A Fast Distributed Approximation Algorithm for Minimum Spanning Trees

Maleq Khan and Gopal Pandurangan

Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA
{mmkhan, gopal}@cs.purdue.edu

Abstract. We give a distributed algorithm that constructs a $O(\log n)$-approximate minimum spanning tree (MST) in arbitrary networks. Our algorithm runs in time $\tilde{O}(D(G) + L(G, w))$ where $L(G, w)$ is a parameter called the *local shortest path diameter* and $D(G)$ is the (unweighted) diameter of the graph. Our algorithm is existentially optimal (up to polylogarithmic factors), i.e., there exists graphs which need $\Omega(D(G) + L(G, w))$ time to compute an H-approximation to the MST for any $H \in [1, \Theta(\log n)]$. Our result also shows that there can be a significant time gap between exact and approximate MST computation: there exists graphs in which the running time of our approximation algorithm is exponentially faster than the *time-optimal* distributed algorithm that computes the MST. Finally, we show that our algorithm can be used to find an approximate MST in wireless networks and in random weighted networks in almost optimal $\tilde{O}(D(G))$ time.

Keywords: Distributed Approximation Algorithm, Minimum Spanning Tree.

1 Introduction

1.1 Background and Previous Work

The distributed minimum spanning tree (MST) problem is one of the most important problems in the area of distributed computing. There has been a long line of research to develop efficient distributed algorithms for the MST problem starting with the seminal paper of Gallager et al [1] that constructs the MST in $O(n \log n)$ time and $O(|E| + n \log n)$ messages. The communication (message) complexity of Gallager et al. is optimal, but its time complexity is not. Hence further research concentrated on improving the time complexity. The time complexity was first improved to $O(n \log \log n)$ by Chin and Ting [2], further improved to $O(n \log^* n)$ by Gafni [3], and then improved to *existentially optimal* running time of $O(n)$ by Awerbuch [4]. The $O(n)$ bound is existentially optimal because there exists graphs where no distributed MST algorithm can do better than $\Omega(n)$ time. This was the state of art till the mid-nineties when Garay, Kutten, and Peleg [5] raised the question of identifying graph parameters that can better capture the complexity (motivated by "universal" complexity) of distributed MST computation. For many existing networks G, their diameter $D(G)$ (or D for short) is significantly smaller than the number of vertices n and therefore is a good candidate to design protocols whose running time is bounded in terms of $D(G)$ rather than

S. Dolev (Ed.): DISC 2006, LNCS 4167, pp. 355–369, 2006.

n. Garay, Kutten, and Peleg [5] gave the first such distributed algorithm for the MST problem that ran in time $O(D(G) + n^{0.61})$ which was later improved by Kutten and Peleg [6] to $O(D(G) + \sqrt{n}\log^* n)$. Elkin [7] refined this result further and argued that the parameter called "MST-radius" captures the complexity of distributed MST better. He devised a distributed protocol that constructs the MST in $\tilde{O}(\mu(G, w) + \sqrt{n})$ time, where $\mu(G, w)$ is the "MST-radius" of the graph [7] (is a function of the graph topology as well as the edge weights). The ratio between diameter and MST-radius can be as large as $\Theta(n)$, and consequently on some inputs this protocol is faster than the protocol of [6] by a factor of $\Omega(\sqrt{n})$. However, a drawback of this protocol (unlike previous MST protocols [6,5,2,3,1]) is that it cannot detect termination in this much time (unless $\mu(G, w)$ is given as part of the input). Finally, we note that the time-efficient algorithms of [6,7,5] are not message-optimal (i.e., they take asymptotically much more than $O(|E| + n \log n)$, e.g., the protocol of [6] takes $O(|E| + n^{1.5})$ messages.

The lack of progress in improving the result of [6], and in particular breaking the $\sqrt{n}$ barrier, led to work on lower bounds for distributed MST problem. Peleg and Rabinovich [8] showed that $\tilde{\Omega}(\sqrt{n})$ is required for constructing MST even on graphs of small diameter and showed that this result establishes the asymptotic near-tight (existential) optimality of the protocol of [6].

While the previous distributed protocols deal with computing the exact MST, the next important question addressed in the literature concerns the study of distributed *approximation* of MST, i.e., constructing a spanning tree whose total weight is near-minimum. From a practical perspective, given that MST construction can take as much as $\tilde{\Omega}(\sqrt{n})$ time, it is worth investigating whether one can design distributed algorithms that run faster and output a near-minimum spanning tree. Peleg and Rabinovich [8] was one of the first to raise the question of devising faster algorithms that construct an approximation to the MST and left it open for further study. To quote their paper: "To the best of our knowledge nothing nontrivial is known about this problem...". Since then, the most important result known till date is the *hardness* results shown by Elkin [9]. This result showed that *approximating* the MST problem on graphs of small diameter (e.g., $O(\log n)$) within a ratio H requires essentially $\Omega(\sqrt{n/HB})$ time (assuming B bits can be sent through each edge in one round), i.e., this gives a time-approximation trade-off for the distributed MST problem: $T^2H = \Omega(\sqrt{n/B})$. However, not much progress has been made on designing time-efficient distributed approximation algorithms for MST. To quote Elkin's survey paper [10]: "There is no satisfactory approximation algorithm known for the MST problem". To the best of our knowledge, the only known distributed approximation algorithm for MST is given by Elkin in [9]. This algorithm gives an H-approximation protocol for the MST with running time $O(D(G)+\frac{\omega_{max}}{H-1}\cdot\log^* n)$, where ω_{max} is the ratio between the maximal and minimal weight of an edge in the input graph G. Thus this algorithm is not independent of the edge weights and its running time can be quite large.

1.2 Distributed Computing Model and Our Results

We present a fast distributed approximation algorithm for the MST problem. We will first briefly describe the distributed computing model that is used by our algorithm (as

well as the previous MST algorithms [2,1,5,6,4,3,7] mentioned above) which is now standard in distributed computing literature (see e.g., the book by Peleg [11]).

Distributed computing model. We are given a network modeled as an undirected weighted graph $G = (V, E, w)$ where V is the set of nodes (vertices) and E is the set of communication links between them and $w(e)$ gives the weight of the edge $e \in E$. Without loss of generality, we will assume that G is connected. Each node hosts a processor with limited initial knowledge. Specifically, we make the common assumption that nodes have unique identity numbers (this is not really essential, but simplifies presentation) and at the beginning of the computation each vertex v accepts as input its own identity number, the identity numbers of its neighbors in G (i.e., nodes that share an edge with v), and the weights of the edges that are adjacent to v. Thus a node has only *local* knowledge limited to itself and its neighbors. The vertices are allowed to communicate through the edges of the graph G. We assume that the communication is synchronous and occurs in discrete pulses (time steps). (This assumption is not essential for our time complexity analysis. One can use a *synchronizer* to obtain the same time bound in an asynchronous network at the cost of some increase in the message (communication) complexity [11].) During each time step, each node v is allowed to send an arbitrary message of size $O(\log n)$ through each edge $e = (v, u)$ that is adjacent to v, and the message will arrive at u at the end of the current pulse. (We note that if unbounded-size messages are allowed, then MST problem can be trivially solved in $O(D(G))$ time[11].) The weights of the edges are at most polynomial in the number of vertices n, and therefore the weight of a single edge can be communicated in one time step. This model of distributed computation is called the $\mathcal{CONGEST}(\log n)$ model or simply the $\mathcal{CONGEST}$ model [11] (the previous results on distributed MST cited in Sect. 1.1 are for this model). We note that, more generally, $\mathcal{CONGEST}(B)$ allows messages of size at most $O(B)$ to be transmitted in a single time step across an edge. Our algorithm can straightforwardly be applied to this model also. We will assume $B = \log n$ throughout this paper.

Overview of results. Our main contribution is an almost existentially optimal (in both time and communication complexity) distributed approximation algorithm that constructs a $O(\log n)$-approximate minimum spanning tree, i.e., whose cost is within a $O(\log n)$ factor of the MST. The running time[1] of our algorithm is $\tilde{O}(D(G)+L(G,w))$ where $L(G, w)$ is a parameter called as the *local shortest path diameter* (we defer the definition of $L(G, w)$ to Sect. 2.2). Like the MST-radius, $L(G, w)$ depends both on the graph topology as well as on the edge weights. $L(G, w)$ always lies between 1 and n. $L(G, w)$ can be smaller or larger than the diameter and typically it can be much smaller than n or even $\sqrt{n}$ (recall that this is essentially a lower bound on distributed (exact) MST computation). In fact, we show that there exists graphs for which any distributed algorithm for computing the MST will take $\tilde{\Omega}(\sqrt{n})$ time, while our algorithm will compute a near-optimal approximation in $\tilde{O}(1)$ time, since $L(G, w) = \tilde{O}(1)$ and $D = \tilde{O}(1)$ for these graphs. Thus there exists an exponential gap between exact MST and $O(\log n)$-approximate MST computation. However, in some graphs $L(G, w)$ can

[1] We use the notations $\tilde{O}(f(n))$ and $\tilde{\Omega}(f(n))$ to denote $O(f(n) \cdot polylog(f(n)))$ and $\Omega(f(n)/polylog(f(n)))$, respectively.

be asymptotically much larger than both the diameter as well as $\sqrt{n}$. By combining the MST algorithm of Kutten and Peleg [6] with our algorithm in an obvious way we can obtain an algorithm with the same approximation guarantee but with running time $\tilde{O}(D(G) + \min(L(G,w), \sqrt{n}))$.

The parameter $L(G,w)$ is not arbitrary. We show that it captures the hardness of distributed approximation quite precisely: there exists a family of n-vertex graphs where $\Omega(L(G,w))$ time is needed by any distributed approximation algorithm to approximate MST within a H-factor ($1 \leq H \leq O(\log n)$) (cf. Theorem 5). This implies that our algorithm is existentially optimal (upto polylogarithmic factors) and in general no other algorithm can do better. We note that the existential optimality our algorithm is with respect to $L(G,w)$ instead of n as in the case of Awerbuch's algorithm [4]. Our algorithm is also existentially optimal (upto polylogarithmic factors) with respect to communication (message) complexity — takes $\tilde{O}(|E|)$ messages, since $\Omega(|E|)$ messages is clearly needed in some graphs to construct any spanning tree[12,13].

One of our motivations for this work is to investigate whether fast distributed algorithms that construct (near-optimal) MST can be given for special classes of networks. An important consequence of our result is that networks with low $L(G,w)$ value (compared to $O(D(G))$ admit a $\tilde{O}(D(G))$ time $O(\log n)$-approximate distributed algorithm. In particular unit disk graphs have $L(G,w) = 1$. Unit disk graphs are commonly used models in wireless networks. We also show $L(G,w) = O(\log n)$ with high probability in any arbitrary network whose edge weights are chosen independently at random from an arbitrary distribution (cf. Theorem 7).

2 Distributed Approximate MST Algorithm

2.1 Nearest Neighbor Tree Scheme

The main idea of our approach is to construct a spanning tree called as the *Nearest Neighbor Tree (NNT)* efficiently in a distributed fashion. In our previous work [14], we introduced the Nearest Neighbor Tree and showed that its cost is within a $O(\log n)$ factor of the MST. The scheme used to construct a NNT (henceforth called *NNT scheme*) is as follows: (1) each node first chooses a unique identity number called *rank* and (2) each node (except the one with the highest rank) connects (via the *shortest path*) to the *nearest* node of higher rank. We showed that the NNT scheme constructs a spanning subgraph in any weighted graph whose cost is at most $O(\log n)$ times that of the MST, irrespective of how the ranks are selected (as long as they are distinct) [14]. Note that cycles can be introduced in step 2, and hence to get a spanning tree we need to remove some edges to break the cycles. Our NNT scheme is based on the approximation algorithm for the *traveling salesman problem* (coincidentally called Nearest Neighbor algorithm) analyzed in a classic paper of Rosenkrantz, Lewis, and Stearns [15]. Imase and Waxman [16] also used a scheme based on [15] (their algorithm can also be considered a variant of NNT scheme) to show that it can maintain a $O(\log n)$-approximate Steiner tree dynamically (assuming only node additions, but not deletions.) However, their algorithm will not work in a distributed setting (unlike our NNT scheme) because one cannot connect to the shortest node (they can do that since the nodes are added one

by one) as this can introduce cycles. The approach needed for distributed implementation is very different (cf. Sect. 2.3).

The advantage of NNT scheme is this: each node, individually, has the task of finding its own node to connect, and hence no explicit coordination is needed between nodes. However, despite the simplicity of the NNT scheme, it is not clear how to efficiently implement the scheme in a general weighted graph. In our previous work [14], we showed NNT scheme can be implemented in a *complete metric* graph G (i.e., $D(G) = 1$). Our algorithm takes only $O(n \log n)$ messages to construct a $O(\log n)$-approximate MST as opposed to $\Omega(n^2)$ lower bound (shown by Korach et al [17]) needed by any distributed MST algorithm in this model. If time complexity needs to be optimized, then NNT scheme can be easily implemented in $O(1)$ time (using $O(n^2)$ messges), as opposed to the best known time bound of $O(\log \log n)$ for (exact) MST [18]. These results suggest that NNT scheme can yield faster and communication-efficient algorithms compared to the algorithm that compute the exact MST. However, efficient implementation in a general weighted graph is non-trivial and was left open in [14]. Thus, a main contribution of this paper is giving an efficient implementation of the scheme in a general network. The main difficulty is in efficiently finding the nearest node of higher rank in a distributed fashion because of congestion (since many nodes are trying to search at the same time) and avoiding cycle formation. We use a technique of "incremental" neighborhood exploration that avoids congestion and cycle formation and is explained in detail in Sect. 2.3.

2.2 Preliminaries

We use the following definitions and notations concerning an undirected weighted graph $G = (V, E, w)$. We say that u and v are neighbors of each other if $(u, v) \in E$.

Notations:

$|Q(u, v)|$ or simply $|Q|$ — is the number of edges in path Q from u to v.

$w(Q(u, v))$ or $w(Q)$ — is the weight of the path Q, which is defined as the sum of the weights of the edges in path Q, i.e., $w(Q) = \sum_{(x,y)\in Q} w(x, y)$. $P(u, v)$ — is a shortest path (in the weighted sense) from u to v.

$d(u, v)$ — is the (weighted) distance between u and v, and defined by $d(u, v) = w(P(u, v))$.

$N_\rho(v)$ — set of all *neighbors* u such that $w(u, v) \leq \rho$, i.e., $N_\rho(v) = \{u \mid (u, v) \in E \wedge w(u, v) \leq \rho\}$.

$W(v)$ — is the weight of the largest edge adjacent to v. $W(v) = \max_{(v,x)\in E} w(v, x)$

$l(u, v)$ — is the minimum length (number of the edges) shortest path from u to v. Note that there may be more than one shortest path from u to v. Thus $l(u, v)$ is the number of edges of the shortest path having the least number of edges, i.e, $l(u, v) = \min\{|P(u, v)| \mid P(u, v)$ is a shortest path from u to $v\}$.

Definition 1. ***ρ-neighborhood.*** *ρ-neighborhood of a node v, denoted by $\Gamma_\rho(v)$, is the set of the nodes that are within distance ρ from v. $\Gamma_\rho(v) = \{u \mid d(u, v) \leq \rho\}$.*

Definition 2. ***(ρ, λ)-neighborhood.*** *(ρ, λ)-neighborhood of a node v, denoted by $\Gamma_{\rho,\lambda}(v)$, is the set of all nodes u such that there is a path $Q(v, u)$ such that $w(Q) \leq \rho$ and $|Q| \leq \lambda$. Clearly, $\Gamma_{\rho,\lambda}(v) \subseteq \Gamma_\rho(v)$.*

Definition 3. ***Shortest Path Diameter (SPD).*** *SPD is denoted by $S(G, w)$ (or S for short) and defined by $S = \max_{u,v \in V} l(u, v)$.*

Definition 4. ***Local Shortest Path Diameter (LSPD).*** *LSPD is denoted by $L(G, w)$ (or L for short) and defined by $L = \max_{v \in V} L(v)$, where $L(v) = \max_{u \in \Gamma_{W(v)}(v)} l(u, v)$.*

Notice that $L \leq S \leq n$ in any graph. However, there exists graphs, where L is significantly smaller than both S and the (unweighted) diameter of the graph, D. For example, in a chain of n nodes (all edges with weight 1), $S = n$, $D = n$, and $L = 1$.

2.3 Distributed NNT Algorithm

We recall that the basic NNT scheme is as follows. Each node v selects a unique rank $r(v)$. Then each node finds the nearest node of higher rank and connects to it via the shortest path.

Rank selection. The nodes select unique ranks as follows. First a leader is elected by a leader election algorithm. Let s be the leader node. The leader picks a number $p(s)$ from the range $[b - 1, b]$, where b is a number arbitrarily chosen by s, and sends this number $p(s)$ along with its ID (identity number) to its neighbors. A neighbor v of the leader s, after receiving $p(s)$, picks a number $p(v)$ from the open interval $[p(s) - 1, p(s))$, thus $p(v)$ is less than $p(s)$, and then transmits $p(v)$ and $ID(v)$ to all of its neighbors. This process is repeated by every node in the graph. Notice that at some point, every node in the graph will receive a message from at least one of its neighbors since the given graph is connected; some nodes may receive more than one message. As soon as a node u receives the first message from a neighbor v, it picks a number $p(u)$ from $[p(v) - 1, p(v))$, so that it is smaller than $p(v)$, and transmits $p(u)$ and $ID(u)$ to its neighbors. If u receives another message later from another neighbor v', u simply stores $p(v')$ and $ID(v')$, and does nothing else. $p(u)$ and $ID(u)$ constitute u's rank $r(u)$ as follows. For any two nodes u and v, $r(u) < r(v)$ iff i) $p(u) < p(v)$, or ii) $p(u) = p(v)$ and $ID(u) < ID(v)$.

At the end of execution of the above procedure of rank selection, it is easy to make the following observations.

Observation 1

1. *Each node knows the ranks of all of its neighbors.*

2. *The leader s has the highest rank among all nodes in the graph.*

3. *Each node v, except the leader, has one neighbor u, i.e. $(u, v) \in E$, such that $r(u) > r(v)$.*

Connecting to a higher-ranked node. Each node v (except the leader s) executes the following algorithm simultaneously to find the nearest node of higher rank and connect to it. Each node v needs to explore only the nodes in $\Gamma_{W(v)}(v)$ to find a node of higher rank.

Each node v executes the algorithm in *phases*. In the first phase, v sets $\rho = 1$. In the subsequent phases, it doubles the value of ρ; that is, in the ith phase, $\rho = 2^{i-1}$. In a phase of the algorithm, v explores the nodes in $\Gamma_\rho(v)$ to find a node u (if any) such that $r(u) > r(v)$. If such a node with higher rank is not found, v continues to the next phase

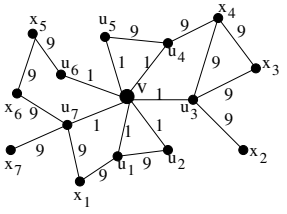

Fig. 1. A network with possible congestion in the edges adjacent to v. Weight of the edges (v, u_i) is 1 for all i, and 9 for the rest of the edges. Assume $r(v) < r(u_i)$ for all i.

with ρ doubled. By Observation 3 of 1, v needs to increase ρ to at most $W(v)$. Each phase of the algorithm consists of one or more *rounds*. In the first round, v sets $\lambda = 1$. In subsequent rounds, values for λ are doubled. In a particular round, v explores all nodes in $\Gamma_{\rho,\lambda}(v)$. At the end of each round, v counts the number of nodes it has explored. If the number of nodes remain the same in two successive rounds of the same phase (that is, v already explored all nodes in $\Gamma_\rho(v)$), v doubles ρ and starts the next phase. If at any point of time v finds a node of higher rank, it then terminates its exploration.

Since all of the nodes explore their neighborhoods simultaneously, many nodes may have overlapping ρ-neighborhoods. This might create congestion of the messages in some edges that may result in increased running time of the algorithm, in some cases by a factor of $\Theta(n)$. Consider the network given in Fig. 1. If $r(v) < r(u_i)$ for all i, when $\rho \geq 2$ and $\lambda \geq 2$, an exploration message sent to v by any u_i will be forwarded to all other u_is. Note that values for ρ and λ for all u_is will not necessarily be the same at a particular time. Thus congestion at any edge (v, u_i) can be as much as the number of such nodes u_i, which can be, in fact, $\Theta(n)$ in some graphs. However, to improve the running time of the algorithm, we keep congestions on all edges bounded by $O(1)$ by sacrificing the quality of the NNT, but only by a constant factor. To do so, v decides that some lower ranked u_is can connect to some higher ranked u_is and informs them instead of forwarding their message to the other nodes (details are given below). Thus v forwards messages from only *one* u_i and this avoids congestion. As a result, a node may not connect to the nearest node of higher rank. However, our algorithm guarantees that distance to the connecting node is not larger than four times the distance to the nearest node of higher rank. The detailed description is given below.

1. Exploration of ρ-neighborhood to find a node of higher rank:

Initiating exploration. Initially, each node v sets radius $\rho \leftarrow 1$ and $\lambda \leftarrow 1$. v explores the nodes in $\Gamma_{\rho,\lambda}(v)$ in a BFS-like manner to find if there is a node $x \in \Gamma_{\rho,\lambda}(v)$ such that $r(v) < r(x)$. v sends *explore* messages $< explore, v, r(v), \rho, \lambda, pd, l >$ to all $u \in N_\rho(v)$. In the message $< explore, v, r(v), \rho, \lambda, pd, l >$, v is the originator of the *explore* message; $r(v)$ is its rank, ρ is its current phase value; λ is its current round number in this phase; pd is the weight of the path traveled by this message so far (from v to the current node), and l is the number of links that the message can travel further. Before v sends the message to its neighbor u, v sets $pd \leftarrow w(v, u)$ and $l \leftarrow \lambda - 1$.

Forwarding *explore* messages. Any node y may receive more than one *explore* message from the same originator v via different paths for the same round. Any subsequent message is forwarded only if the later message arrived through a shorter path than the

previous one. Any node y, after receiving the message $< explore, v, r(v), \rho, \lambda, pd, l >$ from one of its neighbors, say z, checks if it previously received another message $< explore, v, r(v), \rho, \lambda, pd', l' >$ from z' with the same originator v such that $pd' \leq pd$. If so, y sends back a *count* message to z' with count = 0. The purpose of the *count* messages is to determine the number of nodes explored by v in this round. Otherwise, if $r(v) < r(y)$, y sends back a *found* message to v containing y's rank. Otherwise, If $N_{\rho-pd}(y) - \{z\} = \phi$ or $l = 0$, y sends back a *count* message with count = 1 and sets a marker $counted(v, \rho, \lambda) \leftarrow TRUE$. The purpose of the marker $counted(v, \rho, \lambda)$ is to make sure that y is counted only once for the same source v and in the same phase and round of the algorithm. If $r(v) > r(y)$, $l > 0$, and $N_{\rho-pd}(y) - \{z\} \neq \phi$, y forwards the message to all of its neighbors $u \in N_{\rho-pd}(y) - \{z\}$ after setting $pd \leftarrow pd + w(y, u)$ and $l \leftarrow l - 1$.

Controlling Congestion. If at any time step, a node v receives more than one, say $k > 1$, *explore* messages from different originators u_i, $1 \leq i \leq k$, it takes the following actions. Let $< explore, u_i, r(u_i), \rho_i, \lambda_i, pd_i, l_i >$ be the *explore* message from originator u_i. If there is a u_j such that $r(u_i) < r(u_j)$ and $pd_j \leq \rho_i$, v sends back a *found* message to u_i telling that u_i can connect to u_j where weight of the connecting path $w(Q(v, v')) = pd_i + pd_j \leq 2\rho_i$. In this way, some of the u_is will be replied back a *found* message and their *explore* messages will not be forwarded by v.

Let u_s be the node with lowest rank among the rest of u_is (i.e., those u_is which were not sent a *found* message by v), and u_t be an arbitrary node among the rest of u_is and let $t \neq s$. Now it must be the case that ρ_s is strictly smaller than ρ_t , i.e., u_s is in an earlier phase than u_t. This can happen if, in some previous phase, u_t exhausted its ρ-value with smaller λ-value leading to a smaller number of rounds in that phase and quick transition to the next phase. In such a case, we keep u_t waiting for at least one round without affecting the overall running time of the algorithm. To do this, v forwards *explore* message of u_s only and sends back *wait* messages to all u_t.

Each *explore* message triggers exactly one reply (either *found*, *wait*, or *count* message). These reply-back messages move in similar fashion as of *explore* messages but in reverse direction and they are aggregated (convergecast) on the way back as described next. Thus those reply messages also do not create any congestion in any edge.

Convergecast of the Replies of the *explore* Messages. If any node y forwards the *explore* message $< explore, v, r(v), \rho, \lambda, pd, l >$ received from z for the originator v to its neighbors in $N_{\rho-pd}(y) - \{z\}$, eventually, at some point later, y will receive replies to these *explore* messages, which y forwarded to $N_{\rho-pd}(y) - \{z\}$. Each of these replies is either a *count* message, *wait* message, or a *found* message. Once y receives replies from all nodes in $N_{\rho-pd}(y) - \{z\}$, it takes the following actions. If at least one of the replies is a *found* message or a *wait* message, y ignores all of the *count* messages and sends the *found* message or the *wait* message to z towards the destination v. If all of the replies are *count* messages, y adds the count values of these messages and sends a single *count* message to v with the aggregated count. Also, y adds itself to the count if the marker $counted(v, \rho, \lambda) = FALSE$ and sets $counted(v, \rho, \lambda) \leftarrow TRUE$. At the very beginning, y initializes $counted(v, \rho, \lambda) \leftarrow FALSE$. The *count* messages (also the *wait* and *found* messages) travel in the opposite direction of the *explore* messages

using the same paths toward v. Thus the *count* messages form a convergecast as opposed to the (controlled) broadcast of the *explore* messages.

Actions of the Originator after Receiving the Replies of the *explore* messages. At some time step, v receives replies of the *explore* messages originated by itself from all nodes in $N_\rho(v)$. Each of these replies is either a *count* message, *wait* message, or a *found* message. If at least one of the replies is a *found* message, v is done with exploration and makes the connection as described in Item 2 below. Otherwise, if there is a *wait* message, v again initiates exploration with same ρ and λ. If all of them are *count* messages: (a) if $\lambda = 1$, v initiates exploration with $\lambda \leftarrow 2$ and the same ρ; (b) if $\lambda > 1$ and count-value for this round is larger than that of the previous round, v initiates exploration with $\lambda \leftarrow 2\lambda$ and the same ρ; (c) otherwise v initiates exploration with $\lambda \leftarrow 2\lambda$ and $\rho \leftarrow 2\rho$.

2. Making Connection:

Let u be a node with higher rank that v found by exploration. If v finds more than one node with rank higher than itself, it selects the nearest one among them. Let $Q(v, u)$ be the path from v to u. The path $Q(v, u)$ is discovered when u is found in the exploration process initiated by v. The edges in $Q(v, u)$ are added in the resulting spanning tree as follows. To add the edges in $Q(v, u)$, v sends a *connect* message to u along this path. During the exploration process, the intermediate nodes in the path simply keeps tracks of the predecessor and successor nodes for this originator v. Let $Q(v, u) =< v, \ldots, x, y, \ldots, u >$. By our choice of u, note that all the intermediate nodes will have rank lower than $r(v)$. When the connect message passes through the edge (x, y), node x uses (x, y) as its connecting edge regardless of x's rank. If x did not find its connecting node yet, x stops searching for such nodes as the edge (x, y) serves as x's connection. If x is already connected using a path, say $< x, x_1, x_2, \ldots, x_k >$, the edge (x, x_1) is removed from the tree, but the rest of the edges in this path still remains in the tree. All nodes in path $Q(v, u)$ including v upgrade their ranks to $r(u)$; i.e., they assumes a new rank which is equal to u's rank. It might happen that in between exploration and connection, some node x in path $Q(v, u)$ changed its rank due to a connection by some origin other than v. In such a case, when the connect message travels through x, if x's current rank is larger than $r(v)$, x accepts the connections as the last node in the path and returns a *rank-update* message toward v instead of forwarding the connect message to the next node (i.e., y) toward u. This is necessary to avoid cycle creation.

2.4 Analysis of Algorithm

In this section, we analyze the correctness and performance of the distributed NNT algorithm. The following lemmas and theorems show our results.

Lemma 1. *Let, during exploration, v found a higher ranked node u and the path $Q(v, u)$. If v's nearest node of higher rank is u', then $w(Q) \leq 4d(v, u')$.*

Proof. Assume that u is found when v explored a (ρ, λ)-neighborhood for some ρ and λ. Then $d(v, u') > \rho/2$, otherwise, v would find u' as a node of higher rank in the previous phase and would not explore the ρ-neighborhood. Now, u could be found by v in two ways. i) The *explore* message originated by v reached u and u sent back a

found message. In this case, $w(Q) \leq \rho$. ii) Some node y received two *explore* messages originated by v and u via the paths $R(v, y)$ and $S(u, y)$ respectively, where $r(v) < r(u)$ and $w(S) \leq \rho$; and y sent a *found* message to v (see "Controlling Congestion" in Item 1). In this case, $w(Q) = w(R) + w(S) \leq 2\rho$, since $w(R) \leq \rho$. Thus for both cases, we have $w(Q) \leq 4d(v, u')$.

Lemma 2. *The algorithm adds exactly $n - 1$ edges to the NNT.*

Proof. Let a node v connect to another node u using the path $Q(v, u) = < v, \ldots, x, y, z, \ldots, u >$. When a connect message goes through an edge, say (x, y) (from x to y), in this path, the edge (x, y) is added to the tree. We say the edge (x, y) is associated to node x (not to y) based on the direction of the flow of the connect message. If, previously, x was associated to some other edge, say (x, y'), the edge (x, y') was removed from the tree. Thus each node is associated to at most one edge.

Except the leader s, each node x must make a connection and thus at least one connect message must go through or from x. Then, each node, except s, is associated to some edge in the tree.

Thus each node, except s, is associated to exactly one edge in NNT; and s cannot be associated to any node since a connect message cannot be originated by or go through s.

Now, to complete the proof, we need to show that no two nodes are associated to the same edge. Let x be associated to edge (x, y). When the connect message went through (x, y) from x to y, $r(x)$ and $r(y)$ became equal. Later if another connect message increased $r(x)$, then either $r(y)$ also increased to the same value or x became associated to some edge other than (x, y). Thus, while keeping (x, y) associated to x, it must be true that $r(x) \leq r(y)$. Then any new connect message that might make (x, y) associated to y by passing the connect message from y to x, must pass through x toward some node with rank higher than $r(y)$ (i.e., the connect message cannot terminate at x). This will make x associated to some other edge than (x, y). Therefore, no two nodes are associated to the same edge.

Lemma 3. *The edges in the NNT added by the given distributed algorithm does not create any cycle.*

Proof. Each node has a unique rank and it can connect only to a node with higher rank. Thus if each node can connect to a node of higher rank using a direct edge (as in a complete graph), it is easy to see that there cannot be any cycle. However, in the above algorithm, a node u connects to a node of higher rank, v, $r(u) < r(v)$, using shortest path $P(u, v)$, which may contain more than one edge and in such a path, ranks of the intermediate nodes are smaller than $r(u)$. Thus the only possibility of creating a cycle is when some other connecting shortest path goes though these intermediate nodes. For example, in Fig. 2, the paths $P(u, v)$ and $P(p, q)$ both go through a lower ranked node x.

In Fig. 2, if p connects to q using path $< p, x, q >$ before u makes its connection, x gets a new rank which is equal to $r(q)$. Thus u finds a higher ranked node, x, at a closer distance than v and connects to x instead of v. Note that if x is already connected to some node, it releases such connection and takes $< x, q >$ as its new connection, i.e., q is x's new parent. Now y_2 uses either (y_2, x) or (y_2, v), but not both, for its connection. Thus there is no cycle in the resulting graph.

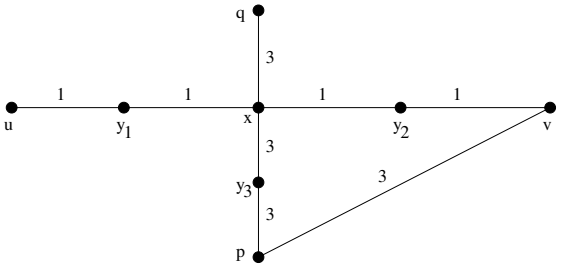

Fig. 2. A possible scenario of creating cycle and avoiding it. Nodes are marked with letters. Edge weights are given in the figure. Let $r(u) = 11, r(v) = 12, r(p) = 13, r(q) = 14$, and ranks of the rest of the nodes are smaller than 11. u connects to v, v connects to p, and p connects to q.

Now, assume that u already made its connection to v, but p is not connected yet. At this moment, x's rank is upgraded to $r(v)$ which is still smaller than $r(p)$. Thus p finds q as its nearest node of higher rank and connects using path $< p, x, q >$. In this connection process, x removes its old connecting edge (x, y_2) and gets (x, q) as its new connecting edge. Again, there cannot be any cycle in the resulting graph.

If x receives the connection request messages from both u (toward v) and p (toward q) at the same time, x only forwards the message for the destination with highest rank; here it is q. u's connection only goes up to x. Note that x already knows the ranks of both q and v from previous exploration steps.

From Lemmas 2 and 3 we have the following theorem.

Theorem 1. *The above algorithm produces a tree spanning all nodes in the graph.*

We next show that the spanning tree found is an $O(\log n)$-approximation to the MST (Theorem 2).

Theorem 2. *Let the NNT be the spanning tree produced by the above algorithm. Then the cost of the tree $c(NNT) \leq 4\lceil \log n \rceil c(MST)$.*

Proof. Let $H = (V_H, E_H)$ be a *complete* graph constructed from $G = (V, E)$ as follows. $V_H = V$ and weight of the edge $(u, v) \in E_H$ is the weight of the shortest path $P(u, v)$ in G. Now, the weights of the edges in H satisfy triangle inequality. Let NNT_H be a nearest neighbor tree and MST_H be a minimum spanning tree on H. We can show that $c(NNT_H) \leq \lceil \log n \rceil c(MST_H)$ [14].

Let NNT' be a spanning tree on G, where each node connects to the nearest node of higher rank. Then it is easy to show that $c(NNT') \leq c(NNT_H)$ and $c(MST_H) \leq c(MST)$.

By Lemma 1, we have $c(NNT) \leq 4c(NNT')$. Thus we get,

$$c(NNT) \leq 4c(NNT_H) \leq 4\lceil \log n \rceil c(MST_H) \leq 4\lceil \log n \rceil c(MST).$$

Theorem 3. *The running time of the above algorithm is $O(D + L \log n)$.*

Proof. Time to elect leader is $O(D)$. The rank choosing scheme takes also $O(D)$ time.

In the exploration process, ρ can increase to at most $2W$; because, within distance W, it is guaranteed that there is a node of higher rank (Observation 3 of 1). Thus, the number of phases in the algorithm is at most $O(\log W) = O(\log n)$.

In each phase, λ can grow to at most $4 * L$. When $L \leq \lambda < 2L$ and $2L \leq \lambda < 4L$, in both rounds, the count of the number of nodes explored will be the same. As a result, the node will move to the next phase.

Now, in each round, a node takes at most $O(\lambda)$ time; because the messages travel at most λ edges back and forth and at any time the congestion in any edge is $O(1)$. Thus any round takes time at most

$$\sum_{\lambda=1}^{\log(4L)} O(\lambda) = O(L).$$

Thus time for the exploration process is $O(L \log W)$. Total time of the algorithm for leader election, rank selection, and exploration is $O(D + D + L \log n) = O(D + L \log n)$.

Theorem 4. *The message complexity of the algorithm is* $O(|E| \log L \log n) = O(|E| \log^2 n)$.

Proof. The number of phases in the algorithm is at most $O(\log L)$. In each phase, each node executes at most $O(\log W) = O(\log n)$ rounds. In each round, each edge carries $O(1)$ messages. That is, number of messages in each round is $O(|E|)$. Thus total messages is $O(|E| \log L \log n)$.

3 Exact vs. Approximate MST and Near-Optimality of NNT Algorithm

Comparison with Distributed Algorithms for (Exact) MST. There can be a large gap between the local shortest path diameter L and $\tilde{\Omega}(\sqrt{n})$, which is the lower bound for exact MST computation. In particular, we can show that there exists a family of graphs where NNT algorithm takes $\tilde{O}(1)$ time, but *any* distributed algorithm for computing (exact) MST will take $\tilde{\Omega}(\sqrt{n})$ time. To show this we consider the parameterized (weighted) family of graphs called $\mathcal{J}_m^K$ defined in Peleg and Rabinovich [8]. (One can also show a similar result using the family of graphs defined by Elkin [9].) The size of $\mathcal{J}_m^K$ is $n = \Theta(m^{2K})$ and its diameter $\Theta(Km) = \Theta(Kn^{1/(2K)})$. For every $K \geq 2$, Peleg and Rabinovich show that any distributed algorithm for the MST problem will take $\Omega(\sqrt{n}/BK)$ time on some graphs belonging to the family. The graphs of this family have $L = \Theta(m^K) = \sqrt{n}$. We modify this construction as follows: the weights on all the highway edges except the first highway (H^1) is changed to 0.5 (originally they were all zero); all other weights remain the same. This makes $L = \Theta(Km)$, i.e., same order as the diameter. One can check that the proof of Peleg and Rabinovich is still valid, i.e., the lower bound for MST will take $\Omega(\sqrt{n}/BK)$ time on some graphs of this family, but NNT algorithm will take only $\tilde{\Omega}(L)$ time. Thus we can state:

Theorem 5. *For every $K \geq 2$, there exists a family of $n-$vertex graphs in which NNT algorithm takes $O(Kn^{1/(2K)})$ time while any distributed algorithm for computing the exact MST requires $\tilde{\Omega}(\sqrt{n})$ time. In particular, for every $n \geq 2$, there exists a family of graphs in which NNT algorithm takes $\tilde{O}(1)$ time whereas* any *distributed MST algorithm will take $\tilde{\Omega}(\sqrt{n})$ time.*

Such a large gap between NNT and any distributed MST algorithm can be also shown for constant diameter graphs, using a similar modification of a lower bound construction given in Elkin [9] (which generalizes and improves the results of Lotker et al [19]).

Near (existential) optimality of NNT algorithm. We show that there exists a family of graphs such that any distributed algorithm to find a $H(\leq \log n)$-approximate MST takes $\Omega(L)$ time (where L is the local shortest path diameter) on some of these graphs. Since NNT algorithm takes $\tilde{O}(D+L)$, this shows the near-tight optimality of NNT (i.e., tight up to a $polylog(n)$ factor). This type of optimality is called *existential optimality* which shows that our algorithm cannot be improved in general.

To show our lower bound we look closely at the hardness of distributed approximation of MST shown by Elkin [9]. Elkin constructed a family of weighted graphs $\mathcal{G}^{\omega}$ to show a lower bound on the time complexity of any $H-$approximation distributed MST algorithm (whether deterministic or randomized). We briefly describe this result and show that this lower bound is *precisely the local shortest path diameter L of the graph*. The graph family $\mathcal{G}^{\omega}(\tau, m, p)$ is parameterized by 3 integers τ, m, and p, where $p \leq \log n$. The size of the graph $n = \Theta(\tau m)$, the diameter is $D = \Theta(p)$ and the local shortest path diameter can be easily checked to be $L = \Theta(m)$. Note that graphs of different size, diameter, and $LSPD$ can be obtained by varying the parameters τ, m, and p. (We refer to [9] for the detailed description of the graph family and the assignment of weights.) We now slightly restate the results of [9] (assuming the $\mathcal{CONGEST}(\mathcal{B})$ model):

Theorem 6 ([9]). *1. There exists graphs belonging to the family $\mathcal{G}^{\omega}(\tau, m, p)$ having diameter at most D for $D \in 4, 6, 8, \ldots$ and LPSD $L = \Theta(m)$ such that any randomized H-approximation algorithm for the MST problem on these graphs takes $T = \Theta(L) = \Omega((\frac{n}{H \cdot D \cdot B})^{1/2-1/(2(D-1))})$ distributed time.*
2. If $D = O(\log n)$ then the lower bound can be strengthened to $\Theta(L) = \Omega(\sqrt{\frac{n}{H \cdot B \cdot \log n}})$.

Using a slightly different weighted family $\tilde{\mathcal{G}^{\omega}}(\tau, m)$ parameterized by two parameters τ and m, where size $n = \tau m^2$, diameter $D = \Omega(m)$ and LSPD $L = \Theta(m^2)$, one can strengthen the lower bound of the above theorem by a factor of $\sqrt{\log n}$ for graphs of diameter $\Omega(n^{\delta})$.
The above results show the following two important facts:

1. There are graphs having diameter $D << L$ where any H-approximation algorithm requires $\Omega(L)$ time.

2. More importantly, for graphs with very different diameters — varying from a constant (including 1, i.e., exact MST) to logarithmic to polynomial in the size of n — the lower bound of distributed approximate-MST is captured by the local shortest path parameter. In conjunction with our upper bound given by the NNT algorithm which takes $\tilde{O}(D + L)$ time, this implies that the LPSD L captures in a better fashion the complexity of distributed $O(\log n)$-approximate-MST computation.

4 Special Classes of Graphs

We show that in unit disk graphs (a commonly used model for wireless networks) $L = 1$, and in random weighted graphs, $L = O((\log n))$ with high probability. Thus our algorithm will run in near-optimal time of $\tilde{O}(D(G))$ on these graphs.

Unit Disk Graph (UDG). Unit disk graph is an euclidian graph where there is an edge between two nodes u and v if and only if general $dist(u, v) \leq R$ for some R (R is typically taken to be 1). Here $dist(u, v)$ is the euclidian distance between u and v which is the weight of the edge (u, v). For any node v, $W(v) \leq R$. Now if there is node u such that $d(u, v) \leq R$, then $dist(u, v) \leq R$ by triangle inequality. Thus $(u, v) \in E$ and the edge (u, v) is the shortest path from u to v. As a result, for any UDG, $L = 1$. For a 2-dimensional UDG, diameter can be as large as $\Theta(\sqrt(n))$.

Graph with Random Edge Weights. Consider any graph G (topology can be arbitrary) with edge weights chosen randomly from an arbitrary distribution (i.e., each edge weight is chosen i.i.d from the distribution). The following theorem shows that L and S is small compared to the diameter for such a graph.

Theorem 7. *Consider a graph G where the edge weights are chosen randomly from a (arbitrary) distribution with a constant (independent of n) mean. Then: (1) $L = O(\log n)$ with high probability (whp), i.e., probability at least $1 - 1/n^{\Omega(1)}$; and (2) the shortest path diameter $S = O(\log n)$ if $D < \log n$ and $S = O(D)$ if $D \geq \log n$ whp.*

Proof. Without loss of generality, we can assume that edge weights are randomly drawn from $[0, 1]$ with mean μ. Otherwise the edge weights can be normalized to this range without affecting the desired result. For any node v, $W(v) \leq 1$. Consider any path with $m = k \log n$ edges, for some constant k. Let the weights of the edges in this path be $w_1, w_2, \cdots, w_m$. For any i, $E[w_i] = \mu$. Since $\frac{1}{2}\mu k \log n \geq 1$ for sufficiently large k, we have

$$\Pr\{\sum_{i=1}^{m} w_i \leq 1\} \leq \Pr\{\sum_{i=1}^{m} w_i \leq \frac{1}{2}\mu k \log n\} = \Pr\{\mu - \frac{1}{m}\sum_{i=1}^{m} w_i \geq \frac{1}{2}\mu\}.$$

Using Hoeffding bound [20] and putting $k = \frac{6}{\mu^2}$,

$$\Pr\{\mu - \frac{1}{m}\sum_{i=1}^{m} w_i \geq \frac{1}{2}\mu\} \leq e^{-m\mu^2/2} = \frac{1}{n^3}.$$

Thus if it is given that the weight of a path is at most 1, then the probability that the number of edges $\leq \frac{6}{\mu^2}\log n$ is at most $\frac{1}{n^3}$. Now consider all nodes u such that $d(v, u) \leq W(v)$. There are at most $n - 1$ such nodes and thus there are at most $n - 1$ shortest paths leading to those nodes from v.

Thus using union bound, $\Pr\{L(v) \geq \frac{6}{\mu^2}\log n\} \leq n \times \frac{1}{n^3} = \frac{1}{n^2}$.

Using $L = \max\{L(v)\}$ and union bound, $\Pr\{L \geq \frac{6}{\mu^2}\log n\} \leq n \times \frac{1}{n^2} = \frac{1}{n}$.

Therefore, with probability at least $1 - \frac{1}{n}$, L is smaller than or equal to $\frac{6}{\mu^2}\log n$.

Proof of part 2 is similar.

References

1. Gallager, R., Humblet, P., Spira, P.: A distributed algorithm for minimum-weight spanning trees. ACM Transactions on Programming Languages and Systems **5**(1) (1983) 66–77
2. Chin, F., Ting, H.: An almost linear time and $O(n \log n + e)$ messages distributed algorithm for minimum-weight spanning trees. In: Proc. 26th IEEE Symp. Foundations of Computer Science. (1985) 257–266
3. Gafni, E.: Improvements in the time complexity of two message-optimal election algorithms. In: Proc. of the 4th Symp. on Principles of Distributed Computing. (1985) 175–185
4. Awerbuch, B.: Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In: Proc. 19th ACM Symp. on Theory of Computing. (1987) 230–240
5. Garay, J., Kutten, S., Peleg, D.: A sublinear time distributed algorithm for minimum-weight spanning trees. SIAM J. Comput. **27** (1998) 302–316
6. Kutten, S., Peleg, D.: Fast distributed construction of k-dominating sets and applications. J. Algorithms **28** (1998) 40–66
7. Elkin, M.: A faster distributed protocol for constructing minimum spanning tree. In: Proc. of the ACM-SIAM Symp. on Discrete Algorithms. (2004) 352–361
8. Peleg, D., Rabinovich, V.: A near-tight lower bound on the time complexity of distributed mst construction. In: Proc. of the 40th IEEE Symp. on Foundations of Computer Science. (1999) 253–261
9. Elkin, M.: Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In: Proc. of the ACM Symposium on Theory of Computing. (2004) 331 – 340
10. Elkin, M.: An overview of distributed approximation. ACM SIGACT News Distributed Computing Column **35**(4) (2004) 40–57
11. Peleg, D.: Distributed Computing: A Locality Sensitive Approach. SIAM (2000)
12. Korach, E., Moran, S., Zaks, S.: Optimal lower bounds for some distributed algorithms for a complete network of processors. Theoretical Computer Science **64** (1989) 125–132
13. Tel, G.: Introduction to Distributed Algorithms. Cambridge University Press (1994)
14. Khan, M., Kumar, V.A., Pandurangan, G.: A simple randomized scheme for constructing low-cost spanning subgraphs with applications to distributed algorithms. In: Technical Report, Dept. of Computer Science, Purdue University. (2005. http://www.cs.purdue.edu/homes/gopal/localmst.pdf)
15. Rosenkrantz, D., Stearns, R., Lewis, P.: An analysis of several heuristics for the traveling salesman problem. SIAM J. Comput. **6(3)** (1977) 563–581
16. Imase, M., Waxman, B.: Dynamic steiner tree problem. Siam J. Discrete Math **4(3)** (1991) 369–384
17. Korach, E., Moran, S., Zaks, S.: The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors. SIAM Journal of Computing **16(2)** (1987) 231–236
18. Lotker, Z., Patt-Shamir, B., Pavlov, E., Peleg, D.: Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. SIAM J. Comput. **35**(1) (2005) 120–131
19. Lotker, Z., Patt-Shamir, B., Peleg, D.: Distributed mst for constant diameter graphs. In: Proc. of the 20th ACM Symp. on Principles of Distributed Computing. (2001) 63–72
20. Hoeffding, W.: Probability for sums of bounded random variables. J. of the American Statistical Association **58** (1963) 13–30