Labeling Schemes for Tree Representation

Reuven Cohen[1], Pierre Fraigniaud[2] ⋆, David Ilcinkas[2] ⋆, Amos Korman[1], and David Peleg[1]

[1] Dept. of Computer Science, Weizmann Institute, Israel
{r.cohen,amos.korman,david.peleg}@weizmann.ac.il
[2] CNRS, LRI, Université Paris-Sud, France
{pierre,ilcinkas}@lri.fr

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 numbering 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 assignments. We conclude by discussing some applications for our tree representation schemes.

1 Introduction

This paper deals with compact label-based representations for trees. 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 numbers of all its incident tree edges. An *upward* tree representation has to ensure that each node in the graph knows the port number of the tree edge connecting it to its parent. Such representations find applications in the areas of data structures, distributed computing, communication networks and others.

⋆ Supported by project "PairAPair" of the ACI Masses de Données, project "Fragile" of the ACI Sécurité et Informatique, and project "Grand Large" of INRIA.

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 edge e of T leading from v toward the root. We use the standard binary representation of positive integers to store the port numbers.

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 numbering) 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., $M_{all}(T)$).

This paper studies the following problem. Given a graph G and a predefined port labeling for it, with the ports of each node v numbered $0, \ldots, \deg(v) - 1$, 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 numbering, 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 numbering 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 assignments. For arbitrary graph, we show a weaker algorithm, constructing for a given graph G (with its port numbering) 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 numbering, constructs a tree T minimizing $S_{all}(T)$. In contrast, the problem of deciding, for a given graph G with a port numbering 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. [11, 1, 12]) 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, [3] 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 number 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 $\lceil \log \deg(v) \rceil$ bits. This implies the following bounds. (Throughout, some proofs are omitted.)

Lemma 1. *For every n-vertex graph G of maximum degree Δ, and for every spanning tree T of G, we have (1) $M_{up}(T) \le \lceil \log \Delta \rceil$, and (2) $S_{up}(T) \le \sum_v \lceil \log \deg(v) \rceil$.*

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 $\boldsymbol{G} = (V, X)$ 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 $\boldsymbol{G}$ are weighted according to the port numbering of the edges in G, i.e., the arc (u, v) of $\boldsymbol{G}$ has weight

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

where p is the port number 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[3] port number p.

Finding a spanning tree T minimizing $M_{up}(T)$ is easy by identifying the smallest k such that the digraph $\boldsymbol{G}_k$ obtained from $\boldsymbol{G}$ by removing all arcs of

[3] 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. [8]) and therefore the overall results are also valid for such encoding.

weight greater than $\lfloor \log k \rfloor + 1$, contains a spanning tree directed toward the root. Thus we have the following.

Proposition 1. *There is a polynomial time algorithm that, given a graph G and a port numbering, constructs a spanning tree T for G minimizing $M_{up}(T)$.*

Similarly, applying any Minimum-weight Spanning Tree (MST) algorithm for digraphs (cf. [2, 7]) on $\boldsymbol{G}$ with weight function ω, we get the following.

Proposition 2. *There is a polynomial time algorithm that, given a graph G and a port numbering, constructs a spanning tree T for G minimizing $S_{up}(T)$.*

There are graphs for which the bound on M_{up} specified in Lemma 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 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. [4]). The algorithm maintains a collection of rooted directed tree 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 $\text{size}(T)$ denote the size (number of nodes) of the tree T. A tree T is *small* for phase $k \geq 1$ if $\text{size}(T) < 2^k$.

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: $\mathcal{T}_{\text{small}}(k) = \{T \mid \text{size}(T) < 2^k\}$. Second, for each tree $T \in \mathcal{T}_{\text{small}}(k)$ 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. 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. $\sum_{j=1}^{m_k} \text{size}(T_j^k) = n$ for every $k \geq 1$;

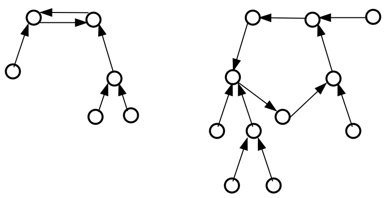

Fig. 1. Two 1-factors.

2. $\mathrm{size}(T_j^1) = 1$ for every $1 \le j \le n$ (observe that $m_1 = n$);
3. $\mathrm{size}(T_j^k) \ge 2^{k-1}$ for every $k \ge 1$ and $1 \le j \le m_k$;
4. $m_k \le n/2^{k-1}$ for every $k \ge 1$.
5. The number of phases is at most $\lceil \log n \rceil$.

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 $\mathrm{size}(T_j^k) - 1$ 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 number used for $e(T_j^k)$ is at most $\mathrm{size}(T_j^k) - 1$, hence:

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

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

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

By Part 4 of Claim 2.2, $C_k \le kn/2^{k-1}$, and the total weight C of the resulting tree satisfies $C = \sum_{k \ge 1} C_k \le \sum_{k \ge 1} kn/2^{k-1} \le 4n$. We have the following.

Proposition 3. *On the complete graph (with an arbitrary port numbering), 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 assignment.

Proposition 4. *On graphs with symmetric port assignments (i.e., where for every edge $e = \{u, v\}$, the port numbers 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 assignments, 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 3. 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 number) 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 3. □

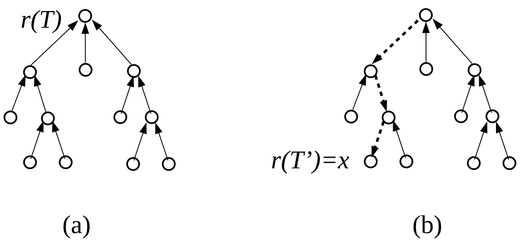

Fig. 2. (a) The tree T. (b) The tree T'.

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 3. As in the proof of Property 4, 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 $1 \le i \le j-1$, we add the edge (v_i, v_{i+1}) of $p(T)$ to T. In turn, for $2 \le i \le j$, 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.)

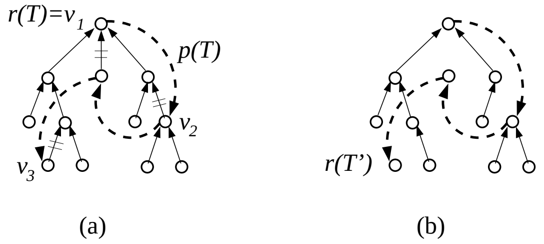

Fig. 3. (a) The tree T and the escape path $p(T)$ (dashed). (b) The tree T'.

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 $A_k = \bigcup_{T \in \mathcal{T}_{\text{small}}(k)} A(T)$.

Partition the nodes of the graph G into classes by their degrees, setting

$$D_\ell = \{v \mid 2^{\ell-1} < \deg(v) \le 2^\ell\}$$

for $\ell \ge 0$. Define $A^\ell(T) = A(T) \cap D_\ell$ and $A_k^\ell = A_k \cap D_\ell = \bigcup_{T \in \mathcal{T}_{\text{small}}(k)} A^\ell(T)$.

Claim. $\sum_{v \in A(T)} \deg(v) \le 3 \cdot \text{size}(T)$ for every phase $k \ge 1$ and tree $T \in \mathcal{T}_{\text{small}}(k)$.

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 $3 \cdot \text{size}(T)$. □

Claim. $|A_k^\ell| \le 3n/2^{\ell-1}$ for every phase 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. Denote this cost by $X(v)$. (For nodes that did not incur such costs at all throughout the execution, let $X(v) = 0$.) For a set of nodes W, let $X(W) = \sum_{v \in W} X(v)$.

Claim. For every ℓ, (1) $X(D_\ell) \leq \ell \cdot |D_\ell|$, and (2) $X(D_\ell) \leq 3\ell n \lceil \log n \rceil / 2^{\ell-1}$.

Proof. By the definition of D_ℓ, we have $X(v) \leq \lceil \log \deg(v) \rceil \leq \ell$ for every node $v \in D_\ell$. Part (1) of the claim follows. For part (2), we first note that Claim 2.2 holds for the setting of the T_j^k in arbitrary graphs. Hence, the number of phases is at most $\lceil \log n \rceil$ by Claim 2.2, item 5. Therefore $X(D_\ell) \leq \sum_{k=1}^{\lceil \log n \rceil} X(A_k^\ell) \leq \sum_{k=1}^{\lceil \log n \rceil} \ell \cdot |A_k^\ell|$. Using Claim 2.3 we get $X(D_\ell) \leq \lceil \log n \rceil \cdot \ell \cdot 3n/2^{\ell-1}$. □

Claim. The total additional cost incurred by the nodes is $O(n \log\log n)$.

Proof. Partition the total additional cost X into $X = X_L + X_H$ where $X_L = \sum_{\ell \leq \log\log n} X(D_\ell)$ and $X_H = \sum_{\ell > \log\log n} X(D_\ell)$. Note that by item (1) of Claim 2.3, $X_L \leq \sum_{\ell \leq \log\log n} \ell \cdot |D_\ell| \leq n \log\log n$. Also, by item (2) of Claim 2.3, $X_H \leq \sum_{\ell > \log\log n} 3\ell n \lceil \log n \rceil / 2^{\ell-1} \leq 3n \lceil \log n \rceil \cdot \frac{2 \log\log n}{\log n} = O(n \log\log n)$. □

Consequently, we have the following.

Theorem 1. *There is a polynomial time algorithm that given a graph G and a port numbering constructs a tree T spanning 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 numberings 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 $\omega(x) = 1$ for $x = 0$ and $\lfloor \log x \rfloor + 1$ for $x \geq 1$, to every edge e where l and l' are the port numbers of e at its two endpoints. It is easy to check that running any MST algorithm returns a tree T minimizing $S_{all}(T)$. Thus, we have the following.

Proposition 5. *There is a polynomial time algorithm that given a graph G and a port numbering constructs a tree T minimizing $S_{all}(T)$.*

On the other hand, by a reduction from the Hamiltonian path problem in 3-regular planar graphs, we have the following negative result.

Proposition 6. *The following decision problem is NP-hard.*
Input: *A graph G with a port numbering, and an integer k;*
Question: *Is there a spanning tree T of G satisfying $M_{all}(T) \le k$.*
This result holds even restricted to cubic planar graphs, and even for fixed $k = 3$.

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. [9]) that a spanning tree with maximum degree at most $\delta_{min} + 1$ can be computed in polynomial time. Hence we have the following.

Theorem 2. *There is a polynomial time algorithm that given a graph G and a port numbering constructs a spanning tree T for G satisfaying $M_{all}(T) = O(\delta_{min} \log \Delta)$.*

On the other hand, any tree T^* minimizing M_{all} in a graph G has a degree $\Delta_{T^*} \ge \delta_{min}$. Thus $M_{all}(T^*) \ge \sum_{i=1}^{\Delta_{T^*}} \log i \ge \sum_{i=1}^{\delta_{min}} \log i \ge \Omega(\delta_{min} \log \delta_{min})$. 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. Hereafter 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.

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. [11, 1, 12]). 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 5 or Theorem 2, respectively, yields the following bounds.

Corollary 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.*

Upcast and convergecast The basic *upcast* process involves collecting information upwards to the root over a spanning tree. 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 [12]. 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 1 we get the following.

Corollary 2. *For any graph G, it is possible to construct an upwards tree representation using $O(n \log \log n)$ memory bits over the graph in a way that will allow performing an upcast 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 $f_{w_i} = f(X_{w_i})$, $1 \leq i \leq k$, 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 [12].

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 1 we get the following.

Corollary 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 numbers. 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,[3] 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 [6], 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 [6], 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 [6], using the all-ports labeling scheme $\mathcal{L}_{all}$, we get the following.

Corollary 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 $2n$ 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 2. *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.*

By Lemma 2, using a labeling $\mathcal{L}_{up}$ on an arbitrary spanning tree and relying on Lemma 1 and Theorem 1, we get the following.

Corollary 5. *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 $4m$.*

By Lemma 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 $O(1)$.

References

1. H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw-Hill, 1998.
2. Y. Chu and T. Liu. On the shortest arborescence of a directed graph. *Science Sinica* 14, pages 1396–1400, 1965.
3. 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, pages 335–346, 2005.
4. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.
5. A. Czumaj and W.-B. Strothmann. Bounded-degree spanning tree. In *Proc. 5th European Symp. on Algorithms (ESA)*, LNCS 1284, pages 104–117, 1997.
6. 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.
7. J. Edmonds. Optimum branchings. *J. Research of the National Bureau of Standards* 71B, pages 233–240, 1967.
8. P. Elias. Universal Codeword Sets and Representations of the Integers. *IEEE Trans. Inform. Theory* 21(2):194–203, 1975.
9. M. Fürer and B. Raghavachari. Approximating the minimum degree spanning tree within one from the optimal degree. In *Proc. 3rd Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 317–324, 1992.
10. M. Garey, D. Johnson, and R. Tarjan. The planar Hamiltonian circuit is NP-complete. *SIAM Journal on Computing* 5(4):704–714, 1976.
11. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1995.
12. D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.