DISTRIBUTED APPROXIMATION ALGORITHMS FOR MINIMUM SPANNING TREES AND OTHER RELATED PROBLEMS WITH APPLICATIONS TO WIRELESS AD HOC NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Md Abdul Maleq Khan

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2007

Purdue University

West Lafayette, Indiana

Dedicated to the memory of my father, Nayeb Uddin Khan, who, in my childhood, prepared me for the long journey toward this degree.

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Gopal Pandurangan, for his continuous guidance, encouragement, and active participation in developing the ideas in this thesis. I also like to thank Prof. V.S. Anil Kumar of Virginia Tech for his collaboration in some part of this research work. We are very grateful to the referees of the papers [1–4] containing the results of this thesis published in various conferences and journals, for their careful reading of the papers and detailed comments which helped greatly in improving the presentation of the results.

I also thank Prof. Wojciech Szpankowski, Prof. Mikhail Atallah, and Prof. Suresh Jagannathan for serving in the committee of my Ph.D. final examination.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Khan, Md Abdul Maleq. Ph.D., Purdue University, December, 2007. Distributed Approximation Algorithms for Minimum Spanning Trees and Other Related Problems with Applications to Wireless Ad Hoc Networks. Major Professor: Gopal Pandurangan.

Due to the advent of various advanced network technologies, distributed algorithms have become an important and rapidly growing field of research. Many emerging networks such as wireless ad hoc networks and peer-to-peer networks operate under inherent resource constraints (energy, bandwidth, etc.). Topologies of these networks can also change dynamically. For these networks, it is necessary to develop efficient (in both time and message complexities) distributed algorithms even if the solutions are sub-optimal (approximate). In this dissertation, we develop and analyze a class of distributed approximation algorithms to solve two fundamental network optimization problems: minimum spanning trees (MST) and minimum-cost k-connected subgraphs. We design and analyze a simple randomized scheme called Nearest Neighbor Tree (NNT) for efficient construction of approximate MSTs. We show that our scheme constructs a $O(\log n)$-approximate MST in any weighted graph and a *constant* approximation for uniform distribution of nodes on a plane. Then we apply the NNT scheme to design local distributed algorithms for MST in complete networks, arbitrary networks, and wireless ad hoc networks. Our main contribution is the first non-trivial distributed $O(\log n)$-approximation algorithm for MST in an arbitrary network which takes $\tilde{O}(D + L)$ time and $\tilde{O}(E)$ messages, where L is a parameter called the *local shortest path diameter* and D is the (unweighted) diameter of the graph. L always lies between 1 and n but can be much smaller than $\sqrt{n}$ in most of the graphs. In addition, we develop an algorithm for a complete graph that takes $O(\log n)$ time and expected $O(n \log n)$ messages and an algorithm for wireless ad hoc network that takes $O(\log^2 n)$ time and expected $O(n)$ messages. We also perform extensive simulations of our algorithms for wireless ad hoc networks. Simulations validate the theoretical results and show that the bounds can be much better in practice.

We extend the NNT scheme to develop a simple randomized scheme for constructing low-cost k-connected spanning subgraphs in a weighted complete graph. We show that our algorithm gives an approximation ratio of $O(k \log n)$ for a metric graph, $O(k)$ for a random graph with nodes uniformly randomly distributed in $[0,1]^2$ and $O(\log \frac{n}{k})$ for a graph with random edge weights $U(0,1)$. We then design an efficient local distributed algorithm for constructing a k-connected spanning subgraph (for any $k \geq 1$) in a point-to-point distributed model, where the processors form a complete network. This algorithm takes $O(\log \frac{n}{k})$ time and expected $O(nk \log \frac{n}{k})$ messages.

1 INTRODUCTION

Many emerging networks such as wireless ad hoc networks and peer-to-peer networks operate under inherent resource constraint (energy, bandwidth, etc.). A distributed algorithm which exchanges a large number of messages can consume a relatively large amount of energy (and also time) is not suitable in an resource-constrained ad hoc wireless sensor network. This is especially true in a *dynamic* setting – when the network needs to be reconfigured (e.g., due to mobility or failures) frequently and quickly. Reconfiguration is also necessary to evenly distribute the energy consumption among all the nodes [5] in a wireless network and thus to increase network lifetime. Thus it is necessary to develop simple, local, distributed algorithms which are efficient (in both time and message) even at the cost of being *sub-optimal* (see e.g., [6–8] for such algorithms in the context of wireless sensor networks). This adds a new dimension to the design of distributed algorithms for such networks. Thus we can potentially *tradeoff* optimality of the solution to the efficiency of the algorithm.

In this dissertation, we develop and analyze a class of distributed approximation algorithms to solve two fundamental network optimization problems: minimum spanning trees (MST) and minimum-cost k-connected subgraphs.

Minimum Spanning Tree (MST) Problem. The MST problem is one of the most important problems in the area of distributed computing and a commonly occurring primitive in the design and operation of data and communication networks. A minimum spanning tree is spanning tree with minimum weight among all possible spanning trees of the graph, where the weight (or cost) of a spanning tree is the sum of the weights of the edges in the spanning tree. Minimum spanning trees has many applications in networking. For instance, in ad hoc sensor networks, MST is the optimal routing tree for data aggregation [9, 10]. The classical distributed MST algorithm due to Gallager, Humblet, and Spira (henceforth referred to as the GHS algorithm) [11] uses $\Theta(n \log n + |E|)$ messages, and is essentially optimal with respect to the message complexity. There are distributed algorithms that find the MST (for e.g., see [12,13]) and are essentially optimal in terms of time complexity: they run in $O(Diam(G) + \sqrt{n})$ time, and there are matching lower bounds. However, these time-optimal algorithms involve a lot of message transfers (much more than GHS). Even for a wireless network modeled by a unit disk graph (or even a ring) any distributed algorithm to construct a MST needs $\Omega(n \log n)$ messages [6,14]. Despite their theoretical optimality, these algorithms are fairly involved, require synchronization and a lot of book keeping; such algorithms are impractical for resource-constraint networks [6]. For example, consider sensor networks — an ad hoc network formed by large numbers of small, battery-powered, wireless sensors. We develop time, message, and energy efficient distributed approximation algorithms to build a low-cost spanning tree which is a very close approximation to MST [1,3].

k-Connected Subgraph Problem. Computing the low weight spanning subgraphs of a given graph $G(V, E)$ with non-negative edge weights is a fundamental problem in network design (e.g., see [15,16] for an extensive survey). One important problem in this setting is the *k-vertex connectivity problem* (henceforth simply the *k-connectivity* problem): find a spanning subgraph of minimum weight that is k-vertex-connected, i.e., there exists k vertex-disjoint paths between every pair of vertices. Finding an optimal k-connected spanning subgraph is NP-hard for $k \geq 2$ even if the weights of the edges satisfy the triangle inequality, or even when the graph is a complete Euclidean graph [17]. There has been a lot of work on designing approximation algorithms for the k-connectivity problem. Most of these algorithms are centralized algorithms which are quite sophisticated and their main goal is to obtain a polynomial time algorithms with the best possible approximation ratio (see e.g., [18–20]). Distributed algorithms for the k-connectivity ($k \geq 2$) problem has received limited attention thus far — this is especially true for the weighted version. In fact, to the best of our knowledge there is no known efficient distributed algorithm for $k \geq 2$ for weighted graphs. In this thesis, we present a simple scheme to construct low-cost k-connected subgraph and its efficient distributed implementation in a complete weighted network [2].

NNT-Scheme. We present a simple and local scheme called *Nearest Neighbor Tree Scheme (NNT-Scheme)* to construct a low-cost spanning tree in a distributed network — each node chooses a unique rank (thus the ranks define a permutation on the nodes), and connects to the closest node of higher rank (details are in Chapter 2). While ranks can be chosen in many ways, in this thesis, we mainly focus on a simple randomized way of choosing ranks: each node chooses a rank uniformly and independently at random from a totally ordered set. Thus, we call this scheme as *Random-NNT*. In the context of geometric setting, we also devise another ranking of the nodes using the coordinates of the nodes, which is called *Co-NNT*. In Chapter 3, we generalize the NNT-scheme to construct a low-cost k-connected subgraph. This generalized scheme is called *Nearest Neighbor Scheme (NN-Scheme)*.

Using the NNT-scheme and the generalized NN-scheme, we develop distributed approximation algorithm for the minimum spanning tree problem and the k-connected subgraph problem. The implementation and the time and message complexity of the NNT-scheme depends on the network model. In this thesis, we consider the following two models of distributed computation.

Point-to-point Communication Model. We are given a network modeled as an undirected weighted graph $G = (V, E, w)$ where V is the set of the nodes (vertices) and E is the set of the communication links between them and $w(e)$ is the weight of the edge $e \in E$. Each node hosts a processor with limited initial knowledge. The communication between any two nodes happens by sending/receiving messages along the edge between them in 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 complexity [13].) We assume that $O(\log n)$ bits can be transferred in one step per edge and a node can send messages through all its incident links at the same time (see e.g., [13, 21]).

Wireless Local Broadcast Model. For a wireless network, we consider wireless local broadcast model. We assume that each node has an omni-directional antenna and a single transmission can be received by *any* node within within the transmission radius (called *local broadcasting*), which is assumed to be a disk of appropriate radius centered at the node. We utilize this broadcasting property to reduce the communications needed in our algorithms. If the maximum transmission power is not enough to communicate with a node directly, then it can communicate through multihop wireless links by using intermediate nodes to relay the message. We also assume that while a node is transmitting a message, no other node within its transmission radius is allowed to transmit.

In Chapter 2, we present the NNT-scheme to construct a low-cost spanning tree, called nearest neighbor tree (NNT) and show that NNT using any ranking of the nodes gives an $O(\log n)$-approximation to MST in any weighted graph. Then we show that Random-NNT gives an $O(1)$-approximation to an MST in a random graph with nodes uniformly randomly distributed in $[0, 1]^2$. For geometric instances, Random-NNT constructs low-degree spanning trees which have $O(\log n)$ maximum degree, with high probability.

In Chapter 3, we present the generalized NN-Scheme for constructing low-weight k-connected spanning subgraphs in a complete metric graph. Then, based on NN-Scheme, we give a distributed approximation algorithm using the point-to-point comunication model to construct a minimum-weight k-connected spanning subgraph in a weighted complete graph [2]. We show that our algorithm gives an approximation ratio of $O(k \log n)$ for a metric graph, $O(k)$ for a random graph with nodes uniformly randomly distributed in $[0, 1]^2$ and $O(\log \frac{n}{k})$ for a complete graph with random edge weights $U(0, 1)$. Our algorithm takes $O(\log \frac{n}{k})$ time and expected $O(nk \log \frac{n}{k})$ messages. With $k = 1$, this algorithm is simply an approximation algorithm for MST in a complete graph.

In Chapter 4, we present a distributed algorithm that constructs an $O(\log n)$-approximate minimum spanning tree (MST) in any arbitrary network (with arbitrary topology and arbitrary edge-weights) [1]. This 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. $L(G,w)$ always lies between 1 and n. The parameter $L(G,w)$ can be smaller or larger than the diameter and typically it can be much smaller than $\sqrt{n}$. 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 exact MST.

In Chapter 5, we present an energy efficient implementation of the NNT-Scheme for construction of MSTs in a wireless ad hoc setting [3]. We show provable bounds on the performance with respect to both the *quality* of the spanning tree produced and the *energy needed* to construct them. In a wireless network, to transmit a signal over a distance r, the required *radiation energy* is proportional to r^α, where typically α is 2 and can range up to 4 in environments with multiple-path interferences or local noise [5, 22]. Motivated by this consideration, our focus is the following geometric weighted minimum spanning tree problem: given an arbitrary set N of points (nodes) in a plane, find a tree T spanning N such that $\sum_{(u,v)\in T} d^\alpha(u,v))$ is minimized where $d(u,v)$ is the distance between nodes u and v.

We show that when the points are distributed uniformly at random in the unit square, Random-NNT gives an $O(1)$ and $O(\log n)$ approximation, respectively, for the case of $\alpha = 1$ and $\alpha = 2$, respectively. In contrast, Co-NNT gives an $O(1)$ approximation for both $\alpha = 1$ and $\alpha = 2$. In the uniform random setting, we show that NNT algorithms have significantly lower message, time, and work complexity (radiation energy required to run the algorithm) compared to other distributed algorithms which compute the exact MST. We show that the average work complexities for Co-NNT and Random-NNT are $O(1)$ and $O(\log n)$, respectively, for $\alpha = 2$. These work complexities are within a constant of optimum, because for the case of points distributed uniformly in a unit square, the cost of the MST equals these values, within constant factors (see e.g., [23]) and this *lower bounds* the work complexity of *any* algorithm that constructs it. We also show that for both NNT algorithms, the expected message complexity is $O(n)$ (on average) and time complexity is $O(\log^2 n)$ (with high probability) which are essentially the best possible.

In addition to showing theoretical bounds, we also perform extensive simulations of our algorithms. We tested our algorithms on both uniformly random distributions of nodes, and on realistic distributions of nodes in an urban setting. Simulations validate the theoretical results and show that the bounds are much better in practice.

2 NEAREST NEIGHBOR TREE SCHEME

In this chapter, we present and analyze a simple scheme, called nearest neighbor tree scheme (NNT scheme) for constructing low-cost spanning trees. We show that our scheme constructs an $O(\log n)$-approximate minimum spanning tree (MST) in any weighted graph. We also show that our scheme is optimal with respect to the amount of "local information" needed to compute any connected spanning subgraph.

2.1 NNT Scheme

We present a simple and local scheme called *Nearest Neighbor Tree (NNT) scheme* to construct a low cost spanning tree — each node chooses a unique rank (thus the ranks define a permutation on the nodes), and connects to the closest node of higher rank. First we assume that the underlying graph is a complete metric graph. Later we will modify the scheme for an arbitrary graph. The basic NNT-scheme for a complete metric graph is given in Figure 2.1.

For a node v, let $nnt(v)$ denote the node that v connects to, if it exists — if v has the highest rank, $nnt(v)$ is not defined. If $nnt(v)$ is defined for node v, it must be the case that $r(nnt(v)) > r(v)$ and $r(v) > r(w)$, for each node w that is closer to v than $nnt(v)$. If we think of the edges $(v, nnt(v))$ as being directed from v to $nnt(v)$, it is clear that each edge is directed from a low rank node to a higher rank node — this immediately rules out cycles, and gives a spanning tree. Figure 2.2 shows a geometric example of the NNT construction for 5 points in the plane, along with the ranks.

Our NNT scheme is closely related to the approximation algorithm for the *traveling salesman problem* (coincidentally called Nearest Neighbor algorithm) analyzed in a classic paper of Rosenkrantz, Lewis, and Stearns [24]. Imase and Waxman [25] also used a scheme based on [24] (their algorithm can also be considered a variant of NNT

All nodes have a unique label *id* from a totally ordered set.

Output: A spanning tree.

NNT Scheme:

Every node $v \in V$ executes the following steps independently:

1. Each node v chooses a unique (distinct) rank $r(v)$.
2. Connect to the nearest node w such that $r(w) > r(v)$, i.e., add the edge (v, w) (to the spanning tree).

Figure 2.1. Nearest Neighbor Tree (NNT) Scheme.

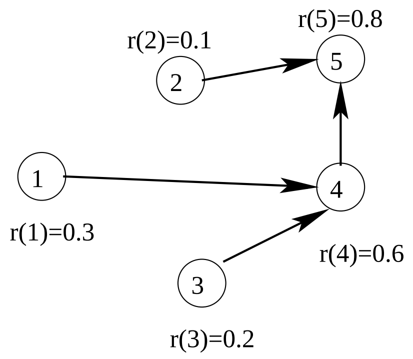

Figure 2.2. An geometric instance with 5 points (in a plane) showing the NNT construction. For each point i, $r(i)$ denotes its rank, and each point i (other than 5) is connected to the closest node of higher rank. Following the definition of $nnt()$, $nnt(1) = 4$, $nnt(2) = 5$, $nnt(3) = 4$ and $nnt(4) = 5$; $nnt(5)$ is not defined, since it has the highest rank.

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 because we 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.

Before showing how the nodes can choose distinct rank in a distributed fashion, we show a general result that NNT gives an $O(\log n)$ approximation to MST irrespective of how the ranks are chosen.

2.2 Cost of a Nearest Neighbor Tree

In this section we show a general result about the quality of a spanning tree constructed by *any* NNT scheme, i.e., ranks can be chosen arbitrarily as long as they are unique. Consider any weighted metric graph $G(V, E)$, i.e., all edges have non-negative weights that satisfy the triangle inequality. The cost of *any* NNT tree (i.e., using any arbitrary ranking of the nodes) is within a factor of $\lceil \log n \rceil$ of the cost (weight) of the MST on G (Theorem 2.2.1). We use log to denote logarithm to the base 2.

To prove Theorem 2.2.1, we use the following Lemma concerning the traveling salesman problem which appears in Rosenkrantz, Stearns, and Lewis [24, Lemma 1].

Lemma 2.2.1 *[24] Let $G = (V, E)$ be a weighted metric (complete) graph on n nodes. Let $d(p, q)$ be the weight of the edge between nodes p and q. Suppose there is a mapping l assigning each node p a number l_p such that the following two conditions hold:*

(a) $d(p, q) \geq min(l_p, l_q)$ for all nodes p and q.

(b) $l_p \leq \frac{1}{2}c(TSP)$ for all nodes p, where $c(TSP)$ is the cost of a optimal (shortest) traveling salesman tour in G.

Then $\sum_{p \in V} l_p \leq \frac{1}{2}(\lceil \log n \rceil + 1)c(TSP)$.

Theorem 2.2.1 *On a complete metric graph G, any NNT scheme constructs a tree whose cost is at most $\lceil \log n \rceil$ times the cost of (optimal) MST.*

Proof: Let $c(MST)$ be the cost of an MST. It can be shown that (e.g., see [26])

$$c(TSP) \leq 2c(MST). \tag{2.1}$$

To apply Lemma 2.2.1, based on the NNT scheme, we define a mapping l as follows:

$l_p = d(p, nnt(p))$ if $nnt(p)$ (the node that p connects to) exists; otherwise (if p is the highest-ranked node), $l_p = \frac{1}{2}c(TSP)$.

Mapping l satisfies condition (a): Let p and q be any two nodes, and without loss of generality, assume $rank(p) < rank(q)$. Then by definition of $nnt()$, $d(p,q) \geq d(p, nnt(p)) = l_p$ (note that p cannot be the highest-ranked node).

Mapping l satisfies also condition (b): It is trivially true for the highest-ranked node. For any other node p, $l_p = d(p, nnt(p))$. There are exactly two disjoint paths between p and $nnt(p)$ in the TSP route. Let S_1 and S_2 be these two paths. Then $c(S_1) + c(S_2) = c(TSP)$, and by triangle inequality, $d(p, nnt(p)) \leq c(S_1)$ and also $d(p, nnt(p)) \leq c(S_2)$. Thus $d(p, nnt(p)) \leq \frac{1}{2}c(TSP)$.

Let p_0 be the highest-ranked node. By construction of NNT and applying Lemma 2.2.1, we have

$$c(NNT) = \sum_{p \in V} l_p - l_{p_0} \leq \frac{1}{2}\lceil \log n \rceil c(TSP). \tag{2.2}$$

From Inequality 2.1 and 2.2, we have

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

□

Remark. We can show that the above bound is tight up to a constant factor. Consider a complete graph of n nodes which are uniformly spread on a straight line of unit length, where $n = 2^k + 1$ for some positive integer k. The nodes are (from left to right) $X_1, X_2, \ldots, X_n$. Assign the ranks to the nodes as follows: $rank(X_1) > rank(X_n) > rank(X_{\lceil n/2 \rceil}) > rank(X_{\lceil n/4 \rceil}) > rank(X_{\lceil 3n/4 \rceil}) > rank(X_{\lceil n/8 \rceil}) > rank(X_{\lceil 3n/8 \rceil}) > rank(X_{\lceil 5n/8 \rceil}) > rank(X_{\lceil 7n/8 \rceil}) \ldots$. The cost of MST and NNT on this graph is 1 and $1 + \frac{1}{2}\log(n-1)$ respectively.

2.3 NNT-Scheme for Arbitrary Graph

If the nodes cannot connect directly and/or edge weights do not satisfy triangle inequality, say, we are given a graph G with arbitrary topology and arbitrary edge weights,we modify the basic NNT scheme so that nodes connect to their closest node of higher rank using a *shortest path.* That is, add all of the edges in this shortest path in NNT. Notice that the resulting graph found by this scheme can contain cycles. To find a spanning tree from this resulting graph we need to remove some edges to break the cycles. The details of how such cycles can be broken are given later in Chapter 4.

We can show that the $O(\log n)$ approximation factor for NNT to MST still hold for any arbitrary graph. We note that the scheme produces a connected subgraph which may, in general, contain cycles. Let's call the subgraph produced by the above scheme as the NNT subgraph. We show next that this subgraph is of cost within a $O(\log n)$ factor of the MST of G. Thus any spanning tree (say, e.g., a breadth-first tree) of this NNT subgraph (this will be our NNT tree) will also have cost at most within $O(\log n)$ factor of MST.

The argument below is similar to the one showing that there is an approximation factor preserving reduction from the Steiner tree problem to the metric Steiner tree problem (see e.g., [27][Chapter 3]). For the purpose of proving the above claim, consider a virtual graph $H(V_h, E_h)$ that is constructed based on the original graph $G(V, E)$ as follows: 1) H has the same set of nodes as in G, i.e., $V_h = V$; 2) make H a complete graph, i.e., for all $u, v \in V_h$, $(u, v) \in E_h$; 3) assign weight to each edge $(u, v) \in E_h$ equal to the weight of the shortest path from u to v in G. Now it is clear that H is a complete metric graph. Let NNT_H and MST_H be the NNT tree and MST in graph H respectively, and NNT_G and MST_G denote the NNT subgraph and the MST in the original graph G. By Theorem 3.4.1, $c(NNT_H) \leq 4(\log n)c(MST_H)$. Since we form NNT subgraph in G by connecting nodes via shortest paths, we have $c(NNT_G) \leq c(NNT_H)$ (in general, it is less in because the same edge can occur in different shortest paths in G). Let $w(u, v)$ and $w_H(u, v)$ be the weights of edge (u, v) in G and H respectively. If $(u, v) \in E$, clearly $w_H(u, v) \leq w(u, v)$. If $(u, v) \notin E$, then we are simply having an extra edge in graph H, comparing to G. Thus, as a direct consequence, we have $c(MST_H) \leq c(MST_G)$, which leads to the conclusion $c(NNT_G) \leq c(NNT_H) \leq 4(\log n)c(MST_H) \leq 4(\log n)c(MST_G)$.

```
All nodes have a unique label $id$ from a totally ordered set.
    Each node $v$ is assigned a unique rank $r(v)$ as follows:
        1) $v$ chooses $p(v)$, a uniform random number $\in [0, 1]$.
        2) $r(w) > r(v)$ if $p(w) > p(v)$ or if $p(w) = p(v)$ and $id(w) > id(v)$.
```

Figure 2.3. Random ranking of the nodes.

2.4 Choosing Unique Rank

While ranks can be chosen in many ways, we mainly focus on a simple randomized way of choosing ranks: each node chooses a rank uniformly and independently at random from a totally ordered set. The detail is given in Figure 2.3. The NNT constructed using random ranking is called *Random-NNT*. Later in this thesis, we also introduce and analyze couple of other ranking scheme.

2.4.1 Average Neighborhood Size in Random-NNT

In NNT scheme, a node has to find the closest node of higher rank to connect to. For a node v, let $v_1, v_2, \ldots v_i, \ldots$ be the nodes that are at increasing distance from v, i.e., v_i is the ith nearest neighbor of v. For a given choice of ranks, let $s(v)$ be the number of nodes that v has to examine (starting from v_1) before it finds a node of higher rank (hence v will connect to $v_{s(v)}$). We call $s(v)$ the *size of the neighborhood* v has to look for, in order to find the connecting edge. The size of the neighborhood measures the amount of *local information* needed by a distributed algorithm. The quantity $s(v)$ has a bearing on the message complexity in distributed implementation of NNT (Section 3.5). For arbitrary choices of ranks, the average neighborhood size (i.e., $(1/n)\sum_v s(v)$) could be $\Omega(n)$. The following lemma shows that the average neighborhood size decreases significantly in Random-NNT.

Lemma 2.4.1 *In the case of Random-NNT, on any graph, the (expected) average neighborhood size is* $\Theta(\log n)$.

Proof: Let a be the random number generated by node v and x_i denotes the random number generated by the i^{th} nearest neighbor of v. Let $X_i = \{x_k | 1 \leq k \leq i\}$. We define, $a > X_i \iff \forall_{1\leq k\leq i}(a > x_k)$. Since the random numbers are generated by the nodes independently, $\Pr\{a > X_i\}$ = probability that a is the largest among $i+1$ independent identically distributed random numbers. That is, $\Pr\{a > X_i\} = \frac{1}{i+1}$. Now, the probability that a node connect to the i^{th} nearest neighbor is

$$\begin{aligned}
& \Pr\{a > X_{i-1}, a < x_i\} \\
= & \Pr\{a > X_{i-1}\}\Pr\{a < x_i | a > X_{i-1}\} \\
= & \Pr\{a > X_{i-1}\}[1 - \Pr\{a > x_i | a > X_{i-1}\}] \\
= & \Pr\{a > X_{i-1}\} - \Pr\{a > x_i, a > X_{i-1}\} \\
= & \Pr\{a > X_{i-1}\} - \Pr\{a > X_i\} \\
= & \frac{1}{i} - \frac{1}{i+1} = \frac{1}{i(i+1)}.
\end{aligned}$$

Let L be a random variable denoting the size of the neighborhood for a node.

$$E[s(v)] = E[L] = \sum_{i=1}^{n-1} i\Pr\{L = i\} = \sum_{i=1}^{n-1}\frac{1}{i+1} = \sum_{i=1}^{n}\frac{1}{i} - 1 = H_n - 1 = \Theta(\log n).$$

By linearity of expectation, the expected average neighborhood size is $\Theta(\log n)$.

□

The above result shows that a *local* distributed algorithm can potentially be developed to find a Random-NNT. Consider an algorithm where each node examines its neighbors beginning from the nearest neighbor until it finds a node of higher rank. Lemma 2.4.1 says that for Random-NNT, on average, each node needs information from $\Theta(\log n)$ nearest neighbors. In fact, this is the optimal local information needed to find any spanning tree on a complete network. Korach et al. [21, 28] showed that any distributed algorithm that constructs a spanning tree in a complete graph uses $\Omega(n \log n)$ edges. That is, on average, each node needs to use $\Omega(\log n)$ edges; i.e., each nodes needs information from at least $\Omega(\log n)$ other nodes. Thus average neighborhood size for any spanning tree is at least $\Omega(\log n)$. As a result, in terms of locality, Random-NNT scheme can be said to be optimal.

Another result by Korach et al., in the same article [21], implies that much larger locality is required to find a MST. They showed that any distributed algorithm to find an MST on a complete weighted graph uses $\Omega(n^2)$ edges. This lower bound can be shown to hold also for a complete metric graph. That is, each node uses information from $\Omega(n)$ other nodes on the average. Thus, the average neighborhood size for MST is $\Omega(n)$, i.e., exponentially more than that needed by Random-NNT.

3 GENERALIZED NNT-SCHEME FOR DISTRIBUTED CONSTRUCTION OF LOW-WEIGHT K-CONNECTED SPANNING SUBGRAPHS

In this chapter, we generalize the NNT-scheme to construct a low cost k-connected subgraph from a complete graph. We show that this scheme gives an approximation ratio of $O(k \log n)$ for a metric graph, $O(k)$ for a random graph with nodes uniformly randomly distributed in $[0,1]^2$ and $O(\log \frac{n}{k})$ for a complete graph with random edge weights $U(0,1)$. We then show that our scheme can be applied to design an efficient distributed algorithm for constructing such an approximate k-connected spanning subgraph (for any $k \geq 1$) in a point-to-point distributed model, where the processors form a complete network. Our algorithm takes $O(\log \frac{n}{k})$ time and expected $O(nk \log \frac{n}{k})$ messages. We also show that for the geometric instances, our randomized scheme constructs low-degree k-connected spanning subgraphs which have $O(k \log n)$ maximum degree, with high probability.

3.1 Introduction

Computing the low weight spanning subgraphs of a given graph $G(V,E)$ with non-negative edge weights is a fundamental problem in network design (e.g., see [15,16] for an extensive survey). One important problem in this setting is the *k-vertex connectivity problem* (henceforth simply the *k-connectivity* problem): find a spanning subgraph of minimum weight that is k-vertex-connected, i.e., there exists k vertex-disjoint paths between every pair of vertices. Finding an optimal k-connected spanning subgraph is NP-hard for $k \geq 2$ even if the weights of the edges satisfy the triangle inequality, or even when the graph is a complete Euclidean graph [17]. There has been a lot of work on designing approximation algorithms for the k-connectivity problem. Most of these algorithms are centralized algorithms which are quite sophisticated and their main goal is to obtain a polynomial time algorithms with the best possible approximation ratio (see e.g., [18–20]). Distributed algorithms for the k-connectivity ($k \geq 2$) problem has received limited attention thus far — this is especially true for the weighted version. In fact, to the best of our knowledge there is no known efficient distributed algorithm for $k \geq 2$ for weighted graphs. In contrast, for $k = 1$ — the minimum spanning tree (MST) problem — optimal distributed algorithms are well-known [12, 13]. With the emergence of the new networking technologies such as ad hoc and sensor networks, there is an increasing need for distributed algorithms that are simple and easily implementable, have low communication complexity, and perform reasonably well (e.g., see [13, 29, 30]). Such simple local algorithms are desirable even for the MST problem, where optimal distributed algorithms are known (see e.g., [11–13]), because these algorithms are quite complex, involve a lot of message complexity and synchronization to implement in a light weight and unreliable environment, such as ad hoc networks. This motivates the question of developing simple, local control, approximate algorithms. This also adds a new dimension to the design of distributed algorithms for such networks: we can potentially *tradeoff* optimality of the solution to the amount of resources (messages, time etc) consumed by the algorithm. This is the motivation for the relatively new area of *distribution approximation* (we refer to the survey by Elkin [12]).

In this chapter, we study a very simple randomized scheme called *Random Nearest Neighbor (Random-NN) scheme* for constructing a low-weight k-connected spanning subgraph (for any $k \geq 1$) and show some of its properties and applications. The Random-NN scheme is based on a simple idea (cf. Section 3.3): each node chooses a unique *rank*, a quantity that is *randomly* chosen from a totally ordered set, and a node connects to its k *nearest* nodes of higher rank. We first show that our scheme gives a simple approximation algorithm to construct a minimum-weight k-connected spanning subgraph in a weighted complete graph, a NP-hard problem even if the weights satisfy the triangle inequality. We show that our algorithm gives an approximation ratio of $O(k \log n)$ for a metric graph, $O(k)$ for a random graph with nodes uniformly randomly distributed in $[0,1]^2$ and $O(\log \frac{n}{k})$ for a complete graph with random edge weights $U(0,1)$. We show that our scheme is optimal with respect to the amount of "local information" (in an average sense — defined precisely in Section 3.3.2) needed to compute any connected spanning subgraph.

We next show that our scheme can be applied to design an efficient distributed algorithm for constructing an approximate k-connected spanning subgraph ($k \geq 1$) in a point-to-point distributed model, where the processors form a complete network. Our algorithm takes $O(\log \frac{n}{k})$ time and expected $O(nk \log \frac{n}{k})$ messages contrasting a result of Korach et al. [21] that shows that $\Omega(n^2)$ is a lower bound of the number of messages required to find an MST (i.e., $k = 1$) in this model. Thus, the expected message complexity of our algorithm is significantly better than the best distributed algorithm that finds the (optimal) MST. The proof of this $\Omega(n^2)$ bound on the number messages for finding MST (cf. Theorem 1 in [21]) can easily be modified to show that $\Omega(n^2)$ is also a lower bound on the message complexity for finding a minimum k-connected spanning subgraph for any $k \leq \lfloor n/2 \rfloor - 1$. This lower bound also holds for the metric weights. This also implies that our algorithm, for this restricted distributed computation model, has provably better asymptotic message complexity than the best distributed algorithm that finds a minimum k-connected subgraph, for any $k = o(n)$. However, the price for this gain is that our algorithm has a somewhat weaker approximation ratio compared to the best-known centralized algorithms.

We also show that for the geometric instances (these are relevant, for example, in the ad hoc sensor network applications [31]), our scheme constructs low-degree k-connected spanning subgraphs (these are useful in many applications e.g., see [17]) which have $O(k \log n)$ maximum degree, with high probability.

The rest of the chapter is organized as follows. In Section 3.2, we discuss related work. In Section 3.3, we present the Random-NN scheme, to construct a k-connected spanning subgraph on a given weighted complete graph. The analysis of the weight of k-connected subgraph produced by Random-NN scheme and approximation ratios for various graph models are given in Section 3.4. In Section 3.5, we describe a distributed implementation of the Random-NN scheme, and analyze its time and message complexities. We conclude in Section 3.6 with a discussion on future work.

3.2 Related Work

The work that is closest in spirit to our work is perhaps that of Imase and Waxman [25]. They consider the dynamic Steiner tree problem, where the objective is to maintain a near-optimal Steiner tree when nodes are added or deleted. They show that, under additions only (no deletions), a simple greedy algorithm which connects the just added node to nearest existing node (by the shortest path, i.e., they assume the triangle inequality) gives a $O(\log n)$ approximation. Their algorithm can be considered as a variant of our NN scheme for finding the spanning tree (i.e., the special case: $k = 1$). However, their algorithm will not work in a distributed (unlike our scheme) because we cannot connect to the shortest node (they can do that since the nodes are added one by one) as this can introduce cycles.

Random ranks were used to construct forests, by a slightly different process by Toroczkai and Bassler [32]. The process defined here chooses a random rank for each node on a graph in $G(n, p)$, and each node connects to the neighbor of highest rank. They show that the resulting forest has a power law degree distribution, which they use as a model for explaining power laws in networks.

The work of Panconesi and Rizzi [33] also uses an approach based on ranking of nodes to design simple, fast, and *deterministic* distributed algorithms to find maximal matchings, edge/vertex-colorings, and maximal independent sets. This approach however is not comparable to our Random-NN scheme because the edge weights play no role in their algorithms (they are for unweighted networks).

We now briefly mention some previous results on the centralized approximation algorithms for the k-connectivity problem ($k \geq 2$). For the general graph setting, where edge weights are arbitrary, a k-approximation algorithm is given in [19]. Cheriyan et al. [18] achieved an approximation ratio of $6H_k = O(\log k)$ for the case where

$k \leq \sqrt{n/6}$. For the case where $k < (1-\epsilon)n$, they achieved an approximation ratio of $\sqrt{n/\epsilon}$. Recently, an $O(\ln^2 k \cdot \min\{\frac{n}{n-k}, \frac{\sqrt{k}}{\ln k}\})$ approximation algorithm was given in [20].

For the metric weights (the edge weights satisfy the triangle inequality), a $2+\frac{k-1}{n}$-approximation algorithm was given in [19]. Czumaj et al. [34] presented a centralized $(1+\epsilon)$-approximation ($\epsilon > 0$) algorithm for the minimum-weight k-connected spanning subgraph problem for a complete Euclidean graph with constant dimension. They also show that there is no polynomial time $(1+\epsilon)$-approximation algorithm for a complete Euclidean graph in dimension $\log n$ or higher unless $P = NP$. This result also implies the same hardness of approximation in a complete metric graph.

We now mention the previous work on distributed algorithms. Most of these algorithms assume that the graph is unweighted and the goal is to find a sparse k-connected subgraph. The algorithm of Cheriyan et al. [35] finds k edge-disjoint breadth first (BFS) forests, which gives a k-connected subgraph. The distributed implementation of this algorithm has time and message complexity as $O(kn \log^3 n)$ and $O(k|E| + kn \log^3 n)$ respectively. Thurimella [36] improved the time complexity to $O(kD + kn^{0.614})$ where D is the diameter of G, but the message complexity was ignored and can be much larger than that of the algorithm given in [35]. Using similar ideas, Jennings et al. [37] developed a distributed algorithm for the k-vertex connected subgraph problem which takes $O(n)$ time and $O(|E|)$ messages. In the same paper, they also presented a distributed algorithm for the k-edge connectivity problem which takes $O((k+D)\log^3 n)$ time and $O(k|E| + kn \log^3 n)$ messages. All of these algorithms [35–37] produces a k-connected subgraph with $O(kn)$ edges from an unweighted k-connected graph G. There is also work on distributed algorithms [38,39] for finding the biconnected components ($k = 2$, unweighted graph). Both of the algorithms given in [38] and [39] take at least linear time.

We now state relevant known distributed algorithms in the complete network model. Korach et al. [21] showed a lower bound of $\Omega(n^2)$ messages for any distributed algorithm computing a minimum weight spanning tree. This result holds even when the weights satisfy the triangle inequality. We note that our algorithm significantly beats the above lower bound at the cost of producing somewhat sub-optimal solutions. In contrast, Korach et al. gave algorithms that needed only $O(n \log n)$ messages for a class of problems that included the spanning tree problem and the leader election problem. In another paper [28], they showed that $\Omega(n \log n)$ messages are necessary for this class of problems. They also showed, however, that for the maximal matching problem and the Hamiltonian circuit problem, $\Omega(n^2)$ messages are necessary and gave the algorithms that matched this lower bound.

3.3 A Scheme to Construct a k-connected Subgraph in a Weighted Clique

We provide a simple scheme to construct a k-vertex connected spanning subgraph in a given complete weighted graph $\mathbb{K}_n$ of n nodes. We assume that there is a non-negative weight, $c(u, v)$ is associated with each edge (u, v) of the graph. The objective is to determine the edges that will be in the k-connected subgraph and to keep the weight of the k-connected subgraph low. The *weight of a k-connected subgraph* is the sum of the weights of the edges in it. In this section, we present a basic scheme (an abstract algorithm) and prove that this scheme indeed constructs a k-connected graph.

The scheme is quite simple. Each node u is given a unique rank $r(u)$. By *unique rank*, we mean that no two nodes have the same rank. Thus a ranking of the nodes corresponds to a permutation of the nodes. For two nodes u and v with $u \neq v$, either $r(u) < r(v)$ or $r(u) > r(v)$. Once the ranks of the nodes are chosen, they remain unchanged throughout the execution of the algorithm. Later, we will see how such ranks can be chosen. To form a k-connected graph, each node u is connected to k nearest nodes w_i, such that $r(u) < r(w_i)$ for all $1 \leq i \leq k$. Node v_1 is nearer than v_2, to u, if $c(u, v_1) < c(u, v_2)$. If $c(u, v_1) = c(u, v_2)$, break the tie arbitrarily, i.e., choose any one of v_1 and v_2 arbitrarily. The nearest nodes are chosen to minimize the weight of the constructed subgraph. However, connecting to any k higher ranked nodes produces a k-connected graph as shown below (Proposition 3.3.1).

If a node does not have enough nodes of higher rank to get connected to, it is connected to the available higher ranked nodes. For example, to form a 2-connected graph, the highest ranked node does not have any such node. The second highest ranked node has only one such node, the highest ranked node. Every other node has at least two nodes to connect to. Obviously, the highest ranked node is connected to at least two other nodes, but it is not the initiator of any connection. By *"u is connected to v"*, we mean that u (the lower ranked node) is the initiator of the connection to v. We use $\eta(u)$ to denote the set of the nodes whom u is connected to.

Consider an enumeration of the nodes, $v_1, v_2, \ldots, v_n$, where v_i be the node of ith rank; for any $i > j$, $r(v_i) > r(v_j)$. In the above scheme, each node v_i is connected to the nearest $\min\{k, n-i\}$ neighbors in $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$. Clearly, $|\eta(v_i)| = \min\{k, n-i\}$ and for the highest ranked node v_n, $\eta(v_n) = \phi$. We call this scheme *nearest neighbor scheme* or *NN-scheme.*

The following known proposition (Proposition 3.3.1) ensures that the NN-scheme constructs a k-connected subgraph.

Proposition 3.3.1 *Let $G = (V, E)$ be a graph on $V = \{v_1, v_2, \ldots, v_n\}$ with $n \geq k+1$ so that every v_i has at least $\min\{k, n-i\}$ neighbors in $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$. Then G is k-connected.*

Proof: If $n = k+1$, then G is a complete graph. Assume that $n \geq k+2$ and suppose to the contrary that G is not k-connected. Then there is a $C \subseteq V$ with $|C| \leq k-1$ so that $G - C$ is disconnected. Let X, Y be two distinct connected components of $G - C$, and let $x = \max_{v_i \in X} i$ and $y = \max_{v_i \in Y} i$. For any $i > x$, if v_i is a neighbor of v_x, then v_i must be in C. Now v_x has at most $k-1$ (since $|C| \leq k-1$) and at least $\min\{k, n-x\}$ neighbors in $\{v_{x+1}, v_{x+2}, \ldots, v_n\}$. Thus, we must have $\{v_{x+1}, v_{x+2}, \ldots, v_n\} \subseteq C$; hence $x > y$. The same argument applied on v_y gives $y > x$. Thus we have a contradiction. $\square$

Before finding the above shorter proof, we developed the following inductive proof. Let $C_G(v_i)$ be the number of connections by node v_i to higher ranked nodes in G. Observe that the above scheme satisfies the following.

$$\begin{aligned} C_G(v_i) &= i-1 \quad \text{for } 1 \le i \le k, \\ &= k \qquad\ \ \text{for } k+1 \le i \le n, \end{aligned}$$

where n is the number of nodes. The following theorem proves that the above scheme constructs a k-connected subgraph.

Theorem 3.3.1 *If $C_G(v_i) \ge i-1$ for $1 \le i \le k$ and $C_G(v_i) \ge k$ for $k+1 \le i \le n$, then G is k-connected.*

Proof: The proof is given by induction on k. The base case is $k = 1$. When $k = 1$, the number of edges in G is at least $n-1$, where each node (except the highest ranked node) is connected to *at least* one higher ranked node. It is easy to see that when each node (except the highest ranked node) is connected to *exactly* one higher ranked node, there is no cycle in the resultant graph and it is connected; in fact, it is a tree. Therefore, G is indeed 1-connected.

Induction hypothesis: If $C_G(v_i) \ge i-1$ for $1 \le i \le k-1$ and $C_G(v_i) \ge k-1$ for $k \le i \le n$, G is $(k-1)$-connected.

To apply our hypothesis we make use of the following property of a k-connected graph (see e.g., [40, Theorem 3.3.5]). A k-connected graph is a connected graph such that removing any $k-1$ vertices and their incident edges leaves the resulting graph connected. Inductively, this can also be stated as: A k-vertex-connected graph is a connected graph such that removing any one vertex and its incident edges leaves the resulting graph $k-1$ connected.

Let G' be the graph formed by deleting an arbitrary node v_j and its adjacent edges from G. For clarity, let v_i and v'_i denote the ith ranked nodes in G and G' respectively.

After removing v_j, for $1 \le i \le j-1$, the number of connections for v_i remains unchanged; that is, for $1 \le i \le j-1$, $C_{G'}(v'_i) = C_G(v_i)$. For $j+1 \le i \le n$, v_i can loose at most one connection and the ith ranked node in G becomes the $(i-1)$st ranked node in G' i.e., the node v_i in G is now node v'_{i-1} in G'. Therefore, for $j+1 \leq i \leq n$, $C_{G'}(v'_{i-1}) \geq C_G(v_i) - 1$.

For $2 \leq i \leq j$, $C_{G'}(v'_{i-1}) = C_G(v_{i-1}) \geq C_G(v_i) - 1$. That is, for $2 \leq i \leq n$, $C_{G'}(v'_{i-1}) \geq C_G(v_i) - 1$.

Now for $1 \leq i \leq k-1$, $C_{G'}(v'_i) \geq C_G(v_{i+1}) - 1 \geq (i+1) - 1 - 1 = i - 1$.

For $k \leq i \leq n-1$, $C_{G'}(v'_i) \geq C_G(v_{i+1}) - 1 \geq k - 1$.

By induction hypothesis, G' is $(k-1)$-connected. That is, G is k-connected. $\square$

Nearest Neighbor Tree (NNT). When $k = 1$, the NN-scheme produces a spanning tree. If $k = 1$, each node (except the highest ranked node) connects to exactly one higher ranked one. Thus there are $n - 1$ edges in the resulting graph. If we consider each edge is directed from the lower ranked node to the higher ranked node, it is easy to see that there is no cycle in this graph. Therefore, the resulting graph is a tree spanning all n nodes. We call this spanning tree a *nearest neighbor tree*, or in short, NNT.

We use the following definitions and notations in the rest of the chapter.

Let $\mathbb{N}_u(i)$ denote the i^{th} nearest neighbor of u in the given complete graph $\mathbb{K}_n$.

Definition 3.3.1 ***i-neighborhood.*** *The i-neighborhood of a node u, denoted by $\Gamma_u(i)$, is the set of the i nearest neighbors of u in $\mathbb{K}_n$; i.e., $\Gamma_u(i) = \{\mathbb{N}_u(t) | 1 \leq t \leq i\}$. We define $\Gamma_u(0) = \phi$.*

Definition 3.3.2 ***jth connection.*** *Let $w_1, w_2, \ldots, w_{|\eta(u)|}$, in non-decreasing order of $c(u, w_t)$, be the nodes in $\eta(u)$. The connection u makes to w_j, for any $1 \leq j \leq |\eta(u)|$, i.e., the edge (u, w_j), is called the jth connection of u.*

3.3.1 Random Ranking

Now we analyze the NN-Scheme using random ranking. Random ranking is defined in the previous chapter. Note that the identifiers of the nodes also constitute a ranking of the nodes. However, here we are interested in a random ranking. We will see later, using random ranking, in contrast to an arbitrary ranking, we can have a better bound on the weight of the k-connected subgraph given by the NN-scheme and on the time and message complexity of the distributed implementation of the NN-scheme. Henceforth, we call the NN scheme with the random ranking as the *Random-NN scheme.*

Later, in the analysis of weight, time, and message complexity, we will use the following lemma regarding the random ranking of the nodes.

Lemma 3.3.1 *When a random ranking is used, the probability that an arbitrary node u makes the jth connection to $\mathbb{N}_u(i)$ is $\frac{j}{i(i+1)}$ for $i \geq j$.*

Proof: Node u makes the jth connection to $\mathbb{N}_u(i)$ if and only if $r(\mathbb{N}_u(i)) > r(u)$ and there are exactly $j-1$ nodes in $\Gamma_u(i-1)$ with ranks higher than $r(u)$. That is, $r(u)$ is exactly $(j+1)$st among the ranks of these $i+1$ nodes (u and the i nodes in $\Gamma_u(i)$) and $\mathbb{N}_u(i)$ is one of the j highest ranked nodes among the i nodes in $\Gamma_u(i)$.

Thus, the desired probability is $\frac{1}{i+1} \times \frac{j}{i} = \frac{j}{i(i+1)}$ for $i \geq j$. $\square$

Remarks: 1) It is not possible for u to make the jth connection to a node closer than $\mathbb{N}_u(j)$.

2) The probability that u is able to make the jth connection is $\sum_{i=j}^{n-1} \frac{j}{i(i+1)} = 1 - \frac{j}{n}$. That is, j out of n nodes do not have their jth connection.

3.3.2 Average Neighborhood Size in Random-NN Scheme

In the NN-scheme, a node has to find the k closest nodes of higher rank to connect to. For a node u, let $v_1, v_2, \ldots v_i, \ldots$ be the nodes, in non-decreasing order of $c(u, v_i)$, i.e., v_i is the ith nearest neighbor of u. For a given choice of ranks, let $s(u)$ be the number of nodes that u has to examine (starting from v_1) before it finds the required number of nodes of higher rank. We call $s(u)$ the *size of the neighborhood*, which u has to look for, in order to find the connecting edges. The size of the neighborhood measures the amount of *local information* needed by a distributed algorithm. The quantity $s(u)$ has a bearing on the message complexity in distributed implementation (Section 3.5). For arbitrary choices of ranks, the average neighborhood size (i.e., $(1/n)\sum_u s(u)$) could be $\Omega(n)$. The following lemma shows that the average neighborhood size decreases significantly if we use the random ranking (Random-NN scheme). The notation H_n is used to denote the harmonic series $\sum_{i=1}^{n}\frac{1}{i} = \Theta(\log n)$.

Lemma 3.3.2 *Let an arbitrary node u makes the k-th connection to $\mathbb{N}_u(L)$. Then $E[L] = k(H_n - H_k) = \Theta(k \log \frac{n}{k})$.*

Proof: Using Lemma 3.3.1,

$$E[L] = \sum_{i=k}^{n-1} \frac{k}{i(i+1)} i = k(H_n - H_k).$$

□

The above result shows that an efficient distributed algorithm can potentially be developed for the Random-NN scheme. Consider an algorithm where each node examines its neighbors beginning from the nearest neighbor until it finds the connecting edges. Lemma 3.3.2 says that using a random ranking, on average, each node needs information from $\Theta(k \log \frac{n}{k})$ nearest neighbors. This is optimal in general, because this is the optimal local information needed to find any spanning tree ($k = 1$) on a complete network. Korach et al. [21, 28] showed that any distributed algorithm that constructs a spanning tree in a complete graph uses $\Omega(n \log n)$ edges. That is, on average, each node needs to use $\Omega(\log n)$ edges; i.e., each nodes needs information from at least $\Omega(\log n)$ other nodes. Thus average neighborhood size for any spanning tree is at least $\Omega(\log n)$. As a result, in terms of locality, Random-NN scheme can be said to be optimal in general.

Another result by Korach et al. [21] implies that a much larger locality is required to find a minimum spanning tree (MST). They showed that any distributed algorithm to find an MST on a complete weighted graph uses $\Omega(n^2)$ edges. The proof of this $\Omega(n^2)$ bound on the number of messages for finding an MST (cf. Theorem 1 in [21]) can easily be modified to show that $\Omega(n^2)$ is also a lower bound on the number of messages for finding an optimal k-connected spanning subgraph for any $k \leq \lfloor n/2 \rfloor - 1$. This lower bound can be shown to hold also for a complete metric graph. That is, each node uses information from $\Omega(n)$ other nodes on the average. Thus, the average neighborhood size to find an optimal k-connected subgraph is $\Omega(n)$, which is exponentially larger than that needed by the Random-NN scheme.

3.4 Weight of the k-Connected Subgraph

We analyze the weight of the k-connected graph constructed by the NN scheme with respect to the minimum weight k-connected (sub)graph. Throughout the rest of the chapter, we use G_k and MKG to denote the k-connected graph constructed by the NN scheme and a minimum weight k-connected graph, respectively.

Let $G = (V, E, W)$ be any weighted undirected graph, where V is the set of vertices, E is the set of edges and $W =< c(u, v) >$, where $c(u, v) \geq 0$ is the weight of the edge $(u, v) \in E$. The weight of G is defined by $c(G) = \sum_{(u,v) \in E} c(u, v)$.

Using the following known proposition (Proposition 3.4.1), we have $c(MKG) \geq \frac{k}{2} c(MST)$. Later, we use this lower bound of $c(MKG)$ to obtain an upper bound for the approximation ratio $c(G_k)/c(MKG)$.

Proposition 3.4.1 *Any k-edge-connected graph G has a spanning tree T with $c(T) \leq 2c(G)/k$.*

Proof: Let D be the bidirection of G; i.e., for each edge (u, v) in the undirected graph G, there are two directed edges (u, v) and (v, u) in the directed graph D. Let w be any node in G. In the graph G, there are k edge-disjoint paths from w to any other node. Then, in D, there are k edge-disjoint directed paths from w to any other node. Edmonds [41] proved that if a directed graph has k edge disjoint paths from a node w to any other node, then it contains k edge-disjoint arborescences rooted at w. Thus D contains k edge-disjoint arborescences rooted at w. Let T be the underlying tree of the least weight arborescence among them. Then $c(T) \leq c(D)/k = 2c(G)/k$. $\square$

Before finding the above proof which depends on the some previous results, we had the following self-contained proof. We wanted to make the alternative proof available to the readers.

Proof: For $k \leq 2$, the inequality is trivially true. For $k = 1$ and 2, $c(MKG) \geq c(MST)$, otherwise we can construct a spanning tree with lower cost than a minimum spanning tree. The following proof is for $k \geq 3$.

We remove edges from MKG until we get a spanning tree, say T, of MKG. Initially, in MKG, each vertex (and edge) is in some cycle because there are at least $k \geq 3$ edge-disjoint paths between any two vertices. After deleting some edges from MKG, there can be edges and vertices which are not in any cycle. Such vertices (edges) are called *out-of-cycle* vertices (edges). The other vertices (edges) are called *in-cycle* vertices (edges). By *largest edge*, we refer to the edge having the largest weight.

We assume the following process of removing edges from MKG to form the spanning tree T.

1. Select the largest edge from all in-cycle edges. If there is more than one largest edge, select one arbitrarily.

2. Remove the selected edge from the graph.

3. Repeat steps 1 and 2 until there is no cycle in the resulting graph.

The above edge removal process never disconnects the graph since an edge is removed from a cycle only. At the end of the process, there is no cycle. Thus the resulting graph T is a spanning tree. Once an edge becomes out-of-cycle, it can never be removed and will eventually be in T.

The following lemma is needed to complete the proof.

Lemma 3.4.1 *For any integer $t \geq 1$, after removing exactly $(k-2)t$ edges, there are at most $2t-2$ out-of-cycle edges in the remaining graph.*

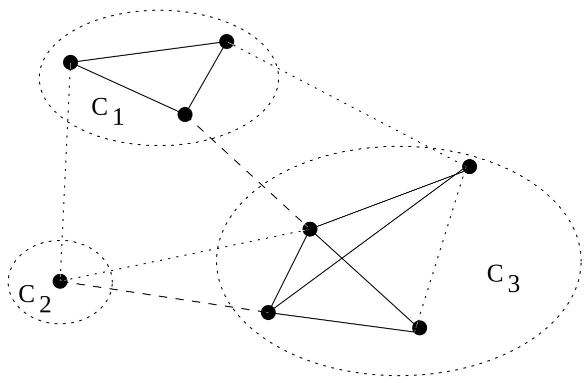

Figure 3.1. In this graph, three types of edges are shown: solid, dashed, and dotted lines. MKG contains all of the edges (solid, dashed, and dotted). Here $k = 3$, i.e., MKG is 3-connected. Dotted edges are removed by the edge removal process and thus G_r contains solid and dashed edges only. The two dashed edges are out-of-cycle edges ($m = 2$). If we remove these two out-of-cycle edges from G_r, we have three components C_1, C_2, and C_3.

Proof: Let G_r be the remaining graph after removing exactly $(k-2)t$ edges from MKG and m be the number of out-of-cycle edges in G_r.

We assume that $m \geq 1$. If $m = 0$, the lemma holds vacuously. If we remove all out-of-cycle edges from G_r, the resulting graph is disconnected and the number of components is exactly $m+1$ (see Figure 3.1). Let these components be $C_1, C_2, \ldots, C_{m+1}$. Now consider a partitioning of MKG into $m+1$ partitions where the ith partition P_i contains exactly the same vertices as in C_i. Each partition P_i shares at least k cross edges with other partitions. Otherwise, MKG is not k-connected. We have at least $\lceil\frac{(m+1)k}{2}\rceil$ cross edges. Each of these cross edges can only be either one of the m out-of-cycle edges (dashed lines in the figure) in G_r or one of the $(k-2)t$ removed edges (dotted lines in the figure).

Thus

$$\lceil\frac{(m+1)k}{2}\rceil \leq m + (k-2)t$$
$$\Rightarrow m \leq 2t - \frac{k}{k-2}.$$

For $k \geq 3$, $\frac{k}{k-2} > 1$, i.e., $m < 2t - 1$. □

Let λ_i be the weight of the ith largest edge in MKG, $1 \leq i \leq |E_k|$, where E_k is the set of edges in MKG. If more than one edge has the same weight, we break ties arbitrarily. Thus $\lambda_i \geq \lambda_{i+1}$ for all $1 \leq i \leq |E_k| - 1$. In a similar fashion, let μ_j be the weight of the jth largest edge in T for $1 \leq j \leq n-1$, where n is the number of vertices.

By using Lemma 3.4.1, among $(k-2)t + 2t - 2 = kt - 2$ largest edges in MKG, T can have at most $2t-2$ edges. Thus $\mu_{2t-1} \leq \lambda_{kt-1}$ and $\mu_{2t} \leq \lambda_{kt}$ for $1 \leq t \leq \lceil \frac{n}{2} \rceil - 1$, and also $\mu_{n-1} \leq \lambda_{kn/2-1}$ when n is even.

For odd n,

$$c(T) = \sum_{t=1}^{\lceil \frac{n}{2} \rceil - 1} (\mu_{2t-1} + \mu_{2t}) \leq \sum_{t=1}^{\lfloor \frac{n}{2} \rfloor} (\lambda_{kt-1} + \lambda_{kt}).$$

For even n,

$$c(T) = \sum_{t=1}^{\frac{n}{2}-1} (\mu_{2t-1} + \mu_{2t}) + \mu_{n-1} \leq \sum_{t=1}^{\lfloor \frac{n}{2} \rfloor} (\lambda_{kt-1} + \lambda_{kt}).$$

For any n,

$$c(T) \leq \sum_{t=1}^{\lfloor \frac{n}{2} \rfloor} (\lambda_{kt-1} + \lambda_{kt}) \leq \sum_{t=1}^{\lfloor \frac{n}{2} \rfloor} \frac{2}{k} \sum_{l=0}^{k-1} (\lambda_{kt-l}) = \frac{2}{k} \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor k} \lambda_i.$$

Now $|E_k| \geq \lceil \frac{kn}{2} \rceil \geq \lfloor \frac{n}{2} \rfloor k$, and thus

$$c(T) \leq \frac{2}{k} \sum_{i=1}^{|E_k|} \lambda_i = \frac{2}{k} c(MKG).$$

That is, $c(MKG) \geq \frac{k}{2} c(T) \geq \frac{k}{2} c(MST)$. $\square$

We can find an example where $c(MKG)$ is exactly equal to $\frac{k}{2} c(MST)$. This shows that this lower bound for the weight of MKG is tight. It is possible to construct a k-connected graph having exactly $\frac{kn}{2}$ edges. Consider a k-cube graph where weight of each edge is one unit. Number of nodes in a k-cube graph is $n = 2^k$. Each node is uniquely identified by a k-tuple $< b_1, b_2, \ldots, b_k >$ where $b_i \in \{0, 1\}$ for $1 \leq i \leq k$. There is an edge between any two nodes u and v if and only if the k-tuples of u and v differ in exactly one component. A k-cube graph is k-connected and the degree of each node is k. Thus, the number of edges is $\frac{kn}{2}$. The weight of this k-connected graph is $\frac{kn}{2}$ and the weight of an MST on this graph is $n-1$. The ratio of these weights is $\frac{kn}{2(n-1)}$, which approaches $\frac{k}{2}$ as $n \to \infty$.

Next we analyze the weight of G_k (output of the NN scheme) and its approximation ratios to MKG for graphs with edge weights satisfying various characteristics.

3.4.1 Metric Graph

A metric graph is a complete weighted graph where the weights of the edges satisfy the triangle inequality. We show that for a metric graph, using any arbitrary ranking of the nodes, the NN scheme outputs a k-connected subgraph with approximation ratio of $O(k \log n)$ to MKG (Theorem 3.4.1).

In the rest of this section, we use I_k to denote the sum of the first k positive integers, i.e., $\sum_{i=1}^{k} i = \frac{1}{2}k(k+1)$.

Theorem 3.4.1 *On a metric graph G of n nodes, for any arbitrary ranking of the nodes, the weight of the k-connected graph G_k constructed by the NN-scheme, $c(G_k) = O(k \lg n)c(MKG)$, where MKG is a minimum k-connected subgraph of G.*

Proof: The theorem holds trivially for $n \leq k$. The following proof is constructed for $n \geq k+1$.

Construct a hamiltonian path S such that $c(S) \leq 2c(MST)$, where MST is a minimum spanning tree on G. Such a path S can be constructed as follows (e.g., see [26]): select any node to be the root of the MST and perform a preorder tree walk on the MST. Let the order of the nodes, as they are visited in the preorder walk, be $v_1, v_2, \ldots, v_n$. (Note that this order of the nodes is used only to construct S. To construct G_k, we assume an arbitrary ranking, which can be different from this ordering, of the nodes.) Now, add the edges (v_i, v_{i+1}) to S, for $i = 1, 2, \ldots, n-1$.

For any i, j such that $1 \leq i \leq j \leq n$, let $S_{i,j}$ be the sub-path $< v_i, v_{i+1}, \ldots, v_j >$ and $V_{i,j}$ the subset $\{v_i, v_{i+1}, \ldots, v_j\}$. Let $G_{i,j}$ be the subgraph of G induced by $V_{i,j}$, and $F_{i,j}$ be the k-connected subgraph produced by the NN scheme running on $G_{i,j}$.

Now, by induction on the number of nodes $|V_{i,j}|$, we show that for any i and j such that $|V_{i,j}| \geq k+1$,

$$c(F_{i,j}) \leq 2I_k c(S_{i,j}) \lg |V_{i,j}|. \tag{3.1}$$

The basis of the induction is any i, j such that $k+1 \leq |V_{i,j}| \leq 2k+1$. The number of edges in $F_{i,j}$ is $k|V_{i,j}| - I_k$. Since the weights of the edges satisfy the triangle inequality, the weight of any edge in $F_{i,j}$ is at most $c(S_{i,j})$. Thus, we have

$$c(F_{i,j}) \leq (k|V_{i,j}| - I_k)c(S_{i,j}) \leq (k(2k+1) - I_k)c(S_{i,j}) \leq 2I_k c(S_{i,j}) \lg |V_{i,j}|$$

by assuming $|V_{i,j}| \geq 3$. For $|Vi,j| = 2$, Inequality 3.1 holds trivially for any $k \geq 1$.

Now we show the induction step. Consider any i, j such that $|V_{i,j}| \geq 2k+2$. Let $m = |V_{i,j}|$ and $x = \lfloor (i+j)/2 \rfloor$. By the induction hypothesis,

$$c(F_{i,x}) \leq 2I_k c(S_{i,x}) \lg |V_{i,x}| = 2I_k c(S_{i,x}) \lg \lceil m/2 \rceil,$$
$$c(F_{x+1,j}) \leq 2I_k c(S_{x+1,j}) \lg |V_{x+1,j}| = 2I_k c(S_{x+1,j}) \lg \lfloor m/2 \rfloor.$$

For any node $v \in V_{i,x}$, if w_1, w_2 are the t^{th} closest (to v) nodes of higher rank in $V_{i,x}$ and $V_{i,j}$, respectively, then $c(v, w_2) \leq c(v, w_1)$; a similar statement holds for any node in $V_{x+1,j}$. Therefore, for any node v, the weight of the t^{th} connection chosen by v in $F_{i,x}$ or $F_{x+1,j}$ is at least as much as that in $F_{i,j}$. Graph $F_{i,j}$ has I_k more edges than the combined edges of $F_{i,x}$ and $F_{x+1,j}$. The weight of each such edge is at most $c(S[i,j])$. Therefore,

$$\begin{aligned}
c(F_{i,j}) &\leq c(F_{i,x}) + c(F_{x+1,j}) + I_k c(S_{i,j}) \\
&\leq 2I_k c(S_{i,x}) \lg \lceil m/2 \rceil + 2I_k c(S_{x+1,j}) \lg \lfloor m/2 \rfloor + I_k c(S_{i,j}) \\
&\leq 2I_k \{c(S_{i,x}) + c(S_{x+1,j})\} \lg \lceil m/2 \rceil + I_k c(S_{i,j}) \\
&\leq 2I_k c(S_{i,j}) \lg \lceil m/2 \rceil + I_k c(S_{i,j}) \\
&\leq 2I_k c(S_{i,j}) \lg |V_{i,j}|,
\end{aligned}$$

where the last inequality holds for $|V[i,j]| \geq 3$. Therefore, by construction of S,

$$c(G_k) = c(F_{1,n}) \leq 2I_k c(S) \lg n \leq 4I_k c(MST) \lg n. \tag{3.2}$$

The weight of the optimal k-connected graph $c(MKG) \geq \frac{k}{2}c(MST)$. Thus, we have

$$c(G_k) \leq 4(k+1)(\lg n)c(MKG).$$

□

Remarks. 1. Using $k = 1$ in Inequality 3.2, we get $c(NNT) = c(G_1) \leq 4(\lg n)c(MST)$. However, for this special case, $k = 1$, with the help of a lemma by Rosenkrantz, Stearns, and Lewis [24, Lemma 1] concerning the traveling salesman problem, we can achieve a better bound of $\lceil \log n \rceil c(MST)$ (see Theorem 2.2.1), improved by a factor of 4.

2. The above bound is asymptotically tight in general. Consider a geometric instance where n nodes are placed on a straight line equally apart by a unit distance and the weight of the edge between any two nodes is their distance on the line. There is a ranking of the nodes, for which, the weight of the NNT (i.e., $k = 1$) is $\Theta(n \log n)$. In fact, a random ranking of nodes (i.e., the Random-NN scheme) can be shown to give a spanning tree of the expected weight $\Theta(n \log n)$. The weight of MST on this geometric instance is $\Theta(n)$, which gives an approximation factor of $\Theta(\log n)$.

Notice that the above theorem also applies to an important special case, namely that of a geometric graph: the nodes are coordinates in a d-dimensional space and the weight of the edge between any two nodes is the Euclidean distance (or any Minkowski distance) between them. In the next section, using the Euclidean distance, we show that the algorithm yields a better approximation of $O(k)$ when nodes are randomly distributed in a 2-dimensional space.

3.4.2 Random Graph with Uniform Distribution of Nodes on a Plane

In this section, we analyze the weight of the k-connected graph given by the Random NN-scheme in a complete geometric graph where n nodes are randomly and uniformly distributed in a unit square $[0, 1]^2$ and the weight of the edge between any two nodes is the Euclidean distance between them. In this model, the probability that a particular node lies within a particular region inside the unit square is directly proportional to the area of the region. We show the following theorem:

Theorem 3.4.2 *For n points distributed randomly and uniformly in $[0,1]^2$, the approximation guarantee of the Random-NN scheme is $E[c(G_k)]/E[c(MKG)] = O(k)$.*

To show the above theorem we first upper bound the weight of the k-connected subgraph constructed by the Random-NN scheme.

Lemma 3.4.2 *For n points distributed randomly and uniformly in $[0,1]^2$, the expected weight of G_k, the subgraph constructed by the Random NN-scheme, is $O(k^2\sqrt{n})$, i.e., $E[c(G_k)] = O(k^2\sqrt{n})$.*

Proof: Consider an arbitrary node u, and the concentric circles centered at u with radii $r_i = \frac{2^i}{\sqrt{n}}$ for $i = 1, 2, \ldots, m$. Considering a unit square, the maximum distance between any two nodes is $\sqrt{2}$. Thus, $r_{m-1} < \sqrt{2} \leq r_m$, i.e., the maximum number of these circles is $m < \frac{1}{2}\lg n + \frac{3}{2}$. Let C_i be the set of the nodes in the circle with the radius r_i and $R_i = C_i - C_{i-1}$ for $i \geq 2$ and $R_i = C_i$ for $i = 1$. For a node $v \in R_i$, The weight of the edge (u,v) is $c(u,v) \leq r_i$.

Let A_i be the event that u makes the jth connection to a node $v \in R_i$. By Lemma 3.3.1, the probability that u makes the jth connection to any node in $\Gamma_u(y-1) - \Gamma_u(x-1)$ is $\sum_{i=x}^{y-1} \frac{j}{i(i+1)} = \frac{j}{x} - \frac{j}{y}$, where $j \leq x < y$. For $i \geq 2$, $|C_{i-1}| \geq 1$ since C_{i-1} contains at least one node, which is u. Considering the fact that u can be close to the border of the unit square, the probability that a particular node, other than u, is in C_{i-1} is $p \geq \frac{1}{4}$ of the area of $C_{i-1} = \frac{1}{4}\pi r_{i-1}^2 = \frac{2^{2i}\pi}{16n}$. Thus for $i \geq 2$,

$$\begin{aligned}
\Pr\{A_i\} &= \sum_{x=j}^{n}\sum_{y=x}^{n}\left(\frac{j}{x}-\frac{j}{y}\right)\Pr\{|C_{i-1}|=x \wedge |C_i|=y\} \\
&\leq \sum_{x=1}^{n}\sum_{y=x}^{n}\frac{j}{x}\Pr\{|C_{i-1}|=x \wedge |C_i|=y\} \\
&= \sum_{x=1}^{n}\frac{j}{x}\Pr\{|C_{i-1}|=x\} \\
&= \sum_{x=1}^{n}\frac{j}{x}\binom{n-1}{x-1}p^{x-1}(1-p)^{n-x} \\
&= \frac{j}{np}\left\{1-(1-p)^n\right\} \leq \frac{j}{np} \leq \frac{16j}{2^{2i}\pi}.
\end{aligned}$$

Let $c_j(u)$ be the weight of the edge given by the jth connection of u. We get

$$\begin{aligned}
E[c_j(u)] &\leq \Pr\{A_1\}r_1 + \sum_{i=2}^{m}\Pr\{A_i\}r_i \\
&\leq r_1 + \sum_{i=2}^{m}\frac{16j}{2^{2i}\pi}r_i \\
&= \frac{1}{\sqrt{n}}\left(2+\frac{8j}{\pi}-\frac{4\sqrt{2}j}{\pi\sqrt{n}}\right)
\end{aligned}$$

By linearity of expectation for all connections of n nodes,

$$E[c(G_k)] = n \times \sum_{j=1}^{k}E[c_j(u)] \leq \sqrt{n}\left\{2+\frac{8I_k}{\pi}\right\} - \frac{4\sqrt{2}I_k}{\pi} = O(k^2\sqrt{n}).$$

□

Proof: (of Theorem 3.4.2) It is well-known that the weight of an MST in the above graph model is $\Theta(\sqrt{n})$ (e.g., [23]). The weight of the optimal k-connected graph $c(MKG) \geq \frac{k}{2}c(MST) = \Theta(k\sqrt{n})$. Thus from Lemma 3.4.2, we have an approximation ratio of $O(k)$. □

3.4.3 Graph with Random Edge Weights

In this section, we analyze the weight of the k-connected subgraphs in another well-studied random graph model (e.g., see [42–44]) where the weights of the edges are selected randomly from $[0, 1]$ according to a uniform distribution, i.e., $U(0, 1)$. The following theorem shows the approximation guarantee of Random-NN scheme.

Theorem 3.4.3 *The approximation guarantee of the Random NN-scheme on a complete graph $\mathbb{K}_n$, where the weights of the edges are chosen randomly following the distribution $U(0, 1)$ is $2H_n - 2H_{k+1} + 1 = O(\log \frac{n}{k})$.*

We note that this model does not necessarily generate a metric graph, but our algorithm still gives a significantly better approximation of $O(\log \frac{n}{k})$. Frieze [42] showed that in this model, the expected weight of the MST converges to a constant $\zeta(3) = 1.202\cdots$ as $n \to \infty$. Here we show a lower bound of $\frac{1}{2}I_k$ for the expected weight of the MKG (Lemma 3.4.4) and show that the expected weight of G_k is $I_k(H_n - H_{k+1} + \frac{1}{2})$ (Lemma 3.4.5). Thus, we have an approximation ratio of $2H_n - 2H_{k+1} + 1 = O(\log \frac{n}{k})$. We now proceed to show the following lemmas, which prove the above theorem.

The proof of Lemma 3.4.3 can be found in [45, Page 195].

Lemma 3.4.3 *[45] Let X_i be the ith smallest number among n independent uniform random variables over $[0, 1]$. Then $E[X_i] = \frac{i}{n+1}$.*

Lemma 3.4.4 *Let MKG be a minimum weight k-connected subgraph on a complete graph $\mathbb{K}_n$, where the weights of the edges are randomly chosen according to the uniform distribution $U(0, 1)$. Then $E[c(MKG)] \geq \frac{1}{2}I_k$.*

Proof: Consider an arbitrary node u. Let the weights of the $n - 1$ edges adjacent to u in $\mathbb{K}_n$ be $e_1, e_2, \ldots, e_{n-1}$ in non-decreasing order. These edge weights are chosen randomly and independently from $U(0, 1)$. Thus, by Lemma 3.4.3, $E[e_i] = \frac{i}{n}$. Since the MKG is k-connected, the degree of each node in the MKG is at least k. Thus the sum of the weights of the edges adjacent to u in MKG is at least $\sum_{i=1}^{k} e_i$ and the expected sum of the weights is at least

$$E\left[\sum_{i=1}^{k} e_i\right] = \sum_{i=1}^{k} E[e_i] = \frac{1}{n} I_k$$

Using the fact that each edge is counted by at most two nodes and by linearity of expectation for n nodes,

$$E[c(MKG)] \geq \frac{1}{2} \times n \times \frac{1}{n} I_k = \frac{1}{2} I_k$$

$\square$

Lemma 3.4.5 *Let G_k be the k-connected subgraph given by the Random-NN scheme on a complete graph $\mathbb{K}_n$, where the weights of the edges are chosen randomly according to the distribution $U(0,1)$. Then $E[c(G_k)] = I_k(H_n - H_{k+1} + \frac{1}{2})$.*

Proof: Again, consider an arbitrary node u. Let the weight of the $(n-1)$ edges adjacent to u in $\mathbb{K}_n$ be $e_1, e_2, \ldots, e_{n-1}$ in non-decreasing order. Then $E[c(u, \mathbb{N}_u(i))] = E[e_i] = \frac{i}{n}$ (Lemma 3.4.3).

The event that u makes the jth connection to $\mathbb{N}_u(i)$, $j \leq i$, is independent of the weights of the edges adjacent to u. By using Lemma 3.3.1, the expected weight of the jth connection by u is

$$\sum_{i=j}^{n-1} \frac{j}{i(i+1)} E[e_i] = \frac{j}{n}(H_n - H_j)$$

Using linearity of expectation, the expected total weight of all connections by the n nodes is

$$E[c(G_k)] = n \sum_{j=1}^{k} \frac{j}{n}(H_n - H_j) = I_k H_n - \sum_{j=1}^{k} j H_j$$

Using the identity $\sum_{j=1}^{k} j H_j = I_k(H_{k+1} - 1/2)$ (see [46], Page 56, Eq. 2.57),

$$E[c(G_k)] = I_k(H_n - H_{k+1} + 1/2)$$

$\square$

3.4.4 Maximum Degree in the Geometric Instances

We assume that the nodes are points in a d-dimensional space and the weight of an edge between any two nodes is the Euclidean distance between them. We show the following theorem:

Theorem 3.4.4 *In a geometric graph, the maximum degree of a node in the k-connected spanning subgraph constructed by the Random-NN scheme is $O(k \log n)$ with high probability, i.e., with probability at least $1 - 1/n^{\Omega(1)}$.*

We show the result assuming $d = 2$, i.e., the nodes (points) are on a plane; however, this result can be generalized to any constant d. Note that for analyzing the maximum degree of a node, we do not assume any particular distribution of the nodes; we consider an arbitrary placement of the nodes in a plane. To show the desired bound on the maximum degree, we first need the following lemma.

Lemma 3.4.6 *Let V be the set of the nodes in the plane. If a node $v \in V$ makes its longest connection, i.e., the $|\eta(v)|^{th}$ connection, to node w, we say that a charge of 1 is placed on every node u in the closed ball $\mathcal{B}(v, c(v,w))$, where $c(u,w)$ is the weight of the edge (u,w), i.e., the distance between u and w. Then, the total charge on any node u is $O(k \log n)$, with high probability.*

Proof: Consider any node u, and partition the 2π angle around u into 6 cones with each of the angles be $\pi/3$. Consider one such cone. We prove that the total charge on u from the nodes in this cone is $O(k \log n)$, with high probability. Order the points in the cone as $v_1, v_2, v_3, \ldots$ in non-decreasing order of their distances from u (see Fig. 3.2). Node v_i places a charge on u *only if* the rank of v_i is in the top $|\eta(v_i)|$ among the ranks of the nodes $v_1, v_2, \ldots v_i$. Thus, the probability that v_i places a charge on u is at most $|\eta(v_i)|/i \leq k/i$. Therefore, the total expected charge on u from these nodes is at most $\sum_{i=1}^{n-1}(k/i) \leq k \log n$.

In order to bound the maximum charge on any node, we use a variant of the Chernoff bound [Lemma 3.4.7] that holds in the presence of dependencies among the variables.

Lemma 3.4.7 *[47] Let $X_1, X_2, \ldots, X_l \in \{0,1\}$ be random variables such that for all i, and for any $S \subseteq \{X_1, \ldots, X_i\}$, $\Pr[X_{i+1} = 1 | \bigwedge_{j \in S} X_j = 1] \leq \Pr[X_{i+1} = 1]$. Then for any $\delta > 0$, $\Pr[\sum_i X_i \geq \mu(1+\delta)] \leq (\frac{e^\delta}{(1+\delta)^{1+\delta}})^\mu$, where $\mu = \sum_i E[X_i]$.*

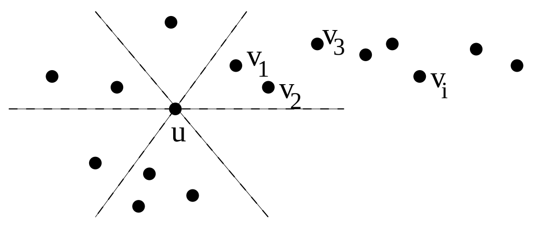

Figure 3.2. Each wedge around the node u is 60°. $v_1, v_2, v_3 \ldots$ are the nodes in one wedge in non-decreasing order of their distances from u.

Let $\mathcal{E}(v)$ be the event that v places a charge on u. In order to use the Chernoff bound, we need to show that, for any i, and any subset $S \subset \{v_1, \ldots, v_i\}$, $\Pr[\mathcal{E}(v_{i+1}) | \bigwedge_{w \in S} \mathcal{E}(w)] \leq \Pr[\mathcal{E}(v_{i+1})]$.

First, suppose $c(w, v_{i+1}) \geq c(w, u)$ for each $w \in S$. Then, the events $\bigwedge_{w \in S} \mathcal{E}(w)$ do not place any constraint on $rank(v_{i+1})$, relative to $rank(v_j), j \leq i$, and therefore, $\Pr[\mathcal{E}(v_{i+1}) | \bigwedge_{w \in S} \mathcal{E}(w)] = \Pr[\mathcal{E}(v_{i+1})]$.

Next, suppose $c(w, v_{i+1}) < c(w, u)$ for some $w \in S$. If the event $\mathcal{E}(w)$ occurs, then $rank(w)$ is in the top $|\eta(w)|$ ranks among the ranks of the nodes $v_1, v_2, \ldots v_{i+1}$, and the probability of $rank(v_{i+1})$ being in the top $|\eta(v_{i+1})|$ ranks goes down; that is, $\Pr[\mathcal{E}(v_{i+1}) | \bigwedge_{w \in S} \mathcal{E}(w)] \leq \Pr[\mathcal{E}(v_{i+1})]$.

Next, we apply the Chernoff bound with $\delta = \frac{5k \log n}{\mu} - 1$, where μ is the expected charge on u. Since $\mu \leq k \log n$, $\delta > 0$. Let X be the total charge on u. Then,

$$\Pr\{X \geq 5k \log n\} = \Pr\{X \geq (1+\delta)\mu\} < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu \leq \left(\frac{e}{1+\delta}\right)^{(1+\delta)\mu} \leq \frac{1}{n^{3k}}.$$

Thus, with probability at least $1 - 1/n^{3k}$, where $k \geq 1$, the total charge on u is $O(k \log n)$. Using the union bound, this holds simultaneously for all nodes with probability at least $1 - 1/n^{2k}$. □

Proof: (of Theorem 3.4.4) If a node u connects to v, u must place a charge on v (see Lemma 3.4.6). Thus, the total charge on v is an upper bound on the number of nodes that are connected to v. Further, $\eta(v) \leq k$. Thus, the degree of v is at most $k + O(k \log n) = O(k \log n)$ with probability at least $1 - 1/n^{2k}$. □

3.5 Distributed Implementation

In this section, we give an efficient distributed implementation of the Random-NN scheme. Our distributed algorithm takes $O(\log \frac{n}{k})$ time and expected $O(nk \log \frac{n}{k})$ messages to construct a k-connected graph.

Model of distributed computation. We consider the well-studied point-to-point communication model, where we are given a complete network of n nodes (processors) with distinct identifiers (we assume $O(\log n)$-size *ids*) and each node knows the (nonnegative) weights associated with its incident edges (bidirectional communication links) but not the identifiers of its neighbors (see e.g., [13, 21]). The communication between any two nodes happens by sending/receiving messages along the edge between them and all nodes perform the same algorithm. We assume that $O(\log n)$ bits can be transferred in one step per edge and a node can send messages through all its incident links at the same time (see e.g., [13]).

The following distributed algorithm, in Figure 3.3, is a realization of the Random NN-scheme in a distributed complete network. Here, each node chooses its rank by choosing a number uniformly and independently at random from $[0, 1]$.[1] Then each node, in rounds, keeps sending FIND messages to its neighbors beginning with the nearest neighbor, in non-decreasing order of the edge weights, until it receives k ACCEPT messages. The FIND messages contain the sender's random number (chosen from $[0, 1]$) and *id*. The receiver of a FIND message compares its rank with the rank of the sender. If the receiver's rank is higher than the sender's rank, the receiver sends an ACCEPT message back to the sender of the FIND message. Note that we do not make any assumption about the weights of the edges in designing the distributed algorithm and in analyzing its time and message complexity. However, as we have seen in the previous section, the quality (the weight) of the k-connected subgraph constructed by this algorithm, with respect to the quality of the optimal k-connected subgraph, depends on the properties satisfied by the weights of the edges.

Message and Time Complexity. It is interesting to analyze the message complexity and the time complexity, and their tradeoffs in the distributed model we consider (i.e., point to point communications with all processors forming a clique). A naive method for finding the k nearest higher ranked nodes is: each node probes one neighbor at a time, to find the ranks of its neighbors, in nondecreasing order of edge

[1]The ranks can be also chosen uniformly from, say, $[1, n^4]$ and the ranks will be unique with high probability. Or, as is done in the algorithm, we assume that each node has a unique label which is used to break the ties. This does not alter any of our proofs or the results.

Distributed k-connected graph algorithm

Input: A complete graph $\mathbb{K}_n = G(V, E)$. We assume each node has a unique id from a totally ordered set.
Output: A k-connected subgraph G_k. On termination, each node knows which of its adjacent edges are in G_k.

Each node $u \in V$ executes the following protocol independently and simultaneously:

1. Choose the rank $r(u)$ as follows: generate a random number $p(u) \in [0, 1]$. We say $r(v) > r(u)$ if and only if $[p(v) > p(u)]$ or $[p(v) = p(u)$ and $id(v) > id(u)]$.
2. Find $|\eta(u)|$ nearest nodes w with $r(w) > r(u)$, and add the edges (u, w) to G_k. Find the w's as follows:

```
$t \leftarrow 1$                                  ▶ $t$ is the round number
REPEAT                                            ▶ A FIND message includes $p(u)$ and $id(u)$
    If $t = 1$, $u$ sends FIND messages to all $v \in \Gamma_u(k)$ simultaneously;
    If $t \geq 2$, $u$ sends FIND messages to all $v \in [\Gamma_u(2^{t-1}k) - \Gamma_u(2^{t-2}k)]$ simultaneously;
    $t \leftarrow t + 1$
UNTIL $u$ received $k$ ACCEPT messages or probed all of its neighbors.
```

3. Upon receipt of a FIND message from any v, send back an ACCEPT message to v iff $r(u) > r(v)$.

Figure 3.3. Distributed implementation of the Random-NN scheme.

weights. By Lemma 3.3.2, the expected number of the messages each node needs to exchange is $O(k \log \frac{n}{k})$ to find the k higher ranked nodes (Note that a node made its k^{th} connection means that it already made all the required connections). This gives an expected total of $O(kn \log \frac{n}{k})$ messages. However, the time complexity of this implementation is $\Theta(n)$ since there will be a node (the highest ranked node) which has to probe all its $(n-1)$ neighbors. On the other hand, if we want to get a better time complexity at the expense of more messages, consider a different protocol: each node sends its rank (the random number and the *id*) to all its neighbors in one step (one round); this finishes in $O(1)$ time, but consumes $\Theta(n^2)$ messages.

To reduce both the time complexity and the message complexity, we consider the *hybrid* protocol given in Figure 3.3, where in the first round, a node probes the first k nearest neighbors and in the subsequent rounds $t \geq 2$, it probes the next $2^{t-2}k$ nearest neighbors until it succeeds in finding the k nearest higher ranked neighbors. Below we present the analysis of the time and message complexity of this protocol.

Theorem 3.5.1 *The protocol of Figure 3.3 takes $O(\lg \frac{n}{k})$ time and uses expected $O(kn \lg \frac{n}{k})$ messages.*

Proof: A node u needs $1 + \lceil \lg \frac{n-1}{k} \rceil$ rounds to probe all of its $n-1$ neighbors. Therefore, the protocol takes at most $1 + \lceil \lg \frac{n-1}{k} \rceil \leq 2 + \lg \frac{n}{k}$ time. To bound the message complexity, we calculate the expected number of the messages a node sends before it finds the k neighbors of higher ranks.

In the tth round for $t \geq 2$, a node u sends FIND messages to all nodes in $\Gamma_u(2^{t-1}k) - \Gamma_u(2^{t-2}k)$. Using Lemma 3.3.1, the probability that u makes the kth connection in the round t is

$$\begin{aligned}
& \sum_{i=2^{t-2}k+1}^{2^{t-1}k} \frac{k}{i(i+1)} \\
= \quad & k \left\{ \frac{1}{2^{t-2}k+1} - \frac{1}{2^{t-1}k+1} \right\} \\
= \quad & k \left\{ \frac{2}{2^{t-1}k+2} - \frac{1}{2^{t-1}k+1} \right\} \\
\leq \quad & \frac{k}{2^{t-1}k+1} \leq \frac{1}{2^{t-1}}.
\end{aligned}$$

Notice that the above upper bound for the probability can also be used for $t = 1$ as $1/2^{t-1}$ evaluates to 1 when $t = 1$. The number of SEND messages u sends in the first t rounds is $2^{t-1}k$. Thus, the expected number of SEND messages by u is at most

$$\sum_{t=1}^{1+\lceil \lg \frac{n-1}{k} \rceil} (2^{t-1}k)\frac{1}{2^{t-1}} \leq 2k + k \lg \frac{n}{k}.$$

Moreover, u receives at most k ACCEPT messages. Thus, using linearity of expectation for n nodes, the expected total number of the messages is $3kn + kn \lg \frac{n}{k}$. $\square$

Remarks. **1.** In the distributed model we consider (i.e., point to point communication with all processors forming a clique), a modification of the proof given by Korach, Moran, and Zaks in [21] (which was given for MST) shows a lower bound of $\Omega(n^2)$ on the number of the messages needed to construct an optimal k-connected spanning subgraph (for any $1 \leq k \leq \lfloor n/2 \rfloor - 1$) in a complete weighted metric graph; this lower bound is independent of the length of the messages. Thus, in general, the expected message complexity of our randomized algorithm is significantly better than the deterministic lower bound. Also, the message complexity of our algorithm is optimal in the sense that $\Omega(n \log n)$ is a lower bound on the number of the messages needed to construct any spanning tree [28]. A lot of work had been devoted to finding spanning tree (equivalent to leader election) algorithms having $O(n \log n)$ message complexity in this model (see e.g., [28, 48, 49]) and our protocol also gives a very simple spanning tree and leader-election protocol that has $O(n \log n)$ (expected) message complexity.
2. It is also quite easy to adapt the above algorithm for a "broadcast" setting which is a typical model for wireless networks (see e.g., [31]). In such a setting, nodes are assumed to be in a geometric space (e.g., a plane) and a node communicates with its neighbors by broadcasting a message. All nodes within the broadcast range can receive the message (ignoring collisions). To implement our algorithm, a node has to progressively increase its broadcast range (in a similar doubling fashion) till it finds the nearest nodes of higher ranks. We analyze such a strategy in detail in

Chapter 5 which also contains experimental results in the context of the wireless sensor networks [3].

3.6 Conclusion and Further Work

We showed and analyzed a simple randomized approximation scheme for constructing a low-weight k-connected spanning subgraph. We also presented its efficient implementation in a complete network of processors. The proposed algorithm has low time and message complexity while giving a relatively good approximation ratio for the metric graphs, random geometric graphs, and random edge-weight graphs. It is interesting to see whether the ideas in this chapter can be used to design an efficient distributed algorithm for the more challenging problem of finding a k-connected subgraph in an arbitrary general graph (need not be complete). The local nature of the NN-scheme seems suitable for designing a simple and efficient *dynamic* algorithm (especially in a distributed setting), where the goal is to maintain a k-connected graph of good quality, as nodes are added or deleted. This looks promising for future work.

4 A FAST DISTRIBUTED APPROXIMATION ALGORITHM FOR MINIMUM SPANNING TREES IN AN ARBITRARY GRAPH

In this chapter, we present a distributed algorithm that constructs a spanning tree with $O(\log n)$ approximation to minimum spanning tree (MST) in any arbitrary network. This 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 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.

4.1 Introduction

4.1.1 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 [11] that constructs the MST in $O(n \log n)$ time and $O(|E| + n \log n)$ messages. The communication (message) complexity of the Gallager et al. algorithm 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 [50], further improved to $O(n \log^* n)$ by Gafni [51], and then improved to *existentially optimal* running time of $O(n)$ by Awerbuch [52]. The $O(n)$ bound is existentially optimal in the sense that there exists graphs where no distributed MST algorithm can do better than $\Omega(n)$ time. This was the state of the art till the mid-nineties when Garay, Kutten, and Peleg [53] 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 n. Garay, Kutten, and Peleg [53] gave the first such distributed algorithm for the MST problem with running time $O(D(G) + n^{0.61})$, which was later improved by Kutten and Peleg [54] to $O(D(G) + \sqrt{n} \log^* n)$. Elkin [55] refined this result further and argued that a parameter called "MST-radius" captures the complexity of distributed MST algorithms 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 [55] (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 [54] by a factor of $\Omega(\sqrt{n})$. However, a drawback of this protocol (unlike the previous MST protocols [11, 50, 51, 53, 54]) is that it cannot detect the termination of the algorithm in this much time (unless $\mu(G, w)$ is given as part of the input). Finally, we note that the time-efficient algorithms of [53–55] are not message-optimal (i.e., they take asymptotically much more than $O(|E| + n \log n)$ messages, e.g., the protocol of [54] takes $O(|E| + n^{1.5})$ messages).

The lack of progress in improving the result of [54], and in particular breaking the $\sqrt{n}$ barrier, led to work on lower bounds for the distributed MST problem. Peleg and Rabinovich [56] showed that $\tilde{\Omega}(\sqrt{n})$ time is required for constructing an MST even on graphs of small diameter and showed that this result establishes the asymptotic near-tight (existential) optimality of the protocol of [54].

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 the 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 [56] 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 [12]. 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 the MST problem. To quote Elkin's survey paper [57]: "There is no satisfactory approximation algorithm known for the MST problem". To the best of our knowledge, the only known distributed approximation algorithm for the MST problem is given by Elkin in [12]. This algorithm gives an H-approximation to the MST with running time $O(D(G) + \frac{\omega_{max}}{H-1} \cdot \log^* n)$, where ω_{max} is the ratio between the maximum and minimum weights of the edges in the input graph G. Thus, this algorithm is not independent of the edge weights and its running time can be quite large.

4.1.2 Distributed Computing Model

We present a fast distributed approximation algorithm for the MST problem. First, we briefly describe the distributed computing model that is used by our algorithm (as well as the previous MST algorithms [11, 50–55] mentioned above) which is now standard in the distributed computing literature (see e.g., the book by Peleg [13]).

We are given a network modeled as an undirected weighted graph $G = (V, E, w)$ where V is the set of the nodes (vertices) and E is the set of the communication links between them and $w(e)$ is the weight of the edge $e \in E$. Without loss of generality, we assume that G is connected. Each node hosts a processor with limited initial knowledge. Specifically, we make the common assumption that each node has unique identity numbers (this is not really essential, but simplifies presentation) and at the beginning of computation, each vertex v accepts as input its own identity number and the weights of the edges 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 complexity [13].) In each time step, each node v can send an arbitrary message of size $O(\log n)$ through each edge $e = (v, u)$ that is adjacent to v, and the message arrives at u by the end of this time step. (If unbounded-size messages are allowed, the MST problem can be trivially solved in $O(D(G))$ time [13].) 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 the distributed computation is called the $\mathcal{CONGEST}(\log n)$ model or simply the $\mathcal{CONGEST}$ model [13] (the previous results on the distributed MST problem cited in Section 4.1.1 are for this model). We note that more generally, $\mathcal{CONGEST}(B)$ model 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 chapter.

4.1.3 Overview of the Results

Our main contribution is an almost existentially optimal (in both time and message complexity) distributed approximation algorithm that constructs an $O(\log n)$-approximate minimum spanning tree, i.e., whose cost is within an $O(\log n)$ 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 the *local shortest path diameter* (we defer the definition of $L(G,w)$ to Sect. 4.2.2). $L(G,w)$ depends on the graph topology. $L(G,w)$ always lies between 1 and n. The parameter $L(G,w)$ can be smaller or larger than the diameter and typically it can be much smaller than $\sqrt{n}$ (recall that this is essentially a lower bound on distributed (exact) MST computation). In fact, we show that there exist some graphs for which any distributed algorithm for computing an MST will take $\tilde{\Omega}(\sqrt{n})$ time, while our algorithm will compute a near-optimal MST 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 be asymptotically larger than both the diameter and $\sqrt{n}$. By combining the MST algorithm of Kutten and Peleg [54] 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 the MST within an H-factor, $1 \leq H \leq O(\log n)$ (cf. Theorem 4.3.1). This implies that our algorithm is existentially optimal (upto a polylogarithmic factor) and in general, no other algorithm can do better. We note that the existential optimality of our algorithm is with respect to $L(G,w)$ instead of n as in the case of Awerbuch's algorithm [52]. Our algorithm is also existentially optimal (upto a polylogarithmic factor) with respect to the communication (message) complexity — takes $\tilde{O}(|E|)$ messages, since $\Omega(|E|)$ messages is clearly needed in some graphs to construct any spanning tree [28, 58].

One of the motivations for this work is to investigate whether a fast distributed algorithm that construct a (near-optimal) MST can be developed for some special

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

classes of networks. An important consequence of our results is that the networks with low $L(G, w)$ value (compared to $O(D(G))$ admit a $\tilde{O}(D(G))$ time $O(\log n)$-approximation distributed algorithm. In particular, the unit disk graphs have $L(G, w)$ $= 1$. The unit disk graph model is a commonly used model in the wireless networks. We also show that $L(G, w) = O(\log n)$ with high probability in any arbitrary network whose edge weights are chosen independently at random from any arbitrary distribution (cf. Theorem 4.4.2).

4.2 Distributed Approximate MST Algorithm

4.2.1 Nearest Neighbor Tree Scheme

The main objective of our approach is to construct a spanning tree, called the *Nearest Neighbor Tree (NNT)*, efficiently in a distributed fashion. In Chapter 2, we introduced the Nearest Neighbor Tree and showed that its cost is within an $O(\log n)$ factor of the cost of the MST. The scheme is used to construct an NNT (henceforth called *NNT scheme*) as follows: (1) each node first chooses a unique *rank* from a totally-ordered set; a ranking of the nodes corresponds to a permutation of the nodes; (2) each node (except the one with the highest rank) connects (via the *shortest path*) to the *nearest* node of higher rank. We show 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) 2. Note that some cycles can be introduced in step 2, and hence to get a spanning tree we need to remove some edges to break the cycles.

The main advantage of the NNT scheme is that each node, individually, has the task of finding its nearest node of higher rank to connect to, and hence no explicit coordination is needed among the 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 Chapter 3, we showed how the NNT scheme can be implemented in a *complete metric* graph G (i.e., $D(G) = 1$). This algorithm takes only $O(n \log n)$

messages to construct an $O(\log n)$-approximate MST as opposed to the $\Omega(n^2)$ lower bound (shown by Korach et al [21]) on the number messages needed by any distributed MST algorithm in this model. If the time complexity needs to be optimized, then NNT scheme can easily be implemented in $O(1)$ time (using $O(n^2)$ messages), as opposed to the best known time bound of $O(\log\log n)$ for the (exact) MST [59]. These results suggest that the NNT scheme can yield faster and communication-efficient algorithms compared to the algorithm that compute the exact MST. However, an efficient implementation in a general weighted graph is non-trivial and was left open in [2]. Thus, a main contribution of this chapter is an efficient implementation of the scheme in a general network. The main difficulties are avoiding the congestions in finding the nearest node of higher rank efficiently in a distributed fashion (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. 4.2.3.

4.2.2 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. We call $|Q|$ the *length of the path* Q.

$w(Q(u, v))$ 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 $d(u, v) = w(P(u, v))$.

$N_\rho(v)$ — is the set of all *neighbors* of v within the distance ρ, 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, e.g., $W(v) = \max_{(v,x)\in E} w(v, x)$.
$l(u, v)$ — is the number of edges in the minimum-length 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,

$$l(u, v) = \min\{|P(u, v)| \mid P(u, v) \text{ is a shortest path from } u \text{ to } v\}.$$

Definition 4.2.1 ***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 4.2.2 ***(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 4.2.3 ***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 4.2.4 ***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 $1 \leq L \leq S \leq n$ 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 $L = 1$.

4.2.3 Distributed NNT Algorithm

We recall the basic NNT scheme 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. Now we describe each of these steps in detail.

Rank selection. The nodes select unique ranks as follows. First, a leader is elected by using 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 identity number ID(s) to all of 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 transmit $p(u)$ and ID(u) to the 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.

Definition 4.2.5 Rank. *The rank of a node u is defined as $r(u) = (p(u), ID(u))$ and for any two nodes u and v,*

$$r(u) < r(v) \text{ iff } p(u) < p(v) \text{ or } [p(u) = p(v) \text{ and } ID(u) < ID(v)].$$

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

Observation 4.2.1 *Each node knows the ranks of all of its neighbors.*

Proof: Once a node receives the rank from one of its neighbors, it selects it own rank and sends it to all of its neighbors. Since the underlying graph G is connected, eventually (within time D, where D is the diameter of the graph), each node receives the messages containing the ranks from all of its neighbors. □

Observation 4.2.2 *Each node u, except the leader s, has at least one neighbor v, i.e., $(u, v) \in E$, such that $r(u) < r(v)$.*

Proof: Each node $u \neq s$, selects its own rank $r(u)$ such that $r(u) < r(v)$ only after receiving $r(v)$ from some neighbor v. □

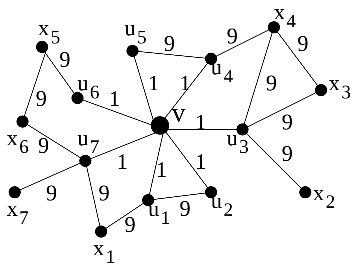

Figure 4.1. A network with possible congestion in the edges adjacent to v. The weight of the edge (v, u_i) is 1 for every i, and 9 for the rest of the edges. Assume $r(v) < r(u_i)$ for all i.

Observation 4.2.3 *The leader s has the highest rank among all nodes in the graph.*

Proof: Since the leader s is the initiator of this rank selection process, we have $r(s) > r(v)$ for any $v \in V$ where $v \neq s$. □

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. By Observation 4.2.2, we can conclude that for any node v, exploring the nodes in $\Gamma_{W(v)}(v)$ is sufficient 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 with ρ doubled. By Observation 4.2.2, 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 the subsequent rounds, the values for λ are doubled, i.e., in the j^{th} round, $\lambda = 2^{j-1}$. In a particular round, v explores all nodes in $\Gamma_{\rho,\lambda}(v)$. At the end of each round, v counts the number of the 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. 4.1. 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 the values for ρ and λ for all u_is may not necessarily be the same at a particular time. Thus, the 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 the congestion. As a result, a node may not connect to the nearest node of higher rank. However, our algorithm guarantees that the 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 $\rho \leftarrow 1$ and $\lambda \leftarrow 1$. Node 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 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 arrives 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 *explore* 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, $1 \leq i \leq k$, v forwards only one *explore* message and replies back to the other u_is as follows. 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 the weight of the connecting path $w(Q(u_i, u_j)) = pd_i + pd_j \leq 2\rho_i$. In this way, some of the u_is are replied back a *found* message and their *explore* messages will not be forwarded by v.

Now, there are at least one u_i left, to which v did not send the *found* message back. If there is exactly one such u_i, v forwards its *explore* message; otherwise, v takes the following actions. Let u_s be the node with lowest rank among the rest of the u_is (i.e., those u_is which were not sent a *found* message by v), and u_t, with $t \neq s$, be an arbitrary node among the rest of u_is. Now, it must be the case that ρ_s is strictly smaller than ρ_t (otherwise, v would send a *found* message to u_s), 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 a 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 the 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, from the nodes in $N_{\rho-pd}(y) - \{z\}$. Each of these replies is either a *count* message, a *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, y ignores all *wait* and *count* messages, and sends the found message to z toward the originator v. If there are more than one *found* messages, select the one with the minimum path weight and ignore the rest. Now, if there is no found message and at least one *wait* message, y sends back only one wait message to z toward the originator v and ignore the *count* messages. 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, these reply-back 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, a *wait* message, or a *found* message. If at least one of the replies is a *found* message, v is done with the exploration and makes the connection as described in Item 2 below. Otherwise, if there is a *wait* message, v again initiates exploration with the same ρ and λ. If all of them are *count* messages, v calculates the total count by adding the count values of these messages and does the following:

(a) if $\lambda = 1$, v initiates exploration with $\lambda \leftarrow 2$ and the same ρ (2nd round of the same phase);

(b) if $\lambda > 1$ and the total count for this round is larger than that of the previous round, v initiates exploration with $\lambda \leftarrow 2\lambda$ and the same ρ (next round of the same phase);

(c) otherwise, v initiates exploration with $\lambda \leftarrow 1$ and $\rho \leftarrow 2\rho$ (first round of the next phase).

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 the rank of itself, then v selects the nearest one among them (break the ties arbitrarily). 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. During the exploration process, the intermediate nodes in the path simply keep track of the predecessor and successor nodes in the path $Q(v, u)$ for this originator 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. Let $Q(v, u) =< v, \ldots, x, y, \ldots, u >$. Note that by our choice of u, all of the intermediate nodes in this path 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 the ranks of x and y. If x is still doing exploration to find a higher ranked node, x stops the exploration process 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. Once u receives the *connect* message originated by v, u sends a *rank-update* message back to v. All

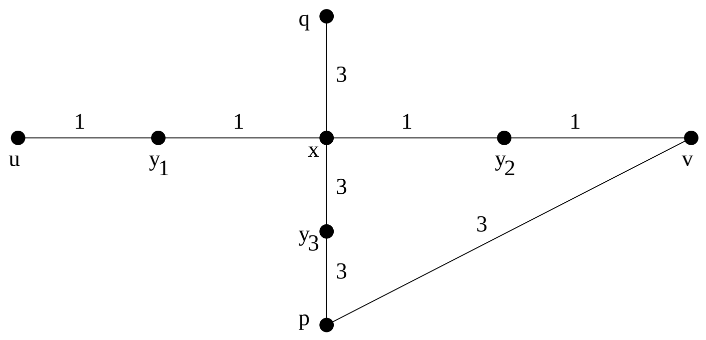

Figure 4.2. A possible scenario of creating cycle and avoiding it. Nodes are marked with letters. Edge weights are given in the figure. Let $r(u) = 11, r(v) = 12, r(p) = 13, r(q) = 14$, and ranks of the rest of the nodes are smaller than 11. u connects to v, v connects to p, and p connects to q.

nodes in the path $Q(v, u)$ including v upgrade their ranks to $r(u)$; i.e., they assumes a new rank which is equal to the rank of u.

It might happen that in between exploration and connection, some node x in the path $Q(v, u)$ changed its rank due to a connection by some originator other than v. In such a case, when the *connect* message originated by v travels through x, if x's current rank is larger than $r(v)$, x accepts the connection as the last node in the path and returns a *rank-update* message with $r(x)$ toward v instead of forwarding the *connect* message to the next node (i.e., y) toward u. This is necessary to avoid cycle creation.

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 a path $Q(u, v)$, 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 path goes though these intermediate nodes. For example, in Fig. 4.2, the paths $P(u, v)$ and $P(p, q)$ both go through a lower ranked node x.

In Fig. 4.2, 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.

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. In the next section, a formal and robust proof is given to show that there is no cycle in the resulting NNT (Lemma 4.2.4).

4.2.4 Analysis of the Algorithm

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

Lemma 4.2.1 *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 the (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 reaches u and u sends back a *found* message. In this case, $w(Q) \leq \rho$. ii) Some node y receives 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 (on behalf of u) 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, in both cases, we have $w(Q) \leq 4d(v, u')$. □

Lemma 4.2.2 *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 the NNT; and s cannot be associated to any node since a *connect* message cannot be originated by or go through s; s can only be the destination (the last node in the path) since s has the highest rank.

Now, to complete the proof, we need to show that no two nodes are associated to the same edge. To show this, we use the following lemma.

Lemma 4.2.3 *Whenever x is associated to the edge (x, y), at that point of time, $r(x) \leq r(y)$.*

Proof: Node x become associated to the edge (x, y) only after a *connect* message passes through (x, y) from x to 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 increases $r(x)$, then either $r(y)$ is also increased to the same value or x become associated to some edge other than (x, y). Thus, while keeping (x, y) associated to x, it must be true that $r(x) \leq r(y)$. [The end of the proof of Lemma 4.2.3] □

Only the nodes x and y can be associated to the edge (x, y). Let x be associated to the edge (x, y). By Lemma 4.2.3, $r(x) \leq r(y)$. 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., this connect message cannot terminate at x). This will make x associated to some edge other than (x, y). Therefore, no two nodes are associated to the same edge. □

Lemma 4.2.4 *The edges in the NNT added by the given distributed algorithm does not create any cycle.*

Proof: Suppose to the contrary that $< v_0, v_1, v_3 \ldots, v_k, v_0 >$ be a cycle created by the edges added to the NNT. Then either one of the following must be true.

- v_i is associated to (v_i, v_{i+1}) for $0 \leq i \leq k-1$, and v_k is associated to (v_k, v_0).
- v_i is associated to (v_i, v_{i-1}) for $1 \leq i \leq k$, and v_0 is associated to (v_0, v_k).

For both cases, by Lemma 4.2.3, we have $r(v_0) = r(v_1) = \ldots = r(v_k)$. Here, we have a contradiction. The ranks of all nodes in this cycle can never be the same. Initially, the ranks are distinct. Later, when a node u connects to the node v via the *connecting path* $Q(u, v)$, the ranks of the nodes in the path $Q(u, v)$ are upgraded to $r(v)$. Notice that the rank of a node cannot be decreases. It can only be increased. It is easy to see that a connecting path cannot contain any cycle. The above cycle must be created by at least two connecting paths. Let $Q(u, v)$ be the last connecting path that completes this cycle, and v_i and v_j be the first and last node in the path $Q(u, v)$ among the nodes that are common both in this path and the cycle. The path $Q(u, v)$ goes beyond v_i; that means $r(v) > r(v_i)$. Since $r(v_i) = r(v_j)$, $r(v) > r(v_j)$; this implies that $v \neq v_j$ and the path $Q(u, v)$ goes beyond v_j. As a result, this connecting path (u, v) will upgrade the ranks of v_i and v_j to $r(v)$, which is higher than the ranks of the other nodes in the cycle. This leads to a contradiction. Thus, there cannot be any cycle in the NNT. □

From Lemmas 4.2.2 and 4.2.4 we have the following theorem.

Theorem 4.2.1 *The above algorithm produces a tree spanning all nodes in the graph.*

We next show that the spanning tree found by our algorithm is an $O(\log n)$-approximation to the MST (Theorem 4.2.2).

Theorem 4.2.2 *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 the triangle inequality. Let NNT_H be a nearest neighbor tree and MST_H be a minimum spanning tree on H. We can show that $c(NNT_H) \leq \lceil \log n \rceil c(MST_H)$ (Theorem 2.2.1).

Let NNT' be a spanning tree on G, where each node connects to the nearest node of higher rank via a shortest path. By Lemma 4.2.1, we have $c(NNT) \leq 4c(NNT')$. Further, it is easy to show that $c(NNT') \leq c(NNT_H)$ and $c(MST_H) \leq c(MST)$. Thus, we have

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

□

Remark: With the help of Theorem 2.13 in [60], an alternative upper bound of $12\lceil \log \frac{w_{max}}{w_{min}} \rceil c(MST)$ for $c(NNT)$ can be achieved, where w_{max} and w_{min} are the maximum and minimum edge weights, respectively. This bound is independent of the number of nodes n, but depends on the weights of the edges.

Theorem 4.2.3 *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 4.2.2). 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 $4L$. 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 4.2.4 *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 $O(|E| \log L \log n)$. □

4.3 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 [56] (see Section 4.1 and 5.3 in [56] for a description of how to construct $\mathcal{J}_m^K$). (One can also show a similar result using the family of graphs defined by Elkin in [12].) 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 4.3.1 *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 [12] (which generalizes and improves the results of Lotker et al [61]).

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 [12]. Elkin constructed a family of weighted graphs $\mathcal{G}^\omega$ (Figure 1, Section 3.1 in [12]) 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 [12] for the detailed description of the graph family and the assignment of weights.) We now slightly restate the results of [12] (assuming the $\mathcal{CONGEST}(\mathcal{B})$ model):

Theorem 4.3.2 *[12] 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.

4.4 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; that is the weight of the edge (u, v). Theorem 4.4.1 shows that for any UDG, $L = 1$. For a 2-dimensional UDG, the diameter D can be as large as $\Theta(\sqrt{(}n))$.

Theorem 4.4.1 *In any UDG, the local shortest path diameter L is* 1*.*

Proof: For any node v, $W(v) \leq R$ by definition of UDG. Now if there is node u such that $d(u, v) \leq R$, then $dist(u, v) \leq R$ by the triangle inequality. Thus, (v, u) is in E and the edge (v, u) is the shortest path from v to u; as a result, $l(v, u) = 1$. Therefore, for any v, $L(v) = \max_{u \in \Gamma_{W(v)}(v)} l(v, u) = 1$, and $L = \max_{v \in V} L(v) = 1$. □

Graph with Random Edge Weights. Consider any graph G (topology can be arbitrary) with edge weights chosen randomly from $[0, 1]$ following any 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 4.4.2 *Consider a graph G where the edge weights are chosen randomly from $[0, 1]$ following any (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: Let the edge weights are randomly drawn from $[0, 1]$ with mean μ. 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 [62] 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. □

4.5 Conclusion and Future Work

We presented and analyzed a simple approximation algorithm for constructing a low-weight spanning tree. We also presented its efficient implementation in an arbitrary network of processors.

The local nature of the NNT-scheme seems to be suitable for designing an efficient distributed *dynamic* algorithm, where the goal is to maintain an NNT of good quality, as nodes are added or deleted. Moreover, it is interesting to see whether the ideas in this chapter can be extended to design an efficient distributed algorithm for the more challenging problem of finding a k-connected subgraph. These look promising for future work.

5 DISTRIBUTED ALGORITHMS FOR CONSTRUCTING APPROXIMATE MINIMUM SPANNING TREES WITH APPLICATIONS TO WIRELESS SENSOR NETWORKS

In this chapter, we design and analyze a class of simple and local distributed algorithms called Nearest Neighbor Tree (NNT) algorithms for energy-efficient construction of MSTs in a wireless ad hoc setting. We show provable bounds on the performance with respect to both the *quality* of the spanning tree produced and the *energy needed* to construct them. For uniform distribution of nodes, we show that our algorithms give a *constant* approximation; we also show that the energy needed to construct these approximate spanning trees is within a constant factor of the minimum possible energy that is needed to construct a MST. We also perform extensive simulations of our algorithms. We tested our algorithms on both uniformly random distributions of nodes, and on realistic distributions of nodes in an urban setting. Simulations validate the theoretical results and show that the bounds are much better in practice.

5.1 Overview

5.1.1 Introduction and Motivation

The classical distributed MST algorithm due to Gallager, Humblet, and Spira (henceforth referred to as the GHS algorithm) [11] uses $\Theta(n \log n + |E|)$ messages, and is essentially optimal with respect to the message complexity. There are distributed algorithms that find the MST (for e.g., see [12,13]) and are essentially optimal in terms of time complexity: they run in $O(Diam(G) + n^{\epsilon})$ time, and there are matching lower bounds. However, these time-optimal algorithms involve a lot of message transfers (much more than GHS). Even for a wireless network modeled by a unit disk graph (or even a ring) any distributed algorithm to construct a MST needs $\Omega(n \log n)$ messages [6,14]. Despite their theoretical optimality, these algorithms are fairly involved, require synchronization and a lot of book keeping; such algorithms are impractical for ad hoc and sensor networks [6]. For example, consider sensor networks — an ad hoc network formed by large numbers of small, battery-powered, wireless sensors. In many applications, the sensors are typically "sprinkled" liberally in the region of interest and the network is formed in an ad hoc fashion by local self-configuration. Since each sensor usually knows only its (local) neighbors, the network management and communication has to be done in a *local and distributed* fashion. Additionally, because of battery limitations, energy is a very crucial resource. A distributed algorithm which exchanges a large number of messages can consume a relatively large amount of energy (and also time) is not suitable in an energy-constrained ad hoc wireless sensor network. This is especially true in a *dynamic* setting – when the network needs to be reconfigured (e.g., due to mobility or failures) frequently and quickly. Reconfiguration is also necessary to evenly distribute the energy consumption among all the nodes [5] and thus to increase network lifetime.

Thus it is necessary to develop simple, local, distributed algorithms which are energy-efficient, and preferably also time-efficient, even at the cost of being *sub-optimal* (see e.g., [6–8] for such algorithms in the context of wireless sensor networks — discussed more below). This adds a new dimension to the design of distributed algorithms for such networks. Thus we can potentially *tradeoff* optimality of the solution to work done by the algorithm. In a sensor network, the total energy required (*"energy complexity"*) in a distributed algorithm typically depends on the time needed, the number of messages exchanged, and the radiation energy needed to transmit the messages over a certain distance (see e.g., [8, 63, 64]). The radiation energy needed to transmit a message is typically assumed proportional to some *work function* f (typically square or some small power) of the distance between the sender and the receiver [5, 22, 65]. Thus it becomes important to measure efficiency of a distributed algorithm in terms of energy, besides the number of messages.

While there has been a lot of recent work on local algorithms for construction of *low-weight* connected subgraphs in the context of wireless ad hoc networks (motivated by topology control and energy-efficient routing) [66–70], to the best of our knowledge, there has been little work on localized construction of exact or approximate MSTs, especially in the context of wireless ad hoc networks. A structure is *low weight* if its total edge length is within a small factor of the total edge length of the MST (but the structure may have cycles). It is easy to show that MST cannot be constructed in a purely localized manner, i.e., each node cannot determine which edge is in the defined structure by using only the information of the nodes within some constant hops. For example, Li, Hou, and Shia [7] proposed a method to build what they call a *local minimum spanning tree (LMST)* that is guaranteed to be connected, and has bounded degree, but is *not* a low-weight structure. In fact, the paper by X. Li et al. [6] mentions the difficulty in constructing an MST and gives a localized algorithm to construct a low-weight connected subgraph (that can have cycles) for topology control in wireless ad hoc networks.

In this chapter, we study a class of *simple, local, distributed, approximation* algorithms called the *Nearest Neighbor Tree (NNT) algorithms* that are provably good: they build slightly sub-optimal trees with low energy complexity and are easy to maintain dynamically. A fundamental step in all existing algorithms for the MST is *cycle detection*: given an edge, one needs to determine whether the edge would form a cycle with the edges already chosen. This deceptively simple operation leads to a big overhead: a significant amount of book keeping and message passing needs to be done in order to maintain the components, and answer such queries. Our algorithms bypass such a step completely by a very simple idea: each node chooses a unique *rank*, a quantity from a totally ordered set, and a node connects to the *nearest* node of higher rank. Observe that this immediately precludes cycles, and the only information that needs to be exchanged is the rank; also, this information does not have to be updated continuously over the course of the algorithm.

5.1.2 MST and Work Complexity

Formally, our focus is the following geometric weighted minimum spanning tree problem: given an arbitrary set N of points (nodes) [1] in a plane [2], find a tree T spanning N such that $\sum_{(u,v)\in T} d^{\alpha}(u,v))$ is minimized where $d(u,v)$ is the length of the edge $(u,v) \in T$ according to some norm (we use the Euclidean norm) and α is a small positive number. The motivation for this objective function comes from energy requirements in a wireless communication paradigm (see also next Section): to transmit a signal over a distance r, the required *radiation energy* is proportional to r^{α}, where typically α is 2 and can range up to 4 in environments with multiple-path interferences or local noise [5,22]. Thus, given a spanning tree T, the *cost (or quality)* of a spanning tree T is defined by $Q_{\alpha}(T) = \sum_{e\in T} |e|^{\alpha}$; e denotes an edge of T and our goal is to find a tree that minimizes the cost for a given α. It can easily be shown (e.g., using Kruskal's algorithmic construction [26]) that the MST which minimizes $\sum_{(u,v)\in T} d(u,v)$ also minimizes $\sum_{(u,v)\in T} d^{\alpha}(u,v)$ for any $\alpha > 0$. In the rest of the chapter, we use the terms *cost* and *quality* interchangeably.

Two important applications of the MST in wireless networks are broadcasting and data aggregation. The MST can be used as broadcast tree to minimize radiation energy consumption since it minimizes $\sum_{(u,v)\in T} d^{\alpha}(u,v)$. It was shown in [71–73] that broadcasting based on MST consumes energy within a constant factor of the optimum. In data aggregation, the idea is to combine the data coming from different sources enroute to eliminate redundancy and minimize the number of transmissions and thus saving energy; the common aggregate functions are minimum, maximum, average, etc [9]. One popular paradigm for computing aggregates is to construct a

[1]E.g., these may represent sensors. We assume that these have unique labels or ids.

[2]We consider the 2-dimensional setting for concreteness; our results can be generalized for higher dimensions.

(directed) tree rooted at the sink where each node forwards its (locally) *aggregated* data collected from its subtree to its parent [10, 74–76]. Again, in such cases, MST is the optimal data aggregation tree, since it works exactly as a reverse broadcast tree.

Since energy is an important constraint in the setting of sensor networks, a lot of work has focused on constructing low energy subgraphs (see e.g., [8, 31]). However, it is counterproductive to use a lot of resources (e.g. time and energy) in order to compute a low energy subgraph, e.g., an MST — the energy *used* by the algorithm is also an important measure. Motivated by this, in addition to the traditional time and message complexity of distributed algorithms, we consider a complexity term called *work complexity* defined as $w = \sum_{i=1}^{M} r_i^\alpha$ where r_i is the transmission distance for message i and M is the number of messages exchanged by the nodes to run the algorithm/protocol (this is implicit in many papers, see e.g., the survey of [14]). Thus total radiation energy is directly proportional to the work done by the algorithm. Number of messages and work together determines the total energy (energy consumption in transceiver electronics + radiation energy) consumed in running the algorithm/protocol.

5.1.3 Our Contributions and Results

Our main contribution is a detailed theoretical and experimental study of a simple and local class of algorithms to construct an approximate MST, especially in the context of ad hoc and sensor networks. Our algorithms, called the *Nearest Neighbor Tree (NNT) algorithms* use a very simple idea to avoid cycle formation: each node (independently) chooses a distinct rank, and connects to the closest node of higher rank. Depending on how ranks are chosen we study two types of NNT algorithms: Random-NNT (ranks are chosen randomly) and Coordinate-NNT (Co-NNT in short; ranks are based on coordinate information)[3]. Given the simple and local nature of

[3]Both are well motivated: when nodes don't know their geometric coordinates, Random-NNT is natural (in contrast most previous work (e.g., [6,14,31] assume that nodes know their coordinates or their relative locations) but if nodes know their coordinate location (say, using GPS) then Co-NNT is more suitable.

this construction, it is quite surprising to expect trees of reasonable properties. We study both the theoretical and empirical properties of such NNT trees, and show that they have many properties that make them practical in a sensor network setting. Our main results are: (i) The tree produced by such an algorithm (called the NNT tree) has *low cost*, compared to the MST, (ii) The NNT paradigm can be used to design a *simple dynamic algorithm* for maintaining a low cost spanning tree, and (iii) The *time*, *message* and the *work* complexities of the NNT algorithms are close to optimal in several settings.

We theoretically analyze the performance of both these NNT algorithms in two scenarios :(1) nodes placed arbitrary on the plane, and (2) nodes are distributed uniformly at random, in a unit square (this is a popular probabilistic model for ad hoc wireless networks, e.g., see [77]); this model also allows us to analytically quantify the energy complexity of our algorithms in a realistic setting. Our performance analysis is with respect to the following metrics: the *quality of the spanning tree produced* by the algorithm, and the *message, time*, and *work* needed by the algorithm to construct the tree. We now summarize our results in more detail below.

Quality bounds: In Chapter 2, we showed a very general result (Theorem 2.2.1) that *any* NNT tree (i.e., irrespective of how ranks are chosen) has cost within an $O(\log n)$-factor of the MST. In fact, this bound is true even if points are located in a *metric* space. This bound immediately suggests the use of the NNT scheme to construct and maintain low cost spanning trees in sensor network type of settings, because of the very local nature of this construction. For higher values of α, powers of the distances do not satisfy the triangle inequality. However, we show that the cost of a *virtual* tree, constructed by using the shortest path for edges of high weights is within $O(\log n)$-factor of the optimal; this is discussed in more detail in Sections 2.3 and 5.5.3. Our bounds are tight, in the sense that there are instances where they cannot be improved, using these type of algorithms.

When the points are distributed uniformly at random in the unit square, we get much better bounds on the quality of the NNT trees. We show that Random-NNT gives an $O(1)$ and $O(\log n)$ approximation, respectively, for the case of $\alpha = 1$ and $\alpha = 2$, respectively. In contrast, Co-NNT gives an $O(1)$ approximation for both $\alpha = 1$ and $\alpha = 2$. Thus, in an average sense, the NNT algorithms give much better bounds on both the cost and energy of the tree, with Co-NNT being much better than Random-NNT - this shows that at a cost of increased information (i.e., about the coordinates), we can get better approximations.

Maintaining a low cost tree dynamically: We show that the degree of a node is $O(\log n)$ with high probability. This property of low node degree can be used to design a simple dynamic algorithm for maintaining a Random-NNT tree. We show that the expected number of *rearrangements*, i.e., number of nodes whose outgoing edge must change, as a result of a node insertion or deletion is $O(\log k)$, where k is the number of insertions and deletions. Each such rearrangement involves recomputing the NNT neighbor of some node. Our algorithm does not require any complicated data structures or severe constraints on the sensors. The dynamic aspect of the NNT scheme makes them very useful in a sensor network setting, where it is very common for nodes to fail, or become alive asynchronously.

Message, time, and work complexity: In the uniform random setting, we show that NNT algorithms have significantly lower message, time, and work complexity compared to other distributed algorithms which compute the exact MST. We show that the average work complexities for Co-NNT and Random-NNT are $O(1)$ and $O(\log n)$, respectively, for $\alpha = 2$. These work complexities are within a constant of optimum, because for the case of points distributed uniformly in a unit square, the cost of the MST equals these values, within constant factors (see e.g., [23]) and this *lower bounds* the work complexity of *any* algorithm that constructs it. We also show that for both NNT algorithms, the expected message complexity is $O(n)$ (on average) and time complexity is $O(\log^2 n)$ (with high probability) which are essentially the best possible.

Simulation results: We also performed extensive simulations of our algorithms. We tested our algorithms on both uniformly random distributions of points, and on realistic distributions of points in an urban setting obtained from TRANSIMS [78]. Experimental results show that the work and number of messages for NNT algorithms are significantly smaller than that for an optimal MST algorithm, while the quality of the NNT tree is very close to MST. For example, for the TRANSIMS data, we found that the cost of the tree found by the NNT algorithms is within a factor of 2 of the MST, but there is more than a ten-fold saving on the work and about a five-fold saving on the number of messages sent.

5.1.4 Network Model

We consider a wireless network composed of n nodes distributed (either arbitrarily or uniformly randomly) in a two-dimensional plane. We assume that all nodes have distinct identifiers. For the general algorithms, we assume that each node has an omni-directional antenna and a single transmission can be received by *any* node within its vicinity (called *local broadcasting*), which is assumed to be a disk of appropriate radius centered at the node. (However, we assume directional antenna only for dynamic algorithm given in Section 5.4.) We utilize this broadcasting property to reduce the communications needed in our algorithm. If the maximum transmission power is not enough to communicate with a node directly, then it can communicate through multihop wireless links by using intermediate nodes to relay the message. To simplify the presentation of our algorithms we assume that each node *can* communicate directly with all other nodes. In fact, in our algorithms, most of the nodes need to communicate with only a small number of nearby neighbors, but some nodes may need to communicate with distant nodes (as mentioned earlier, it is impossible to construct a MST in a purely localized manner [6]). This may not be possible if the maximum power level (maximum communication range) of a node is not enough to reach a node at that distance. In such a case, a node uses multi-hop communication by using other nodes to relay the message; we will elaborate on this later (cf. Section 5.5.3).

5.1.5 Other Related Work

We briefly discuss other work most relevant to our work.

X. Li et al. [6] give a local algorithm to construct a low-weight subgraph that has many desirable properties: connectivity (but may have cycles), sparseness, spanner, bounded degree, and planarity. A structure is called *low-weight* if its weight is within a constant factor of the total edge weight of MST; however it need not be acyclic. Their model is similar to ours; however, they assume the nodes need to know coordinate or at least relative positions, whereas for Random NNT, no coordinate information is needed. Their algorithm takes $O(n)$ messages which is asymptotically optimal. However their low-weight structure is not a tree and can not be used for applications where a tree is needed, e.g., data aggregation. Moreover their structure is low-weight only if the weight of an edge is interpreted as the distance between two nodes (and not as αth power of the distance, for some $\alpha > 1$). The energy consumption using this structure is within $O(n^{\alpha-1})$. Further, although they use only two-hop information, the number of nodes within the two-hop range can be as much as $O(n)$. Thus each node basically explores $O(n)$ node information, while in our algorithm, each explores expected $O(\log n)$ closest neighbors.

N. Li et al. [7] devised a localized algorithm to build similar structure called local minimum spanning tree (LMST). They use only one hop neighbor information to build LMST. However LMST is not a low-weight structure even for $\alpha = 1$ [6].

Kempe et al. [79] presented an algorithm to construct an approximate Euclidean MST using spatial gossip mechanism. They considered n nodes are located at points spaced (approximately) uniformly in R^D. This algorithm achieves an $O(\log n)$ approximation of the cost of MST on the expectation, where the cost of an edge is the Euclidean distance between two nodes ($\alpha = 1$). They did not show any approximation factor for $\alpha > 1$ in which case approximation ratio can be significantly larger than $O(\log n)$. Both random and coordinate NNT achieve $O(1)$ approximation to MST for $\alpha = 1$. To avoid cycle formation, independent random numbers by the nodes are used (similar to random NNT). However their algorithm cannot guarantee that a node is connected to the nearest possible node and thus getting a worse approximation factor. Moreover, the message and thus energy complexity for the algorithm itself can be very large. To run the Kempe and Kleinberg algorithm, the expected number of messages exchanged is $O(nf(n)\log n)$ where $f(n)$ is some poly-logarithmic function of n, which can even be larger than number of messages in GHS algorithm. Finally, Kempe and Kleinberg algorithm is not local, whereas our attempt is to develop a local distributed algorithm to reduce energy complexity.

5.2 A Local Distributed Algorithm for Construction of (Approximate) MST

in Chapter 2, we showed a very general result that the cost of *any* NNT tree (i.e., using any ordering of the nodes) is within $O(\log n)$ of the cost of the MST on G. In this chapter, based on the NNT-Scheme described in Chapter 2, we present and analyze an NNT algorithm for wireless network using wireless local broadcast model and considering nodes are uniformly distributed in a unit suare. In addition to the random raking (see Chapter 2), In this chapter we introduce and analyze the following ranking of the nodes based on the coordinates.

Coordinate-NNT (or in short, Co-NNT):

1. Assume that V is a set of points in a plane. $rank(v) = (x(v), y(v))$, i.e., the coordinate of v.

2. For two nodes v and w, $rank(w) > rank(v)$ if $x(w) > x(v)$ or if $x(w) = x(v)$ and $y(w) > y(v)$.

We assume that n nodes are uniformly distributed in a unit square. In this setting, we measure the quality of the tree produced by NNT, $Q_\alpha(T) = \sum\limits_{(u,v)\in T} d^\alpha(u,v)$, work $w = \sum\limits_{i=1}^{M} r_i^\alpha$, number of messages, and the time complexities of NNT algorithms. Although our analysis generalizes to any α, for clarity we consider $\alpha = 1$ and 2. In order to quantify the time, message and work complexity, we need to formalize the steps of finding the closest NNT neighbor; this is done in Figure 5.1. To simplify the presentation and analysis of our algorithm, we assume that each node *can* communicate directly with all other nodes by suitably increasing its transmission radius. However, it turns out that most of the nodes need to communicate with only a small number of nearby neighbors, but some nodes may need to communicate with distant nodes. If the maximum power level of a node is not enough to reach a node at that distance, a node uses multi-hop communication by using other nodes to relay the message; we discuss such an implementation in Section 5.5.3.

The algorithm consists of exchanging three types of messages: *request*, *available*, and *connect* among the nodes. Each node begins with broadcasting a *request* for connection message. Considering a unit square, each node broadcasts *request* messages successively *in phases* to the distances $\frac{2}{\sqrt{n}}, \frac{4}{\sqrt{n}}, \frac{8}{\sqrt{n}}, \ldots$, until it finds a node with higher rank. We assume that these phases are synchronized; i.e., if there is a node of higher rank within the transmission radius of a phase, the reply from that higher ranked node is received by the end of the phase. The highest ranked node among all the nodes, can never find a node with higher rank. This node stops transmitting *request* message when it reaches the maximum possible distance between any two nodes. *Request* messages carry rank information (coordinates or random number). The other nodes who can hear the message check their relative rank and send back an *available* message if their rank is higher. The sender of the *request* message selects the nearest node from the senders of *available* messages if more than one available message is received and thus it finds the nearest higher ranked node.

When coordinates are not available (e.g., for Random-NNT), senders can include the transmission power levels in the *available* messages and the recipient can determine the relative distances of the senders from these power levels and the signal-strengths of the received messages. Finally, the node sends a *connect* message to the nearest higher ranked node, that creates an edge between these two nodes.

It is known that $E[Q_1(MST)]$ is asymptotically $\Theta(\sqrt{n})$ and $E[Q_2(MST)]$ is asymptotically $\Theta(1)$ [23, 80, 81]. We show that for Co-NNT, $E[Q_1] = O(\sqrt{n})$ and

```
/* The algorithm is executed by each node $u$ independently and simultaneously. Messages are written in the format ⟨msg name, sender, [recipient], [other info]⟩. When a message is broadcasted, the recipient is not specified. $l$ is the maximum possible distance between any two nodes.*/
$i \leftarrow 1$
Repeat
    Set transmission radius (power level) $r_i \leftarrow \frac{2^i}{\sqrt{n}}$
    If $r_i > l$, set $r_i \leftarrow l$
    Broadcast $\langle \text{request}, u, rankinfo \rangle$   // rankinfo is the random number $p(u)$ & ID
                                                            // for Random-NNT
                                                            // and coordinates $(x_u, y_u)$ for Co-NNT
    $i \leftarrow i + 1$
until (receipt of an available message) or ( $r_i = l$)
For all $v$, upon receipt of $\langle \text{request}, v, rankinfo \rangle$ do
    if $rank(v) > rank(u)$,
        set transmission radius to $distance(u, v)$
        send $\langle \text{available}, u, v \rangle$ to $v$
Upon receipt of "available" message(s):
    Select the nearest node $v$ from the senders
    Send $\langle \text{connect}, u, v \rangle$ to $v$
```

Figure 5.1. Distributed NNT algorithm for wireless networks.

$E[Q_2] = O(1)$ giving an approximation factor of $O(1)$ for both of them. For Random-NNT, $E[Q_1] = O(\sqrt{n})$ and $E[Q_2] = O(\log n)$ giving approximation factors of $O(1)$ and $O(\log n)$ respectively.

We also show that the expected work complexities for Random-NNT and Co-NNT (for $\alpha = 2$) are $O(\log n)$ and $O(1)$ respectively. In conjunction with the quality

$Q_2(MST)$, this is asymptotically optimal. We show that for both NNT algorithms, the expected number of messages is $O(n)$ and time complexity is $O(\log^2 n)$ W.H.P., which are also essentially the best possible.

The following lemmas and theorems prove the above claims.

5.2.1 Random-NNT

Theorem 5.2.1 *For a Random-NNT, $E[Q_\alpha]$ is $O(\lg n)$ for $\alpha = 2$, $O(n^{1-\alpha/2})$ for $\alpha < 2$, and $O(1)$ for $\alpha > 2$.*

Proof: Consider an arbitrary node u, and concentric circles centered at u with radius $r_i = \frac{2^i}{\sqrt{n}}$ for $i = 1, 2, \ldots, m$. Considering a unit square, maximum distance between any two nodes is $\sqrt{2}$. Thus, $r_{m-1} < \sqrt{2} \le r_m$, i.e., the maximum number of circles $m < \frac{1}{2}\lg n + \frac{3}{2}$. Let C_i be the set of nodes in the circle with radius r_i, $R_i = C_i - C_{i-1}$ for $i \ge 2$, and $R_i = C_i$ for $i = 1$. For a node $v \in R_i$, distance $d(u, v) \le r_i$.

Let A_i be the event that u connects to a node $v \in R_i$. By Lemma 2.4.1, the probability that u connects to any node between jth nearest neighbor (NN) and $(k-1)$st NN is $\sum_{i=j}^{k-1} \frac{1}{i(i+1)} = \frac{1}{j} - \frac{1}{k}$, where $j \le k$. For $i \ge 2$, $|C_{i-1}| \ge 1$ since C_{i-1} contains at least one node, which is u. Probability that a particular node, other than u, is in C_{i-1} is $p \ge \frac{1}{4}\pi r_{i-1}^2 = \frac{2^{2i}\pi}{16n}$ (for a node at the corner or next to the border, probability p can be as low as $\frac{1}{4}$ of the area of the circle with radius r_{i-1}). Thus for $i \ge 2$,

$$\begin{aligned}
\Pr\{A_i\} &= \sum_{j=1}^{n}\sum_{k=j}^{n}\left(\frac{1}{j} - \frac{1}{k}\right)\Pr\{|C_{i-1}| = j \wedge |C_i| = k\} \\
&\le \sum_{j=1}^{n}\frac{1}{j}\Pr\{|C_{i-1}| = j\} \\
&= \sum_{j=1}^{n}\frac{1}{j}\binom{n-1}{j-1}p^{j-1}(1-p)^{n-j} \\
&= \frac{1}{np}\{1-(1-p)^n\} \le \frac{1}{np} \le \frac{16}{2^{2i}\pi}.
\end{aligned}$$

$$\begin{aligned} E[d^\alpha(u,v)] &\le \Pr\{A_1\}r_1^\alpha + \sum_{i=2}^{m} \Pr\{A_i\}r_i^\alpha \\ &\le r_1^\alpha + \sum_{i=2}^{m} \frac{16}{2^{2i}\pi} r_i^\alpha \\ &= n^{-\alpha/2}\left\{2^\alpha + \frac{16}{\pi}\sum_{i=2}^{m} 2^{(\alpha-2)i}\right\}. \end{aligned}$$

By linearity of expectation for n nodes, $E[Q_\alpha] = nE[d^\alpha(u,v)]$.
If $\alpha = 2$,

$$E[Q_\alpha] \le \frac{8}{\pi}\lg n + \frac{24}{\pi} + 4 = O(\lg n)$$

. If $\alpha \ne 2$,

$$E[Q_\alpha] \le \left\{2^\alpha - \frac{2^{2+2\alpha}}{\pi(2^\alpha - 4)}\right\} n^{1-\alpha/2} + \frac{2^{1+5\alpha/2}}{\pi(2^\alpha - 4)}.$$

Thus for $\alpha < 2$, $E[Q_\alpha] = O(n^{1-\alpha/2})$ and for $\alpha > 2$, $E[Q_\alpha] = O(1)$. □

Theorem 5.2.2 *Expected work complexity of Random-NNT algorithm $E[W]$ is $O(\lg n)$ for $\alpha = 2$, $O(n^{1-\alpha/2})$ for $\alpha < 2$, and $O(1)$ for $\alpha > 2$.*

Proof: Again consider an arbitrary node u. First transmission radius for *request* message is $r_1 = 2\sqrt{\frac{1}{n}}$ and for the ith transmission, $r_i = 2r_{i-1} = 2^i\sqrt{\frac{1}{n}}$. Then, the maximum number of transmissions, $m < \frac{1}{2}\lg n + \frac{3}{2}$. Let C_i be the set of nodes in the circle centered at u with radius r_i and $R_i = C_i - C_{i-1}$, the set of nodes in the ith ring. Let $A(v,u,i)$ be the event that v replies to u in phase i. For $i \ge 2$, the event $A(v,u,i)$ occurs iff $v \in R_i$ and $rank(v) > rank(u) > rank(s)$ for all $s \in C_{i-1}$. The probability that a particular node is in C_{i-1} is $p \ge \frac{\pi 2^{2i}}{16n}$, and $\Pr\{v \in R_i\} \le 3p$. Letting $|C_{i-1}| = k$, we have $\Pr\{\forall_{s\in C_{i-1}}[rank(v) > rank(u) > rank(s)]\,|v \in R_i\} = \frac{1}{k(k+1)}$. Then for $i \ge 2$,

$$\begin{aligned} \Pr\{A(v,u,i)\} &\le 3p\sum_{k=1}^{n-1}\frac{1}{k(k+1)}\binom{n-2}{k-1}p^{k-1}(1-p)^{n-k-1} \\ &\le \frac{3}{n(n-1)p} \le \frac{48}{\pi(n-1)2^{2i}}. \\ \Pr\{A(v,u,1)\} &= \Pr\{v \in C_1, rank(v) > rank(u)\} \\ &\le \frac{4\pi}{n}\cdot\frac{1}{2} = \frac{2\pi}{n}. \end{aligned}$$

Potentially, there are $n-1$ nodes that can reply to u. Thus by linearity of expectation, expected work done by the all replies to u is less than or equal to

$$(n-1)\left\{\frac{2\pi}{n}r_1^\alpha + \sum_{i=2}^{m}\frac{48}{\pi(n-1)2^{2i}}r_i^\alpha\right\} \le n^{-\alpha/2}\left\{2\pi 2^\alpha + \frac{48}{\pi}\sum_{i=2}^{m}2^{i(\alpha-2)}\right\} \tag{5.1}$$

Now we calculate the work done by the *request* and *connect* messages. Let T_i denotes the event that u needs ith transmission. $\Pr\{T_1\} = 1$. For $i \ge 2$, u needs ith transmission if and only if rank of u is the largest among all nodes in C_{i-1}. Thus,

$$\Pr\{T_i\} = \sum_{k=1}^{n}\frac{1}{k}\binom{n-1}{k-1}p^{k-1}(1-p)^{n-k} \le \frac{16}{2^{2i}\pi}$$

In each phase, there is 1 *request* message, and at most 1 *connect* message by u. Thus expected work done by u for *request* and *connect* messages is

$$\sum_{i=1}^{m}\Pr\{T_i\}2r_i^\alpha \le n^{-\alpha/2}\left\{2\times 2^\alpha + \frac{32}{\pi}\sum_{i=2}^{m}2^{i(\alpha-2)}\right\} \tag{5.2}$$

From Eq. 5.1 and 5.2, expected total work for node u,

$$E[W_u] \le n^{-\alpha/2}\left\{2(\pi+1)2^\alpha + \frac{80}{\pi}\sum_{i=2}^{m}2^{i(\alpha-2)}\right\}$$

Expected work by the algorithm, $E[W] = nE[W_u]$. Thus,

$$E[W] \le n^{1-\alpha/2}\left\{2(\pi+1)2^\alpha + \frac{80}{\pi}\sum_{i=2}^{m}2^{i(\alpha-2)}\right\} \tag{5.3}$$

This gives the desired result stated in the theorem. □

Corollary 5.2.1 *For $i \ge 2$, number of nodes that needs ith transmission is $n\Pr\{T_i\} \le \frac{16n}{4^i\pi}$ and expected number of required transmissions by a node to find a higher ranked node is $\sum_{i=1}^{m}\Pr\{T_i\} \le 1 + \frac{4}{3\pi}(1-\frac{1}{2n}) \le 1 + \frac{4}{3\pi} < 1.425$.*

Theorem 5.2.3 *Expected message complexity of Random-NNT algorithm is $O(n)$.*

Proof: If we consider work needed for every message is 1, i.e., when $\alpha = 0$, the total work is simply the number of messages, M, exchanged in the algorithm. Thus from Equation 5.3, putting $\alpha = 0$ in the right hand side, we get

$$E[M] \le n\left\{2(\pi+1) + \frac{80}{\pi}\sum_{i=2}^{m}2^{-2i}\right\} = O(n).$$

□

Theorem 5.2.4 *Running time of Random-NNT algorithm is $O(\log^2 n)$ with high probability.*

Proof: We assume that transmission of each message takes one unit of time and while one node is transmitting a message, no other node in its transmission radius (transmission range) is allowed to transmit.

The radius of the first transmission by each node is $r_1 = \frac{2}{\sqrt{n}}$. The expected number of nodes within this radius, $E[|C_1|] \leq \pi r_1^2 n = 4\pi$. Using the following standard Chernoff bound ([82]),

$$\Pr\{x \geq (1+\delta)\mu\} < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$$

with $x = |C_1|, \mu = E[|C_1|]$ and $\delta = \frac{c \log n}{\mu} - 1$, we can show that with high probability, $|C_1| < c \log n$ for sufficiently large constant c, and each node receives at most $c \log n$ *available* messages. Thus total time to complete the first phase is at most $(c \log n)^2 = O(\log^2 n)$ W.H.P.

Now consider an ith transmission phase to distance $r_i = \frac{2^i}{\sqrt{n}}$. After the $(i-1)$st phase, the distance between any two unconnected nodes is at least r_{i-1}; otherwise, one node has lower rank than the other and would connect to that in some previous phase. Thus the maximum number of unconnected nodes in any circle with radius r_i is $O(1)$ (this is the maximum number of nodes that can be packed in a circle with radius $2d$, for any d, such that distance between any two nodes is at least d. Notice that there can be at most one node in any square with side $d/2$). Next we show that each such unconnected node receives at most $O(\log n)$ *available* messages W.H.P.

Consider an arbitrary node u. Let $C_i = y$. Assume that $y \geq 60 \log n$. (If $y < 60 \log n$, then u receives $O(\log n)$ *available* messages with probability 1.) Let x denotes the number of nodes in C_{i-1}. For any node v, $\Pr\{v \in C_{i-1} | v \in C_i\} \geq \frac{1}{4}$ (inequality, instead of equality, comes from the fact that u can be close to the borders). Thus

$E[x] \geq y/4 \geq 15 \log n$. Since the position of the nodes are independent and identically distributed, using standard Chernoff bound ([82]) with $\delta = \frac{1}{2}$ and $\mu = E[x]$, we have

$$\begin{aligned} \Pr\{x < y/8\} &\leq \Pr\{x < (1-\delta)\mu\} \\ &< \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu} \leq \frac{1}{n^{2.3}} \end{aligned}$$

Let $z = |R_i| = y - x$. Then $\Pr\{z \geq 7y/8\} = \Pr\{x < y/8\} < \frac{1}{n^{2.3}}$. Since u is at the ith transmission phase, it is known that u has the largest rank among the x nodes in C_{i-1}. Now u receives exactly t *available* messages iff exactly t out of z nodes in R_i have higher ranks than u. The probability of such event is

$$\binom{z}{t} \frac{t!(y-t-1)!}{y!} \leq \left(\frac{z}{y}\right)^t$$

Let A be the event that u receives more than $20 \log n$ *available* messages, and B be the event that $z < 7y/8$.

$$\begin{aligned} \Pr\{A\} &\leq \sum_{t=\lceil 20\log n \rceil}^{z} \left(\frac{z}{y}\right)^t \leq \frac{y}{y-z}\left(\frac{z}{y}\right)^{20\log n} \\ \Pr\{A|B\} &< \frac{8}{n^{2.6}} \\ \Pr\{A\} &\leq \Pr\{A|B\} + \Pr\{\bar{B}\} < \frac{1}{n^{2.3}} + \frac{8}{n^{2.6}} \end{aligned}$$

Excluding the first phase, there are at most $\frac{1}{2}(\lg n + 1)$ phases. By union bound (i.e., Boole's inequality [82]), the probability that each of n nodes receives more than $20 \log n$ replies in each phase is less than

$$\frac{1}{2}(\lg n + 1)n\left(\frac{1}{n^{2.3}} + \frac{8}{n^{2.6}}\right) = O(1/n^{1.2})$$

Thus with probability at least $1 - O(1/n^{1.2})$, total time taken by all $\frac{1}{2}(\lg n + 1)$ phases is $\frac{1}{2}(\lg n + 1) \times O(1) \times 20 \log n = O(\log^2 n)$. $\square$

We note that only a very few nodes may need to go far to find a node of higher rank. Most of the nodes are connected to the closer neighbors. From Corollary 5.2.1, we see that the number of nodes that need i^{th} transmission is decreasing exponentially with i. Average number of transmissions by a node is at most 1.425. Thus almost

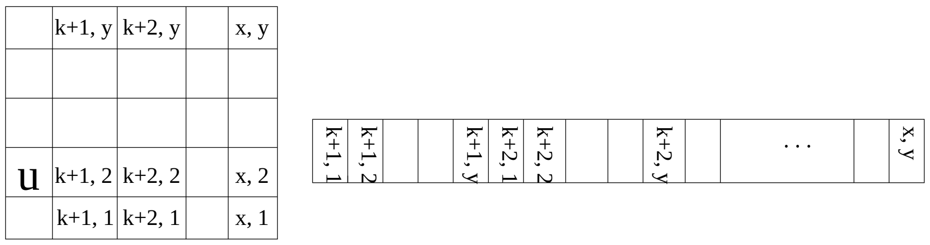

Figure 5.2. Division of the unit square into $\sqrt{n} \times \sqrt{n}$ smaller squares and their rearrangement. Here $x = y = \sqrt{n}$. Node u resides in column k. The cells in columns $k+1, k+2, \ldots, \sqrt{n}$ have been rearranged in a single row.

all of the nodes get connected after the first few transmissions. The radii for the first few transmissions are $\frac{2}{\sqrt{n}}$, $\frac{4}{\sqrt{n}}$, etc., which are very small and decreasing with n. This shows that the proposed algorithm is highly scalable and local in nature. However, if the maximum transmission range of a node is not large enough, those few nodes may not be able to get connected. One way to handle this, is to connect by using multihop communication as described in Section 5.5.3.

5.2.2 Coordinate NNT

We show analogous theorems for Co-NNT. The following theorems give time, message, work complexities, and quality of Co-NNT.

Theorem 5.2.5 *The expected quality of the tree constructed by Co-NNT algorithm for $\alpha = 1$ and 2 are $O\left(\sqrt{n}\right)$ and $O\left(1\right)$ respectively.*

Proof: We upper bound the expected distance that a node needs to connect to some other node. For the purpose of analysis, let us subdivide the unit square into $\sqrt{n} \times \sqrt{n}$ small squares (Fig. 5.2). The length of a side of each small square is $b = \frac{1}{\sqrt{n}}$.

Consider an arbitrary node u. Assume that each node selects the nearest node from the nodes that are in a column at the right of the column containing u. We further rearrange the cells in these columns, along with the nodes in it, in a single row as shown in Fig. 5.2. In this new arrangement, we are moving the nodes further away and increasing the distances among the nodes; and thus increasing the length of the edges comparing to the original Co-NNT. As a result, the expected quality of the original Co-NNT is less than that of the Co-NNT in this new arrangement. Node u connects to a node in the ith next cell, if the next $i-1$ cells are empty and there is a node in the ith next cell. The probability that the next $i-1$ cells are empty is $\left(1-\frac{i-1}{n}\right)^{n-1}$. Let P' be the probability that there is a node in the ith next cell, and P_i be the probability that u connects to a node v in the ith next cell.

$$\begin{aligned}
P_i = \left(1-\tfrac{i-1}{n}\right)^{n-1} P' \leq \left(1-\tfrac{i-1}{n}\right)^{n-1} \leq e^{-\frac{i-1}{n}(n-1)} \leq e^{-\frac{i-1}{2}} \\
E[d^\alpha(u,v)] \leq \sum_{i=1}^{n-1} (ib)^\alpha P_i \leq \sum_{i=1}^{n-1} \left(\tfrac{i}{\sqrt{n}}\right)^\alpha e^{-\frac{i-1}{2}} \\
E[Q_\alpha] = nE[d^\alpha(u,v)] \leq n^{1-\alpha/2} \sum_{i=1}^{n-1} i^\alpha \left(\tfrac{1}{\sqrt{e}}\right)^{i-1} \\
E[Q_1] \leq \sqrt{n}\sqrt{e} \sum_{i=1}^{n-1} i \left(\tfrac{1}{\sqrt{e}}\right)^{i} = O(\sqrt{n}) \\
E[Q_2] \leq \sqrt{e} \sum_{i=1}^{n-1} i^2 \left(\tfrac{1}{\sqrt{e}}\right)^{i} = O(1)
\end{aligned}$$

□

Theorem 5.2.6 *The expected work complexity of Co-NNT algorithm, for $\alpha = 1$ and 2 are $O(\sqrt{n})$ and $O(1)$ respectively.*

Proof: Again we subdivide the area into cells and consider the rearrangement of the cells in a single row as described in the proof of Theorem 5.2.5. Transmission radius for ith phase is $\frac{2^i}{\sqrt{n}}$. Length of each cell is $b = \frac{1}{\sqrt{n}}$. Thus a node, u, need ith transmission if the next 2^{i-1} cells are empty. Let T_i be the event that u needs ith transmission. $\Pr\{T_1\} = 1$ and for $i \geq 2$,

$$\Pr\{T_i\} = \left(1 - \frac{2^{i-1}}{n}\right)^{n-1} \leq e^{-2^{i-1}/2}$$

The number of *available* messages u receives in phase i is the number of nodes in $2^i - 2^{i-1} = 2^{i-1}$ cells that are covered by the transmission i but not by transmission

$i-1$. For $i \geq 2$, the expected number of such nodes in these 2^{i-1} cells, given that the first 2^{i-1} cells are empty, is

$$\frac{2^{i-1}}{n-2^{i-1}}(n-1) \leq 2^{i-1}\frac{n}{n-2^{i-1}} = 2^{i-1}\left(1+\frac{2^{i-1}}{n-2^{i-1}}\right) \leq 2^{i-1}\left(1+2^{i-1}\right).$$

The expected number of replies in the first transmission is $\frac{2}{n}(n-1) \leq 2$. In addition, in each phase, there are at most one *request* message and one *connect* message by u. Thus expected work by u,

$$\begin{aligned} E[W_u] &\leq (2+2)(2b)^\alpha \Pr\{T_1\} + \sum_{i=2}^{\lceil \lg n \rceil} \left\{2+2^{i-1}\left(1+2^{i-1}\right)\right\}(2^i b)^\alpha \Pr\{T_i\} \\ &\leq 4\frac{2^\alpha}{n^{\alpha/2}} + \frac{2^\alpha}{n^{\alpha/2}}\sum_{i=2}^{\infty}(2+i+i^2)i^\alpha\left(\frac{1}{\sqrt{e}}\right)^i \end{aligned}$$

Total work by n nodes, $E[W] = nE[W_u]$. Thus,

$$E[W] \leq 2^\alpha n^{1-\alpha/2}\left\{4+\sum_{i=2}^{\infty}(2+i+i^2)i^\alpha\left(\frac{1}{\sqrt{e}}\right)^i\right\} \tag{5.4}$$

Putting $\alpha = 1$ and 2, we have the desired result. □

Theorem 5.2.7 *The expected message complexity of Co-NNT algorithm is $O(n)$.*

Proof: Again, if we consider work for every message is 1, i.e., when $\alpha = 0$, total work is equal to the number of messages M. Thus from Equation 5.4, by putting $\alpha = 0$ in the right hand side, we get $E[M] = O(n)$. □

Theorem 5.2.8 *Running time of distributed Co-NNT algorithm is $O(\log^2 n)$ with high probability.*

Proof: A part of the proof of this theorem is similar as the proof of the Theorem 5.2.4. Using the same argument as in Theorem 5.2.4, 1) running time for the first phase of Co-NNT algorithm is $O(\log^2 n)$ W.H.P., 2) after $(i-1)$st phase, the maximum number of unconnected nodes in any circle of radius r_i is constant, $O(1)$. Next we show that in phase i, each unconnected node receives $O(\log n)$ *available* messages W.H.P. The number of unconnected nodes in C_i and the number of *available*

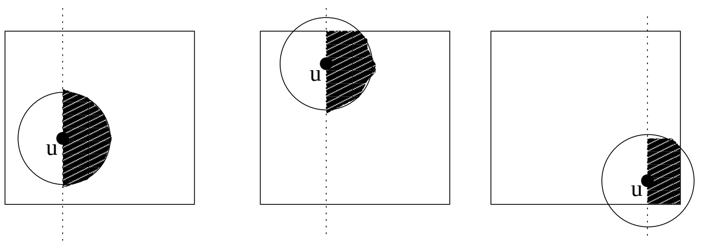

Figure 5.3. The figure shows three cases where u is a node in the unit square. In phase i, radius of the circle centered at u is $r_i = 2^i/\sqrt{n}$. Only the nodes in the shaded region reply back to u in phase i.

messages received by an unconnected node jointly determine the running time of ith phase. Assume a vertical line through node u (the dotted line in Fig. 5.3) divides the plane that contains the unit square (the unit square containing the sensor nodes) into two half-planes. Let B_i denotes the common region (the shaded region in Fig. 5.3) among the right half-plane, the disk with radius r_i centered at u, and the unit square. Let a_i be the area of the region B_i. Using simple geometry, it can easily be shown that $2a_{i-1} \leq a_i \leq 4a_{i-1}$ for any position of u in the unit square. Let n_i be the number of nodes in B_i excluding u. Now we consider two cases.

Case $a_i \leq \frac{12 \log n}{n-1}$: Then $E[n_i] \leq 12 \log n$. By Chernoff bound ([82]) with $\mu = E[n_i]$ and $\delta = \frac{36 \log n}{\mu} - 1$,

$$\begin{aligned} \Pr\{n_i \geq 36 \log n\} &= \Pr\{n_i \geq (1+\delta)\mu\} \\ &< (e^{\delta}(1+\delta)^{-(1+\delta)})^{\mu} \\ &\leq (e/(1+\delta))^{(1+\delta)\mu} \\ &\leq 1/n^3. \end{aligned}$$

The number of replies u receives in phase i cannot be more than n_i. Thus the probability that u receives more than $8 \log n$ replies is at most $1/n^3$.

Case $a_i > \frac{12 \log n}{n-1}$: Then $a_{i-1} \geq \frac{a_i}{4} > \frac{3 \log n}{n-1}$. In this case, the probability that u needs ith transmission, i.e., the probability that B_{i-1} is empty, is

$$(1 - a_{i-1})^{n-1} \leq e^{-(n-1)a_{i-1}} < 1/n^3.$$

Thus with probability at least $1 - \frac{1}{n^3}$, either u does not need phase i or number of replies in phase i is $O(\log n)$. This statement holds simultaneously for all of $O(\log n)$ phases for all n nodes with probability at least $1 - \frac{1}{n}$ (by union bound). Thus, W.H.P., running time of each phase $i \geq 2$ is $O(\log n)$ and total time is $O(\log^2 n)$. $\square$

5.3 Work Complexity of GHS Algorithm

The authors of GHS algorithm [11] shown the message and time complexity of the algorithm as we discussed earlier in this section. Here we compute the lower bound for work complexity of GHS algorithm. First we need the following lemma.

Lemma 5.3.1 *Let r_i be the distance of ith nearest neighbor for an arbitrary node. Then $E[r_i^2] = \frac{ci}{n} = \Theta\left(\frac{i}{n}\right)$, for some constant c and $\frac{1}{\pi} \leq c \leq 2$.*

Proof: To get the lower bound, consider any node u in the unit square. Now consider a circle centered at u with unit area, i.e., $\pi R^2 = 1$ where R is the radius of the circle. Let A_sc be the region that is common to both the unit square and the circle, A_s the region in the square but not in the circle, and A_c the region in the circle but not in the square. Since both the circle and the square have equal area (unit area), the area of A_s is equal to the are of A_c. Now consider a rearrangement (repositioning) of the nodes: keep the nodes in A_sc as they are and move all nodes in A_s to A_c; place the moved nodes in A_c following uniform distribution. Now it is easy to see that the distance to the ith nearest neighbor of u in the original arrangement of the nodes is greater than or equal to that in the new arrangement.

Now, the probability that a particular node (other than u) is within distance r from u (in the new arrangement) is $\frac{\pi r^2}{\pi R^2} = \frac{r^2}{R^2}$. Then the probability that there are at least i nodes within distance r,

$$C_i(r) = 1 - \sum_{k=0}^{i-1} \binom{n-1}{k} \left(\frac{r^2}{R^2}\right)^k \left(1 - \frac{r^2}{R^2}\right)^{n-k-1}.$$

The probability density function,

$$\begin{aligned} P_i(r) &= \frac{d}{dr} C_i(r) \\ &= -\sum_{k=0}^{i-1} \binom{n-1}{k} k \frac{2r}{R^2} \left(\frac{r^2}{R^2}\right)^{k-1} \left(1 - \frac{r^2}{R^2}\right)^{n-k-1} \\ &\quad + \sum_{k=0}^{i-1} \binom{n-1}{k} (n-k-1) \frac{2r}{R^2} \left(\frac{r^2}{R^2}\right)^{k} \left(1 - \frac{r^2}{R^2}\right)^{n-k-2}. \end{aligned}$$

Let $T_k =$ the first term inside the above sum $= \binom{n-1}{k} k \frac{2r}{R^2} \left(\frac{r^2}{R^2}\right)^{k-1} \left(1 - \frac{r^2}{R^2}\right)^{n-k-1}$. Then,

$$\begin{aligned} T_{k+1} &= \binom{n-1}{k+1}(k+1)\frac{2r}{R^2}\left(\frac{r^2}{R^2}\right)^k \left(1-\frac{r^2}{R^2}\right)^{n-k-2} \\ &= \binom{n-1}{k}(n-k-1)\frac{2r}{R^2}\left(\frac{r^2}{R^2}\right)^k \left(1-\frac{r^2}{R^2}\right)^{n-k-2}. \end{aligned}$$

Now $T_0 = 0$, thus

$$P_i(r) = -\sum_{k=0}^{i-1}(T_k - T_{k+1}) = T_i.$$

$$\begin{aligned} E[r_i^2] &\geq \int_0^R r^2 P_i(r)dr \\ &= iR^2\binom{n-1}{i}\int_0^R \frac{2r}{R^2}\left(\frac{r^2}{R^2}\right)^i \left(1-\frac{r^2}{R^2}\right)^{n-i-1} dr \\ &= iR^2\binom{n-1}{i}\sum_{k=0}^{i}\binom{i}{k}(-1)^k \frac{1}{k+n-i}. \end{aligned}$$

Since $n - i > 0$, using the identity $\sum_{k=0}^{n} \binom{n}{k} \frac{(-1)^k}{k+x} = x^{-1}\binom{x+n}{n}^{-1}$ (page 188 in [46]),

$$\begin{aligned} E[r_i^2] &\geq iR^2\binom{n-1}{i}\frac{1}{(n-i)\binom{n}{i}} \\ &= \frac{iR^2}{n} = \frac{i}{n\pi}. \end{aligned}$$

To get the upper bound, we consider a node u in a corner of the unit square and a circle centered at u and with radius $R' = \sqrt{2}$, the length of a diagonal of the square. If we redistribute the nodes in this circle uniformly, the average distance to the ith nearest neighbor can only increase. Thus, $E[r_i^2] \leq \frac{iR'^2}{n} = \frac{2i}{n}$. □

Theorem 5.3.1 *The expected work complexity of GHS algorithm is* $\Omega(\log^2 n)$.

Proof: We analyze work complexity for test/accept/reject messages only (the detail of GHS algorithm can be found in [11]). By the end of exection of the algorithm, each node tests all of its adjacent edges by using test/accept/reject messages through these edges one by one. To have a connected graph with high probability the required number of neighbors is $c \log n$ [77], for some constant c. Thus each node send test messages to these $c \log n$ neighbors. Using Lemma 5.3.1, the expected work by a node is at least $\sum_{i=1}^{c \log n} \frac{i}{n\pi} = \Omega(\frac{\log^2 n}{n})$. For n nodes, by linearity of expectation, total work $w = n \times \Omega(\frac{\log^2 n}{n}) = \Omega(\log^2 n)$.

Notice that if a node can communicate with all these neighbors using a single transmission, the work complexity would be $\Omega(\log n)$. However, that is not possible since there are $\log n$ phases in the algorithm, and in each phase, a node need to send at least one test message to find the minimum out going edge. $\square$

Theorem 5.3.2 *The expected work complexity of GHS algorithm to run on Yao graph is* $\Omega(\log n)$.

Proof: To find this lower bound, we consider initiate and report messages. These messages travel along the MST edges. Thus, the work done by these messages in one phase is $\Theta(1)$; that is, the total work done in $\log n$ phases is $\Theta(\log n)$. Even if the initiate message is broadcasted using a single message by the leader of a fragment, the total work for these messages is still $\Theta(\log n)$. At the ith phase, the average number of fragments is $\frac{n}{2^i}$, therefore, there are $\frac{n}{2^i}$ transmissions by the $\frac{n}{2^i}$ leaders. Again, there are average 2^i nodes in each fragments. Thus, in the ith phase, the leader need to transmit to at least to the distance of 2^ith nearest neighbor. By using Lemma 5.3.1, total work $\geq \sum_{i=1}^{\log n} \frac{n}{2^i} \frac{2^i}{n\pi} = \frac{1}{\pi} \log n$. $\square$

5.4 Dynamic Algorithm for NNT

The local nature of the NNT algorithms naturally allows for simple dynamic versions, where the goal is to maintain a tree of good quality, as nodes are added or deleted. From Theorem 3.4.1, it follows that as long as we maintain an NNT tree, the cost remains within the bounds proven in the previous theorems.

The measure we focus on, in the dynamic setting, is the expected number of rearrangements, when a node is added or deleted. We define the term *number of* *rearrangements* to be the number of the edges in the tree to be deleted plus the number of edges to be added to the tree, to maintain NNT, due to addition or deletion of a node.

The dynamic Random-NNT algorithm is described below. We partition the 2π angle around each node into 6 sectors and assume that the closest node in each cone can be determined - this could be done, for instance, by using directional antennas (such an assumption is made in several papers, e.g. [64]). For any node v, if $nnt(v)$ exists, i.e., v is not the highest ranked node, we say that a charge of 1 is placed on every point u in the closed ball $B(v, d(v, nnt(v)))$. Let $Q(v)$ denote the set of nodes in closed ball $B(v, d(v, nnt(v)))$ and $L(v) = \{u | v \in Q(u)\}$. The dynamic algorithm is given in Figure 5.4.

Next we analyze the number of rearrangements needed for each insertion and deletion. The complexity of a rearrangement depends on the model, as discussed earlier. We first need the following lemma, which bounds the charge placed on any node.

Lemma 5.4.1 *Let V be a set of points in the plane on which the Random-NNT algorithm is run. If node v connects to node w, we say that a charge of* 1 *is placed on every point u in the closed ball $B(v, d(v, w))$ (whether or not there is an actual node at point u). Then, the total charge at any fixed point u is $O(\log n)$, with high probability.*

Proof: Consider any point u, and partition the 2π angle around u into 6 cones, each of angle $\pi/3$. Consider one such cone. We prove that the total charge from points in this cone on u is $O(\log n)$, with high probability. Order the points in the cone as $v_1, v_2, \ldots$, based on increasing distance from u (Fig. 5.5). Node v_i places a charge on u only if $rank(v_i) > rank(v_j)$, for all $1 \leq j < i$. The probability of occurring this event is at most $1/i$ (the probability that a particular number is the largest among i identical random numbers is $1/i$). Thus, the total expected charge on u from these points is at most $\sum_{i=1}^{n-1}(1/i) \leq \log n$.

- If a node v is added:
  1. v chooses a random rank.
  2. v checks its neighbors $v_1, v_2, \ldots$ in non-decreasing order of the weight $d(v, v_i)$, till it finds the closest neighbor v_j of higher rank.
  3. v adds each $v_i,\ i \leq j$ in $Q(v)$ and sends a message to v_i to add it in $L(v_i)$.
  4. v finds the closest node in each of the 6 wedges.
  5. For each of these closest points, w, v sends an UPDATE message to every node in $L(w)$.
  6. If u receives an UPDATE message from some node v:
     (a) Let $u_1, u_2, \ldots$ be the neighbors of u in increasing order of distance from u, and let u is currently connected to u_k.
     (b) If $d(u, v) \leq d(u, u_k)$ and $rank(v) > rank(u)$, u removes u_ℓ from $Q(u)$ and itself from $L(u_\ell)$, $\forall j < \ell \leq k$, where $v = u_j$. Further u connects to v, instead of u_k, and adds v to $Q(u)$ and itself to $L(v)$.
     (c) If $d(u, v) \leq d(u, u_k)$ and $rank(v) < rank(u)$, u adds v to $Q(u)$ and itself to $L(v)$.
- If a node v is deleted:
  1. v deletes itself from $L(w)$ for each $w \in Q(v)$.
  2. v sends a DELETE message to each node $u \in L(v)$.
  3. A node u after receiving a DELETE message from v, deletes v from $Q(u)$. If $nnt(u) = v$, u starts checking the neighbors from v onwards till it finds the closest one of higher rank.

Figure 5.4. Dynamic Algorithm for Random-NNT.

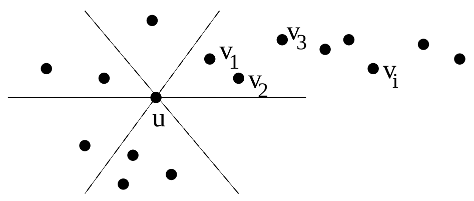

Figure 5.5. Each wedge around node u is 60°. $v_1, v_2, v_3 \ldots$ are the nodes in one wedge in increasing order of distance from u.

In order to bound the maximum charge on any node, we use a variant of the Chernoff bound (Lemma 5.4.2) that holds in the presence of dependencies among the variables.

Lemma 5.4.2 *([47]) Let* $X_1, X_2, \ldots, X_l \in \{0,1\}$ *be random variables such that for all* i*, and for any* $S \subseteq \{X_1, \ldots, X_i\}$, $\Pr[X_{i+1} = 1 | \bigwedge_{j \in S} X_j = 1] \leq \Pr[X_{i+1} = 1]$. *Then for any* $\delta > 0$, $\Pr[\sum_i X_i \geq \mu(1+\delta)] \leq (\frac{e^\delta}{(1+\delta)^{1+\delta}})^\mu$*, where* $\mu = \sum_i E[X_i]$.

Let $\mathcal{E}(v)$ be the event that node v places a charge on u. In order to use the Chernoff bound, we need to show that, for any i, and any subset $S \subset \{v_1, \ldots, v_i\}$, $\Pr[\mathcal{E}(v_{i+1}) | \bigwedge_{w \in S} \mathcal{E}(w)] \leq \Pr[\mathcal{E}(v_{i+1})]$.

First, suppose $d(w, v_{i+1}) \geq d(w, u)$ for each $w \in S$. Then, the events $\bigwedge_{w \in S} \mathcal{E}(w)$ do not place any constraint on $rank(v_{i+1})$, relative to $rank(v_j), j \leq i$, and therefore, $\Pr[\mathcal{E}(v_{i+1}) | \bigwedge_{w \in S} \mathcal{E}(w)] = \Pr[\mathcal{E}(v_{i+1})]$.

Next, suppose $d(w, v_{i+1}) < d(w, u)$ for some $w \in S$. Then occurrence of the event $\mathcal{E}(w)$ implies that $rank(w) > rank(v_{i+1})$. Further $d(v_{i+1}, w) < d(w, u) \leq d(v_{i+1}, u)$. Therefore, $\Pr[\mathcal{E}(v_{i+1}) | \bigwedge_{w \in S} \mathcal{E}(w)] = 0 \leq \Pr[\mathcal{E}(v_{i+1})]$.

Next, we apply the Chernoff bound with $\delta = \frac{5 \log n}{\mu} - 1$, where μ is the expected charge on u. Since $\mu \leq \log n$, $\delta > 0$. Let X be the total charge on u. Then,

$$\begin{aligned} \Pr\{X \geq 5 \log n\} &= \Pr\{X \geq (1+\delta)\mu\} \\ &< (e^\delta (1+\delta)^{-(1+\delta)})^\mu \\ &\leq (e/(1+\delta))^{(1+\delta)\mu} \leq 1/n^3. \end{aligned}$$

Thus, with probability at least $1 - 1/n^3$, charge on u is $O(\log n)$. Using union bound, this holds simultaneously for all nodes with probability at least $1 - 1/n^2$. □

Corollary 5.4.1 *Degree of any node* v *in the NNT is* $O(\log n)$ *W.H.P.*

Proof: Degree of a node cannot be larger than the charge on the node. Thus the corollary immediately follows from Lemma 5.4.1. □

Theorem 5.4.1 *For a sequence of k node insertions or deletions, the number of rearrangements per insertion or deletion is $O(\log k)$ W.H.P.*

Proof: When a node v is deleted, only the nodes u such that $nnt(u) = v$ needs to find a new parent to connect to. Let $D(v)$ be the degree of v. Deletion of v results in deletion of $D(v)$ edges and addition of $D(v) - 1$ new edges. Thus the number of rearrangement is $2D(v) - 1 = O(\log k)$ W.H.P. (Corollary 5.4.1). When a node v is added, a node $u \in L(v)$ may need to change its connection. For any other node $w \notin L(v)$, $d(w, nnt(w)) < d(w, v)$ and such node w does not need to change its connecting edge. Thus, due to addition of v, the number of rearrangements is at most $2|L(v)| = O(\log k)$ W.H.P. (Lemma 5.4.1). □

5.5 Simulation Results

We perform extensive simulations of our algorithms to understand their empirical performance. Our experimental setup is the following:

Number of Nodes: Varying from 50 to 5000.

Node distributions: Uniformly random distributions in the unit square and several realistic distributions of points in an urban setting obtained from TRANSIMS [78].

Number of Runs: 50

Measures: We compare the NNT trees and the MST, with respect to the quality $Q_\alpha(T) = \sum_{(u,v)\in T} d^\alpha(u, v)$ for $\alpha = 1$ and 2. We also compare the performance of the NNT algorithms and GHS, with and without the Yao graph information, with respect to the following measures: (i) Number of messages, and (ii) Work, $W = \sum_{i=1}^{M} r_i^\alpha$ for $\alpha = 2$. The GHS algorithm is message optimal in the point-to-point setting, and achieves this by means of a complicated synchronization.

To be fair to GHS which does not exploit geometry per se (to compare with Co-NNT, which uses coordinate information of the nodes) we run the GHS algorithm on the Yao graph (see e.g., [14, 83]). Yao graph is constructed as follows: the 2π angle around each node u is divided into six wedges of equal size (Fig. 5.6), and the edge

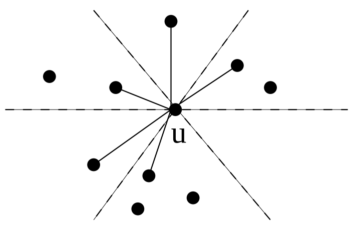

Figure 5.6. Construction of Yao graph. Each wedges is 60°. Node u have an edge with the nearest node in each wedge.

between u and the nearest node (if any) in each wedge is included in Yao graph. A Yao graph is sparse (each node has degree at most 6) and contains the MST [83]. Running GHS on Yao graph reduces message complexity to $O(n \log n)$.

In our simulations, we ignore the effects of the MAC layer. Our main results are enumerated below, and validate our theoretical results in earlier sections.

1. The Co-NNT algorithm always outperforms the Random-NNT algorithm, with respect to the quality, number of messages and the work.

2. Both Directional and Random-NNT give a very good approximation to the MST- in particular, Co-NNT is always within about 10% of the MST.

3. For $\alpha = 2$, Random-NNT does not give a very good approximation, but Co-NNT remains within a factor of 2.

4. The number of messages and the work done by both Directional and Random-NNT is very close, and significantly smaller than that by GHS or GHS with the Yao graph.

5. When the power level of nodes is bounded, and a node cannot reach its NNT neighbor, we use a multihop shortest path to make this connection. We observe that number of messages and work for NNT algorithms in this multihop setting increases only by a constant factor over the non-multihop setting and still remain significantly smaller than those of the GHS algorithm.

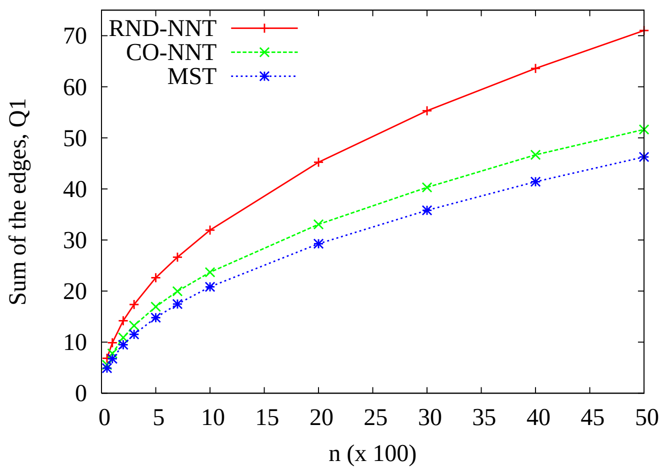

Figure 5.7. Sum of the lengths of the edges, $Q_1(T)$, for MST, Random-NNT, and Co-NNT.

5.5.1 Quality of the Spanning Trees

We present the simulation results of the quality $Q_\alpha(T)$ for $\alpha = 1$ and 2. As Figure 5.7 shows, both Random-NNT and Co-NNT compare very well with the MST. As shown earlier, the MST cost is $\Theta(\sqrt{n})$ for $\alpha = 1$, and both NNTs seems to be within a small constant factor of this value; Figure 5.8 demonstrates this by showing the values as a fraction of $\sqrt{n}$, and the plot for the NNTs are straight lines.

Figure 5.9 shows $Q_2(T)$, the sum of squares of the edge lengths, for the NNT algorithms and the optimum. Quality Q_2 for both the MST and Co-NNT are constant, and $Q_2(Co - NNT)$ is within a factor of 2 of $Q_2(MST)$. However, the value of Q_2 for Random-NNT increases with n as the asymptotic bound is $O(\log n)$- this becomes clear from Figure 5.10.

5.5.2 Work and Message Complexities to Construct the Spanning Trees

In this section, we compare work w for $\alpha = 2$ and number of messages needed by the algorithms. The NNT algorithms are compared with GHS, both with and without

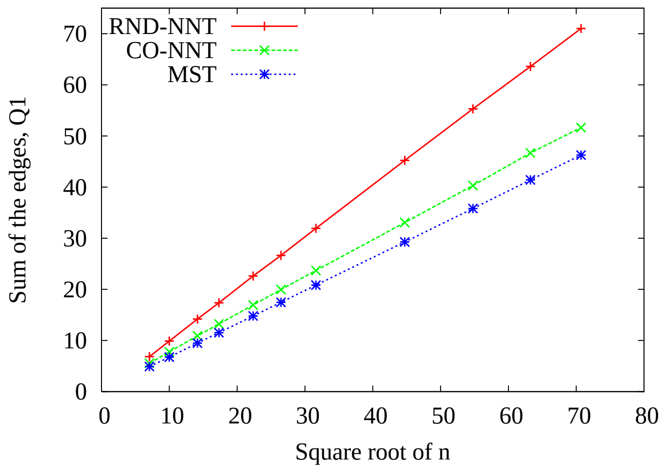

Figure 5.8. Sum of the lengths of the edges, $Q_1(T)$, plotted with $\sqrt{n}$ for MST, Random-NNT, and Co-NNT.

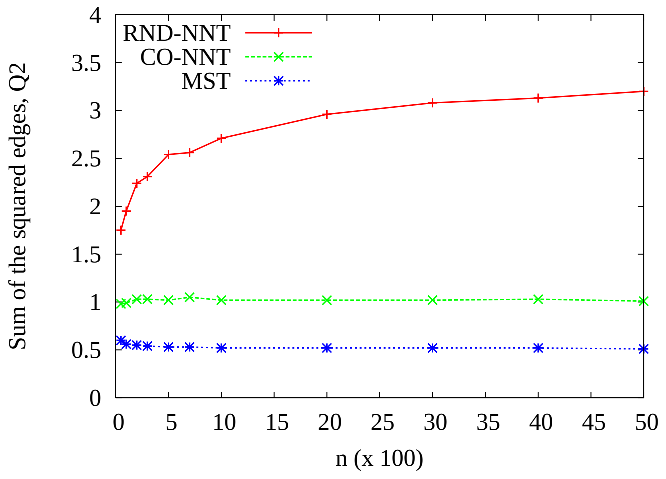

Figure 5.9. Sum of the squares of the edge lengths, $Q_2(T)$ for MST, Random-NNT, and Co-NNT.

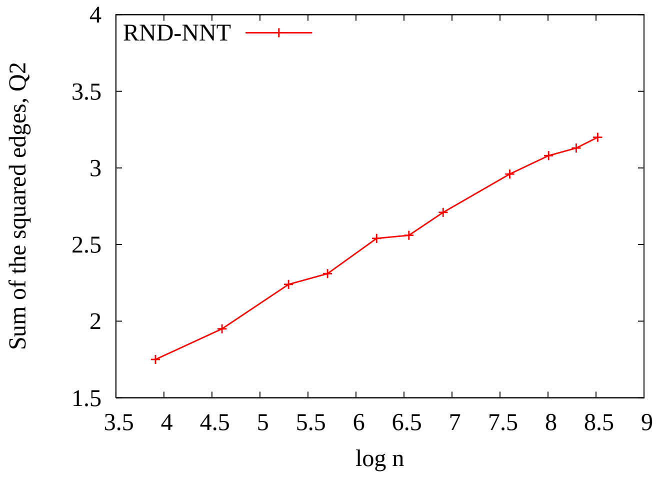

Figure 5.10. $Q_2(T)$ for Random-NNT with respect to $\log n$.

the Yao graph. The input to the GHS algorithm must be a connected graph to obtain MST. To have a connected graph, with high probability, in a wireless ah-hoc network, when the nodes are uniformly distributed, each node must be connected to the nodes which are within distance $\Theta(\sqrt{\frac{\log n}{n}})$ [77]. We consider the radius of the neighborhood to be $1.6\sqrt{\frac{\log n}{n}}$, the minimum required for connectivity. Since each node sends at least one message to each of its neighbor (test message - to check if the neighbor is in the same fragment), cost of GHS algorithm increases as the number of neighbors of the nodes increases. Here we note that the way GHS algorithm works, it needs to test the neighbors sequentially. Thus it cannot take advantage of broadcasting. Even if we consider that a node tests all of its neighbors by broadcasting a single message to all of its neighbors, each neighbor must reply to this test message individually. Thus number of messages related to testing the neighbors is still no less than the number of neighbors.

To determine the neighbors, each node can broadcast a message to distance $1.6\sqrt{\frac{\log n}{n}}$ and consider another node as a neighbor if the node can hear the message from the other node. However, we did not incur any cost on GHS algorithm to find the neighbors (thus favoring GHS). We assumed that each node knows its neighbors and their distances. In addition, we also simulate GHS on the Yao graph. Each node finds its Yao neighbors first, then executes GHS algorthm.

Fig. 5.11 depicts the number of messages needed to construct the tree. We see that the number of messages for NNT algorithms is significantly smaller than GHS. Moreover, the number of messages for NNT algorithms increases linearly. On the other hand, the number of messages for GHS increases at a slightly higher rate. In fact, message complexity for GHS is $O(n \log n)$.

Required work for NNT algorithms is also significantly less than that of GHS algorithm (Fig. 5.12). In addition, with the number of nodes, work for NNT algorithms increases in a lower rate than GHS. In terms of both number of messages and work, GHS with Yao graph is more efficient than GHS without Yao graph. However, it is still much less efficient than NNT algorithms.

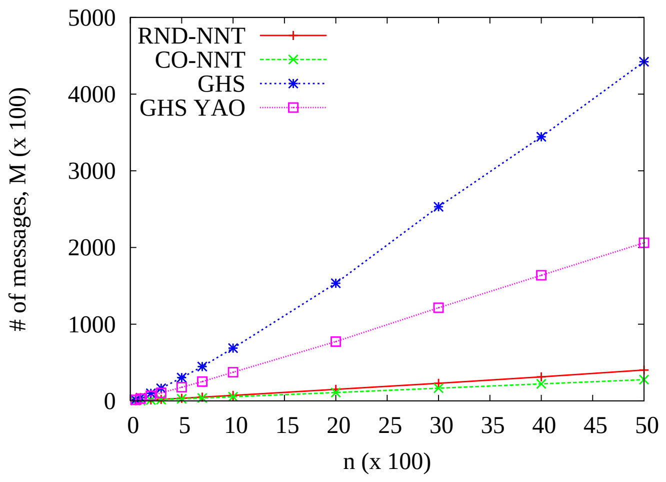

Figure 5.11. Number of messages needed to construct the spanning trees.

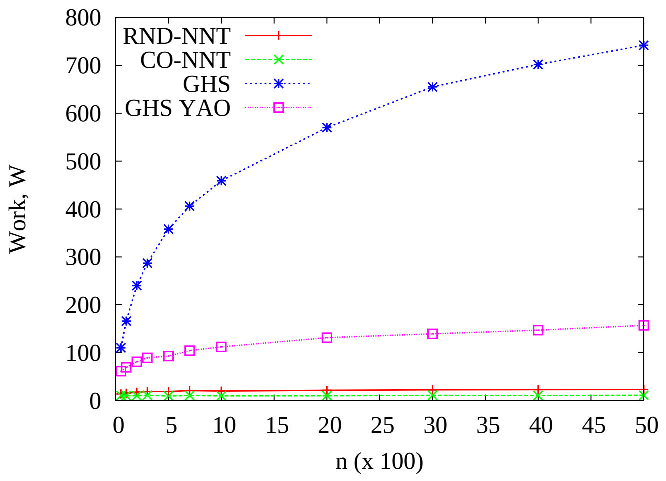

Figure 5.12. Work done by the algorithms.

Analytically, we know that for $\alpha = 2$, the work complexities for Co-NNT, Random-NNT, and GHS and GHS-YAO are $O(1)$, $O(\log n)$, $\Omega(\log^2 n)$ (Throrem 5.3.1), and $\Omega(\log n)$(Theorem 5.3.2) respectively. We can also observe these results from experimental data. Let work $w = c \log^a n$. Then $\log w = \log c + a \log \log n$. Thus if we plot $\log w$ vs. $\log \log n$ the graph is an straight line and the slope of the line is a, the power of log. In Fig. 5.13, the slope for GHS is greater than 2. For GHS-YAO, the slope is about 1 and for Random-NNT, it is less than 1. For Co-NNT the slope is 0 which indicates work is $O(1)$.

5.5.3 Implementing NNT Algorithms in a Multihop Setting

As mentioned earlier, in the basic NNT scheme few nodes have to go a large distance to connect to their nearest node of higher rank. In a practical setting, the maximum power levels of nodes are limited and hence cannot directly communicate with distant nodes in a single hop. In this case, we connect via shortest paths. Given a node u, to find its nearest neighbor of higher rank, the node uses a shortest path finding algorithm implemented in a multihop manner. In our implementation, we use a multihop algorithm based on a straightforward distributed implementation of Dijkstra's shortest path algorithm [13]. Although, clearly, the number of messages will be more in a multihop implementation, all the messages are transmitted through short distances and hence we expect the work complexity to be comparable to the non-multihop (direct connection) setting. Our simulation results show that this is indeed the case.

Fig. 5.14 and 5.15 present the simulation results for multihop algorithm using uniform distribution of the nodes. In this simulation, we assume that the maximum transmission range of a node is $\Theta(\sqrt{\frac{\log n}{n}})$, which is necessary to have a connected graph. We see that number of messages and work for NNT algorithms in this multihop setting increases only by a constant factor over the non-multihop setting (results presented above) and still significantly smaller than those of GHS algorithm.

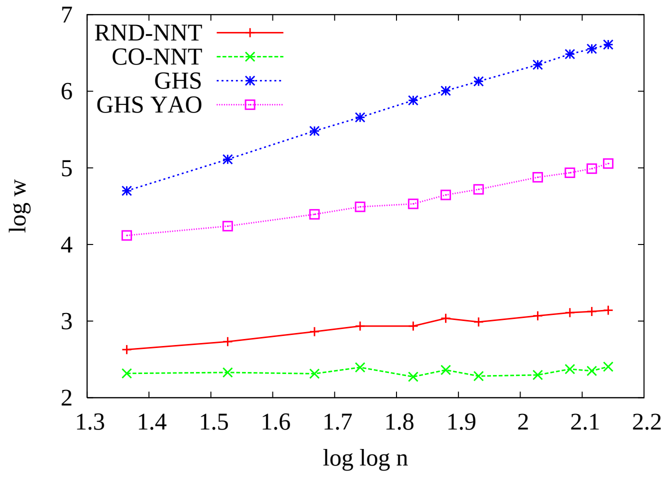

Figure 5.13. Slope of the lines indicate the powers of log in work complexity.

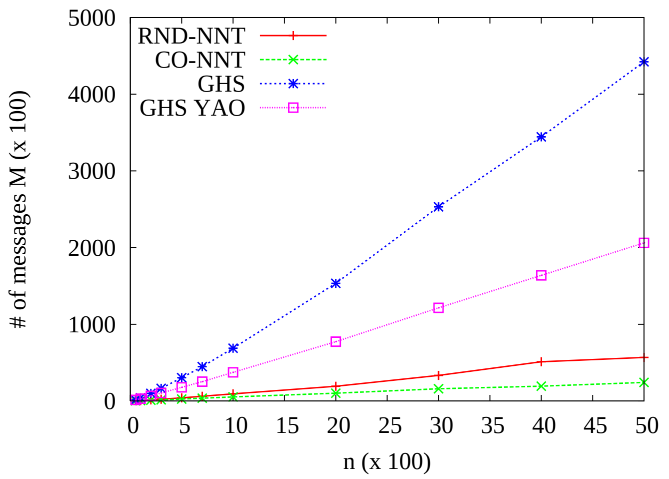

Figure 5.14. Number of messages needed to construct the trees in Multi-hop Communication.

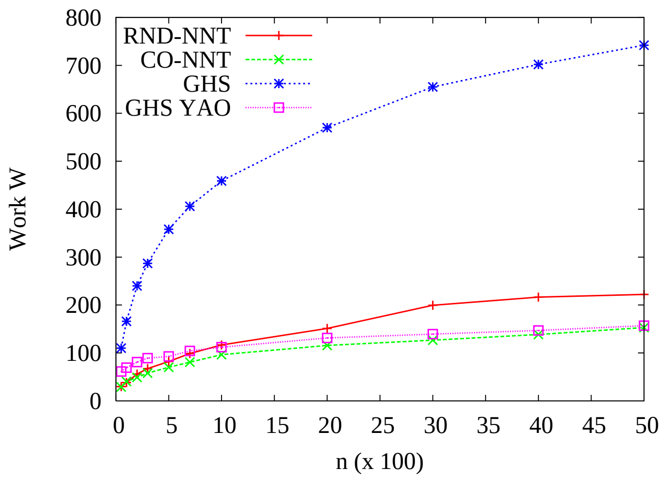

Figure 5.15. Work needed to construct the spanning trees in Multi-hop Communication.

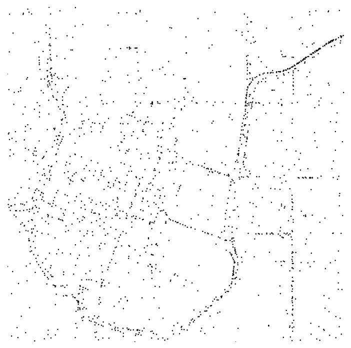

Figure 5.16. The distribution of nodes in one of the snapshots.

5.5.4 Experiments on Real Data

We consider a distribution of points in a section of downtown Portland, OR, measuring $2900m \times 2950m$ approximately 9 square KM. The distribution of points, corresponding to cars on the roadway, was obtained from the TRANSIMS simulation [78], which does a very detailed modeling of urban traffic, combining a variety of data sources, ranging from census data to activity surveys to land use data. We use three snapshots, at one minute intervals. The number of nodes (or cars) in these snapshots are different as some cars are moving in and out of this section. The distribution of nodes at one of the snapshots is shown in Fig. 5.16. Experimental results on these three snapshots are given in Table 5.1. Where the original data was in meters, we converted into KM. Work is computed for $\alpha = 2$.

We see that work and number of messages are significantly larger for GHS algorithm. Work is about 10 times larger and number messages is about 5 times larger than NNT algorithms. On the other hand, both Q_1 and Q_2 for Co-NNT is within 2-approximation. Although approximation for Q_2 in Random-NNT is large, for Q_1, Random-NNT also provides a close approximation. In this experiment, we only considered the Yao graph assuming that the nodes know their coordinates. If the coordinates are not available, for GHS algorithm input need to be a complete graph (each node is a neighbor of the others) to make sure connectivity since the points does not follow any particular (say uniform) distribution. Thus GHS would incur large work and messages. In that case, Random-NNT will still be a good choice over GHS, by sacrificing quality.

5.6 Implementation of NNT-Scheme in a Wireless Sensor Network Modeled as a Unit Disk Graph

In a disk graph model [14], we are given a connected graph $G = (V, E)$; V is the set of the sensor nodes and $(u, v) \in E$, i.e., u and v are neighbors of each other, if and only if $d(u, v) \leq R$, where $d(u, v)$ is the distance between nodes u and v, and

Table 5.1
Experiment results for Snapshot 1, 2, and 3

	Snapshot 1				Snapshot 2				Snapshot 3			
	Q_1	Q_2	Work	Msg	Q_1	Q_2	Work	Msg	Q_1	Q_2	Work	Msg
Co-NNT	38.72	6.77	90.54	4832	39.39	8.18	92.28	4647	38.32	6.25	83.42	4668
Rnd-NNT	50.75	14.13	131.42	5241	52.97	20.12	137.91	5250	52.57	18.47	148.88	5229
GHS-Yao	33.16	3.73	1271.11	20592	33.52	3.82	1083.99	20417	33.27	3.78	1083.99	20417

R is the transmission radius of the nodes. We assume a wireless broadcast model, i.e., when v transmit a message, all node u, such that $d(u, v) \leq R$ can receive the message and the edges are bidirectional. We also assume that each node knows only its neighbors and can communicate with them only. The distributed NNT algorithm runs on G. Initially each node v knows only its own $ID(v)$ and the following algorithm is executed by each node simultaneously. At the end of the execution of the algorithm, both u and v knows whether (u, v) is an edge in Random-NNT. Typically, in a sensor network, a special node called sink gathers data from the sensors and supposed to be the root of the tree. Thus we assume that there is a special node s, the sink, is designated to the root of NNT[4].

Our NNT algorithm is very local in nature. Each node uses information (random number and ID) from its immediate neighbors only. Using this information each node u can decide deterministically whether (u, v) is an edge of NNT. This local nature makes NNT algorithm message-optimal and energy-efficient.

5.6.1 The Algorithm

The algorithm is executed in two phases. In the first phase, the nodes choose their ranks randomly as follow. This phase is initiated by the sink s. The sink generates a random number, $p(s)$ and sends this number along with its ID to its neighbors in one transmission. A neighbor v of the sink s, after receiving $p(s)$, picks a random number $p(v)$ smaller than $p(s)$, and transmit $p(v)$ 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 neighbor if the given unit disk graph is connected and some nodes may receive more than one messages. As soon as a node u receives the first message from a neighbor v, it generates a random number $p(u)$ 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

[4]If there is no such node is designated, a leader election algorithm can be executed before running the distributed NNT algorithm.

stores $p(v')$ and $ID(v')$, and do nothing else. $p(u)$ and $ID(u)$ constitute u's rank $r(u)$. At the end of the first phase, each node knows the ranks of all of its neighbors. Once u receives messages from all of its neighbors, it executes the second phase of the algorithm.

It is easy to see that at the end of the first phase, i) each node knows the ranks of all of its neighbors, ii) each node u, except the sink s, has at least one neighbor v such that $rank(u) < rank(v)$, and iii) the sink has the highest rank.

In the second phase, each node $u \neq s$ selects the nearest[5] node w among its neighbors such that $r(u) < r(w)$ and sends a message to w to inform that (u, w) is an edge in the spanning tree.

5.6.2 Analysis

Theorem 5.6.1 *The above algorithm produces an NNT correctly.*

Proof: To prove the correctness of the above algorithm it is sufficient to show that (i) each node u, except the root s, connects to another node w such that $r(u) < r(w)$ and (ii) there is no node w' such that $r(u) < r(w')$ and $d(u, w') < d(u, w)$.

To show part (i), observe that in the second phase of the algorithm u choose a node only with higher rank than its own. Further, at the end of the first phase, it is guaranteed that each node u has at least one neighbor with higher rank. u chooses its random number $p(u)$ after receiving a message from one of its neighbor v satisfying $p(u) < p(v)$, which implies $r(v) < r(u)$.

To show part (ii), observe that in the second phase of the algorithm, u chooses the nearest of all neighbors with higher rank. Thus if such a node w' exists, u would pick w' instead of w. $\square$

[5]If the distances of the neighbors are not known before executing the algorithm, a node can determine the distance of all of its neighbors from the signal strengths of the messages received from them in the first phase.

Quality of the NNT. When nodes are uniformly distributed, to have a connected graph with high probability, it is necessary and sufficient that R be $\Theta(\sqrt{\frac{\log n}{n}})$ [77]. Thus, we assume $R = \Theta(\sqrt{\frac{\log n}{n}})$.

Since each node has at least one neighbor with higher rank, the length of any edge of an NNT is at most R. Then $Q_\alpha(NNT) \leq (n-1)R^\alpha = O(n^{1-\alpha/2}\log^{\alpha/2} n)$; that is $E[Q_1(NNT)] = \Theta(\sqrt{n \log n})$ and $E[Q_2(NNT)] = O(\log n)$. Since $E[Q_1(MST)] = \Theta(\sqrt{n})$ and $E[Q_2(MST)] = \Theta(1)$, we have approximation ratio of $O(\sqrt{\log n})$ and $O(\log n)$ for $\alpha = 1$ and 2 respectively.

Message Complexity. In each phase each node transmit exactly one message. Thus total message is at most $2n = O(n)$, which is essentially message-optimal. At least $\Omega(n)$ transmissions are necessary to build any spanning tree over n nodes.

Work Complexity. Each node transmit a message to a distance of at most R. Hence total work for $2n$ message is $W \leq 2nR^2 = O(\log n)$.

Time Complexity. Let D be the diameter of the given unit disk graph. It can be shown that the number of nodes within distance R from any point is $O(\log n)$ with high probability. Then the running time for the first phase is $O(D + \log n)$ WHP, and the running time for the second phase is $O(\log n)$ WHP. Thus, total time is $O(D + \log n)$ WHP.

5.7 Conclusion and Further Work

We have shown that the NNT paradigm is a simple and local scheme for constructing and maintaining low cost trees in a sensor network setting. It does not require any complex synchronization, and is naturally robust. We study various properties, such as quality, degree and dynamic complexity for different NNT trees, both from a theoretical and an empirical perspective, and it shows very promising results.

Among the questions for further work, the most interesting ones are: (i) Improve the time and message complexity of the NNT algorithms for arbitrary point distributions, (ii) For the energy analysis, we make the simplifying assumption that each node can connect directly to all nodes. It will be nice to analyze the algorithm without this assumption, i.e., nodes have only limited transmission range and hence cannot directly communicate to all nodes. Hence, nodes connect to the shortest paths by multihop communication. (iii) Quantify the tradeoff between the amount of local information needed, and the quality of the tree produced, and (iv) It will be nice to compare the performance of NNT with the best algorithm that achieves the same approximation ratio. For example, one can try to find a good *lower bound* on the energy/messages needed by any approximation algorithm that achieves $O(\log n)$ approximation ratio. To the best of our knowledge nothing is known about this for the model addressed here. For example, a concrete open question is: what is the lower bound on the energy complexity needed to compute a $O(\log n)$ approximate MST under uniform distribution of nodes?

LIST OF REFERENCES

LIST OF REFERENCES

[1] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*, September 2006.

[2] M. Khan, G. Pandurangan, and V.S.A. Kumar. A simple randomized scheme for constructing low-weight k-connected spanning subgraphs with applications to distributed algorithms. *Theoretical Computer Science*, 2007. Article in press, DOI: 10.1016/j.tcs.2007.05.028.

[3] M. Khan, G. Pandurangan, and V.S.A. Kumar. Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*. To appear.

[4] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. *Distributed Computing*. Article in press.

[5] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, October 2002.

[6] X. Li, Y. Wang, W. Song, and O. Frieder. Localized low-weight graph and its applications in wireless ad hoc networks. In *Proceedings of INFOCOM, IEEE International Conference*, 2004.

[7] N. Li, J. Hou, and L. Sha. Design and analysis of an MST-based topology control algorithm. In *Proceedings of INFOCOM, IEEE International Conference*, 2003.

[8] M. Cheng, M. Cardei, X. Cheng, L. Wang, Y. Xu, and D. Du. Topology control of ad hoc wireless networks for energy efficiency. *IEEE Transactions on Computers*, 53(12):1629–1635, December 2004.

[9] B. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. In *Proceedings of the International Workshop on Distributed Event-Based Systems*, July 2002.

[10] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[11] R. Gallager, P. Humblet, and P. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, January 1983.

[12] M. Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 331 – 340, June 2004.

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

[14] X. Li. Localized construction of low weighted structure and its applications in wireless ad hoc networks. *ACM Wireless Network*, 11(6), November 2005.

[15] G. Kortsarz and Z. Nutov. Approximating min-cost connectivity problems. *Handbook on Approximation Algorithms and Metaheuristics*, Ed. T. Gonzalez, Chapman & Hall, 2006.

[16] A. Frank. Connectivity and network flows. *Handbook of Combinatorics*, pages 111–177, Eds. R. Graham, M. Grotschel and L. Lovasz, MIT Press, 1995.

[17] D. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, MA, 1996.

[18] J. Cheriyan, S. Vempala, and A. Vetta. Approximation algorithms for minimum-cost k-vertex connected subgraph. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 206–312, May 2002.

[19] G. Kortsarz and Z. Nutov. Approximating node connectivity problem via set covers. In *Proceddings of the 3rd International workhop on approximation algorithms for combinatorial optimization (APPROX)*, pages 194–205, September 2000.

[20] G. Kortsarz and Z. Nutov. Approximation algorithm for k-node connected subgraphs via critical graphs. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 206–312, June 2004.

[21] E. Korach, S. Moran, and S. Zaks. 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):231–236, 1987.

[22] K. Delin and S. Jackson. Sensor web for in situ exploration of gaseous biosignatures. In *Proceedings of IEEE Aerospace Conference*, March 2000.

[23] J. Steele. Asymtotics for Euclidian minimal spanning trees on random points. *Probability Theory and Related Fields*, 92:247–258, 1992.

[24] D. Rosenkrantz, R. Stearns, and P. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal of Computing*, 6(3):563–581, 1977.

[25] M. Imase and B. Waxman. Dynamic Steiner tree problem. *SIAM Journal of Discrete Mathematics*, 4(3):369–384, 1991.

[26] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[27] V. Vazirani. *Approximation Algorithms.* Springer Verlag, 2004.

[28] E. Korach, S. Moran, and S. Zaks. Optimal lower bounds for some distributed algorithms for a complete network of processors. *Theoretical Computer Science*, 64:125–132, 1989.

[29] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally. In *Proceedings of the Symposium on Principles of Distributed Computing (PODC)*, July 2004.

[30] R. Rajaraman. Topology control and routing in ad hoc networks: A survey. *SIGACT News*, 33:60–73, 2002.

[31] X. Li. Algorithmic, geometric and graphs issues in wireless networks. *Journal of Wireless Communications and Mobile Computing*, 3(2):119–140, 2003.

[32] Z. Toroczkai and K. Bassler. Network dynamics: Jamming is limited in scale-free systems. *Nature*, 428(6984):716, April 2004.

[33] A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001.

[34] A. Czumaj and A. Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. In *Proceedings of 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 74–83, January 2002.

[35] J. Cheriyan, M. Kao, and R. Thurimella. Scan-first search and sparse certificates: an improved parallel algorithm for k-connectivity. *SIAM Journal of Computing*, 22(1):157–174, 1993.

[36] R. Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components (extended abstract). In *Proceedings of the Symposium on Principles of Distributed Computing (PODC)*, pages 28–37, August 1995.

[37] E. Jennings and L. Motyckova. Distributed algorithms for sparse k-connectivity certificates. In *Proceedings of the Symposium on Principles of Distributed Computing (PODC)*, May 1996.

[38] S. Huang. A new distributed algorithm for the biconnectivity problem. In *Proceedings of the International Conference on Parallel Processing*, volume III, pages 106–103, 1989.

[39] W. Hohberg. How to find biconnected components in distributed networks. *Journal of Parallel and Distributed Computing*, 9(4):374–386, 1990.

[40] R. Diestel. *Graph Theory.* Springer-Verlag New York, Inc., 1997.

[41] J. Edmonds. Edge-disjoint branchings. *Combinatorial Algorithms*, pages 91–96, Ed. R. Rustin, Academic Press, New York, 1973.

[42] A. Frieze. On the vaule of a random minimum spanning tree problem. *Discrete Applied Mathematics*, 10(1):47–56, 1985.

[43] J. Fill and J. Steele. Exact expectations of minimal spanning trees for graphs with random edge weights. In *Stein's Method and Applications*, pages 169–180, Eds. A. Barbour and L. Chen, World Scientific Publications, 2005.

[44] D. Gamarnik. The expected value of random minimal spanning tree of a complete graph. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, January 2005.

[45] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithm and Probabilistic Analysis.* Cambridge University Press, first edition, 2005.

[46] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Publishing Company, Inc., second edition, 1989.

[47] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SIAM Journal on Computing*, 26:350–368, 1997.

[48] Y. Afek and E. Gafni. Simple and efficient distributed algorithms for election in complete networks. In *Proceedings of the 22nd Annual Allerton Conference on Communication, Control, and Computing*, pages 689–698, 1984.

[49] P. Humblet. Selecting a leader in a clique in $O(n \log n)$ messages. In *Proceedings of the 23rd conference on decision and control*, pages 1139–1140, 1984.

[50] F. Chin and H.F. Ting. An almost linear time and $O(n \log n + e)$ messages distributed algorithm for minimum-weight spanning trees. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 257–266, 1985.

[51] E. Gafni. Improvements in the time complexity of two message-optimal election algorithms. In *Proceedings of the 4th Symposium on Principles of Distributed Computing (PODC)*, pages 175–185, 1985.

[52] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proceedings of the 19th ACM Symposium on Theory of Computing (STOC)*, pages 230–240, May 1987.

[53] J. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal of Computing*, 27:302–316, 1998.

[54] S. Kutten and D. Peleg. Fast distributed construction of k-dominating sets and applications. *Journal of Algorithms*, 28:40–66, 1998.

[55] M. Elkin. A faster distributed protocol for constructing minimum spanning tree. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 352–361, 2004.

[56] D. Peleg and V. Rabinovich. A near-tight lower bound on the time complexity of distributed MST construction. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 253–261, 1999.

[57] M. Elkin. An overview of distributed approximation. *ACM SIGACT News Distributed Computing Column*, 35(4):40–57, December 2004.

[58] G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 1994.

[59] Z. Lotker, B. Patt-Shamir, E. Pavlov, and D. Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM Journal of Computing*, 35(1):120–131, 2005.

[60] F. Kuhn and R. Wattenhofer. Dynamic analysis of the arrow distributed protocol. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures (SPAA)*, pages 294–301, 2004.

[61] Z. Lotker, B. Patt-Shamir, and D. Peleg. Distributed mst for constant diameter graphs. In *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 63–72, August 2001.

[62] W. Hoeffding. Probability for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[63] W. Song, Y. Wang, X. Li, and O. Frieder. Localized algorithms for energy efficient topology in wireless ad hoc networks. In *Proceedings of The ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, May 2004.

[64] R. Wattenhofer, L. Li, P. Bahl, and Y. Wang. Distributed topology control for power efficient operation in multihop wireless ad hoc networks. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, April 2001.

[65] L. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc. Power consumption in packet radio networks. *Theoretical Computer Science*, 243(1-2):289–305, July 2000.

[66] X. Li, G. Calinescu, and P. Wan. Distributed construction of planar spanner and routing for ad hoc wireless networks. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, June 2002.

[67] X. Li, P. Wan, Y. Wang, and O. Frieder. Sparse power efficient topology for wireless networks. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS)*, January 2002.

[68] R. Ramanathan and R. Rosales-Hain. Topology control of mutlihop wireless networks using transmit power adjustment. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, March 2000.

[69] Y. Wang and X. Li. Geometric spanner for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(5):1–14, May 2003.

[70] Y. Wang, X. Li, and O. Frieder. Distributed spanner with bounded degree for wireless ad hoc networks. *IEEE Transactions on Computers*, 53(12):1629–1635, December 2004.

[71] C. Ambuhl. An optimal bound for the MST algorithms to compute energy efficient broadcast trees in wireless networks. In *Proceedings of the 32th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1139–1150, November 2005.

[72] P. Wan, G. Calinescu, X. Li, and O. Frieder. Minimum-energy broadcasting in static ad hoc wireless networks. *Wireless Networks*, 8(6):607–617, November 2002.

[73] A. Clementi, P. Crescenzi, P. Penna, G. Rossi, and P. Vocca. On the complexity of computing minimum energy consumption broadcast subgraph. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 121–131, June 2001.

[74] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and G. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, October 2001.

[75] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networks (MobiCom)*, August 2000.

[76] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, July 2002.

[77] F. Xue and P. Kumar. The number of neighbors needed for connectivity of wireless networks. *Wireless Networks*, 10(2):169–181, March 2004.

[78] Transportation Analysis Simulation Systems, Los Alamos National Lab. transims.tsasa.lanl.gov.

[79] D. Kempe and J. Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, November 2002.

[80] J. Bearwood, H. Halton, and J. Hammersley. The shortest path through many points. *Proceedings of the Cambridge Philosophical Society*, 55:299–327, 1959.

[81] F. Avram and D. Bertsimas. The minimum spanning tree constant in geometrical probability and under the independent model: a unified approach. *The Annals of Applied Probability*, 2(1):113–130, 1992.

[82] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[83] A. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

VITA

VITA

Maleq Khan received the Ph.D. in Computer Science from Purdue University, the M.S. in Computer Science from North Dakota State University, and the B.S. in Computer Science and Engineering from Bangladesh University of Engineering and Technology. His research interests include the design and analysis of algorithms, distributed algorithms, wireless sensor networks, communication networks, and data mining. He received the Best Student Paper Award at the 20th International Symposium on Distributed Computing (DISC 2006) held in Stockholm, Sweden. He was awarded a Bilsland Dissertation Fellowship by Purdue University in 2006.