tree edges. An *upward* tree representation has to ensure that each node in the graph knows the port label of the unique tree edge connecting it to its parent. Such representations find applications in the areas of data structures, distributed computing, communication networks and others.

Corresponding to the two general representation types discussed above, we consider two label-based schemes. The *all-ports* tree labeling $\mathcal{L}_{all}$ labels each node v of G by a label containing the port numbers of all the *tree* edges incident to v. The *upward* tree labeling $\mathcal{L}_{up}$ labels each node v of G by the number of the port connected to the unique edge e of T leading from v toward the root. We use the standard binary representation of positive integers to store the port labels.

Our measure of interest is the worst case or average length of the labels used by tree labeling schemes. Let us formalize these notions. Given a graph G (including a port labeling) and a spanning tree T for G,

- the sum of the label sizes in the labeling $\mathcal{L}_{up}$ (respectively, $\mathcal{L}_{all}$) on T is denoted by $S_{up}(T)$ (resp., $S_{all}(T)$);
- the maximum label size in the labeling $\mathcal{L}_{up}$ (respectively, $\mathcal{L}_{all}$) on T is denoted by $M_{up}(T)$ (resp., $M_{all}(T)$).

The problem studied in this paper is the following: Given a graph G and a predefined port labeling for it, with the ports of each node v numbered by $0, \ldots, \deg(v) - 1$, select a rooted spanning tree T for G minimizing (one of) these measures.

We show that there are polynomial time algorithms that given a graph G and a port labeling, construct a spanning tree T for G minimizing $M_{up}(T)$ or $S_{up}(T)$. Moreover, we conjecture that for every graph G, and any port labeling for G, there exists a tree T spanning G, for which $S_{up}(T) = O(n)$. In other words, we conjecture that there is a tree for which the upward labeling requires a constant number of bits per node on average.

We establish the correctness of this conjecture in the cases of complete graphs with arbitrary labeling and arbitrary graphs with symmetric port labeling. For arbitrary graph, we show a weaker algorithm, constructing for a given graph G (with its port labeling) a spanning tree T with $S_{up}(T) = O(n \log \log n)$.

Turning to all-port labeling schemes, for any spanning tree T the labeling $\mathcal{L}_{all}$ has average label size $O(\log \Delta)$ in graphs of maximum degree Δ, which is optimal on some n-node graphs of maximum degree Δ. It turns out that here there is a difference between the measures $S_{all}(T)$ and $M_{all}(T)$. We show that there is a polynomial time algorithm that given a graph G and a port labeling, constructs a tree T minimizing $S_{all}(T)$. In contrast, the problem of deciding, for a given graph G with a port labeling and an integer k, whether there exists a spanning tree T of G satisfying $M_{all}(T) \leq k$ is NP-hard. This holds even restricted to 3-regular planar graphs, and even for fixed $k = 3$. Nevertheless, denoting the smallest maximum degree of any spanning tree for the graph G by δ_{min}, there is a polynomial time approximation of the tree of minimum $M_{all}(T)$, up to a multiplicative factor of $O(\log \Delta / \log \delta_{min})$.

We conclude by discussing some applications for our tree representation schemes, including basic distributed operations such as broadcast, convergecast and graph exploration. A number of well-known solutions to these problems (cf. [21, 2, 22]) are based on maintaining a spanning tree for the network and using it for efficient communication. All standard spanning tree constructions that we are aware of do not take into account the memory required to store the spanning tree, and subsequently, the resulting tree may in general require a total of up to $O(n \log \Delta)$ memory bits over an n-node network of maximum degree Δ. Using the tree representations developed herein may improve the memory requirements of storing the tree representation. For instance, for applications that require only an upward tree representation, our construction yields a total memory requirement of $O(n \log \log n)$ bits, which is lower in high degree graphs. These applications are discussed in more detail in Section 4.

The all-port labeling scheme is particularly convenient for broadcast applications because it minimizes the number of messages. For less demanding tasks such as graph exploration, more compact labeling schemes can be defined. In particular, [4] describes a labeling scheme which uses only three different labels and allows a finite automaton to perform exploration in time at most $O(m)$ on m-edge graphs.

2 Upward tree labeling schemes with short labels

2.1 Basic properties

Let us first establish a naive upper bound on $S_{up}(T)$ and $M_{up}(T)$. In the basic upwards tree labeling scheme, the label kept at each node v is the port label of the tree edge leading from v toward the root. Hence no matter which tree is selected, the label assigned to each node v by the upwards tree labeling scheme uses at most $\lfloor \log \deg(v) \rfloor + 1$ bits. This implies the following bounds. (Throughout, some proofs are omitted.)

Lemma 2.1 *Let $n \geq 1$. For every n-vertex connected graph G of maximum degree Δ, and for every spanning tree T of G, we have*

1. $M_{up}(T) \leq \lfloor \log \Delta \rfloor + 1$ *and*
2. $S_{up}(T) \leq \sum_v (\lfloor \log \deg(v) \rfloor + 1)$.

Note that the second part of the lemma implies that in graph families with a linear number of edges, such as planar graphs, the average label size for any spanning tree is at most $O(1)$.

Given $G = (V, E)$, let $\vec{G} = (V, X)$ be the directed graph in which every edge $\{u, v\}$ in E corresponds to two arcs (u, v) and (v, u) in X. The arcs of $\vec{G}$ are weighted according to the port labeling of the edges in G, i.e., the arc (u, v) of $\vec{G}$ has weight

$$\omega(u, v) = \begin{cases} 1, & p = 0, \\ \lfloor \log p \rfloor + 1, & p \geq 1, \end{cases}$$

where p is the port label at u of the edge $\{u, v\}$ in G. That is $\omega(u, v)$ is the number of bits in the standard binary representation of positive integers required to encode* port label p.

Finding a spanning tree T minimizing $M_{up}(T)$ is easy by identifying the smallest k such that the digraph $\vec{G}_k$ obtained from $\vec{G}$ by removing all arcs of weight greater than $\lfloor \log k \rfloor + 1$, contains a spanning tree directed toward the root.

A distributed implementation of this procedure may be based on a binary search for the correct k value. In each experiment, "removing" an arc can be implemented by assigning it an infinite weight; if the resulting Directed Minimum weight Spanning Tree (DMST) has infinite weight then this k value is infeasible. Such implementation requires the use of a distributed algorithm for constructing a DMST, e.g., the algorithm of [16].

Thus we have the following.

Proposition 2.2 *There is a polynomial time algorithm that, given a graph G and a port labeling, constructs a spanning tree T for G minimizing $M_{up}(T)$. The algorithm is amenable to distributed implementation.*

*Note that this encoding is not a prefix coding and therefore might not be decodable. However, efficient encoding methods exist which are asymptotically optimal (cf. [10]) and therefore the overall results are also valid for such encoding.

Similarly, applying any sequential or distributed Minimum-weight Spanning Tree (MST) algorithm for digraphs (cf. [3, 9, 16]) on $\vec{G}$ with weight function ω, we get the following.

Proposition 2.3 *There is a polynomial time algorithm that, given a graph G and a port labeling, constructs a spanning tree T for G minimizing $S_{up}(T)$. The algorithm is amenable to distributed implementation.*

There are graphs for which the bound on M_{up} specified in Lemma 2.1 is reached for any spanning tree T (e.g., a graph composed of two Δ-regular graphs linked by a unique edge labeled Δ at both of its extremities). However, this is not the case for S_{up}, and we will show that, for any graph, there is a spanning tree T for which $S_{up}(T)$ is much smaller than the bound in Lemma 2.1.

2.2 Complete and symmetric graphs

First, consider the case of a complete graph with arbitrary labeling. We show that there exists a spanning tree T of it, for which $S_{up}(T) = O(n)$. We establish the claim by presenting an algorithm that yields a labeling of this cost. The algorithm is a variant of Kruskal's minimum-weight spanning tree (MST) algorithm (cf. [6]). The algorithm maintains a collection of rooted directed trees with the edges of each tree directed towards its root. Initially, each vertex forms a tree on its own. The algorithm merges these trees into larger trees until it remains with a single tree giving the solution.

The algorithm operates in phases. Let $\mathrm{size}(T)$ denote the size (number of nodes) of the tree T. A tree T is *small* for phase $k \geq 1$ if $\mathrm{size}(T) < 2^k$.

Each phase k of the algorithm consists of four steps. At the beginning of the phase, we identify the collection of small trees for the phase: $\mathcal{T}_{\mathrm{small}}(k) = \{T \mid \mathrm{size}(T) < 2^k\}$. Second, for each tree $T \in \mathcal{T}_{\mathrm{small}}(k)$ with root $r(T)$, we look at the set $S(T)$ of outgoing edges that connect $r(T)$ to nodes in other trees $T' \neq T$, and select the edge $e(T)$ of minimum weight in $S(T)$. (Note that $S(T) \neq \emptyset$ since the graph is complete.) Third, we add these edges to the collection of trees, thus merging the trees into 1-factors. Formally, a *1-factor* is a weakly-connected directed graph of out-degree 1. Intuitively, a 1-factor is a directed subgraph consisting of a directed cycle and a collection of directed trees rooted at the nodes of the cycle. Figure 1 illustrates two 1-factors. Finally, for the last of the four steps, in each 1-factor we arbitrarily select one of the edges on the cycle and erase it, effectively transforming the 1-factor back into a rooted directed tree.

This process is continued until a single tree remains, which is the desired tree.

Claim 2.4 *Denote the collection of trees at the beginning of the kth phase, $k \geq 1$, by $T_1^k, \ldots, T_{m_k}^k$.*

1. $\sum_{j=1}^{m_k} \mathrm{size}(T_j^k) = n$ *for every* $k \geq 1$*;*

2. $\mathrm{size}(T_j^1) = 1$ *for every* $1 \leq j \leq n$ *(observe that* $m_1 = n$*);*

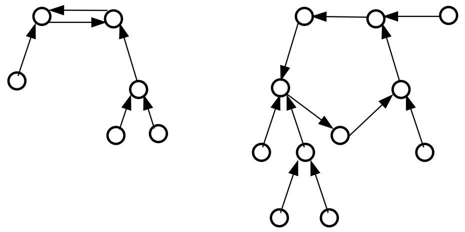

Figure 1: Two 1-factors.

3. $\text{size}(T_j^k) \geq 2^{k-1}$ *for every* $k \geq 1$ *and* $1 \leq j \leq m_k$*;*

4. $m_k \leq n/2^{k-1}$ *for every* $k \geq 1$.

5. *The number of phases is at most* $\lceil \log n \rceil$.

Proof. Parts 1 and 2 are immediate from the choice of the initial trees, and the way trees are merged at each phase. Part 3 is proved by induction on k. The basis is clear. Assuming the claim for phase k, we prove it for phase $k+1$ relying on the observation that every tree that is large for phase k already satisfies the required size bound for phase $k+1$, and each small tree of phase k is merged into a 1-factor consisting of at least two trees of phase k. Parts 4 and 5 readily follow from Part 3, and the fact that trees are disjoint. ■

Observe that when selecting the outgoing edge $e(T_j^k)$ for the root $r(T_j^k)$ on the kth phase, the only outgoing edges of $r(T_j^k)$ excluded from consideration are the $\text{size}(T_j^k) - 1$ edges leading to the other nodes in T_j^k. Hence even if all of these edges are "lighter" than the edges leading outside the tree, the port label used for $e(T_j^k)$ is at most $\text{size}(T_j^k) - 1$, hence:

$$\begin{cases} \omega(e(T_j^k)) = 1 & \text{if } k = 1 \\ \omega(e(T_j^k)) \leq \lfloor \log(\text{size}(T_j^k) - 1) \rfloor + 1 & \text{if } k > 1 \end{cases}$$

Moreover, we have $\log \text{size}(T_j^k) < k$ because outgoing edges are selected only for small trees, and thus we have $\omega(e(T_j^k)) \leq k$. Hence the total weight C_k of the edges added to the structure throughout the kth phase satisfies

$$C_k \leq \sum_{T_j^k \in \mathcal{T}_{\text{small}}(k)} k \quad = k \cdot |\mathcal{T}_{\text{small}}(k)| \leq k \cdot m_k.$$

By Part 4 of Claim 2.4, $C_k \leq kn/2^{k-1}$, and the total weight C of the resulting tree satisfies $C = \sum_{k \geq 1} C_k \leq \sum_{k \geq 1} kn/2^{k-1} \leq 4n$. We have thus shown the following.

Proposition 2.5 *On the complete graph (with an arbitrary port labeling), there exists a spanning tree* T *for which* $S_{up}(T) = O(n)$.

Next, we consider another interesting and potentially applicable special case, namely, arbitrary graphs with symmetric port labeling. We prove the following:

Proposition 2.6 *On graphs with symmetric port labeling (i.e., where for every edge $e = \{u, v\}$, the port labels of e at u and v are identical), there exists a spanning tree T for which $S_{up}(T) = O(n)$.*

Proof. For graphs with symmetric port labeling, we again present an algorithm that yields a labeling of cost $O(n)$. The algorithm is a variant of the one used for proving Property 2.5. The general structure of the algorithm is the same, i.e., it is based on maintaining a collection of rooted directed tree and merging them until remaining with a single tree. The main difference has to do with the fact that since the graph is not complete, it may be that for the small tree T under consideration, the set $S(T)$ is empty, i.e., all the outgoing edges of the root $r(T)$ go to nodes *inside* T.

Therefore, an additional step is needed, transforming T into a tree T' on the same set of vertices, with the property that the new root, $r(T')$, has an outgoing edge to a node outside T'.

This is done as follows. We look for the lightest (least port label) outgoing edge from some node x in T to some node outside T. Note that such an edge must exist so long as T does not span the entire graph G, as G is connected. Let $p(T) = (v_1, v_2, \ldots, v_j)$ be the path from $r(T)$ to x in T, where $r(T) = v_1$ and $v_j = x$. Transform the tree T into a tree T' rooted at x by reversing the directions of the edges along this path. (See Figure 2 where dashed edges represent the path from the original root to T.) Observe that by symmetry, the cost of T' is the same as that of T, so the proof can proceed as for Property 2.5. ■

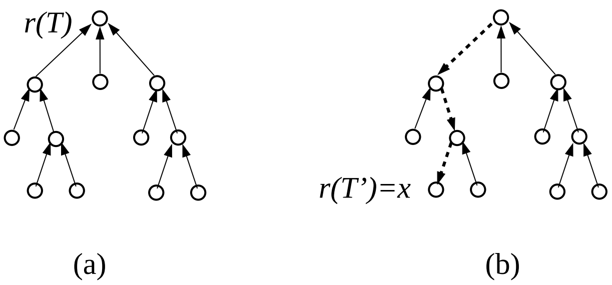

Figure 2: (a) The tree T. (b) The tree T'.

2.3 Arbitrary graphs

For the general setting, we show the universal bound of $O(n \log \log n)$ on S_{up}. Again, the algorithm yielding this cost is a variant of the one used for proving Property 2.5. As in the proof of Property 2.6, since the graph is not complete, it may be that for the small tree T under consideration, all the outgoing edges of the root $r(T)$ go to nodes *inside* T. It is thus necessary to transform T into a tree T' on the same set of vertices so that the new root $r(T')$ has an outgoing edge to a node outside T'. However, it is not enough to pick an arbitrary outgoing edge and make its internal endpoint the new root because, in the absence of symmetry, the reversed route may be much more expensive than the original path, thus causing the transformed tree to be too costly.

Instead, the transformation is performed as follows (cf. Fig. 3). We look for the shortest path (in hops) from the current root $r(T)$ to the node in T that is the closest to the root, and that has an outgoing edge to a node outside T. Moreover, all the nodes of the path must be in T. (Such a path must exist so long as T does not span the entire graph G, as G is connected.) Let this path be $p(T) = (v_1, v_2, \ldots, v_j)$, where (1) $r(T) = v_1$, (2) $v_1, \ldots, v_j \in T$, and (3) v_j has a neighbor $z \notin T$. For every $1 \leq i \leq j-1$, we add the edge (v_i, v_{i+1}) of $p(T)$ to T. In turn, for $2 \leq i \leq j$, we remove from T the (unique) outgoing edge of v_i in T, (v_i, w_i). The resulting subgraph is a directed tree T' rooted at $r(T') = v_j$. (Note that in case the original root $r(T)$ has an outgoing edge to some node z outside T, this transformation uses $p(T) = (r(T))$ and leaves T unchanged.)

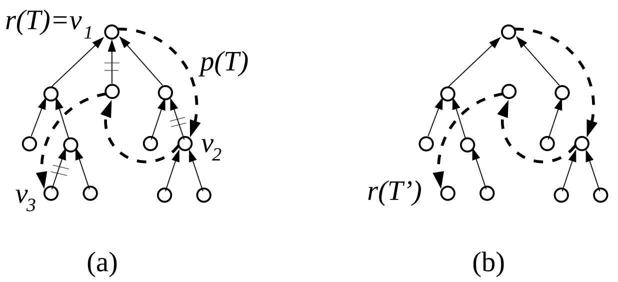

Figure 3: (a) The tree T and the escape path $p(T)$ (dashed). (b) The tree T'.

Clearly, applying these transformations on the small trees in each phase incurs additional costs. To estimate them, we bound from above the additional cost incurred by adding the paths $p(T)$ for every tree $T \in \mathcal{T}_{\text{small}}(k)$ in every phase k. For such a tree T with $p(T) = (v_1, v_2, \ldots, v_j)$, denote the set of nodes whose outgoing edge was replaced (hence whose labels may increase) by $A(T) = \{v_1, v_2, \ldots, v_{j-1}\}$, and let $A_k = \bigcup_{T \in \mathcal{T}_{\text{small}}(k)} A(T)$ and $A = \bigcup_k A_k$.

To effectively bound the cost increases, we rely on the following observation. A node v may participate in several paths $p(T)$ throughout the construction. Each time, it may replace its outgoing edge with a new one. Nevertheless, the cost it incurs in the final tree is just the cost of its *final* outgoing edge, since all the other outgoing edges added for it in earlier phases were subsequently replaced. By definition, this cost is at most $\lceil \log \deg(v) \rceil$ and hence, denoting $Z(W) = \sum_{v \in W} \lceil \log \deg(v) \rceil$ for a set of nodes W, the total additional cost incurred by the nodes of A is bounded by $Z(A)$. We would like to prove that $Z(A) = O(n \log \log n)$.

Claim 2.7 $\sum_{v \in A(T)} \deg(v) \leq 3 \cdot \text{size}(T)$ *for every tree* $T \in \mathcal{T}_{small}(k)$ *in every phase* $k \geq 1$.

Proof. Note that the nodes of $A(T)$ have all their neighbors inside T, hence their degrees in the (undirected) subgraph $G(T)$ induced by the nodes of T are the same as their degrees in G. Since $p(T)$ is a shortest path from v_1 to v_j in $G(T)$, we have that every node w in $G(T)$ has at most 3 neighbors in $p(T)$ (otherwise it would provide a shortcut yielding a shorter path between v_1 and v_j in $G(T)$, contradicting the assumption). Thus the number of edge ports in the nodes of $p(T)$ is at most $3 \cdot \text{size}(T)$. ■

It follows that $\sum_{v \in A_k} \deg(v) \leq 3n$ for every phase $k \geq 1$, and as the number of phases is at most $\lceil \log n \rceil$ by Claim 2.4, item 5, we have that $\sum_{v \in A} \deg(v) \leq 3n \lceil \log n \rceil$. By the arithmetic-geometric means inequality,

$$\frac{\sum_{v \in A} \log \deg(v)}{n} \leq \log \frac{\sum_{v \in A} \deg(v)}{n} \leq \log(3 \log n).$$

Therefore, $\sum_{v \in A} \log \deg(v) \leq n \log(3 \log n)$, yielding $Z(A) = O(n \log \log n)$.

Consequently, we have the following.

Theorem 2.8 *For every graph G and every port labeling, there exists a spanning tree T for G in which $S_{up}(T) = O(n \log \log n)$.*

3 All-ports tree labeling schemes with short labels

Let us now turn our attention to S_{all} and M_{all}. Any spanning tree T enables the construction of a labeling $\mathcal{L}_{all}$ with average label size $O(\log \Delta)$ in graphs of maximum degree Δ. This is optimal in the sense that there are n-node graphs of maximum degree Δ and port labeling for which $S_{all}(T) = \Omega(n \log \Delta)$ for any spanning tree T. For instance, take a bipartite graph $G = (V_1, V_2, E)$ where $V_i = \{(i, x), x = 0, \ldots, n-1\}$, $i = 1, 2$, and $\{(1, x), (2, y)\} \in E$ if and only if $(y - x) \bmod n \leq \Delta - 1$. Then, label any $\{(1, x), (2, y)\} \in E$ by $l = (y - x) \bmod n$ at $(1, x)$, and by $\Delta - l$ at $(2, y)$. For any tree T spanning G, at least one of the two labels at the extremity of every edge of T is larger than $\lfloor \Delta/2 \rfloor$, and therefore $S_{all}(T) \geq \Omega(n \log \Delta)$.

However, for many graphs, one can do better by selecting an appropriate spanning tree T. Assign a weight $\omega(l) + \omega(l')$, where

$$\omega(x) = \begin{cases} 1, & x = 0, \\ \lfloor \log x \rfloor + 1, & x \geq 1, \end{cases}$$

to every edge e where l and l' are the port labels of e at its two endpoints. It is easy to check that running any undirected MST algorithm returns a tree T minimizing $S_{all}(T)$. For a distributed implementation, one may use any distributed MST algorithm, e.g., the algorithm of [13] for optimizing message efficiency, or those of [14, 20] for optimizing time efficiency. Thus, we have the following.

Proposition 3.1 *There is a polynomial time algorithm that given a graph G and a port labeling constructs a tree T minimizing $S_{all}(T)$. The algorithm is amenable to distributed implementation.*

On the other hand, we have the following negative result.

Proposition 3.2 *The following decision problem is NP-hard.*
Input: *A graph G with a port labeling, and an integer k;*
Question: *Is there a spanning tree T of G satisfying $M_{all}(T) \leq k$.*
This result holds even restricted to 3-regular planar graphs, and even for fixed $k = 3$.

Proof. It is known [15] that it is NP-complete to decide whether a 3-regular planar graph has a Hamiltonian path (see also [7]). We reduce the Hamiltonian path problem in 3-regular planar graphs to our problem. In 3-regular graphs, ports are labeled 0, 1, and 2 at each node, using respectively 1, 1, and 2 bits. If there exists a spanning tree T such that $M_{all}(T) \leq 3$ then its maximum degree satisfies $\Delta_T \leq 2$ since $1 + 1 + 2 = 4$. Conversely, if $M_{all}(T) > 3$ then $\Delta_T > 2$ since $1 + 2 = 3$. Hence $M_{all}(T) \leq 3$ iff the graph has an Hamiltonian path. ■

Obviously, one way to obtain a tree T with small $M_{all}(T)$ is to construct a spanning tree with small maximum degree. Finding a spanning tree with the smallest maximum degree δ_{min} in an arbitrary graph G is NP-hard. However, it is known (cf. [12]) that a spanning tree with maximum degree at most $\delta_{min} + 1$ can be computed in polynomial time. Hence we have the following.

Theorem 3.3 *There is a polynomial time algorithm that given a graph G and a port labeling constructs a spanning tree T for G in which $M_{all}(T) = O(\delta_{min} \log \Delta)$.*

On the other hand, any tree T^* minimizing M_{all} in a graph G has a degree $\Delta_{T^*} \geq \delta_{min}$. Thus

$$M_{all}(T^*) \geq \sum_{i=1}^{\Delta_{T^*}} \log i \geq \sum_{i=1}^{\delta_{min}} \log i \geq \Omega(\delta_{min} \log \delta_{min}).$$

Hence we obtain a polynomial time approximation of the optimal tree for $\mathcal{L}_{all}$, up to a multiplicative factor of $O(\log \Delta / \log \delta_{min})$.

4 Applications of tree labeling schemes

Let us now discuss the applicability of our tree representation schemes in various application domains, mainly in the context of distributed network algorithms.

Throughout this section we consider an n-vertex m-edge graph G of maximum degree Δ, such that the smallest maximum degree of any spanning tree for G is δ_{min}.

4.1 Information dissemination on spanning trees

A number of fundamental distributed processes involve collecting information upwards or disseminating it downwards over a spanning tree of the network. Let us start with applications of our tree representation schemes for these operations.

4.1.1 Broadcast

The broadcast operation requires disseminating an information item initially available at the root to all the vertices in the network. Given a spanning tree of the graph, this operation can be performed more efficiently than by the standard flooding mechanism (cf. [21, 2, 22]). Specifically, whereas flooding requires $O(m)$ messages, broadcasting on a spanning tree can be achieved using only $O(n)$ messages.

Broadcast over a spanning tree can be easily performed given an all-ports tree representation scheme, with no additional communication overheads. Consider the overall memory requirements of storing such a representation. Using an arbitrary spanning tree may require a total of $O(n \log \Delta)$ memory bits throughout the entire network and a maximum of $O(\Delta \log \Delta)$ memory bits per node. In contrast, using the constructions of Property 3.1 or Theorem 3.3, respectively, yields the following bounds.

Corollary 4.1 *For any graph G, it is possible to construct an all-port spanning tree representation using either optimal total memory over the entire graph or maximum memory $O(\delta_{min} \log \Delta)$ per node, in a way that will allow performing a broadcast operation on the graph using $O(n)$ messages.*

4.1.2 Upcast and convergecast

The basic *upcast* process involves collecting information upwards over a spanning tree, towards the root. This task is rather general, and refers to a setting where each vertex v in the tree has an input item x_v and it is required to communicate all the different items to the root. Analysis and applications of this operation can be found, e.g., in [22]. Any representation for supporting such operation must allow each vertex to know its parent in the tree.

Again, using an arbitrary spanning tree may require a total of $O(n \log \Delta)$ memory bits throughout the network. Observe, however, that the upcast process does not require knowing the children so it can be based on an upwards tree representation scheme. Given such a representation, the upcast process can be implemented with no additional overheads in communication. Hence using the construction of Theorem 2.8 we get the following.

Corollary 4.2 *For any graph G, it is possible to construct an upwards tree representation using a total of $O(n \log\log n)$ memory bits over the entire graph in a way that will allow performing an upcast operation on the graph using $O(n)$ messages.*

A more specialized process, known as the *convergecast* process, involves collecting information of the same type upwards over a spanning tree. This process may include the computation of various types of global functions. Suppose that each vertex v in the graph holds an input x_v and we would like to compute some global function $f(x_{v_1}, \ldots, x_{v_n})$ of these inputs. Suppose further that f is a *semigroup function*, namely, it enjoys the following two properties:

1. $f(Y)$ is well-defined for any subset $Y \subseteq \{x_{v_1}, \ldots, x_{v_n}\}$ of the inputs,

2. f is associative and commutative.

A semigroup function f can be computed efficiently on a tree T by a convergecast process, in which each vertex v in the tree sends upwards the value of the function on the inputs of the vertices in its subtree T_v, namely, $f_v = f(X_v)$ where $X_v = \{x_w \mid w \in T_v\}$. An intermediate vertex v with k children $w_1, \ldots, w_k$ computes this value by receiving the values $f_{w_i} = f(X_{w_i})$, $1 \le i \le k$, from its children, and applying $f_v \leftarrow f(x_v, f_{w_1}, \ldots, f_{w_k})$, relying on the associativity and commutativity of f. The message and time complexities of the convergecast algorithm on a tree T are $O(n)$ and $O(Depth(T))$, respectively, matching the obvious lower bounds. For a more detailed exposition of the convergecast operation and its applications see [22].

Examples for semigroup functions include addition, maximum or logical condition computation. The convergecast process can also be applied in order to collect global knowledge from local information. Specifically, suppose each vertex v holds a bit variable x_v. It is then possible to accumulate this information efficiently, ending up with the root knowing, for instance, whether there exists some vertex v with $x_v = 1$ or whether all vertices v satisfy $x_v = 1$. The former can be achieved by setting f to the logical "or" function and performing a global convergecast computation as above. Computing $\forall v, x_v = 1$ is achievable in a similar way, setting f to be the logical "and" function. In particular, the common acknowledgement process used for "broadcast with echo" can be implemented by this type of convergecast by setting $x_v = 1$ iff v has received the message. (This is a rather trivial use of the logical "and" computation, as all inputs are 1 once defined, hence so is the output.) For a more detailed exposition of the convergecast operation and its applications see [22].

Observe that the convergecast process requires each vertex to receive messages from all its children before it can send a message upwards to its parent. This implies, in particular, that a vertex needs to know the number of children it has in the tree. This means that when using the spanning tree T, the label size at each node v has another component of $\log(\deg_T(v))$. Hence the maximum label size increases by $\log \delta_{min}$, and the average label size increases by $\frac{1}{n}\sum_v \log(\deg_T(v)) = O(1)$.

Here, too, using an arbitrary spanning tree would require a total of $O(n \log \Delta)$ memory bits throughout the network. In contrast, using the construction of Theorem 2.8 we get the following.

Corollary 4.3 *For any graph G, it is possible to construct an upwards tree representation using a total of $O(n \log\log n)$ memory bits over the entire graph in a way that will allow performing a convergecast operation on the graph in time at most $Diam(G)$ using $O(n)$ messages.*

4.2 Fast graph exploration

Graph exploration is an operation carried out by a finite automaton, simply referred to in this context as a *robot*, moving in an unknown graph $G = (V, E)$. The robot has no a priori information about the topology of G and its size. The robot can distinguish between the edges of the currently visited node by their port labels. The robot has a transition function f, and a finite number of states. If the robot enters a node of degree d through port i in state s, then it switches to state s' and exits the node through port i', where $(s', i') = f(s, i, d)$. The objective of the robot is to *explore* the graph, i.e., to visit all its nodes.

The tree labeling schemes allow fast exploration. Specifically, the all-ports labeling scheme $\mathcal{L}_{all}$ allows exploration to be performed in time at most $2n$ in n-node graphs. The upward labeling scheme $\mathcal{L}_{up}$ allows exploration to be performed in time at most $4m$ in m-edge graphs.

More compact labeling schemes can be defined for graph exploration. In particular,[4] describes a labeling scheme using only 2 bits per node. However, this latter scheme yields slower exploration protocols, i.e., ones requiring $20m$ steps in m-edge graphs.

Suppose our graph G has a spanning tree T. As a consequence of [8], if the labels allow the robot to infer at each node v, for each edge e incident to v in G, whether e belongs to T, then it is possible to traverse G perpetually, and traversal is ensured after time at most $2n$. Indeed, the exploration procedure in [8], which applies to trees only, specifies that when the robot enters node v by port i, it leaves the node by port $(i+1) \bmod d$ where $d = \deg(v)$. In the case of general graphs, exploration is performed as follows. When the robot enters node by port i, it looks for the first j in the sequence $i+1, i+2, \ldots$ such that port $j \bmod d$ is incident to a tree-edge and leaves the node by port $j \bmod d$.

Clearly, this exploration procedure performs a DFS traversal of T. Hence, as a corollary of [8], using the all-ports labeling scheme $\mathcal{L}_{all}$, we get the following.

Corollary 4.4 *It is possible to label the nodes of every graph G in polynomial time, with labels of maximum size $O(\delta_{min} \log \Delta)$ and average size $O(\log \Delta)$, in a way that will allow traversal of the graph in time at most $2n$ by a robot with no memory.*

The following result shows that exploration can be performed with smaller labels, using the upward labeling scheme on a spanning tree of the graph.

Lemma 4.5 *Consider a node-labeled m-edge graph G, with a rooted spanning tree T. It is possible to perform traversal of G within time at most $4m$, terminating at the root of T.*

Proof. The exploration will use the upward labeling scheme $\mathcal{L}_{up}$ on T. At node u, we denote by $\mathrm{root}(u)$ the port-label corresponding to the edge of T leading to the root. At the root r, $\mathrm{root}(r) = -1$. If not initially at the root, the robot follows the edges leading to the root in state TO_ROOT. Once at the root, it starts the traversal and leaves the root by port 0 in state DOWN.

When the robot enters a node u of degree d by port i in state DOWN, it proceeds as follows. If $i \neq \mathrm{root}(u)$, then the robot backtracks through port i in state UP. If $i = \mathrm{root}(u)$, then the robot leaves u through port $(i+1) \bmod d$ in state DOWN, unless $d = 1$, in which case, it leaves u by port 0 in state UP.

When the robot enters a node u of degree d by port i in state UP, it proceeds as follows. Let $j = (i+1) \bmod d$. If $\mathrm{root}(u) = -1$, then the robot stops if $j = 0$, and leaves u by port

j otherwise. If $\text{root}(u) \neq -1$, then the robot leaves u via port j in state UP if $j = \text{root}(u)$, else it leaves u via port j in state DOWN.

During the traversal, every tree-edge is traversed twice, once in each direction. Every non tree-edge $\{u, v\}$ is traversed four times: from u to v, and backtrack from v to u; and from v to u, and backtrack from u to v. Initially, the robot traverses at most $n-1$ edges to go to the root. ■

By Lemma 4.5, using a labeling $\mathcal{L}_{up}$ on an arbitrary spanning tree and relying on Lemma 2.1 and Theorem 2.8, we get the following.

Corollary 4.6 *It is possible to label the nodes of every graph G with labels of maximum size $O(\log \Delta)$ and average size $O(\log \log n)$ in a way that will allow traversal of the graph in time at most $4m$.*

Note that by Lemma 2.1, the scheme uses labels of *total* size at most $\sum_v \lceil \log \deg(v) \rceil$. This means, in particular, that in graph families with a linear number of edges, such as planar graphs, the average label size for any spanning tree is at most $O(1)$.

4.3 Representing goal-seeking game trees

The upward tree labeling scheme $\mathcal{L}_{up}$ can be used to succinctly represent the solution tree for certain games whose course progresses only upwards in the tree. This includes various goal-seeking games in which, starting from an arbitrary initial position, the goal of the game is to reach a unique predefined end-position. (A concrete example for such a game is Rubik's cube.)

References

[1] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. In *Proc. 16th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 20–24. ACM Press, July 2004.

[2] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw-Hill, 1998.

[3] Y. Chu and T. Liu. On the shortest arborescence of a directed graph. *Science Sinica* 14, pp. 1396–1400, 1965.

[4] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman and D. Peleg. Label-Guided Graph Exploration by a Finite Automaton. In *Proc. 32nd Int. Colloq. on Automata, Languages & Prog. (ICALP)*, LNCS 3580, 335–346, 2005.

[5] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman and D. Peleg. Labeling Schemes for Tree Representation. Proc. of 7th International Workshop on Distributed Computing (IWDC), LNCS 3741, 13–24, 2005.

[6] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[7] A. Czumaj and W.-B. Strothmann. Bounded-degree spanning tree. In *Proc. 5th European Symp. on Algorithms (ESA)*, LNCS 1284, pages 104–117, 1997.

[8] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree Exploration with Little Memory. In *Proc. 13th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 588–597, 2002.

[9] J. Edmonds. Optimum branchings. *J. Research of the National Bureau of Standards* 71B, pp. 233–240, 1967.

[10] P. Elias. Universal Codeword Sets and Representations of the Integers. *IEEE Trans. Inform. Theory* 21(2):194–203, 1975.

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

[12] M. Fürer and B. Raghavachari. Approximating the minimum degree spanning tree within one from the optimal degree. In *Proc. 3rd ACM-SIAM Sump. on Discrete Algorithms (SODA)*, pages 317–324, 1992.

[13] Robert G. Gallager, Pierre A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. on Programming Lang. and Syst.*, 5(1):66–77, 1983.

[14] J. Garay, S. Kutten, and D. Peleg. A sub-linear time distributed algorithm for minimum-weight spanning trees. *SIAM J. on Computing*, 302–316, 1998.

[15] M. Garey, D. Johnson, and R. Tarjan. The planar Hamiltonian circuit is NP-complete. *SIAM Journal on Computing* 5(4):704–714, 1976.

[16] P. Humblet. A distributed algorithm for minimum weighted directed spanning trees. *IEEE Trans. on Communications*, v.COM-31, n.6, 1983, pp.756–762.

[17] S. Kannan, M. Naor, and S. Rudich. Implicit Representation of Graphs. In *SIAM J. on Descrete Math* **5**, (1992), 596–603.

[18] M. Katz, N.A. Katz, A. Korman and D. Peleg. Labeling Schemes for Flow and Connectivity. *SIAM Journal on Computing* **34** (2004), 23–40.

[19] A. Korman, S. Kutten, and D. Peleg. Proof Labeling Schemes. In *Proc. the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, Las Vegas, NV, USA (2005).

[20] S. Kutten and D. Peleg. Fast distributed construction of k-dominating sets and applications. *J. of Algorithms*, pages 40–66, 1998.

[21] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1995.

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

[23] M. Thorup. Compact Oracles for Reachability and Approximate Distances in Planar Digraphs. *J. of the ACM* **51**, (2004), 993–1024.