Efficient Distributed Approximation Algorithms via Probabilistic Tree Embeddings *

Maleq Khan † Fabian Kuhn ‡ Dahlia Malkhi § Gopal Pandurangan ¶
Kunal Talwar §

Abstract

We present a uniform approach to design efficient distributed approximation algorithms for various fundamental network optimization problems. Our approach is randomized and based on a probabilistic tree embedding due to Fakcharoenphol, Rao, and Talwar [16] (FRT embedding). We show how to efficiently compute an (implicit) FRT embedding in a decentralized manner and how to use the embedding to obtain efficient expected $O(\log n)$-approximate distributed algorithms for various problems, in particular the generalized Steiner forest problem (including the minimum Steiner tree problem), the minimum routing cost spanning tree problem, and the k-source shortest paths problem.

The distributed construction of the FRT embedding is based on the computation of least elements (LE) lists, a distributed data structure that is of independent interest. Assuming a global order on the nodes of a network, the LE-list of a node stores the smallest node (w.r.t. the given order) within every distance d (cf. Cohen [8], Cohen and Kaplan [9]). Assuming a random order on the nodes, we give a distributed algorithm for computing LE-lists on a weighted graph with time complexity $O(S \log n)$, where S is a graph parameter called the shortest path diameter which can be considered the weighted counterpart of the diameter D of the graph. For unweighted graphs, our LE-lists computation has asymptotically optimal time complexity of $O(D)$. As a byproduct, we get an improved synchronous leader election algorithm for general networks that is both time-optimal and almost message-optimal with high probability.

Keywords: Distributed Approximation Algorithms, Generalized Steiner Forests, Shortest Paths, Optimum Routing Cost Spanning Trees, Least Element Lists, Metric Spaces, Network Optimization, Leader Election, Probabilistic Tree Embeddings

*A preliminary version of this paper appeared in the Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing (**PODC**), 2008, Toronto, Canada, pp. 263-272.

†Network Dynamics and Simulation Science Laboratory, Virginia Bioinformatics Institute, Virginia Tech, Blacksburg, VA 24061, USA. E-mail: maleq@vbi.vt.edu. M. Khan was supported in part by NSF Award CNS-0626964.

‡Faculty of Informatics, University of Lugano, 6904 Lugano, Switzerland. E-mail: fabian.kuhn@usi.ch

§Microsoft Research, Mountain View, CA 94043, USA. E-mail: {dalia, kunal}@microsoft.com

¶Division of Mathematical Sciences, Nanyang Technological University, Singapore 648477. E-mail: gopalpandurangan@gmail.com. Supported in part by Nanyang Technological University grant M58110000. Part of the work done when the author was at Purdue University and supported in part by NSF grant CCF-0830476.

1 Introduction and Overview

Distributed approximation algorithms trade-off optimality of the solution for the amount of resources (messages, time etc.) consumed by the distributed algorithm. Besides a fundamental theoretical interest in understanding the algorithmic complexity of distributed approximation, there also is a practical motivation in studying distributed approximation algorithms. Emerging networking technologies such as ad hoc wireless sensor networks and peer-to-peer (P2P) networks operate under inherent resource constraints, e.g., energy (in sensor networks), bandwidth (in P2P networks) etc. A distributed algorithm which exchanges a large number of messages and takes a lot of time can consume a relatively large amount of resources, and is not very suitable in a resource-constrained network. Further the topology of these networks can change dynamically, e.g., due to the mobility of the nodes in wireless sensor networks, or the arrivals/departures of nodes in P2P networks. Communication cost and running time are especially crucial in such a dynamic setting.

The above motivations makes it critical to design efficient distributed algorithms for various network optimization problems that have low communication (message) and time complexity even if this comes at the cost of a reduced quality of the solution. (For example, there is not much point in having an optimal algorithm if it takes too much time, since the topology could have changed by that time.) For this reason, in the distributed context, such algorithms are motivated even for network optimization problems that are not NP-hard, e.g., minimum spanning tree, shortest paths etc. There is a large body of work on distributed approximation algorithms for classical graph optimization problems such as minimum spanning tree, shortest path, minimum edge-coloring, minimum dominating set, minimum vertex cover, maximum matching, and positive linear programs [15, 19, 21, 22, 25, 27, 29, 32, 42, 12, 34, 5, 26]. We refer to the surveys by Elkin [13] and Dubhashi et al. [11] that summarize many results regarding efficient distributed approximation algorithms and the hardness of distributed approximation for various classical optimization problems.

This paper is concerned with design and analysis of efficient distributed approximation algorithms for some important network optimization problems including the minimum Steiner tree problem, the shortest paths problem and their generalizations. These are fundamental problems in distributed computing and are widely used primitives in distributed communication networks. We present a uniform approach based on probabilistic tree embeddings to design efficient distributed approximation algorithms for the generalized Steiner forest problem (a generalization of the minimum Steiner tree problem), the shortest paths problem, and the optimum communication spanning tree problem. These problems are formally defined and described in detail in Section 3.

Developing uniform approaches to distributed approximation algorithm design is an important goal and leads to powerful tools for designing algorithms for various problems. Such approaches have been developed in the theory of approximation algorithms — LP-based approaches (primal-dual, randomized rounding) and metric embeddings — that yield (centralized) approximation algorithms for many problems [41]. In general, it is not clear how such paradigms can be used efficiently in a distributed setting. While LP-based techniques have been successfully used to design distributed approximation algorithms for minimum dominating set, minimum vertex cover, maximum matching (e.g., see [13, 11]), no such results are known for the problems that are addressed here.

When dealing with a certain class of metric optimization problems, such as those addressed here, a standard approach to obtain approximation algorithms is to use low-distortion *metric embeddings*. First, the given metric space is embedded into a host space with a simpler structure (e.g., an ℓ_p-metric or a tree metric). By solving the optimization problem on the host metric, approximately or exactly, one gets an approximation algorithm for the problem on the original metric. The approximation ratio suffers from the *distortion* of the embedding. While using embeddings to obtain approximation algorithms is a standard approach in a non-distributed context, to the best of our knowledge, the technique has not been used in distributed approximation.

In our approach we use a probabilistic tree embedding due to Fakcharoenphol, Rao, and Talwar [16]

(referred to as FRT embedding), where a given metric is approximately embedded over a distribution of tree metrics. We show that the FRT embedding can be used to design fast distributed (expected) $O(\log n)$-approximation algorithms for a variety of problems. Probabilistic tree embeddings were introduced by Bartal [3], who showed that any metric can be approximated by a distribution over dominating tree metrics with distortion $O(\log^2 n)$; that is, for any two points in the metric space, the expected distance between the two points in the tree is at most $O(\log^2 n)$ times their distance in the original metric. The bound on distortion was improved to $O(\log n \log\log n)$ in [4, 7] and to $O(\log n)$ in the FRT embedding. Using the FRT embedding, it is possible to obtain centralized polylogarithmic approximation algorithms for many optimization problems such as generalized Steiner forest problem, metric labeling, buy-at-bulk network design, minimum cost communication network problem, group Steiner tree problem, etc. (see [16] and references therein for a detailed list). The stretch bound of the FRT embedding is existentially tight: there are metric spaces for which any probabilistic tree embedding has distortion $\Omega(\log n)$.

The FRT embedding algorithm of [16] is centralized. We show how to efficiently compute an (implicit) FRT embedding in a distributed manner and how to use the embedding to obtain distributed algorithms for the generalized Steiner forest problem, the minimum routing cost spanning tree problem, and the k-source shortest paths problem with expected approximation ratio $O(\log n)$. Our algorithms are the first known distributed approximation algorithms for the generalized Steiner forest and the minimum routing cost spanning tree problems. Our algorithm for k-source shortest paths is significantly more efficient than previous algorithms at the cost of giving a logarithmic approximation.

At the heart of our approach is the construction of a distributed data structure called least elements (LE) lists that was first described by Cohen in [8] in the context of designing Monte-Carlo algorithms for transitive closure and reachability. It turns out that LE-lists exactly capture FRT embedding in an implicit fashion and our approach exploits this somewhat surprising connection between LE-lists and FRT embedding to design a fast distributed algorithm to compute an FRT embedding. For arbitrary weighted graphs, we give a distributed algorithm for computing LE-lists, assuming a random order on the nodes. Since LE-lists are useful in various contexts [8, 9], this result can have other applications as well.

The rest of the paper is organized as follows. Distributed computing model, notations and definitions are given in Section 2. In Section 3, we provide formal problem statements and our results. The distributed constructions of LE-list and FRT tree embedding are given in Section 4 and 5, respectively. Using the distributed FRT tree embedding, we present distributed approximation algorithms for the generalized Steiner forest problem, the minimum routing cost spanning tree problem, and the k-source shortest paths problem in Section 6. We conclude in Section 7.

2 Preliminaries

2.1 Distributed Computing Model

The network is modeled as a connected undirected weighted graph $G = (V, E, w)$; V is the set of the n nodes (vertices), E is the set of m communication links (edges), and $w(e)$ is the weight of edge $e \in E$; the edge weights are assumed to be non-negative real numbers. The weight of an edge can also denote some cost associated with the edge. Each node hosts a processor. The nodes communicate with each other by exchanging messages. We assume that each node has a unique identity number. At the beginning of computation, each node knows its own identity number and the weights of its adjacent edges. Thus, a node initially has only *local* knowledge limited to itself and its neighbors. At the end of the distributed computation, each node knows the set of the adjacent edges that belong to the solution. For example, in the MST problem, on termination, each node must know which of its incident edges belong to the MST.

We assume that the communication is synchronous and occurs in discrete time steps (also known as "rounds"). We use the widely-used $\mathcal{CONGEST}(\log n)$ model [37, 33] for communication, where a node v

can send an arbitrary message of size at most $O(\log n)$ through an edge in each time step. (Notice that if unbounded-size messages are allowed through an edge in one time step, then the problems considered here can be trivially solved in $O(D)$ time by collecting information to one node v from all other nodes, solving the problem locally by v, and then broadcasting the results to the other nodes [37].) The weight of an edge is assumed to be bounded by a polynomial of n so that the weight of a single edge can be communicated in one time step. As is standard, we also assume that local computation within a node is free as long as computation time is polynomial in n. Our focus is on the time complexity (total number of rounds) and the message complexity (the total number of messages exchanged).

2.2 General Notations and Definitions

Throughout the paper, we use the following definitions and notations concerning an undirected weighted graph $G = (V, E, w)$ with $|V| = n$ nodes, $|E| = m$ edges, and non-negative edge-weights $w(e)$ for all edges $e \in E$. In the rest of the paper, we use the terms "hop-length", "unweighted length" or simply "length" interchangeably to denote the number of edges in a path and "weighted distance" or simply "distance" to denote the sum of the weights of the edges in a path.
$Q(u, v)$ — is a path from node u to node v.
$|Q(u, v)|$ or simply $|Q|$ — is the number of edges in path $Q(u, v)$. We call $|Q|$ the *length of the path* Q.
$w(Q(u, v))$ or $w(Q)$ — the *weight of the path* Q, is the sum of the edge weights in Q, i.e., $w(Q) = \sum_{(u,v)\in Q} w(u, v)$.
$P(u, v)$ — is a *shortest path* from u to v, a path with minimum weight.
$d(u, v)$ — is the (weighted) *distance* between u and v, defined by $d(u, v) = w(P(u, v))$.
$l(u, v)$ — is the number of edges in the minimum-length shortest path from u to v. If there are more than one shortest path from u to v, $l(u, v)$ is the number of edges in the shortest path having the least number of edges, i.e, $l(u, v) = \min\{|P(u, v)| \mid P(u, v)$ is a shortest path from u to $v\}$.
D — denotes the *diameter* of G; that is $D = \max_{u,v} \min_Q |Q(u, v)|$.
Δ — denotes the *weighted diameter*; that is $\Delta = \max_{u,v} w(P(u, v))$.
$\Gamma_\rho(v)$ — the *ρ-neighborhood of a node v* — the set of the nodes that are within weighted distance ρ from v, i.e., $\Gamma_\rho(v) = \{u \mid d(u, v) \leq \rho\}$.

Definition 2.1 ***Shortest Path Diameter (SPD) [22].*** *The SPD is denoted by $S(G, w)$ (or S for short) and defined as $S = \max_{u,v\in V} l(u, v)$.*

Shortest path diameter plays an important role in the performance measure of our algorithms. Note that $1 \leq D \leq S \leq n$ for every connected graph and $S = D$ for unweighted graphs. Informally, S can be thought of as the weighted counterpart of the diameter D.

Definition 2.2 ***Embedding.***

Definition 2.3 ***Tree Embedding.***

We use the term "with high probability" (WHP) to mean with probability at least $1 - 1/n^{\Omega(1)}$.

3 Our Results and Related Work

3.1 Problems Addressed and Results

In this section, we formally define the problems we addressed in this paper and discuss the results we obtained. Table 1 summarizes the number of rounds, the number of messages and the approximation ratio for each of the problems.

Table 1: Summary of the results

Problem	Approx.	Weighted Graph		Unweighted Graph	
		Rounds	Messages	Rounds	Messages
LE-list	–	$O(S \log n)$	$O(S\|E\| \log n)$	$O(D)$	$O(\|E\| \min(D, \log n))$
Leader election	–	$O(D)$	$O(\|E\| \min(D, \log n))$	$O(D)$	$O(\|E\| \min(D, \log n))$
FRT tree	$O(\log n)$	$O(S \log n)$	$O(S\|E\| \log n)$	$O(D)$	$O(\|E\| \min(D, \log n))$
Gen. Steiner forest	$O(\log n)$	$O(Sk \log^2 n)$	$O(S\|E\| \log n)$	$O(kD \log^2 n)$	$O(D\|E\| \log n)$
Min. rt. cost tree	$O(\log n)$	$O(S \log^2 n)$	$O(S\|E\| \log n)$	$O(D \log^2 n)$	$O(D\|E\| \log n)$
k-source SP	$O(\log n)$	$O(Sk \log n)$	$O((S\|E\| + kn) \log n)$	$O(kD \log n)$	$O((\|E\| + kn) \log n)$

Distributed Computation of LE-Lists:
The first part of the paper gives an efficient distributed algorithm for computing least element (LE) lists of all nodes, a key procedure of our unified approach. Assuming a global order on the nodes of a network, the LE-list of a node stores the smallest node (with respect to the given order) within every distance d [8]. Assume that each v is assigned a unique rank $\mathbb{R}(v)$. The *least element* in $\Gamma_\rho(v)$, denoted by $L_\rho(v)$, is a node $u \in \Gamma_\rho(v)$ such that for all $u' \in \Gamma_\rho(v)$ and $u' \neq u$, $\mathbb{R}(u) < \mathbb{R}(u')$. For every node $v \in V$, we want to compute the least element in $\Gamma_\rho(v)$ for every distance $\rho \in [0, \Delta]$. These least elements are maintained as a list of ordered pairs, called the *least-element list (LE-list)*. The LE-list of v is the set of all nodes u so that u has the lowest rank among nodes within distance $d(v, u)$ from v, as defined below.

Definition 3.1 ***LE-list.*** *The LE-list of a node $v \in V$, $\mathbb{L}(v) = \{\langle u, \rho\rangle \mid \rho = d(v, u)$ and $u = L_\rho(v)\}$.*

Several properties follow from the definition of LE-lists. Let $\langle u_i, \rho_i\rangle$ be the i^{th} element in the sorted order of the elements of $\mathbb{L}(v)$ in increasing order of ρ, i.e., $\rho_i < \rho_{i+1}$ for $1 \leq i < |\mathbb{L}(v)|$. We have (a) $L_\rho(v) = u_i$ for any $\rho \in [\rho_i, \rho_{i+1})$, for $1 \leq i \leq |\mathbb{L}(v)|$ assuming $\rho_{|\mathbb{L}(v)|+1} = \Delta + \epsilon$ with any $\epsilon > 0$, (b) $\mathbb{R}(u_i) > \mathbb{R}(u_{i+1})$ for $1 \leq i < |\mathbb{L}(v)|$, and (c) $u_{|\mathbb{L}(v)|}$ is the least element in V. Note that $\mathbb{L}(v)$ might contain $\langle v, 0\rangle$ only (if v has the lowest rank among all nodes).

Choosing the ranks of the nodes according to a uniform random order, we give a distributed algorithm, called LE-Dist algorithm, for computing LE-lists. In weighted graphs, our algorithm terminates in $O(S \log n)$ rounds with a message complexity $O(S|E| \log n)$ where S is the shortest path diameter. It is easy to see that it can take up to $\Omega(S)$ time in some graphs (e.g., unweighted graphs, where $S = D$) and hence the above running time is existentially optimal up to a polylogarithmic factor. In unweighted graphs, our algorithm terminates in $O(D)$ rounds and uses $O(|E| \min(D, \log n))$ messages WHP. A consequence of our distributed LE-lists algorithm is an improved leader election algorithm in arbitrary (synchronous) networks which improves over the previous best known result due to Peleg [36]. Peleg's algorithm (which is a deterministic algorithm) takes $O(D)$ time and $O(D|E|)$ messages, while our algorithm takes $O(D)$ time (deterministically) and $O(|E| \min(D, \log n))$ messages WHP. Peleg [36] raised an important open question of whether there exists an algorithm that achieves both optimal message complexity $\Omega(|E| + n \log n)$ and optimal time complexity $O(D)$ and our result makes progress toward this direction.

Distributed Approximation Algorithms:
Using the distributed LE-lists, we construct distributed FRT tree embedding with expected stretch $O(\log n)$. The construction of distributed FRT tree embedding takes $(S \log n)$ rounds and $(|E|S \log n)$ messages. Using this tree embedding, we obtain distributed approximation algorithms for the following problems.

1. *Generalized Steiner forest (GSF) problem:* The GSF problem is a generalization of the minimum Steiner tree problem and is an important problem in the theory of approximation algorithms [41]. Given a weighted graph $G = (V, E, w)$, and a collection of k disjoint subsets (groups) of V: $V_1, V_2, \ldots, V_k$, the GSF problem is to find a minimum weight subgraph in which each pair of nodes belonging to the same group V_j is connected. We present a distributed algorithm that takes $O(S\,k \log^2 n)$ time and $O(S\,|E| \log n)$ messages and computes an (expected) $O(\log n)$-approximate Steiner subgraph.

The parameter S can be shown to capture the hardness of distributed approximation quite precisely. Using the hardness results of Elkin [15], one can show that there exists a family of n-node graphs where $\Omega(S)$ time is needed by any distributed approximation algorithm to approximate the MST within an H-factor, for any $H \in [1, O(\log n)]$ [22]. Since the MST problem is a special case of the GSF problem, the above bound also applies to the GSF problem. Thus for a small k, with $k = O(\log^c n)$ for some constant c, our algorithm is existentially time-optimal up to a polylogarithmic factor. The existential optimality of our algorithm is with respect to S instead of n as in the case of Awerbuch's distributed MST algorithm[2], for example. S can be much smaller than $\sqrt{n}$; e.g., in networks where edge weights are chosen uniformly and independently from an arbitrary distribution, $S = O(D + \log n)$ WHP [22]. Note that $\tilde{\Omega}(\sqrt{n})$ is a lower bound on the time needed to compute exact MST or Steiner tree [38, 22].

2. *Minimum routing cost spanning tree:* The minimum routing cost spanning tree problem is a special case of the optimum cost communication tree problem. Given a weighted graph G and a collection of R of pairs of nodes, the optimum cost communication tree problem is to find a tree T in G so that the sum of the distances in T of the pairs in R is minimized. When R consists of all pairs of the nodes, this problem is the minimum routing cost spanning tree problem; in which case, T is a spanning tree in G. Finding a spanning tree with minimum routing cost is NP-hard for general weighted graphs [44, 43]. We present a distributed (expected) $O(\log n)$-approximation algorithm that takes $O(S \log^2 n)$ time and $O(S\,|E| \log n)$ messages.

3. *k-source shortest paths problem:* Given a weighted graph $G = (V, E, w)$ and a subset $K \subseteq V$ of k nodes, the goal is to compute shortest paths between all pairs of nodes in $V \times K$. In the distributed context, the problem is to construct routing tables in the nodes such that any node can route to any source node or vice versa. We give a distributed algorithm that computes (expected) $O(\log n)$-approximate k-source shortest paths in $O(kD \log n)$ time using $O(|E|(\min[D, \log n]) + kn \log n)$ messages in an unweighted graph and in $O(kS \log n)$ time using $O(|E|S \log n + kn \log n)$ messages in a weighted graph.

3.2 Related Work and Comparison

The problems studied here have a rich history and we confine ourselves to work most relevant to this paper.

LE-Lists: LE-lists were studied by Cohen [8] in the context of designing fast Monte-Carlo algorithms for neighborhood size estimation and have since been shown to be useful in other applications as well (e.g., [9]). Cohen [8] presented a centralized strategy for computing LE-lists that incurs $O(|E| \log n)$ memory operations. The strategy in [8] requires nodes to spread their rank values sequentially in ascending order. This is not trivial to emulate in a distributed setting. Cohen and Kaplan [9] compute LE-list in a distributed setting. However, their protocol still assumes sequential spreading of rank values, and only their order is determined in a distributed manner.

GSF: The GSF problem is a generalization of the minimum Steiner tree problem which in turn is a generalization of the minimum spanning tree (MST) problem, both of which are fundamental problems in distributed computing and communication networks. There is a long line of research on time-efficient distributed algorithms for the (exact) MST problem (see e.g. [17, 2, 30, 14]). The best known algorithms take $O(D + \sqrt{n} \log^* n)$ time [30, 14]. In [22], Khan and Pandurangan presented a fast distributed approximation algorithm that constructs an $O(\log n)$-approximate minimum spanning tree. The algorithm of [22] can be easily modified to yield an $O(\log n)$ approximation to the minimum Steiner tree problem with the same time

bound. When applied to the GSF problem, the technique however only yields an $O(k \log n)$-approximation, where k is the number of disjoint sets. The question of getting an efficient distributed $O(\log n)$-approximation was left open, which is a motivation for the new approach in this paper.

For the minimum Steiner tree problem ($k = 1$), there is a well-known centralized 2-approximation algorithm based on computing the MST on the node set V_1 [41]. Using this fact, an $O(n \log n)$-time 2-approximate distributed algorithm was designed in [6] based on the classical distributed MST algorithm due to Gallager et al [17]. This algorithm is not time optimal, and the approach cannot be generalized to the GSF problem. The work of [31] gives distributed approximation algorithms based on the primal-dual method for some problems including the GSF problem. This algorithm is a distributed implementation of the corresponding centralized 2-approximation primal dual algorithm and takes $O(nD)$ rounds and $O(n^2 D \log D)$ messages. These bounds are not directly comparable to our bounds for the GSF problem for two reasons. Firstly, our bounds are with respect to S, k, and $|E|$ and not n and is better when S and k are small compared to n. Secondly and more importantly, the algorithm of [31] makes an assumption (Assumption 1 in [31]) regarding certain global information being pre-known at the beginning of the distributed algorithm and thus is different from the standard model assumed in this paper (i.e., nodes start with only local information).

Shortest paths: The single-source shortest paths ($k = 1$) and all-pairs shortest paths ($k = n$) are special cases of the k-source shortest path problem that have been very well studied (e.g., see [12] and the references therein). Distributed exact algorithms for shortest paths include the classical Bellman-Ford algorithm and Dijkstra algorithm [37, 33]. The distributed complexity of the 1-source problem is well-understood. In unweighted graphs, this problem is equivalent to computing the breadth-first tree (BFT) rooted at the source, and the best-possible time and communication complexities (in the synchronous setting) are $O(D)$ and $O(|E|)$, respectively, which are achievable [37]. Note that any protocol for the 1-source case can be converted into a protocol for the k-source problem with time complexity $O(kD)$ and message complexity $O(k|E|)$. For weighted graphs, the time and communication complexities for the 1-source case are $O(S)$ and $(S|E|)$, respectively, using the Bellman-Ford algorithm [33]. For all-pairs shortest paths, there are many algorithms proposed in the literature [40], which essentially take $O(Sn)$ time and $O(Sn|E|)$ messages for weighted graphs. For unweighted graphs, there is an algorithm with $O(n^2 \log n)$ message complexity but its time complexity can be as high as $O(n^2 D \log n)$ [1].

For the general k-source case, Elkin [12] gave a distributed algorithm that computes a path that is $(1 + \epsilon)$ times the shortest path plus an additive constant. On unweighted graphs, the algorithm runs in $O(kD+n^{1+\delta/2})$ time and uses $O(|E|n^{\rho} + kn^{1+\delta})$ messages (in the synchronous model) where ϵ, ρ, and δ are arbitrarily small positive constants (note that this algorithm improves message complexity over BFT-based algorithm mentioned above). On weighted graphs, the algorithm runs in $O(w_{max}n^{1+\delta/2}+kD)$ time using $O(|E|n^{\rho}+kn^{1+\delta})$ messages and the additive error depends on w_{max}, the ratio between the largest and smallest weight in the network. Since the algorithm's time complexity depends on the edge weights, its running time can be quite large. Our algorithm improves on both the time and messages complexities of Elkin's algorithm for unweighted graphs. For weighted graphs, our result has a substantially better time complexity, with a somewhat weaker message complexity. This improvement however comes at the cost of an $O(\log n)$-factor in the quality of the solution. Our algorithm also substantially improves over the communication complexity of exact algorithms. For the all-pairs shortest paths problem, the classical distributed Bellman-Ford algorithm, which computes an exact solution, takes $O(nS)$ time and $O(nS|E|)$ messages while our algorithm takes $O(nS \log n)$ time and $O(|E|S \log n + n^2 \log n)$ messages — the message complexity is better by a factor of $O(n)$, while the time complexity is larger by only a logarithmic factor.

Related lower bounds: The recent work of [10] gives non-trivial lower bounds on the time complexity of distributed approximation for several problems including MST, GSF, minimum routing cost spanning tree, shortest paths, SPT, minimum cut etc. It shows that $\Omega(\sqrt{n/\log n})$ time is required for approximating the above problems for *any* given approximation factor. In other words the above lower bound holds seamlessly

for exact algorithms also as well as for any H-approximation algorithm (where H is the approximation ratio). This is done by showing that there exists a graph where any distributed algorithm (randomized or deterministic) will take this amount of time. Our $O(\log n)$-approximation algorithm does not violate this lower bound, as the running time is in terms of S. In particular, for the graph used to prove the lower bound of [10], $S = \Omega(\sqrt{n/\log n})$.

4 LE-Dist: Distributed Algorithm to Compute the LE-Lists

We present a distributed algorithm called *LE-Dist* for computing the LE-lists of all nodes. The nodes choose their ranks randomly. The basic strategy is to let each node flood its rank to the network. A node v needs to forward a rank message origniated by w to its neighbors only if w is the lowest ranking node within distance $d(v, w)$ from v. All nodes execute the algorithm simultaneously in phases, where a phase consists of several communication rounds. Initially, each node only knows its own rank and distance (0). In each phase, a node exchanges rank information with its neighbors. At the end of each phase, the node updates its knowledge of ranks in the network based on the messages it receives from neighbors. These changes are forwarded to neighbors in the next phase. It is easy to see that after phase k, a node has rank information regarding all nodes within hop-count k. Thus, after S phases (S is the shortest path diameter), every node has full rank information and can compute $\mathbb{L}(v)$. Note that, in a weighted graph, at the time when node v gets w's rank, v may not have information about all the nodes within distance $d(v, w)$. Therefore w's rank message may get forwarded by v, and later, v may receive a rank message originated by a lower ranking node within distance $d(v, w)$. This may result in having extraneous messages being forwarded, and also in extra delays, since only one message may be sent over each link per time step. The challenge is to keep the communication and completion time bounded. Subsequent sections address the analysis of time and message complexity.

More specifically, a node v locally maintains a data structure le_v which stores v's current view of $\mathbb{L}(v)$. Initially, it contains the LE-list of $\{v\}$, the node itself. At the beginning of round k, it stores the LE-list for v of all nodes at hop-distance $k-1$ from v. For convenience, each entry in le_v contains, in addition to the pair $\langle u, R(u)\rangle$, the following information:

- ρ, the minimal distance that u's rank message traveled to reach v;
- u', the last node, a neighbor of v, on the minimal-distance path from u to v;
- a new/old flag.

The details for the LE-Dist algorithm are given by Algorithm 1. In an initialization step, each node v chooses its ranks $p(v)$ uniformly at random from $\{1, \ldots, n^c\}$ so that all nodes v choose different $p(v)$ WHP. Initially, le_v contains the single entry $\langle v, \mathbb{R}(v), 0, v, new\rangle$. The algorithm works correctly for any arbitrary ranking of the nodes; for example, each node can use its ID as its rank. However, as we will see later, by choosing uniform random ranking, we can have improved time and message complexity WHP. After the initialization, the algorithm runs in phases. In phase k, node v exchanges all items marked *new* in its list with its neighbors, and incorporates their rank messages into le_v. At the end of a phase, a node v proceeds to the next phase as long as it has not received a control message notifying it of termination. The control messages needed for termination are omitted from the pseudo-code of the algorithm. We discuss termination detection in the following section.

4.1 Detecting Termination

As we already argued above, after phase S, every node v has rank information from the entire network, and le_v does not change any more. Therefore, no more rank messages are generated after phase S ends. Since

Algorithm 1 LE-Dist Algorithm (at node v)

Initialization:

choose $p(v)$ uniformly at random from $\{1, \ldots, n^c\}$
rank $\mathbb{R}(v) := (p(v), \mathrm{ID}(v))$ *// use lexicographical order when comparing ranks*
insert $\langle v, \mathbb{R}(v), 0, v, new\rangle$ into le_v

Phases: *// execute phase code until termination message received*

for phase $k := 1, 2, \ldots$ **do**
  for each *new* item $\langle u, \mathbb{R}(u), \rho, u', new\rangle$ in le_v **do**
    send *rank* message $\langle u, \mathbb{R}(u), \rho\rangle$ to all neighbors except u'
  send *end-of-phase* control message to all neighbors
  mark the *new* items in le_v as *old*
  wait until *end-of-phase* message received from all neighbors
  for all neighbors u' **do**
    for all *rank* message $\langle u, \mathbb{R}(u), \rho\rangle$ received from neighbor u' **do**
      $\rho' := \rho + w(u', v)$
      if there is a $\langle u_i, \mathbb{R}(u_i), \rho_i, u_i', *\rangle \in \mathsf{le}_v$ with $\rho_i \leq \rho'$ and $\mathbb{R}(u_i) \leq \mathbb{R}(u)$ **then**
        // here "$*$" stands for "any value" of the corresponding field
        delete (ignore) *rank* message $\langle u, \mathbb{R}(u), \rho\rangle$
      else
        for all $\langle u_i, \mathbb{R}(u_i), \rho_i, u_i', *\rangle \in \mathsf{le}_v$ **do**
          if $\rho_i \geq \rho'$ and $\mathbb{R}(u_i) \geq \mathbb{R}(u)$ **then**
            delete $\langle u_i, \mathbb{R}(u_i), \rho_i, u_i', *\rangle \in \mathsf{le}_v$ from le_v
        insert $\langle u, \mathbb{R}(u), \rho', u', new\rangle$ into le_v

S is not known to the nodes initially, relying on the *rank* messages only, a node cannot determine whether there are any outstanding *rank* messages in the network and does not know when to terminate.

To detect termination, we introduce an additional mechanism. If v ignores a rank message it receives from u', it *echo*es it back. Otherwise, it waits for echoes from all neighbors other than u', and then echoes it back. Note that, if u' is v's only neighbor, then the waiting vacuously completes immediately, and v echoes back the message. Every rank message is eventually echoed back, either when it cannot travel farther or when it is ignored. At the latest, after the end of Phase S, any remaining rank message is echoed. It may take up to S additional phases for echoes to reach their origins. After the end of Phase $2S$, every node has received echoes for its own rank message from all of its neighbors. Still, nodes do not know when all of their peers have received their echoes.

Consider a node v whose rank is not the lowest. Its rank message is ignored at least by one other node (the lowest ranking). A node that ignores v's rank message forwards at the next phase information about a lower rank. That lower rank, or an even lower rank message that collides with it, reaches v at the latest one phase after the echo message reaches v. Therefore, a node w that receives its own echoes from all of its neighbors at phase r, and does not receive any lower rank message at phase $r + 1$, knows that it is the lowest ranking node in the network. Node w then floods the network with a *leader* message. A node that receives a leader message stores the last (parent) hop, and forwards the leader message to all other neighbors. This builds a BFS tree rooted at the lowest ranking node w.

After having collected *echo* message for its rank message from all of its neighbors, a *leader* message, and *done* messages from every child neighbor, a node sends a *done* message to the parent neighbor. The leader waits for *done* messages from all of its children, and then floods the network with a *termination* signal.

More precisely, in a phase, after having exchanged rank messages as described above, a node v exchanges messages with neighbors as follows.

Echo:

a) When v ignores a *rank* message $\langle u, \mathbb{R}(u), \rho\rangle$ received from a neighbor u' (see Algorithm 1, Line 14), it sends an *echo* message, for this *rank* message, back to u'.

b) If v forwards the *rank* message $\langle u, \mathbb{R}(u), \rho\rangle$ (Algorithm 1, Line 5-6, 19), it waits until it receives the *echo* messages for this *rank* message from its neighbors (except u', whom v did not forward the *rank* message to). Once v receives these *echo* messages, v sends an *echo* message to u' if it did not do so already. If v does not have any neighbor other than u', i.e., there is no neighbor to forward $\langle u, \mathbb{R}(u), \rho\rangle$ to, v sends the *echo* immediately to u'.

c) If an item $\langle u_i, \mathbb{R}(u_i), \rho_i, u'_i, *\rangle$ gets deleted from le_v (Algorithm 1, Line 18), and if v did not send the echo to u'_i for this *rank* message yet, v sends the echo to u'_i. (Here "$*$" stands for "any value" of the corresponding field.)

One full phase after having received *echo* messages for its own *rank* message from all of its neighbors, a node v starts the following termination protocol.

Termination:

a) If there are no lower rank items in le_v, then v knows it is the leader and sends *leader* message to all its neighbors. It waits for *done* messages back from them, and then sends a *termination* signal to all of them.

b) If v receives a leader message from u, and $parent_v$ is empty, then v sets $parent_v$ to u and forwards the leader message to every neighbor except u.

c) After v receives *done* messages from its children (vacuously holds if $parent_v$ is the only neighbor), it sends a *done* message to $parent_v$.

4.2 Correctness

By applying the following lemma recursively, one can show that using the LE-lists as "routing tables", a node v can trace back a path (which is the weighted shortest path) to the lowest ranked node in $\Gamma_\rho(v)$ for any ρ.

Lemma 4.1 *At the end of the LE-Dist algorithm (Algorithm 1), if* $\langle u, \mathbb{R}(u), \rho, u', *\rangle \in \mathsf{le}_v$, *then* $\langle u, \mathbb{R}(u), \rho - w(u', v), u'', *\rangle \in \mathsf{le}_{u'}$ *for some* u'' *such that* $u'' \neq v$ *and* $(u', u'') \in E$.

Proof: $\langle u, \mathbb{R}(u), \rho, u', *\rangle$ can be inserted in le_v only after v receives *rank* message $\langle u, \mathbb{R}(u), \rho - w(u', v)\rangle$ from neighbor u', and at that time $\langle u, \mathbb{R}(u), \rho - w(u', v), u'', *\rangle$ is in $\mathsf{le}_{u'}$. Clearly $u'' \neq v$, because if u' receives this message from v, u' would not forward it to v. Later $\langle u, \mathbb{R}(u), \rho - w(u', v), u'', *\rangle$ can be deleted from le'_u only if u' receives another message $\langle u_1, \mathbb{R}(u_1), \rho_1\rangle$ from u'_1 with $\mathbb{R}(u_1) \leq \mathbb{R}(u)$ and $\rho_1 + w(u', u'_1) \leq \rho - w(u', v)$. If $u'_1 \neq v$, u' would forward this message to v leading to the deletion of $\langle u, \mathbb{R}(u), \rho, u', *\rangle$ from le_v, and if $u'_1 = v$, v would delete it before forwarding the message to u'. □

Lemma 4.2 *By the time a node* u *receives the echoes of the rank message originated by itself from all of its neighbors, the rank message* $\langle u, \mathbb{R}(u), \rho\rangle$ *has reached every node* v *for which* $\langle u, \mathbb{R}(u), \rho, *, *\rangle$ *is supposed to be in* le_v.

Proof: The *echo* for a *rank* message is generated only when the *rank* messages is not forwarded, and if it is forwarded (by v), v waits until the *echo* messages comes back from the neighbors to which v forwarded the message to. Thus, clearly, the *rank* messages (originated by u) complete their journey before u receives the echoes back. In only other case, v sends an echo early if an item $\langle u, \mathbb{R}(u), \rho, *, * \rangle$ is deleted from le_v. Consider any node v' such that the *rank* message $\langle u, \mathbb{R}(u), * \rangle$ travels from u to v' via v. Since $\langle u, \mathbb{R}(u), \rho, *, * \rangle$ is deleted from le_v, following Lemma 4.1, the item $\langle u, \mathbb{R}(u), \rho, *, * \rangle$ corresponding to the *rank* message $\langle u, \mathbb{R}(u), \rho \rangle$ cannot be in the "final" LE-list of v'. Thus, the lemma follows. □

Lemma 4.3 *If a node u receives the echoes of its own* rank *message from all of its neighbors by the end of phase k, then by the end of phase $k+1$ it has a lower ranked entry in le_v if there exists a lower ranked node in the network.*

Proof: Consider a node v whose rank is not the lowest. Let M be the *rank* message generated by v, and define rank of M, $\mathbb{R}(M) = \mathbb{R}(v)$. Message M is ignored at least by one other node (the lowest ranked node). A node that ignores M must forwards, in the next phase, another rank message M' s.t. $\mathbb{R}(M') < \mathbb{R}(M)$. Then either M' reaches v by the end of phase $k+1$, or M' is ignored (on its way to v) caused by another *rank* message M'', with $\mathbb{R}(M'') < \mathbb{R}(M')$, which in turn reaches v by the end of phase $k+1$ or is ignored due to another even lower ranked message, and so on. □

Lemma 4.4 *At the end of Algorithm 1, $\langle u, \mathbb{R}(u), \rho, u', * \rangle \in \mathsf{le}_v$ if and only if $\rho = d(v,u)$ and $u = L_\rho(v)$.*

Proof: When the algorithm terminates, every node v will have received echoes for its own rank message from all its neighbors. Therefore, by Lemma 4.2, for every node u, such that u in le_v, u's rank message has arrived at v. It remains to show that le_v is computed correctly from all such messages. This follows from lines 16–27 of the algorithm. Notice that a *rank* message, originated by $u = L_\rho(v)$, that follows the shortest path $P(u,v)$, meaning $\rho = d(v,u)$, cannot be ignored or deleted by an intermediate node in path $P(u,v)$ or by v itself. On the other hand, if it does not follow $P(u,v)$ (i.e., $\rho > d(v,u)$), its deletion is caused, if it is not deleted yet, by another *rank* message originated by u that follows $P(u,v)$. If $\rho = d(v,u)$ but $u \neq L_\rho(v)$, i.e., there exist $u_1 \in \Gamma_\rho(v)$ with $\mathbb{R}(u_1) < \mathbb{R}(u)$, the deletion is caused by the *rank* message originated by u_1 that follows $P(u_1, v)$. □

Theorem 4.5 *The LE-Dist algorithm correctly computes the LE-list of each node.*

4.3 Analysis for Weighted Graphs

In this section, we provide an analysis of the number of rounds and messages needed for completion. We will make use of the following notation in our proofs: $\mathsf{le}_v(k)$ denotes the LE-list le_v at the end of phase k.

Lemma 4.6 *$E[|\mathsf{le}_v(k)|] \leq \log n$, and simultaneously for all $v \in V$ and all phases k, $|\mathsf{le}_v(k)| \leq O(\log n)$ WHP.*

Proof: Let node set $H_k(v) = \{u \mid \exists Q(v,u) \text{ s.t. } |Q| \leq k\}$, and edge set $E_k(v) = \{e \mid \exists Q(v,u) \text{ s.t. } |Q| \leq k \wedge e \in Q\}$. The graph G restricted to the nodes in $H_k(v)$ and edges in $E_k(v)$ is denoted $G_k(v)$. Let $M_k(v)$ be a set of ordered pairs defined as $M_k(v) = \{\langle u, \rho \rangle \mid u \in H_k(v) \wedge \rho = \min_{|Q| \leq k} w(Q(v,u))\}$. We have $|M_k(v)| = |H_k(v)| \leq n$. Now, consider a sorted order of the elements in $M_k(v)$ in non-decreasing order of ρ, and let $\langle u_i, \rho_i \rangle$ be the i^{th} element in this sorted order, where $1 \leq i \leq |M_k(v)|$. It is easy to see that the LE-list of v on the restricted graph $G_k(v)$ is equal to the set $\mathbb{L}_k(v) = \{\langle u_i, \rho_i \rangle \mid \forall_{j<i} \left[\mathbb{R}(u_j) > \mathbb{R}(u_i)\right]\}$. The list constructed by the algorithm at the end of phase k at node v is the LE-list of v on $G_k(v)$. Hence, $\mathsf{le}_v(k)$ is $\mathbb{L}_k(v)$.

Let ξ_i be the event that $\langle u_i, \rho_i \rangle \in \mathbb{L}_k(v)$. Event ξ_i occurs if and only if $\mathbb{R}(u_i)$ is the lowest among $\mathbb{R}(u_1), \mathbb{R}(u_2), \ldots, \mathbb{R}(u_i)$. Since the ranks are chosen by the nodes independently at random, we have $\Pr\{\xi_i\} = \frac{1}{i}$, and thus,

$$E[|\mathbb{L}_k(v)|] = \sum_{i=1}^{|M_k(v)|} \frac{1}{i} \leq \log |M_k(v)| \leq \log n.$$

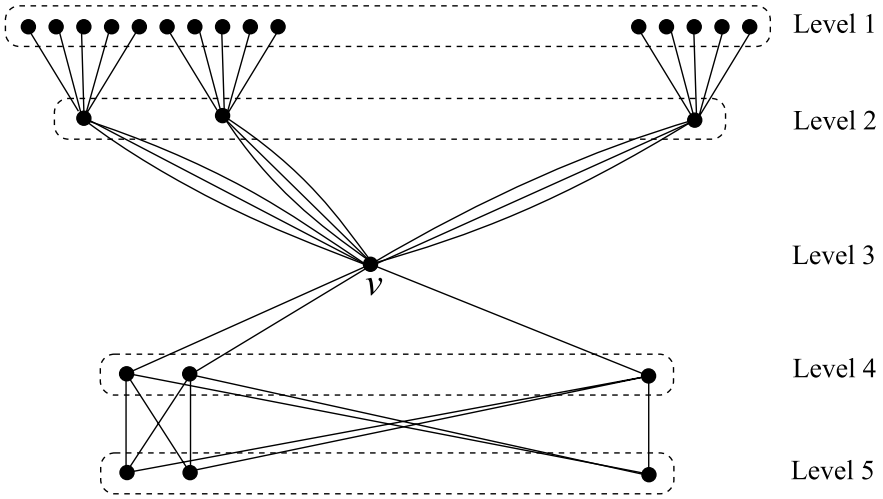

Figure 1: The graph used in the proof of Theorem 4.9.

Notice that the events ξ_is, $1 \leq i \leq |M_k(v)|$, are independent. By using the Chernoff bound, we can show that for a particular v, $|\mathbb{L}_k(v)| < 5 \log n$ with probability at least $1 - \frac{1}{n^3}$. Since $k < n$, by using the union bound, we have $|\mathbb{L}_k(v)| < 5 \log n$ for all v and k simultaneously with probability at least $1 - \frac{n^2}{n^3} \geq 1 - \frac{1}{n}$. □

Lemma 4.7 *In every phase k, a node v exchanges $O(\log n)$ messages with each of its neighbors WHP.*

Proof: Nodes exchange three types of messages during a phase: *rank*, *echo*, and termination-related (*leader, done, termination*). By Lemma 4.6, each node sends its neighbors $O(\log n)$ *rank* messages, and likewise, receives $O(\log n)$ *rank* messages from each neighbor.

We now bound the number of echo messages. During phase k, a node v echoes back to a neighbor u either for the *rank* messages received (and ignored) from u during the phase, and none or some of the *rank* messages that were previously received from u (and not echoed yet). The latter can be at most the number of items in $\mathsf{le}_v(k-1)$, the LE-list of v at the beginning of Phase k, because if a previously-received *rank* message is not in $\mathsf{le}_v(k-1)$ (i.e., ignored or deleted), its echo has already been sent in some previous phase. By Lemma 4.6, the number of *rank* messages arriving in phase k from u is $O(\log n)$ WHP, and by the same lemma, so is the size of $\mathsf{le}_v(k-1)$. Finally, in each phase a node may exchange with each neighbor at most one of each, *leader*, *done*, *termination*, messages. Hence, the lemma follows. □

Theorem 4.8 *The LE-Dist algorithm terminates in $O(S \log n)$ rounds using $O(|E|S \log n)$ messages WHP.*

Proof: Each phase takes $O(\log n)$ communication rounds WHP, by Lemma 4.7. When all nodes complete $S + 1$ phases, no new *rank* messages are generated. Within additional S phases, echo messages travel back to all nodes. Finally, constructing the termination tree by flooding a *leader* notification takes $O(D)$ phases, where D is the unweighted (hop) diameter. The resulting BFS tree has height D. Hence, a converge-cast of *done* messages and a broadcast of the *termination* signal take altogether $O(D)$ phases. In total, since $D \leq S$, the algorithm terminates in $O(S \log n)$ rounds and uses $O(|E|S \log n)$ messages. □

Tightness of Analysis: The analysis of the LE-Dist Algorithm is tight up to constant factors as stated in the following theorem.

Theorem 4.9 *For arbitrary constants $\epsilon, \delta > 0$ such that $3\epsilon + 2\delta < 1$, it is possible to construct a weighted n-node graph G with $S = \Theta(n^\delta)$ and $|E| = \Theta(n^{2-2\epsilon})$ such that the expected time and message complexities of the LE-Dist algorithm are $\Omega(\epsilon S \log n)$ and $\Omega(\epsilon |E| S \log n)$.*

Proof: For parameters N and S, the graph G is constructed as follows. G consists of 5 levels of nodes (Figure 1):

- Level 1: N nodes
- Level 2: $N^{1-\epsilon-2\delta}$ nodes. Every node in Level 2 is connected to $N^{\epsilon+2\delta}$ nodes in Level 1 by an edge of weight 1.
- Level 3: one node v that is connected to every node in Level 2 by S paths with the following properties: The S paths have hop lengths $1, \ldots, S$; the path P of hop length i has (weighted) length $2S - i + X_P$ where X_P is a small random number (absolute value $< 1/2$). The random variables X_P are independent and identically distributed for all $SN^{1-\epsilon-2\delta}$ paths. For simplicity, in figure 1, the S paths connecting v to a node in Level 2 are depicted as a collection of edges.
- Level 4: $N^{1-\epsilon}$ nodes, all connected to v (Level 3) by an edge of weight 1.
- Level 5: $N^{1-\epsilon}$ nodes, connected to Level 4 by a complete bipartite graph with edges of weight 1.

Note that the number of nodes (including the nodes on the paths connecting Levels 3 and 2) is $n = N + 2.5N^{1-\epsilon} + o(N^{1-\epsilon})$. When assigning ranks at random, the probability that there exists a node in levels 3-5 whose rank is among the N^ϵ smallest ranks is less than $2.5N^{1-\epsilon}N^\epsilon/(N + 3.5N^{1-\epsilon}) \leq 5/9$. Assume that we are in this case. These N^ϵ lowest ranks are scattered among the $N^{1-\epsilon-2\delta} \geq N^{2\epsilon}$ clusters at random. The probability of have more than 10 such ranks collide with another in the same cluster is less than $\binom{N^\epsilon}{10}\left(\frac{1}{N^\epsilon}\right)^{10} \leq \left(\frac{e}{10}\right)^{10}$. Again, assume this is the case.

After phase 2 of the LE-Dist algorithm, node v receives the rank information of the Level 1 nodes over the 1-hop paths connecting v to Level 2. Only one rank value per cluster (the smallest) may appear in the LE-list of v. Due to the independent choice of weights on each of the one-hop paths connecting v to Level 2, the LE-list of v after phase two contains $\Theta(\log(N^\epsilon)) = \Theta(\epsilon \log N)$ elements with at least constant probability. In phase 3, v has to sequentially send all these elements to the nodes of Level 4. In phase 3, v also receives new rank information from the Level 1 nodes. Because the 2-hop paths between v and Level 2 are all shorter than the one-hop paths, v has to discard all old information about the N^ϵ smallest rank values and add $\Theta(\epsilon \log N)$ new elements to its LE-list with constant probability. Similarly, node v has to update $\Theta(\epsilon \log N)$ of the values in its LE-list during S phases. All these $O(\epsilon S \log N)$ updates have to be propagated to the Level 4 and 5 nodes. This requires $\Theta(\epsilon S \log N)$ time and $\Theta(N^{2-2\epsilon}\epsilon S \log N) = \Theta(\epsilon |E| S \log n)$ messages (product of the required time and the number of edges in the bipartite graph between Levels 4 and 5). □

4.4 Analysis for Unweighted Graphs

In an unweighted graph, the LE-Dist algorithm can be simplified as described below. On unweighted graphs, the distance ρ traveled by a message is simply the number of edges on the path that the message followed. Initially, at phase $k = 1$, v forwards its own rank to all of its neighbors. By induction on k, we have that i) the messages arriving at v in phase k, have larger ρ values than that of all messages arriving in phases $k' < k$, ii) all messages arriving at v in phase k, have $\rho = k$. As a result, in phase $k + 1$, we only need to forward the message with the lowest-ranked originator among the messages received in phase k and we can ignore all other messages. Hence, a message is never delayed, it is either ignored or forwarded. Consequently, each phase boils down to a single round. We no longer need to use control messages to mark the end of a phase and we do not need to keep the new/old flag. For the message complexity, we therefore only need to count the number of *rank* messages. Note that the number of *echo* messages is exactly equal to the number of *rank* messages.

Theorem 4.10 *For unweighted graphs, LE-lists can be computed in $O(D)$ rounds (for any ranking of the nodes). If the ranks are chosen randomly, the message complexity is $O(|E| \cdot \min\{D, \log n\})$ WHP.*

Proof: Each phase takes one communication round (i.e., there is no delay). Hence the algorithm terminates in $O(S) = O(D)$ rounds.

As we observed, the messages that arrive at node v in phase k have a larger ρ value than that of a message that arrives in a phase $k' < k$. Thus an item in le_v never gets deleted once the item is inserted into le_v. Further, whenever v forwards a message, it inserts the corresponding item in le_v. Thus the total number of messages forwarded by v is bounded by $|\mathbb{L}(v)|$. Node v forwards each message through at most $\deg(v)$ adjacent edges, where $\deg(v)$ is the degree of v. Now, $\|\mathsf{le}_v\| = O(\log n)$ WHP (Lemma 4.6) and also $|\mathsf{le}_v| \leq D$ since in each round, at most one item is inserted in le_v. Thus, the message complexity is $\sum_{v \in V} \deg(v) \cdot O(|\mathsf{le}_v|) \leq \sum_{v \in V} \deg(v) \cdot O(\min\{D, \log n\}) = O(|E| \cdot \min\{D, \log n\})$. □

Leader Election: The LE-Dist algorithm for unweighted graphs is a leader election algorithm since the lowest ranked node can be elected as the leader and the all the rest know it (cf. Section 4.1). Hence the algorithm for unweighted graphs can be used as a synchronous, leader election algorithm for arbitrary graphs. By Theorem 4.10, the time complexity is $O(D)$ (deterministically) and the message complexity is $O(|E| \cdot \min\{D, \log n\})$ WHP. Note that for the sole purpose of leader election, it is sufficient that each node stores only the information about the highest rank among the ranks seen so far in each phase of the algorithm.

5 Probabilistic Tree Embedding

Next we show how to implement FRT embedding in a distributed setting. We refer to [16] for the original sequential algorithm for FRT embedding and its analysis. The β-lists below has been introduced to facilitate distribtued implementation. The sequential algorithm does not require such β-lists.

1. Construction of LE-lists: Construct the LE-lists of the nodes using the LE-Dist algorithm.

2. Construction of β-lists: Given a parameter $\beta \in [1, 2]$, let $\beta_i = 2^{i-1}\beta$ for $i = 0, 1, 2, \ldots, \delta = \lceil \lg \Delta \rceil + 1$. The β-list of a node v, denoted by $\beta(v)$, is a list of pairs $\langle \beta_i, u_i \rangle$ such that u_i is the least element in $\Gamma_{\beta_i}(v)$. The leader selects a real number β uniformly at random in $[1, 2]$ and broadcasts β to the other nodes using the BFS tree computed during the LE-lists computation. This needs $O(D)$ time and $O(n)$ messages. Once a node gets β, it can compute its β-list from its LE-list.

3. Tree Embedding: The β-lists of the nodes define a hierarchical clustering as follows: The top level cluster consists of all nodes in V and we define its level to be δ. Each level-i cluster $C_{i,j}$ is decomposed into level-$(i-1)$ clusters. We define u_i to be the level i cluster center for v if $\langle \beta_i, u_i \rangle \in \beta(v)$. All nodes in $C_{i,j}$ with the same level-$(i-1)$ cluster center form one level-$(i-1)$ cluster. Thus v itself is its level-0 cluster center, i.e., $u_0 = v$, and the center of the root cluster is the least element in V. Note that a particular node may be the center of several different clusters at the same level and at different levels. Also observe that each cluster at level i has diameter at most $2^i\beta$.

Given the hierarchical clustering, one can naturally define a tree as follows: each cluster corresponds to a node in the tree, and we have an edge from cluster $C_{i,j}$ to $C_{i-1,j'}$ iff $C_{i-1,j'}$ is a sub-cluster of $C_{i,j}$. The level-0 clusters that form the leaves of this tree are singleton clusters that correspond to the nodes of V. Thus each $v \in V$ forms a leaf node in the tree (the black nodes in Figure 2). The edge from a level i cluster to its parent has weight $2^i\beta$. It is easy to verify that for any edge in the tree, the graph distance between the corresponding cluster centers is no larger than twice the weight of the edge. Clusters have radii at most $2^{i-1}\beta$. In a distributed setting, we do not explicitly construct this FRT tree. Instead, the β-lists implicitly define the tree. We think of each cluster as being represented by its center.

As the time and message complexity of the above implementation is dominated by that of the distribtued construction of the LE-lists, we have the following theorem.

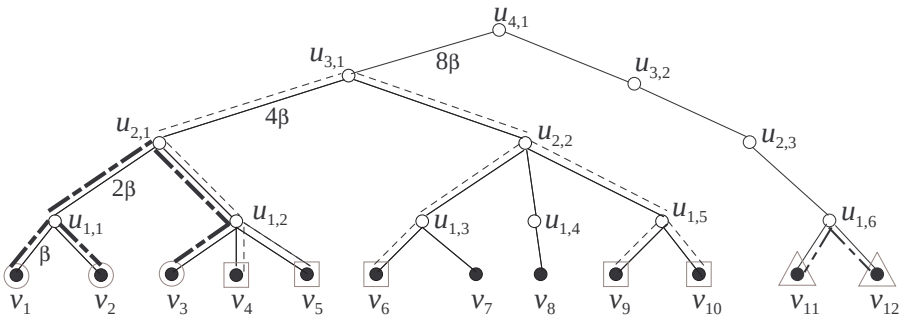

Figure 2: Construction of GSF in an FRT tree. $V = \{v_1, v_2, \ldots, v_{12}\}$, $\delta = 4, V_1 = \{v_1, v_2, v_3\}$, $V_2 = \{v_4, v_5, v_6, v_9, v_{10}\}$, $V_3 = \{v_{11}, v_{12}\}$. The solid black nodes are in V. The white nodes are internal nodes of an FRT tree. Nodes in V_1, V_2 and V_3 are marked by circles, squares, and triangles, respectively. The solid lines are the edges in an FRT tree, and the dotted lines are the edges in the Steiner forest.

Theorem 5.1 *Construction of distribtued FRT tree embedding takes $O(S \log n)$ rounds and $O(|E|S \log n)$ messages.*

Let $d_T(u, v)$ and $d_G(u, v)$ be the distances between nodes u and v in the FRT tree and the original graph G, respectively. In [16] it has been shown that for any $u, v \in V$, $d_T(u, v) \geq d_G(u, v)$ (d_T dominates d_G) and $E[d_T(u, v)] \leq O(\log n) d_G(u, v)$. Following Theorem 4 in [3], for any problem $\mathcal{P}$ where the cost is a linear combination of distances between vertices, if we solve $\mathcal{P}$ in FRT tree exactly, we can get an $O(\log n)$-approximation algorithm for $\mathcal{P}$ in G. In [16], the authors have discussed several problems such as the group Steiner tree problem, the minimum cost communication network problem, the buy-at-bulk network design problem, etc. for which better centralized approximation algorithm can be devised using this tree embedding. In the next section, we present distributed $O(\log n)$-approximation algorithms for the generalized Steiner forest problem, the minimum routing cost spanning tree problem and the k-source shortest path problem.

6 Distributed Approximation Algorithms

6.1 Generalized Steiner Forests

The tree embedding inspires a natural centralized algorithm for the GSF problem (defined in Section 1): Observe that the problem can be solved optimally on a tree by connecting each pair of nodes belonging to the same group V_j (for all $1 \leq j \leq k$) via the nearest common ancestors (Figure 2) and taking the union of all selected edges for all sets V_j, $1 \leq j \leq k$. This solution, applied to the tree resulting from an embedding, can be converted to a solution for the original graph by including, for each edge in the tree solution, the shortest path in the graph between the corresponding cluster centers (note that the resulting subgraph may not be forest). The bound on the expected distortion of the FRT embedding implies that in expectation, the cost of OPT on the tree is no larger than $O(\log n)$ times the optimal solution to the original instance. The fact that the distances in the tree are always larger than corresponding distances in the graph implies that the cost of the solution induced on the graph is no larger than the cost of the solution on the tree. Thus, this algorithm gives an $O(\log n)$-approximation to the problem in expectation. Next, we describe a distributed approximation algorithm based on this approach.

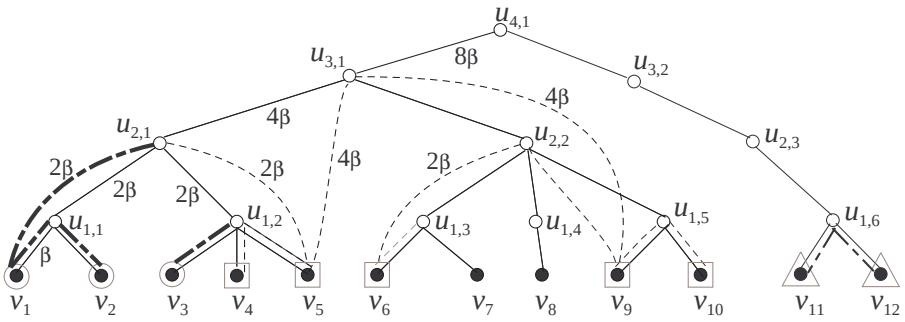

Figure 3: Distributed GST construction. The dotted lines are the paths that are included in the Steiner subgraph. The light-colored dotted lines, e.g., $P(v_6, u_{1,3})$, may or may not be included in the Steiner subgraph.

Distributed Algorithm

We assume that we have an implicit representation of the FRT tree; i.e., each node has its LE-list and β-list. At the end of the algorithm, each node knows which of its incident edges belong to the Steiner subgraph. We focus on finding an $O(\log n)$-approximate Steiner subgraph, not necessarily a forest. However, a Steiner forest can be constructed from the Steiner subgraph, with the same time and message complexities, by a breadth first search.

Before we detail the algorithm, we illustrate the main ideas with an example. In a distributed setting, the internal nodes (the white nodes in Figure 2) of an FRT tree are represented by the centers of the corresponding clusters. An edge of an FRT tree is replaced by the shortest path between the corresponding cluster centers. However, replacing an edge between two white nodes in an FRT tree is not straightforward: using LE-lists as routing tables (Lemma 4.1), we can find the shortest path from a black node to any of its ancestor white nodes but not between two white nodes. To resolve this problem, we pick an arbitrary black descendant of a white node and connect it to the higher level white node. For example, as shown in Figure 3, edge $(u_{1,2}, u_{2,1})$ of the FRT tree can be replaced with the shortest path $P(v_5, u_{2,1})$ in G.

For a group V_j, let $\ell(V_j)$ denote the level of the lowest common ancestor (LCA) of the nodes in V_j, in the tree; i.e., the smallest i such that all nodes in V_j belong to a single level i cluster. For a node v in group V_j, we define $\ell(v)$ to be $\ell(V_j)$ (recall that the groups are disjoint).

The algorithm consists of two parts: the first is a *discovery* part, where each node v computes $\ell(v)$, and the second is a *construction* part where the Steiner subgraph is constructed using this information.

6.1.1 Discovery Part: Finding Lowest Common Ancestor (LCA)

The following algorithm for finding lowest common ancestor can be of independent interest and be used as a subroutine of other distributed algorithms. We start by describing a simplified version of the discovery part of the algorithm, which we will then refine to avoid congestion. We describe the algorithm for a single group V_j; the k groups run the algorithm in parallel. The algorithm consists of $\delta = \lceil \log \Delta \rceil$ synchronized phases. The nodes begin $(i+1)^{st}$ phase after all nodes finish their i^{th} phase. (We describe later how to synchronize the phases.) Each node v computes $\ell(v)$ by sending messages to its level i ancestor u_i, for each i, and determining whether the cluster defined by its level i ancestor contains all of V_j, starting from the top down.

We next describe the i^{th} phase. Let $\bar{i} = \delta - i + 1$. Each node v that hasn't yet determined $\ell(v)$ sends a *find-lca* message to its level $\bar{i}$ cluster center $u_{\bar{i}}$, containing its own identifier, its group identifier, and its level $\bar{i}-1$ cluster center. The level $\bar{i}$ cluster center looks at all the messages it receives, and if the messages contain

at least two distinct level $(\bar{i}-1)$ cluster centers, it declares itself to be the least common ancestor and sends a YES reply to each node it received a message from. On the other hand, if all messages agree on the level $(\bar{i}-1)$ cluster center, the level $\bar{i}$ cluster center sends a NO message, communicating that the least common ancestor is lower down the tree. A node $v \in V_j$ stops as soon as it receives a YES message and sets $\ell(v)$ to $\bar{i}$.

Lemma 6.1 *At the end of δ phases, each node v in each group will determine the correct value of $\ell(v)$.*

Proof: The lemma would be immediate if instead of cluster center $u_{\bar{i}-1}$, we sent a unique identifier for the cluster to the parent cluster center. However, one node u can be the cluster center for several clusters at the same level. We next argue that from the point of view of this algorithm, the cluster centers are in fact unique identifiers.

Let C_{i^*} be the cluster corresponding to the true LCA with center u_{i^*}. Thus $V_j \subseteq C_{i^*}$, so that messages sent to centers u_i for $i > i^*$ all agree on their level $i-1$ cluster and hence on the cluster center. The tree construction is easily seen to have the property that all *children clusters* of a cluster C have distinct cluster centers. Thus the messages to the level i^* cluster center, which contain centers of at least two distinct clusters, must in fact contain at least two distinct level (i^*-1) cluster centers. Thus the response from u_{i^*} will in fact be YES and the nodes in V_j will stop with the correct value of $\ell(v)$. □

We next mention some implementation details and refinements.

Routing: Lemma 4.1 implies that for every $v \in V$, routing from v to its level i cluster center u_i, for any i, can be done using the LE-lists as routing tables. (The route taken will be the weighted shortest path to u_i.) Additionally, whenever a node w gets a message originating from v along edge e, destined for a node u_i, w stores a routing table entry consisting of v and the edge e. These routing table entries enable routing from u_i to v. Thus all messages can be routed without redundant transmissions.

Controlling Congestion: In the above protocol, several nodes can send a message to their level i cluster center simultaneously. Since the paths from these nodes to their parents can intersect, this can lead to congestion on edges. We modify the protocol so as to merge messages belonging to the same group and destined for the same cluster center, according to the following rules:

1) If in any round, a node w gets at least two simple *find-lca* messages all agreeing in $u_{\bar{i}}$ and $u_{\bar{i}-1}$ and j: w picks one of the messages $\langle v, u_{\bar{i}}, u_{\bar{i}-1}, j\rangle$ arbitrarily and forwards it.

2) If in any round, a node w gets at least two simple *find-lca* messages $\langle v, u_{\bar{i}}, u_{\bar{i}-1}, j\rangle$ and $\langle v', u'_{\bar{i}}, u'_{\bar{i}-1}, j\rangle$ such that $u_{\bar{i}} = u'_{\bar{i}}$ but $u_{\bar{i}-1} \neq u_{\bar{i}-1}$: w concatenates two of the messages and forwards a concatenated message $\langle v, u_{\bar{i}}, u_{\bar{i}-1}, j, v', u'_{\bar{i}-1}\rangle$.

3) If in any round, a node w receives more than one message agreeing on $u_{\bar{i}}$ and j, including at least one concatenated message: w forwards a concatenated message.

4) Messages agreeing on the group j but destined to different cluster centers are not combined.

5) Whenever w drops or concatenates messages, it makes a record locally so that when the reply arrives, it can forward it to all the relevant nodes.

Note that now a cluster center u_i learns of only a subset of the nodes in V_j. However, the above rules ensure that if there are at least two nodes u and u' with different level $(\bar{i}-1)$ cluster centers, then u_i gets messages for at least two such nodes. Thus the correctness of the algorithm (i.e., correctness of ℓ values) continues to hold.

We next prove a lemma to help bound the congestion.

Lemma 6.2 *Suppose nodes in a group V_j send a message in phase i. Then they all have the same level $\bar{i}$ ancestor $u_{\bar{i}}$.*

Proof: We show this by induction on i. In the first phase, clearly all messages are meant for the root so the base case holds. Suppose the claim holds for all $i-1$. Since a message is sent in phase i, all the previous responses must be NO . In particular, $u_{\bar{i}+1}$ sent a NO message which implies that all nodes agree on $u_{\bar{i}}$. The claim follows. □

Thus rule 4 above never applies and at most one message is sent per round on each edge.

Synchronizing the Phases: Let S' be the maximum number of edges in the shortest path from any node v to any of v's cluster center. In each phase, a cluster center waits $S'k$ rounds so that it receives all of the *find-lca* messages destined to it before responding to any of them. Note that $S' \leq S$ and S' and k can easily be computed during the execution of LE-dist algorithm. Before starting the next phase, each node waits for another $S'k$ rounds to make sure all nodes receives the responses of their *find-lca* messages. The correctness of this synchronization follows from the following lemma.

Lemma 6.3 *Any message in the first part of the GST algorithm takes at most $S'k$ time to reach its destination.*

Proof: A message travels through at most S' edges and congestion at any edge is at most k. The claim follows. □

Theorem 6.4 gives the time and message complexity for finding LCA.

Theorem 6.4 *Finding lowest common ancestors for k disjoint groups of nodes takes $O(Sk\log n)$ time and $O(Sn\log n)$ messages in a weighted graph and takes $O(Dk\log n)$ time and $O(Dn\log n)$ messages in an unweighted graph.*

Proof: The time complexity follows from the fact that there are $\delta = O(\log n)$ phases and each phase takes $O(Sk)$ time (Lemma 6.3). For the message complexity, notice that in each phase, each node generates at most one *find-lca* message and receives its response. Since each message travels through at most S edges, the number of messages is at most $2Sn\delta$. For the unweighted case, $S = D$, and the claim follows. □

Corollary 6.5 *For a single group ($k = 1$) of $n' \leq n$ nodes, finding lowest common ancestor takes $O(S\log n)$ time and $O(Sn'\log n)$ messages in a weighted graph and takes $O(D\log n)$ time and $O(Dn'\log n)$ messages in an unweighted graph.*

6.1.2 Construction Part: Converting FRT Tree Path to a Path in the Original Graph

The construction part of the algorithm also runs in synchronized phases. Each node $v \in \cup V_j$, tries to construct paths that correspond to the tree path from v to its level $\ell(v)$ ancestor. The definition of ℓ implies that each group V_j is connected in the resulting subgraph.

In phase i, each node v with $\ell(v) \geq i$ ensures that its level $(i-1)$ cluster center is connected to its level i cluster center. This is achieved by picking a node w from each relevant level $(i-1)$ cluster, and connecting w both to its level $i-1$ cluster center and to its level i cluster center. Thus u_{i-1} and u_i get connected via some node w in u_i's cluster (see Figure 3).

However constructing the above subgraph may require too many messages (causing congestion), as each node u_i may represent several cluster centers, and hence may need to connect to several level $(i+1)$ cluster centers in one phase. To alleviate this, we will actually construct only a subset of these edges. More precisely, if there are several different nodes $v_1, \ldots, v_s$ belonging to the same group V_j that want to connect their level i cluster center u_i (possibly representing different clusters), to their level $(i+1)$ cluster centers $u_{i+1}^{(1)}, \ldots, u_{i+1}^{(s)}$ respectively, we select one arbitrarily and connect u_i to u_{i+1}. For concreteness, suppose that we always select the v_x with the lowest rank; we say that $v_y, x \neq y$ is *overruled* by v_x. Let E' be the corresponding subset of pairs (u_i, u_{i+1}). For a pair $(u_i, u_{i+1}) \in E'$, define $w(u_i, u_{i+1})$ to be $2^{i+2}\beta$.

Before we show how this is implemented in the distributed setting, we show that this optimization preserves correctness. It is easy to see that this builds only a subset of edges and hence the approximation guarantee is preserved. We next argue that this still ensures that each V_j is connected in the resulting subgraph.

Lemma 6.6 *Let the subgraph E' be constructed as above. Then each V_j is connected in E'.*

Proof: Consider a particular group V_j, with $\ell(v) = l$ for each v in V_j. Thus each $v \in V_j$ has the same level l ancestor u_l. We show that E' has a path from u_l to every $v \in V_j$, by induction on the rank of v.

Clearly the lowest ranked node in V_j never gets over-ruled, and hence constructs a path from its level i ancestor to its level $(i + 1)$ ancestor, for every $i : 0 \leq i \leq (l - 1)$. This establishes the base case.

Consider a node $v \in V_j$ and assume inductively that all lower ranked nodes have a path to u_l. If v is never overruled, it clearly has a path to u_l. Else it first gets overruled at some step i, say by a node v'. In this case, v has a path to u_i, and v' has a path to u_i as well. Moreover, since v' has a lower rank, it inductively has a path to u_l. Hence v is connected to u_l as well, and the claim follows. □

We next describe how this set E' is constructed by the distributed algorithm. Nodes that have not yet been overruled are considered *selected*, and these nodes send messages to their level i ancestors in phase i. These messages help u_i determine the next level cluster centers that it needs to connect to. For each such u_{i+1}, it picks an arbitrary descendant from the corresponding cluster and asks it to connect to u_{i+1}. Initially, each node v considers itself *selected* for phase 1 if $\ell(v) \neq 0$. Only the *selected* nodes participate in the next phase.

In phase $i = 0, 1, 2, \ldots, \delta$:

- Each *selected* node v belonging to a group V_j sends a *want connected* message $\langle v, u_i, u_{i+1}, j\rangle$ to its level-i cluster center u_i.
- After the cluster center u_i receives *want connected* messages, if any, from its *selected* descendants in group V_j, u_i picks the one with the lowest rank and selects it. Then u_i sends a *selected-for-next-phase* message to it. Additionally, let U_{i+1} be the set of level $(i + 1)$ cluster centers that are received from the selected nodes. For each $u_{i+1} \in U_{i+1}$, u_i picks an arbitrary sender v of that request, and sends v a *chosen to connect* message consisting of u_i and u_{i+1}.
- If v gets a *chosen to connect* message from u_i consisting of u_i and u_{i+1}, it sends a *connect* message to u_{i+1}. When a *connect* message passes through an edge (v_1, v_2), both v_1 and v_2 mark this edge to be included in the Steiner subgraph.
- After v' gets the *selected-for-next-phase* message, v' considers itself *selected* for the next phase if $i < \ell(v')$.

Note that the *selected-for-next-phase*, *chosen to connect*, and *connect* messages correspond uniquely to some *want connected* message, and hence it suffices to analyze the time and message complexity of the *want connected* messages. We next show the following invariants.

Lemma 6.7 *If v sends a* want connected *message to u_i, then v and u_i are connected in the Steiner subgraph.*

Proof: The proof is by induction on i. The base case is trivial as u is connected to itself. Suppose that v is selected in phase i so that it sends a *want connected* message to u_{i+1}. Then $u_{i+1} \in U_{i+1}$ so that there is v' which receives a *chosen to connect* message from u_i consisting of u_{i+1}. Thus at the end of this phase, v' and u_{i+1} are connected in the Steiner subgraph. Inductively, both v and v' were connected to u_i in the Steiner subgraph. The claim follows. □

Lemma 6.8 *If u_i is connected to u_{i+1} in E' above, then the Steiner subgraph built by the distributed algorithm has a u_i—u_{i+1} path. Moreover, the Steiner subgraph has cost no more than $w(E')$.*

Proof: The set of edges incident on u_i and a higher level cluster center in E' consists exactly of the level $i+1$ ancestors of nodes that are minimal ranked in $V_j \cap desc(u_i)$ for some j. This is exactly the set U_{i+1} and the *chosen to connect* messages ensure that the Steiner subgraph has a path from u_i to u_{i+1}, for every $u_{i+1} \in U_{i+1}$. The fact that a unique v' is chosen to connect ensures that bound on the weight of the subgraph. □

These imply that running the above algorithm will give subgraph that connects each of the V_j's and has expected cost at most $O(\log(n) \cdot OPT)$.

Routing: Since messages go from nodes to their cluster centers, and back, routing is easily done as in the first part.

Controlling Congestion: Similar to the discovery part of the algorithm congestion is controlled by combining the *want connected* messages if they meet at some intermediate node u on their way and have the same destination:

- If two or more *want connected* messages belonging to the same group (e.g. messages originated by nodes $x_1, x_2, \ldots, x_r$) with the same destination u_i arrive at u at the same time, u picks the lowest ranked one and forwards it towards u_i.
- If u_i sends a *selected-for-next-phase* or a *chosen to connect* message destined for x_j, u simply forwards it to x_j.

It is easy to verify that while this pruning allows fewer *want connected* messages from reaching u_i, it still ensures that for each node that would get selected in the unpruned run, its *want connected* message is not pruned, and hence the same set of nodes gets selected. Thus the set of edges in the Steiner subgraph is unchanged.

Synchronizing the Phases: In each phase, a cluster center waits $kS'L_{\max}$ rounds so that it receives all of the *connect* messages destined for it, where $L_{\max} = \max_{v \in V} |\mathbb{L}(v)|$. The leader can find k, S' and $L_{\max}$ by using BFS tree as an aggregation tree and broadcasting them to the other nodes. This takes $O(D)$ time and $O(n)$ messages. Before starting the next phase, a *selected* node waits for another $kS'L_{\max}$ rounds. The correctness of this synchronization follows from the following lemma.

Lemma 6.9 *In the Construction Part of the GSF algorithm, congestion at any edge adjacent to v is at most $k|\mathbb{L}(v)|$.*

Proof: A node v' sends a message to one of its cluster centers u_i through the shortest path $P(v', u_i)$. If v is on path $P(v', u_i)$, there must be an entry $\langle u_i, * \rangle$ in $\mathbb{L}(v)$ (cf. Lemma 4.1 and 4.4). Thus, v can be on the shortest path tree for at most $|\mathbb{L}(v)|$ different destinations. The rules above for controlling congestion ensure that at most one message is forwarded per group to each cluster center, and hence the congestion at any edge (v, u') is at most $k|\mathbb{L}(v)|$. □

The following theorem shows the time and message complexity of the Construction Part.

Theorem 6.10 *The Construction Part of the distributed GSF algorithm takes $O(Sk\log^2 n)$ time and $O(Sn\log n)$ messages in a weighted graph and takes $O(Dk\log^2 n)$ time and $O(Dn\log n)$ messages in an unweighted graph.*

Proof: There are $\delta = O(\log n)$ phases and each phase takes $O(S'kL_{\max})$ time. Since $S' \leq S$ and $L_{\max} = O(\log n)$ (Lemma 4.6), we have the time complexity of $O(Sk\log^2 n)$. For the message complexity, notice that in each phase, each node generates at most one *want connected* message and receives its response. Since each message travels through at most S edges, the number of messages is at most $2Sn\delta$. For the unweighted case, the theorem follows from $S = D$. □

Theorem 6.11 *The above distributed GSF algorithm gives an expected $O(\log n)$ approximation and takes $O(Sk\log^2 n)$ time and at most $O(S|E|\log n)$ messages in a weighted graph and takes $O(Dk\log^2 n)$ time and at most $O(Dn\log n)$ messages in an unweighted graph.*

Proof: Notice that the congestion-related pruning does not affect the set of edges added to the Steiner subgraph. Thus, the expected $O(\log n)$ approximation follows from Lemma 6.8 and Theorem 4 in [3] (as discussed in the last paragraph of Section 5).

After constructing the FRT tree, the GSF algorithm executes the Discovery Part and Construction Part in this order. Thus we have the state time and message complexity by adding the time and messages from Theorem 5.1, 6.4, and 6.10. □

6.2 Minimum Routing Cost Spanning Trees

The minimum routing cost spanning tree problem, and its generalization, the optimum cost communication tree problem (defined in Section 3) is trivially solvable optimally in trees. Thus, using the FRT embedding, we can obtain an (expected) $O(\log n)$ approximation for the problem. The approach for the GSF problem works here also, with $k = 1$. This is because, we can regard all nodes in the graph as one group – the resultant spanning tree that connects all the nodes gives the desired approximation. Thus we state the following theorem which follows from Theorem 6.11.

Theorem 6.12 *There is an (expected) $O(\log n)$-distributed approximation algorithm for the routing cost spanning tree problem that takes $O(S\log^2 n)$ time and $O(S|E|\log n)$ messages in a weighted graph and takes $O(D\log^2 n)$ time and at most $O(Dn\log n)$ messages in an unweighted graph.*

6.3 *k*-Source Shortest Paths

We present a distributed approximation algorithm for k-source shortest paths problem (defined formally in Section 3) in an arbitrary weighted network. FRT embedding gives (expected) $O(\log n)$-approximate shortest paths between any pair of nodes. Below is a description and analysis of the algorithm. The algorithm's goal is to construct routing tables for these (approximate) shortest paths. The routing table at a node specifies the next hop neighbor on the path to any given destination; thus a message can be routed from a source to any given destination along the approximate shortest path using the routing tables.

1. All $v \in V$ find their LE-lists using LE-Dist algorithm and compute the β-lists.

2. Construct a BFS tree rooted at the leader, which can be found using the LE-Dist algorithm. Each source $s_j \in K$ broadcasts its LE-list $\mathbb{L}(s_j)$ to all $v \in V$ using the edges of this BFS tree. The tree construction takes $O(|E|)$ messages and $O(|D|)$ time. Since $O(\log n)$ messages (size of a β-list of a source) need to be sent per source, this process takes overall $O(kD\log n)$ rounds and $O(|E| + kn\log n)$ messages for all k sources.

3. Each $v \in V$ computes $\beta(s_j)$, $1 \le j \le k$, from $\mathbb{L}(s_j)$. Now v can determine the lowest common ancestor of $\{v, s_j\}$ (for any j, $1 \le j \le k$) from the β-lists $\beta(v)$ and $\beta(s_j)$. A path from v to source s_j is constructed by concatenating the shortest paths $P(v, u_i)$ and $P(u_i, s_j)$, where cluster center u_i is the lowest common ancestor of v and s_j. By the FRT embedding, it follows that the above path is an $O(\log n)$-approximation to the shortest path between v and s_j (for any j). The shortest path from any node to any of its cluster center can be found using the the LE-lists as routing tables (Lemma 4.1). Thus, $P(v, u_i)$ can be constructed correctly, but we can construct $P(u_i, s_j)$ only in reverse direction. To solve this problem, i.e., to construct the routing tables for the sub-path $P(u_i, s_j)$, each source s_j sends a *dummy* message to all of its cluster centers u_i so that the nodes in this path can track their predecessors for the purpose of constructing the routing table for the reverse paths toward the sources. For one source, this takes $O(S\log n)$ time and $O(S\log n)$ messages because there at most $O(\log n)$ cluster centers and each message goes through the respective shortest path. For all k-sources, this takes $O(kS\log n)$ time and $O(kS\log n)$ messages.

The correctness of the algorithm is clear from the above description. Hence, combining the time and messages needed for steps 1, 2, and 3, we have the following theorem.

Theorem 6.13 *The above algorithm computes an (expected) $O(\log n)$-approximate k-source shortest paths in $O(kD \log n)$ time using $O(|E| \cdot \min D, \log n + kn \log n)$ messages in an unweighted graph and in $O(kS \log n)$ time using $O(|E|S \log n + kn \log n)$ messages in a weighted graph.*

7 Conclusion

Developing uniform approaches to design efficient distributed approximation algorithms for a wide variety of problems is important to the theory of distributed approximation. This paper presented an algorithmic paradigm based on metric tree embeddings, in particular, probabilistic tree embeddings, to design efficient distributed approximation algorithms. We show that a probabilistic tree embedding due to Fakcharoenphol, Rao, and Talwar (FRT) can be used to design fast distributed (expected) $O(\log n)$-approximation algorithms for GSF, shortest paths, and routing cost spanning trees. The distributed construction of the FRT tree embedding is based on the computation of least elements (LE) lists, a distributed data structure that seems to be very useful in designing efficient distributed algorithms for many problems.

We conclude by discussing some issues for further work. A natural direction would be to extend our approach to design distributed approximation algorithms for other network optimization problems. The FRT tree embedding approach use here has the drawback that each cluster may not be internally connected, and therefore a node may have to carry messages for clusters not containing it. A direct distributed implementation of an algorithm for the tree thus leads to large congestion. While we can overcome this hurdle for some problems here, we leave open the question of finding a distributed embedding with internally connected clusters. Moreover, the algorithms based on these tree embeddings can be easily derandomized in the centralized setting. Designing deterministic distributed algorithms for these problems is a natural open question.

References

[1] Y. Afek, M. Ricklin. Sparser: A Paradigm for Running Distributed Algorithms. *J. Algorithms*, 14(2), 316-328, 1993.

[2] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *19th STOC*, 1987.

[3] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th FOCS*, pages 184–193, 1996.

[4] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *30th STOC*, pages 161–168, 1998.

[5] Y. Bartal, J.W. Byers, and D. Raz. Global optimization using local information with applications to flow control, In *Proc. of FOCS*, 1997.

[6] P. Chalermsook and J. Fakcharoenphol. Simple distributed algorithms for approximating minimum steiner trees. In *11th COCOON*, 2005.

[7] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. In *39th FOCS*, pages 379–388, 1998.

[8] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *JCSS*, 55(3):441–453, 1997.

[9] E. Cohen and H. Kaplan. Spatially-decaying aggregation over a network. *JCSS*, 73(3):265–288, 2007.

[10] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed Verification and Hardness of Distributed Approximation, in *Proceedings of the 43rd ACM Symposium on Theory of Computing* **(STOC)**, San Jose, CA 2011.

[11] D. Dubhashi, F. Grandioni, and A. Panconesi. Distributed algorithms via LP duality and randomization. In *Handbook of Approximation Algorithms and Metaheuristics*, T.F. Gonzalez (editor), CRC Press, 2007.

[12] M. Elkin. Computing almost shortest paths. In *20th PODC*, 2001.

[13] M. Elkin. Distributed approximation - a survey. *ACM SIGACT News*, 35(4):40–57, 2004.

[14] M. Elkin. A faster distributed protocol for constructing minimum spanning tree. In *15th SODA*, 2004.

[15] M. Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *36th STOC*, 2004.

[16] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *JCSS*, 69(3):485–497, 2004.

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

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

[19] F. Grandoni, J. Könemann, A. Panconesi, and M. Sozio. Primal-dual based distributed algorithms for vertex cover with semi-hard capacities. In *24th PODC*, 2005.

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

[21] L. Jia, R. Rajaraman, and R. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

[22] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. *Distributed Computing*, 20:391–402, 2008.

[23] 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*, 385(1-3):101–114, 2007.

[24] 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*, 19(2), December 2008.

[25] F. Kuhn and T. Moscibroda. Distributed Approximation of Capacitated Dominating Sets. In *19th SPAA*, 2007.

[26] F. Kuhn and R. Wattenhofer. Distributed Combinatorial Optimization. *Technical Report 426*, Dept. of Computer Science, ETH, Zurich, 2004.

[27] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What Cannot Be Computed Locally! In *23rd PODC*, 2004.

[28] F. Kuhn and R. Wattenhofer. Constant-time distributed set approximation, *Proc. of PODC*, 2003.

[29] F. Kuhn, T. Moscibroda, and R. Wattenhofer. The Price of Being Near-Sighted. In *17th SODA*, 2006.

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

[31] Z. Nutov and A. Sadeh. Distributed primal-dual approximation algorithms for network design problems, Manuscript, 2009. `http://www.openu.ac.il/home/nutov/distributed.pdf`

[32] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds, *SIAM Journal on Computing*, **26**, 1997, 350-368.

[33] G. Pandurangan and M. Khan. Theory of Communication Networks, in *Algorithms and Theory of Computation Handbook*, M.J. Atallah and M. Blanton (editors), CRC Press, 2009.

[34] C. Papadimitriou and M. Yannakakis. Linear programming without matrix, in *Proc. of STOC*, 1993.

[35] N. Lynch. *Distributed Algorithms*, Morgan Kaufmann Publishers, 1996.

[36] D. Peleg. A time optimal leader election algorithm in general networks. *J. Parallel and Distributed Computing*, 8:96–99, 1990.

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

[38] D. Peleg and V. Rabinovich. A near-tight lower bound on the time complexity of distributed mst construction. In *40th FOCS*, 1999.

[39] R. Rajaraman. Topology Control and Routing in Ad hoc Networks: A Survey, *SIGACT News*, 33:60-73, June 2002.

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

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

[42] M. Wattenhofer and R. Wattenhofer. Distributed Weighted Matching. *Proc. of 18th Annual Conference on Distributed Computing (DISC)*, Amsterdam, Netherlands, October 2004.

[43] B. Wu and K. Chao. *Spanning Trees and Optimization Problems*, Chapman & Hall/CRC, 2004.

[44] B. Wu, G. Lancia, V. Bafna, K. Chao, R. Ravi, and C. Tang. A polynomial time approximation scheme for minimum routing cost spanning trees. In *9th SODA*, 1998.