# The Distributed Minimum Spanning Tree Problem

Gopal Pandurangan[*] Peter Robinson[†] Michele Scquizzato[‡]

**Abstract**

This article surveys the distributed minimum spanning tree (MST) problem, a central and one of the most studied problems in distributed computing. In this problem, we are given a network, represented as a weighted graph $G = (V, E)$, and the nodes in the network communicate by message passing via the edges of $G$ with the goal of constructing an MST of $G$ in a distributed fashion, i.e., each node should identify the MST edges incident to itself. This article summarizes the long line of research in designing efficient distributed algorithms and showing lower bounds for the distributed MST problem, including the most recent developments which have focused on algorithms that are simultaneously round- and message-optimal.

[*]Department of Computer Science, University of Houston, Houston TX, USA. E-mail: gopalpandurangan@gmail.com. Supported, in part, by NSF grants CCF-1527867, CCF-1540512, IIS-1633720, and CCF-BSF-1717075, and by US-Israel Binational Science Foundation (BSF) grant 2016419.

[†]Department of Computing and Software, McMaster University, Hamilton, Canada. E-mail: peter.robinson@mcmaster.ca. Peter Robinson acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

[‡]School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden. E-mail: mscq@kth.se. Supported, in part, by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672.

# 1 Introduction

The minimum-weight spanning tree (MST) problem is a classical and fundamental problem in graph theory, with a long line of research dating back to Borůvka's algorithm in 1926. The MST is an important and commonly occurring primitive in the design and operation of communication networks. Formally, the MST problem is as follows. Given an $n$-node connected (undirected) graph with edge weights, the goal is to compute a *spanning* tree of *minimum total weight*. Weights can be used to model fundamental network performance parameters such as transmission delays, communication costs, etc., and hence an MST represents a spanning tree that optimizes these parameters. One of the most common applications of MST is that it can serve as a backbone for efficient communication, e.g., it can be used naturally for *broadcasting*, i.e., sending a message from a (source) node to all the nodes in the network. Any node that wishes to broadcast simply sends messages along the spanning tree. The message complexity is then $O(n)$, which is optimal. The advantage of this method over flooding (which uses all the edges in the networks) is that redundant messages are avoided, as only the edges in the spanning tree are used. In particular, if weights model the cost or the delay for a message to pass through an edge of the network, an MST can minimize the total cost for a node to communicate with all the other nodes of the network.

Because of its fundamental importance in networks, the MST problem is also one of the central and most studied problems in network algorithms, specifically distributed network algorithms. In the distributed MST problem, the goal is to compute a MST in a distributed fashion efficiently, which is achieved by minimizing two fundamental performance measures used for distributed algorithms: message and time complexity. In a distributed network, each node (which represents a processor) is aware only of its incident edges (which represent communication links to its neighbors) and of their weights. That is, each node starts with only local knowledge, and at the end of the distributed algorithm each node should know which of its incident edges belong to the MST, i.e., the output knowledge is also local (e.g., a node need not know about the status of edges that are not incident to it). We will give formal details of the model assumptions in Section 2.

Besides an efficient communication backbone, a distributed MST algorithm can be also be used for leader election, a fundamental *symmetry breaking* problem in distributed computing. In the leader election problem, among many nodes (potentially all) nodes vying to be leader, the goal is to elect one unique leader. A distributed MST algorithm can construct a rooted tree (where parent-child relationships are known to each node), and the root can serve as the leader. The MST algorithms described in this article naturally construct a rooted tree.

In this article, we survey algorithms and lower bounds for the distributed MST problem. The rest of the article is organized as follows. In Section 2, we formally

describe the distributed computing model and its complexity measures and discuss some preliminaries on the distributed MST problem. In section 3, we give a quick historical survey and summarize the state of the art until 2016. In Section 4, we discuss three well-studied distributed MST algorithms. In Section 5, we present *singularly optimal* distributed MST algorithms that were discovered more recently, i.e., distributed MST algorithms that are simultaneously time- and message-optimal. In Section 6 we discuss lower bounds and describe the techniques used to show such results. We conclude with some open questions in Section 7.

# 2 Model, Definitions, and Preliminaries

We first describe the distributed computing model that we consider in this article. The model is called the CONGEST model (see, e.g., [47, 43]), which is now standard in the distributed computing literature.

A point-to-point communication network is modeled as an undirected weighted graph $G = (V, E, w)$, where the vertices of $V$ represent the processors, the edges of $E$ represent the communication links between them, and $w(e)$ is the weight of edge $e \in E$. Without loss of generality, we assume that $G$ is connected. We use $D$ to denote the hop-diameter (that is, the unweighted diameter) of $G$, and, in this article, by *diameter* we always mean hop-diameter. We also assume that the weights of the edges of the graph are all *distinct*. This implies that the MST of the graph is *unique*. [1] Each node hosts a processor with limited initial knowledge. Specifically, we make the common assumption that each node has unique identity numbers, and at the beginning of computation each vertex $v$ accepts as input its own identity number and the weights of the edges incident to it. Thus, a node has only *local* knowledge. Specifically, we assume that each node $u$ has ports (each port having a unique integer port number) and each incident edge of $u$ is connected to one of $u$'s distinct port. A node does not have any initial knowledge of the other endpoint of its incident edge (which node it is connected to or the port number that this node is connected to). This model is referred to as the *clean network model* in [47, 43] and is also sometimes called the $KT_0$ model, i.e., the initial (K)nowledge of all nodes is restricted (T)ill radius 0, c.f. [43, 47]. The $KT_0$ model is a standard model in distributed computing and used by many prior results on distributed MST (e.g., [3, 6, 15, 16, 36, 14, 10]), with a notable exception ([29], discussed in Section 3).

The vertices are allowed to communicate by sending messages to their neighbors through the edges of the graph $G$. Here we assume that communication is

[1] Even if the weights are not distinct, it is easier to make them distinct by using the unique node identifiers of the respective endpoints of the edges to break ties.

*synchronous* and occurs in discrete *rounds* (time steps). In each round, each node $v$ can send an arbitrary message of $O(\log n)$ bits through each edge $e = (v, u)$ incident to $v$, and each message arrives at $u$ by the end of this round. [2] The weights of the edges are at most polynomial in the number of vertices $n$, and therefore the weight of a single edge can be communicated in one time step. This model of distributed computation is called the CONGEST($\log n$) model or simply the CONGEST model [47, 43]. Some of the MST algorithms discussed are randomized and hence also assume that each vertex has access to the outcomes of an unbiased private coin flips.

The efficiency of distributed algorithms is traditionally measured by their time and message (or, communication) complexities. *Time complexity* measures the number of synchronous rounds taken by the algorithm, whereas *message complexity* measures the total number of messages sent and received by all the processors during the execution of the algorithm. Both complexity measures crucially influence the performance of a distributed algorithm. As defined in [45], we say that a problem enjoys *singular optimality* if it admits a distributed algorithm whose time and message complexity are both optimal.

We make an assumption that simplifies our algorithms and analysis: we assume that all edge weights in the graph are distinct. It is easy to show that this implies that the MST is unique. This assumption is without loss of generality, because one can tag each edge weight (additionally) with the node IDs of the endpoints of the edge (which are unique as a pair).[3] This tagging can be used to break ties between edges having the same weight.

As in centralized MST algorithms, distributed MST algorithms also rely on two important properties of an MST: (1) the *cut property* and (2) the *cycle property* [49]:

1. **Cut property:** A cut in a graph is a partition of the vertex set into two disjoint sets. The cut property states that, given any cut in a graph, the *lightest*, i.e. minimum weight, edge crossing the cut belongs to the MST. (Recall that due to the assumption of unique edge weights, there is a unique lightest edge crossing the cut.)
2. **Cycle property:** Consider any cycle in the graph. The heaviest (i.e. maximum weight) edge in the cycle will not be in the MST.

[2] If unbounded-size messages are allowed—this is the so-called LOCAL model—the MST problem can be trivially solved in $O(D)$ time by collecting all the topology information in one node [43].

[3] Note that this tagging involves nodes knowing the IDs of their neighbors. This tagging can be done during the course of an MST algorithm.

# 3 Summary of Prior Research

Given the importance of the distributed MST problem, there has been significant work over the last 30 years on this problem and related aspects. The distributed MST problem has triggered a lot of research in distributed computing, leading to new techniques for the design and analysis of distributed algorithms as well as for showing lower bounds. We first briefly summarize research until 2016 and then discuss very recent results on singularly optimal algorithms.

## 3.1 Research Until 2016

**Early Work.** Distributed MST was one of the very first distributed computing problems that was studied. A long line of research aimed at developing efficient distributed algorithms for the MST problem started more than thirty years ago with the seminal paper of Gallager, Humblet, and Spira [15], which presented a distributed algorithm that constructs an MST in $O(n \log n)$ rounds exchanging a total of $O(m + n \log n)$ messages, where $n$ and $m$ denote the number of nodes and the number of edges of the network, respectively.[4] The message complexity of this algorithm is (essentially) optimal [35], but its time complexity is not. Hence further research concentrated on improving the time complexity, resulting in a first improvement to $O(n \log \log n)$ by Chin and Ting [6], a further improvement to $O(n \log^* n)$ by Gafni [14], and then to $O(n)$ by Awerbuch [3] (see also [13]). The $O(n)$ bound is existentially optimal in the sense that there exist graphs with diameter $\Theta(n)$ for which this is the best possible, even for randomized Monte-Carlo algorithms (see [35]).

This was the state of the art until the mid-nineties when Garay, Kutten, and Peleg [16] raised the question of whether it is possible to identify graph parameters that can better capture the complexity of distributed network computations. In fact, for many existing networks, the hop-diameter $D$ is significantly smaller than the number of vertices $n$, and therefore it is desirable to obtain protocols whose running time is bounded in terms of $D$ rather than in terms of $n$. (We note that $\Omega(D)$ is a lower bound on the running time of any distributed MST algorithm in a "universal" sense [35, 34]). Garay, Kutten, and Peleg [16] gave the first such distributed algorithm for the MST problem with running time $O(D + n^{0.614} \log^* n)$, which was later improved by Kutten and Peleg [36] to $O(D + \sqrt{n} \log^* n)$.[5] However, both these algorithms are not message-optimal,[6]

[4] The original algorithm has a message complexity of $O(m \log n)$, but it can be improved to $O(m + n \log n)$.

[5] The $\log^* n$ factor can, in all respective algorithms, be reduced to $\sqrt{\log^* n}$, by growing components to a size larger by a factor $\sqrt{\log^* n}$ in the respective first phase.

[6] In this paper, "optimal" means "optimal up to a polylog(n) factor."

as they exchange $O(m + n^{1.614})$ and $O(m + n^{1.5})$ messages, respectively.

The lack of progress in improving the result of [36], and in particular breaking the $\tilde{O}(\sqrt{n})$ barrier,[7] led to work on lower bounds for the distributed MST problems, starting out by the result of Peleg and Rubinovich [48] and the more recent extensions of [11, 7], which we discuss in more detail in Section 6.

**State of the Art (until 2016).** We now summarize the state of the art for distributed MST algorithms until 2016: There exist algorithms which are either time-optimal (i.e., they run in $\tilde{O}(D + \sqrt{n})$ time) or message-optimal (i.e., they exchange $\tilde{O}(m)$ messages), *but not simultaneously both*. Indeed, the time-optimal algorithms of [36, 10] (as well as the sublinear time algorithm of [16]) are not message-optimal, i.e., they require asymptotically much more than $\Theta(m)$ messages. In contrast, the known message-optimal algorithms for MST (in particular, [15, 3]) are not time-optimal, i.e., they take significantly more time than $\tilde{O}(D + \sqrt{n})$. In their 2000 SICOMP paper [48], Peleg and Rubinovich raised the question of whether one can design a distributed MST algorithm that is *simultaneously* optimal with respect to time and message complexity. In 2011, Kor, Korman, and Peleg [32] also raised this question and showed that distributed *verification* of MST, i.e., verifying whether a given spanning tree is MST or not, can be done in optimal messages and time, i.e., there exists a distributed verification algorithm that uses $\tilde{O}(m)$ messages and runs in $\tilde{O}(D+\sqrt{n})$ time. However, achieving these bounds for MST construction remained open until 2017.

**Singular Optimality.** As defined in [45], the main question that remained open for distributed MST is whether there is a distributed algorithm that is *singularly optimal* or if the problem exhibits a *time-message trade-off*:

- **Singularly optimal:** A distributed algorithm that is optimal with respect to both measures simultaneously. In this case we say that the problem enjoys *singular optimality*.
- **Time-message trade-off:** When the problem inherently fails to admit a singularly optimal solution, namely, algorithms with better time complexity necessarily incur higher message complexity, and vice versa. In this case we say that the problem exhibits a *time-message trade-off*.

The above question addresses a fundamental aspect in distributed algorithms, namely the relationship between the two basic complexity measures of time and messages. The simultaneous optimization of both time and message complexity has been elusive for several fundamental problems (including MST, shortest

[7] $\tilde{O}(f(n))$ and $\tilde{\Omega}(f(n))$ denote $O(f(n) \cdot \text{polylog}(f(n)))$ and $\Omega(f(n)/\text{ polylog}(f(n)))$, respectively.

paths, and random walks), and consequently research in the last three decades in distributed algorithms has focused mainly on optimizing either one of the two measures separately. However, in various modern and emerging applications such as resource-constrained communication networks and distributed computation of large-scale data, it is crucial to design distributed algorithms that optimize both measures *simultaneously* [30, 24].

Besides the prior work already mentioned, we now discuss other relevant work on distributed MST.

**Other Distributed MST Algorithms.** Elkin [10] showed that a parameter called *MST-radius* captures the complexity of distributed MST algorithms better. The MST-radius, denoted by $\mu(G, w)$, is a function of the graph topology as well as the edge weights and, roughly speaking, is the maximum radius each vertex has to examine to check whether any of its edges is in the MST. Elkin devised a distributed protocol that constructs the MST in $\tilde{O}(\mu(G, w) + \sqrt{n})$ time. 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 [36] by a factor of $\Omega(\sqrt{n})$. However, a drawback of this protocol (unlike the previous MST protocols [36, 16, 6, 14, 15]) is that it cannot detect the termination of the algorithm in that time (unless $\mu(G, w)$ is given as part of the input). On the other hand, it can be shown that for distributed MST algorithms that correctly terminate, $\Omega(D)$ is a lower bound on the running time [48, 34]. (In fact, [34] shows that for every sufficiently large $n$ and every function $D(n)$ with $2 \leq D(n) < n/4$, there exists a graph $G$ of $n' \in \Theta(n)$ nodes and diameter $D' \in \Theta(D(n))$ which requires $\Omega(D')$ rounds to compute a spanning tree with constant probability in the clean network model.) We also note that the message complexity of Elkin's algorithm is $O(m + n^{1.5})$.

Some specific classes of graphs admit efficient MST algorithms that beat the general $\tilde{\Omega}(D + \sqrt{n})$ time lower bound. Perhaps the first result in this direction is due to Khan and Pandurangan [28] who showed that one can obtain a distributed $O(\log n)$-approximate algorithm for MST for special classes of graphs such as unit disk graphs (that are used to model wireless networks) and randomly weighted graphs that runs in $\tilde{O}(D)$ time and $\tilde{O}(m)$ messages. More recently, faster distributed MST algorithms have been presented for planar graphs, graphs of bounded genus, treewidth, or pathwidth [17, 22, 23], and graphs with small mixing time [18]. Faster distributed MST algorithms have also been developed for networks with small constant diameter. An $O(\log n)$ time algorithm for networks with $D = 2$ was given in [38], and an $O(\log \log n)$ time algorithm for networks with $D = 1$ (i.e., complete networks) was given in [37]. In fact, the complete network case, and more in general, MST construction in the so-called *Congested Clique* model, has received significant attention in recent years. (The Congested

Clique is a simple model for overlay networks whereby, unlike the CONGEST model, even non-adjacent nodes can communicate directly.) The deterministic algorithm of Lotker et al. [37] (which improved significantly over an easy-to-construct $O(\log n)$-round algorithm) was later improved by Hegeman et al. [24], who presented an $O(\log \log \log n)$-round randomized Monte Carlo algorithm. The key technique for this result is linear graph sketching [1, 2, 40], which has since been applied in other distributed algorithms (see, e.g., [44]). This paper also gives lower bounds on the message complexity of MST construction in the Congested Clique: $\Omega(n^2)$ in the $KT_0$ model and $\Omega(n)$ in the $KT_1$ model respectively. It also gives a randomized algorithm that uses $\tilde{O}(n)$ messages and runs in $\tilde{O}(1)$ rounds in the $KT_1$ model. Building on the work of Hegeman et al., the time bound for MST construction in the Congested Clique has been improved to $O(\log^* n)$ rounds [19], and then, very recently, to $O(1)$ rounds [27]. All these algorithms are randomized and crucially rely on using graph sketches.

However, there exists a significant gap in the time complexity of the distributed MST algorithms between the cases of network diameters 2 and 3. In [38], it was shown that the time complexity of any distributed MST algorithm is $\Omega(\sqrt[4]{n}/\sqrt{B})$ for networks of diameter 3 and $\Omega(\sqrt[3]{n}/\sqrt{B})$ for networks of diameter 4. These asymptotic lower bounds hold for randomized algorithms as well. (On the other hand, $O(\log n)$ time suffices to compute an MST deterministically for graphs with diameter 2.)

**Time Complexity.** From a practical perspective, given that MST construction can take as much as $\Omega(\sqrt{n}/\log n)$ time even in low-diameter networks, it is worth investigating whether one can design distributed algorithms that run faster and output an approximate minimum spanning tree. The question of devising faster approximation algorithms for MST was raised in [48]. Elkin [11] later established a hardness result on distributed MST approximation, showing that *approximating* the MST problem on a certain family of graphs of small diameter (e.g., $O(\log n)$) within a ratio $H$ requires essentially $\Omega(\sqrt{n/H \log n})$ time. Khan and Pandurangan [28] showed that there can be an exponential time gap between exact and approximate MST construction by showing that there exist graphs where any distributed (exact) MST algorithm takes $\Omega(\sqrt{n}/\log n)$ rounds, whereas an $O(\log n)$-approximate MST can be computed in $O(\log n)$ rounds. The distributed approximation algorithm of [28] is message-optimal but not time-optimal.

Das Sarma et al. [7] settled the time complexity of distributed approximate MST by showing that this problem, as well as approximating shortest paths and about twenty other problems, satisfies a time lower bound of $\tilde{\Omega}(D + \sqrt{n})$. This applies to deterministic as well as randomized algorithms, and to both exact and approximate versions. In other words, any distributed algorithm for computing a

$H$-approximation to MST, for any $H > 0$, takes $\tilde{\Omega}(D + \sqrt{n})$ time in the worst case.

**Message Complexity.** Kutten et al. [35] fully settled the message complexity of leader election in general graphs assuming the clean network (i.e. $KT_0$ model), even for randomized algorithms and under very general settings. Specifically, they showed that any randomized algorithm (including Monte Carlo algorithms with suitably large constant success probability) requires $\Omega(m)$ messages; this lower bound holds in a universal sense: Given any $n$ and $m$, there exists a graph with $\Theta(n)$ nodes and $\Theta(m)$ edges for which the lower bound applies. Since a distributed MST algorithm can also be used to elect a leader (where the root of the tree is the leader, which can be chosen using $O(n)$ messages once a tree is constructed), the above lower bound applies to distributed MST constructions as well, for all $m \geq cn$, where $c$ is a sufficiently large constant.

The above lower bound bound holds even if nodes have initial knowledge of $n, m$, and $D$. It also holds for synchronous networks, where all the nodes wake up simultaneously. Finally, it holds not only for the CONGEST model [47], where sending a message of $O(\log n)$ bits takes one unit of time, but also for the LOCAL model [47], where the number of bits carried in a single message can be arbitrary. On the other hand, it is known from [15, 3] that an MST can be constructed using $\tilde{O}(m)$ messages in synchronous networks.

**The $KT_1$ Variant.** It is important to point out that all the results discussed in this article (including the MST results [3, 6, 15, 16, 36, 14, 10]) assume the so-called *clean network model*, a.k.a. $KT_0$ [47] (cf. Section 2), where nodes do not have initial knowledge of the identity of their neighbors. However, one can assume a model where nodes do have such a knowledge. This model is called the $KT_1$ *model*. Although the distinction between $KT_0$ and $KT_1$ has clearly no bearing on the asymptotic bounds for the time complexity, it is significant when considering message complexity. Awerbuch et al. [4] show that $\Omega(m)$ is a message lower bound for MST in the $KT_1$ model, if one allows only (possibly randomized Monte Carlo) comparison-based algorithms, i.e., algorithms that can operate on IDs only by comparing them.

Awerbuch et al. [4] also show that the $\Omega(m)$ message lower bound applies even to non-comparison based (in particular, algorithms that can perform arbitrary local computations) *deterministic* algorithms in the CONGEST model that terminate in a time bound that depends only on the graph topology (e.g., a function of $n$). On the other hand, for *randomized non-comparison-based* algorithms, it turns out that the message lower bound of $\Omega(m)$ does not apply in the $KT_1$ model. In 2015, King et al. [29] showed a surprising and elegant result: in the $KT_1$ model one can give a randomized Monte Carlo algorithm to construct an MST in $\tilde{O}(n)$ mes-

sages ($\Omega(n)$ is a trivial message lower bound for the $KT_1$ model) and in $\tilde{O}(n)$ time. This algorithm is randomized and not comparison-based. While this algorithm shows that one can achieve $o(m)$ message complexity (when $m = \omega(n \operatorname{polylog} n)$), it is *not* time-optimal (it can take significantly more than $\tilde{\Theta}(D + \sqrt{n})$ rounds). In subsequent work, Mashreghi and King [39] presented another randomized, not comparison-based MST algorithm with round complexity $\tilde{O}(\text{Diam(MST)})$ and with message complexity $\tilde{O}(n)$. It is an open question whether one can design a randomized (non-comparison based) algorithm that takes $\tilde{O}(D + \sqrt{n})$ time and $\tilde{O}(n)$ messages in the $KT_1$ model. Very recently, Gmyr and Pandurangan [20] presented improved algorithms in the $KT_1$ model for MST and several other problems. For the MST problem, they showed that it can be solved in $\tilde{O}(D + n^{1-\delta})$ rounds using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages for any $\delta \in [0, 0.5]$. In particular, for $\delta = 0.5$ they obtain a distributed MST algorithm that runs in optimal $\tilde{O}(D + \sqrt{n})$ rounds and uses $\tilde{O}(\min\{m, n^{3/2}\})$ messages. Notice that this improves over the singularly optimal algorithms ([45, 12, 21]) for the $KT_0$ model.

## 3.2 Recent Results

In 2017, Pandurangan et al. [45] presented the first distributed MST algorithm for the CONGEST model which is simultaneously time- and message-optimal. The algorithm is randomized Las Vegas, and always returns the MST. The running time of the algorithm is $\tilde{O}(D + \sqrt{n})$ and the message complexity is $\tilde{O}(m)$, and both bounds hold with high probability.[8] This is the first distributed MST algorithm that matches *simultaneously* the time lower bound of $\tilde{\Omega}(D + \sqrt{n})$ [11, 7] and the message lower bound of $\Omega(m)$ [35], which both apply even to randomized Monte Carlo algorithms, thus closing a more than thirty-year-old line of research in distributed computing. In terms of the terminology introduced earlier, we can therefore say that the distributed MST problem exhibits singular optimality up to polylogarithmic factors.

The work of Pandurangan et al. [45] raised the open problem of whether there exists a *deterministic* time- and message-optimal MST algorithm. We notice that the algorithm of Pandurangan et al. is *randomized*, due to the use of the randomized cover construction of [10], even though the rest of the algorithm is deterministic. Elkin [12], building on the work of [45], answered this question affirmatively by devising a deterministic MST algorithm that achieves essentially the same bounds, i.e., it uses $\tilde{O}(m)$ messages and runs in $\tilde{O}(D + \sqrt{n})$ time. Another deterministic round- and message-optimal algorithm for MST appeared very recently in [21]. Table 1 summarizes the known upper bounds on the complexity of distributed MST.

[8] Throughout, with high probability (w.h.p.) means with probability $\geq 1 - 1/n^{\Omega(1)}$.

| Reference | Time | Messages | Computation |
|---|---|---|---|
| Gallager et al. [15] | $O(n \log n)$ | $O(m + n \log n)$ | Deterministic |
| Awerbuch [3] | $O(n)$ | $O(m + n \log n)$ | Deterministic |
| Garay et al. [16] | $O(D + n^{0.614} \log^* n)$ | $O(m + n^{1.614})$ | Deterministic |
| Kutten and Peleg [36] | $O(D + \sqrt{n} \log^* n)$ | $O(m + n^{1.5})$ | Deterministic |
| Elkin [10] | $\tilde{O}(\mu(G, w) + \sqrt{n})$ | $O(m + n^{1.5})$ | Randomized |
| Pandurangan et al. [45] | $\tilde{O}(D + \sqrt{n})$ | $\tilde{O}(m)$ | Randomized |
| Elkin [12] | $\tilde{O}(D + \sqrt{n})$ | $\tilde{O}(m)$ | Deterministic |
| Haeupler et al. [21] | $\tilde{O}(D + \sqrt{n})$ | $\tilde{O}(m)$ | Deterministic |

Table 1: Summary of upper bounds on the complexity of distributed MST. Notation $\tilde{O}(\cdot)$ hides polyogarithmic factors in $n$.

# 4 Overview of Classical MST Algorithms

The recent singularly optimal distributed MST algorithm of [45, 12] that we discuss in Section 5 build on prior distributed MST algorithms that were either message-optimal or time-optimal but not both. We now provide a brief overview of three well-known previous MST algorithms: (1) The Gallager-Humblet-Spira (GHS) algorithm; (2) The Pipeline algorithm; (3) Garay-Kutten-Peleg (GKP) algorithm. This overview is based on [43], to which we refer to for a more detailed description of these algorithms.

## 4.1 The Gallager-Humblet-Spira (GHS) algorithm

The first distributed algorithm for the MST problem was given by Gallager, Humblet, and Spira in 1983 [15].

We are given an undirected, connected, weighted graph $G = (V, E, w)$. Let $n$ be the number of nodes and $m$ be the number of edges of $G$. Let $T$ be the (unique) MST on $G$. An *MST fragment* (or simply a *fragment*) $F$ of $T$ is defined as a connected subgraph of $T$, that is, $F$ is a subtree of $T$. An *outgoing edge* of an MST fragment is an edge in $E$ where one adjacent node to the edge is in the fragment and the other is not. The *minimum-weight outgoing edge (MOE)* of a fragment $F$ is the edge with *minimum weight* among all outgoing edges of $F$. As an immediate consequence of the cut property of MSTs, the MOE of a fragment $F = (V_F, E_F)$ is an edge of the MST.

The synchronous GHS algorithm is essentially a distributed implementation of the classical Borůvka's MST algorithm [5]. The GHS algorithm operates in *phases*. In the first phase, it starts with each individual node as a fragment by itself and continues until there is only one fragment left. That is, at the beginning,

there are $|V|$ fragments, and at the end of the last phase, a single fragment which is exactly the sought MST. All fragments find their MOE simultaneously in parallel.

In each phase, the algorithm maintains the following invariant: Each MST fragment has a leader and all nodes know their respective parents and children. The root of the tree will be the leader. Initially, each node (a singleton fragment) is a root node; subsequently each fragment will have one root (leader) node. Each fragment is identified by the identifier of its root, called the fragment ID, and each node in the fragment knows its fragment ID.

### 4.1.1 One phase of GHS

We describe one phase of the GHS algorithm. Each fragment's operation is coordinated by the respective fragment's root (leader). Each phase consists of two major operations: (1) Find MOE of all fragments and (2) Merging fragments via their MOEs.

**Find MOE of all fragments** In a phase, all fragments find their MOE *simultaneously in parallel*. To find the MOE of a fragment, the root in the fragment broadcasts a message ("find MOE") to all nodes in the fragment using the edges in the fragment. Once a node receives "find MOE" message, it finds its minimum outgoing *incident* edge (i.e, the minimum weight outgoing edge among all the incident edges). To find the minimum weight outgoing incident edge, a node checks its neighbors in *increasing order of weight*. If the fragment ID of its neighbor is different from its own, then the edge is an outgoing edge. Note that since edges are checked in increasing order of weight, the first neighbor whose fragment ID is different from its own is the minimum outgoing incident edge. Also, note that the checking can be done (in increasing weight order) starting from the neighbor that was checked *last* in the previous phase. This is because, all edges that were checked earlier would belong to *the same fragment* and will continue to be in the same fragment until the end of the algorithm. Then, each node sends its minimum outgoing incident edge to the root by *convergecasting* the minimum; the root then finds the MOE, which is the minimum among all the edges convergecast. Note that the convergecast process uses the fragment (tree) edges only.

**Merging fragments via their MOEs** Next, fragments are merged via their MOEs. Recall that MOEs belong to the MST (by the cut property).

Once the leader finds the MOE, it broadcasts a "Merge" message to all its fragment nodes (the broadcast is sent along the tree edges); the message contains the MOE edge of the fragment. Hence upon receiving the MOE edge, a node knows whether it is the same as its minimum outgoing incident edge or not. If a

node is incident to the fragment's MOE edge, then the node attempts to combine with its neighbor (which belongs to a different fragment). It sends a "Request to combine" message to its neighbor. If the neighbor has also selected the *same* edge as its MOE then the two neighboring nodes agree to combine through this edge; i.e., both neighboring nodes receive "Request to combine" message from each other. (Otherwise, if only of them receives this message, it ignores it. However, the MOE edge is not ignored: the neighbor marks the edge over which it receives the message as the MOE edge of its neighboring fragment). The node with the higher identifier becomes the root of the combined fragment. The (combined) root broadcasts a "new-fragment" message through the fragment edges and the MOE edges chosen by all the fragments. Each node updates its parent, children, and fragment identifier (which will be the ID of the new root).

To see that the above process correctly combines the fragments, we ascribe a direction to all the MOEs (towards the outgoing way). This creates a "directed tree" of fragments (think of fragments as "super-nodes"). Note that since each fragment has only one outgoing edge, there can *at most one pair* of neighboring nodes (in this directed tree). One of these nodes in the pair will be the root of the combined fragment as described above.

**Analysis of GHS algorithm** It is easy to argue that the total number of phases is $O(\log n)$. This is because, in each phase, the total number of fragments is reduced by at least half: in the worst case, each MOE will be the MOE of both neighboring fragments and they combine into one.

We next argue that each phase takes $O(n)$ time. Hence, overall time complexity is $O(n \log n)$. This is because in each phase, both the major operations take $O(n)$ time (these include broadcast and convergecast), since they happen along the MST edges and the diameter of the MST can be as large as $\Theta(n)$.

We next argue that each phase takes $O(n)$ messages (for convergecast and broadcast) plus the messages needed to find the MOE. The latter takes a total of $O(m + n \log n)$ messages because in each phase a node checks its neighbor in increasing order of weight starting from the last checked node. Thus, except for the last checked node (which takes one message per phase) all other neighbors are checked at most once. Hence, the total message complexity is

$$\sum_{v \in V} 2d(v) + \sum_{i=1}^{\log n} \sum_{v \in V} 1 = O(m + n \log n).$$

## 4.2 The Pipeline Algorithm

Next we discuss the Pipeline algorithm due to Peleg [46], which is slightly better than the GHS algorithm, i.e., it runs in $O(n)$ rounds. Note that this is *existentially*

optimal, since we know that the diameter is a lower bound for MST (even for randomized algorithms) [35] and hence there exists graphs of diameter $\Theta(n)$, where any MST algorithm will require $\Omega(n)$ rounds.

The Pipeline algorithm is essentially an *upcast* algorithm, where we build a BFS tree over the graph and each node upcasts edges to the root of the BFS tree; the root ends up having (enough) global knowledge of the network topology and locally computes the MST and downcasts the MST edges to all nodes in the network. Of course, a naive upcast is for each node to send all its incident edges and this upcast can take $\Theta(m)$ rounds, since there are as many edges. The main idea of the Pipeline MST algorithm is to filter the number of edges broadcast so that the running time is reduced to $O(n)$ rounds. However, the message complexity of the Pipeline algorithm can be as much as $\Theta(n^2)$.

The pipeline algorithm uses the cycle property of MST to filter edges at intermediate nodes. Each node $v$, except the root $r$, maintains two lists of edges, $Q$ and $U$. Initially, $Q$ contains only the edges adjacent to $v$, and $U$ is empty. At each round, $v$ sends the *minimum-weight* edge in $Q$ that does not create a cycle with the edges in $U$ to its parent and moves this edge from $Q$ to $U$. If $Q$ is empty, $v$ sends a terminate message to its parent. The parent after receiving an edge from a child, adds the edge in its $Q$ list. A leaf node starts sending edges upwards at round 0. An intermediate node starts sending at the first round after it has received at least one message from each of its children.

### 4.2.1 Analysis

We give the high-level idea behind the correctness and running time.

**Correctness** The algorithm's correctness follows from the cycle property. Using the cycle property, since only non-MST edges are filtered (note that an edge at node $v$ is not sent upward if it closes a cycle with edges in $U$, i.e., the already sent edges—since edges are sent in non-increasing order, the filtered edges are the respective heaviest edge in a cycle) it can be shown that the root receives all the MST edges (plus possible additional edges) required to compute the MST correctly.

**Running time** It is easy to show that the edges reported by each node to its parent in the tree are sent in non-decreasing weight order, and each node sends at most $n - 1$ edges upward to its parent. This is because if more than $n - 1$ edges are sent through a node, then at least one edge will form a cycle with the edges sent previously and will be filtered by the cycle property. To build the BFS tree, it takes $O(D)$ time. Since the depth of the BFS tree is $D$ and each node sends at

most $n-1$ edges upward, the pipeline algorithm takes $O(D+n) = O(n)$ time. The analysis shows that there is not too much delay (more than $n$) overall before the root receives all the edges that it needs for computing the MST.

The complexity of the Pipeline algorithm is stated in Table 1; for a detailed proof we refer to [43].

## 4.3 The Garay-Kutten-Peleg (GKP) Algorithm

The GKP algorithm [36] runs in $O(D + \sqrt{n}\log^* n)$ time, where $D$ is the diameter of the graph $G$, which is essentially optimal (up to logarithmic factors), due to the existence of the time lower bound of $\tilde{\Omega}(D + \sqrt{n})$ [48, 11, 7]. Note that in graphs with diameter smaller than $\sqrt{n}$, this algorithm can be much faster than the Pipeline algorithm.

We give a high-level overview of the GKP algorithm; for full details we refer to [43].

The GKP algorithm consists of two parts: it combines the *GHS algorithm* and the *Pipeline algorithm* in a judicious way.

### 4.3.1 First part: Controlled-GHS Algorithm

The first part, called the *controlled-GHS algorithm* is similar to the GHS algorithm, with the crucial property of ensuring that the diameter of fragments do not become too big. (Note that in the GHS algorithm, even after the first phase, there can be fragments with diameter as large as $n$). The controlled-GHS begins with each node as a singleton fragment. In every phase, as in the GHS algorithm, each fragment finds its MOE. However, not all fragments are merged along their MOE edges. Only a subset of MOE edges are selected for merging. A crucial property of the merging is the following: in every phase, the number of fragments is reduced by at least a factor of two, while the diameter is not increased by more than a constant factor. The controlled-GHS algorithm continues for about $\log(\sqrt{n})$ phases. At the end of the first part, the following property is guaranteed: the diameter of each fragment is $O(\sqrt{n})$ and there are at most $\sqrt{n}$ fragments. The first part of the algorithm takes $O(\sqrt{n}\log^* n)$ time.

We note that Controlled-GHS as implemented in the time-optimal algorithm of [36] is not message-optimal—the messages exchanged can be $\tilde{O}(m+n^{1.5})$; however, a modified version can be implemented using $\tilde{O}(m)$ messages [43].

### 4.3.2 Second part: Pipeline algorithm

The second part of the algorithm uses the Pipeline algorithm to find the remaining (at most) $\sqrt{n}-1$ MST edges (since there are $\leq \sqrt{n}$ fragments left). As in the

Pipeline algorithm, a breadth-first tree $B$ is built on $G$. Let $r(B)$ be the root of $B$. Using the edges in $B$, root $r(B)$ collects weights of the *inter-fragment* edges, computes the minimum spanning tree $T'$ of the fragments by considering each fragment as a super node. It then broadcasts the edges in $T'$ to the other nodes using the breadth-first tree $B$. Since the depth of $B$ is $O(D)$ and each node sends at most $\sqrt{n}$ edges upward, the Pipeline algorithm takes $O(D + \sqrt{n})$ time; this analysis is very similar to the one of the Pipeline algorithm (by replacing $n$ with $\sqrt{n}$). Thus the overall time complexity of the GKP algorithm is $O(D + \sqrt{n}\log^* n)$. The message complexity can be shown to be $O(m + n^{1.5})$.

The overall algorithms consists of running the controlled-GHS first and then switching to the Pipeline algorithm after $O(\sqrt{n}\log^* n)$ rounds. Combining the two parts, the complexity bounds as stated in Table 1 follow.

# 5 Toward Singular Optimality: Round- and Message-Optimal Distributed MST Algorithms

As mentioned in Section 1, up until recently, all known distributed MST algorithms achieved either optimal time or optimal message complexity, but not both. In this section, we describe the recent progress toward achieving *singular optimality*, as defined in Section 1. We begin by describing the algorithm of [45] in Section 5.1, on which the subsequent work of Elkin is based [12], which is discussed in Section 5.2.

## 5.1 A Randomized Singularly-Optimal Algorithm

We now describe the algorithm of [45] that attains *singular optimality*, i.e., $\tilde{O}(m)$ messages and $\tilde{O}(D+\sqrt{n})$ time. The first part of the algorithm consists of executing Controlled-GHS, described in Section 4.3.1. At the end of this procedure, at most $O(\sqrt{n})$ distinct MST fragments remain, each of which has diameter $O(\sqrt{n})$; we call these *base fragments*.

The second part of the algorithm is different from the existing time-optimal MST algorithms. The existing time-optimal MST algorithms [36, 10], as well as the algorithm of [16], are not message-optimal since they use the Pipeline procedure of [46, 16].

The algorithm of [45] uses a different strategy to achieve optimality in both time and messages. The main novelty is how the (at most) $\sqrt{n}$ base fragments which remain at the end of the Controlled-GHS procedure are merge into one resulting fragment (the MST). Unlike previous time-optimal algorithms [36, 10, 16], this algorithm does not use the Pipeline procedure of [46, 16], since it is not

message-optimal. Instead, it continues to merge fragments and uses two main ideas to implement a Borůvka-style merging strategy efficiently. The first idea is a procedure to efficiently merge when $D$ is small (i.e., $D = O(\sqrt{n})$) or when the number of fragments remaining is small (i.e., $O(n/D)$). The second idea is to use *sparse neighborhood covers* and efficient communication between fragments to merge fragments when $D$ is large *and* the number of fragments is large. Accordingly, the second part of the algorithm can be conceptually divided into three phases, which are described next. The description in the remainder of this subsection closely follows Section 2 in [45].

### 5.1.1 Phase 1: When $D$ is $O(\sqrt{n})$:

At the start of this phase, a BFS tree of the entire network is constructed, and then merging is performed as follows: Each base fragment finds its minimum-weight-outgoing edge (MOE) by convergecasting *within* each of its fragments. This takes $O(\sqrt{n})$ time and $O(\sqrt{n})$ messages per base fragment, leading to $O(n)$ messages overall. The $O(\sqrt{n})$ MOE edges are sent by the leaders of the respective base fragments to the root by *upcasting* (see, e.g., [47]). This takes $O(D + \sqrt{n})$ time and $O(D\sqrt{n})$ messages, as each of the at most $\sqrt{n}$ edges has to be sent along up to $D$ edges to reach the root. The root merges the fragments and sends the renamed fragment IDs to the respective leaders of the base fragments by *downcast* (which has the same time and message complexity as upcast [47]). The leaders of the base fragments broadcast the new ID to all other nodes in their respective fragments. This takes $O(\sqrt{n})$ messages per fragment and hence $O(n)$ messages overall. Thus one iteration of the merging can be done in $O(D + \sqrt{n})$ time and using $O(D\sqrt{n})$ messages. Since each iteration reduces the number of fragments by at least half, the number of iterations is $O(\log n)$. At the end of this iteration, several base fragments may share the same label. In subsequent iterations, each base fragment finds its MOE (i.e., the MOE between itself and the other base fragments which do not have the same label) by convergecasting within its own fragment and (the leader of the base fragment) sends the MOE to the root; thus $O(\sqrt{n})$ edges are sent to the root (one per base fragment), though there is a lesser number of combined fragments (with distinct labels). The root node finds the overall MOE of the combined fragments and initiates the merging. Since the time and message complexity per merging iteration is $O(D + \sqrt{n})$ and $O(D\sqrt{n}) = O(n)$, respectively, the overall complexity is still within the sought bounds.

### 5.1.2 Phase 2: When $D$ and the Number of Fragments are Large:

When $D$ is large (e.g. $n^{1/2+\varepsilon}$, for some $0 < \varepsilon \leq 1/2$) and the number of fragments is large (e.g. $\Theta(\sqrt{n})$) the previous approach of merging via the root of the

global BFS tree does not work directly, since the message complexity would be $O(D\sqrt{n})$. A second idea addresses this issue by merging in a manner that respects *locality*. That is, we merge fragments that are close by using a *local* leader, such that the MOE edges do not have to travel too far. The high-level idea is to use a *hierarchy of sparse neighborhood covers* due to [10] to accomplish the merging. [9] Intuitively, a sparse neighborhood cover is a decomposition of a graph into a set of overlapping clusters that satisfy suitable properties.

The main intuitions behind using a cover are the following: (1) the clusters of the cover have relatively smaller diameter (compared to the strong diameter of the fragment and is always bounded by $D$) and this allows efficient communication for fragments contained within a cluster (i.e., the weak diameter of the fragment is bounded by the cluster diameter); (2) the clusters of a cover overlap only a little, i.e., each vertex belongs only to a few clusters; this allows essentially congestion-free (overhead is at most polylog($n$) per vertex) communication and hence operations can be done efficiently in parallel across all the clusters of a cover. This phase continues till the number of fragments reduces to $O(n/D)$, when we switch to Phase 3. We next give more details on the merging process in Phase 2.

**Communication-Efficient Paths**. An important technical aspect in the merging process is constructing efficient communication paths between nearby fragments, which are maintained and updated by the algorithm in each iteration. The algorithm requires fragments to be communication-efficient, in the sense that there is an additional set of *short paths* between the fragment leader $f$ and fragment members, which may use "shortcuts" through vertices that are not part of the fragment to reduce the distance. A fragment $F$ is *communication-efficient* if, for each vertex $v \in F$, there exists a path between $v$ and $f$ (possibly including vertices in $V(G) \setminus V(F)$) of length $O(\mathsf{diam}_G(F) + \sqrt{n})$, where $diam_G(F)$ is the weak diameter of $F$.

It can be shown that, in each iteration, all fragments find their respective MOE in time $\tilde{O}(\sqrt{n} + D)$ and $\tilde{O}(m)$ messages. Note that it is difficult to merge all fragments along their respective MOE, as this might create long fragment-chains. Instead, [45] compute a maximal matching in the fragment graph $\mathcal{F}_i$ induced by the MOE edges to merge fragments in a controlled manner. The crucial part is to use time- and message communication between the fragment leader and the nodes in the fragment (while finding MOEs) as well as between fragment leaders of adjacent fragments, which is made possible by the existence of the communication-efficient paths.

[9] Neighborhood covers were used by Elkin [10] to improve the running time of the Pipeline procedure of his distributed MST algorithm; on the other hand, in [45] they are used to replace the Pipeline part entirely.

In more detail, [45] use a hierarchy of sparse neighborhood covers to construct communication-efficient fragments. Each cover in the hierarchy consists of a collection of clusters of a certain radius, where the lowest cover in the hierarchy contains clusters of radius $O(\sqrt{n})$ and subsequent covers in the hierarchy have clusters of geometrically increasing radii, whereby the last cover in the hierarchy is simply the BFS tree of the entire graph. Initially, it is easy to construct communication-efficient paths in base fragments, since they have strong diameter $O(\sqrt{n})$. In subsequent iterations, when merging adjacent fragments, the algorithm finds a cluster that is (just) large enough to contain both fragments. The neighborhood property of the cluster allows the algorithm to construct communication-efficient paths between merged fragments (that might take shortcuts outside the fragments, and hence have small *weak diameter*), assuming that the fragments before merging are efficient. Note that it is important to make sure that the number of fragments in a cluster is not too large in relation to the radius of the cluster—otherwise the message complexity would be high. Hence, a key invariant maintained through all the iterations is that the *cluster depth times the number of fragments that are contained in the cluster of such depth is always bounded by* $\tilde{O}(n)$. This invariant is maintained by making sure that the number of fragments per cluster reduces sufficiently to compensate for the increase in cluster radius: At the end of the phase, when the cluster radius is $D$, the number of fragments is $\tilde{O}(n/D)$.

### 5.1.3 Phase 3: When the Cluster Radius is $D$:

Phase 3 employs a merging procedure similar to the one used in Phase 1. Recall that, in Phase 1, in every merging iteration, each base fragment finds their respective MOEs (i.e., MOEs between itself and the remaining fragments) by convergecasting to their respective leaders; the leaders send at most $O(\sqrt{n})$ edges to the root by upcast. The root merges the fragments and sends out the merged information to the base fragment leaders. In Phase 3, it is possible to treat the $O(n/D)$ fragments remaining at the end of Phase 2 as the "base fragments" and repeat the above process. An important difference to Phase 1 is that the merging leaves the leaders of these base fragments intact: in the future iterations of Phase 3, each of these base fragments again tries to find an MOE, whereby only edges that have endpoints in fragments with distinct labels are considered as candidate for the MOE.

Note that the fragment leaders communicate with their respective nodes as well as the BFS root via the hierarchy of communication-efficient routing paths constructed in Phase 2; these incur only a polylogarithmic overhead. This takes $\tilde{O}(D + n/D)$ time (per merging iteration) since $O(n/D)$ MOE edges are sent to the root of the BFS tree via communication-efficient paths (in every merging iteration) and a message complexity of $\tilde{O}(D \cdot n/D) = \tilde{O}(n)$ (per merging iteration) since, in

each iteration, each of the $O(n/D)$ edges takes $\tilde{O}(D)$ messages to reach the root. Since there are $O(\log n)$ iterations overall, the overall bound follows.

## 5.2 Deterministic Singularly-Optimal Algorithms

Shortly after the work of Pandurangan et al. [45], Elkin[12] presented a new simultaneously optimal distributed MST algorithm which 1) is deterministic, thus answering an explicit question raised in [45], and 2) is simpler and allows for a simpler analysis [12].

The main novelty in Elkin's algorithm is to grow fragments up to (strong) diameter $O(D)$, as opposed to $O(\sqrt{n})$, in the first phase of the algorithm. This results in an $(O(n/D), O(D))$-MST forest as the base forest, as opposed to an $(O(\sqrt{n}), O(\sqrt{n}))$-MST forest. The algorithm in [45] is then executed on top of this base forest.

This simple change brings benefits to the case $D \geq \sqrt{n}$, for which now the complexities of finding minimum-weight outgoing edges and of their subsequent upcast to the root of the auxiliary BFS tree of the network are within the desired time and message bounds. Hence Phase 2 of Part 2 of Pandurangan et al.'s algorithm can be bypassed, and since this phase contains the sole randomized portion of the algorithm (that is, the randomized cover construction of [10]), the final result is a fully deterministic algorithm.

A different approach toward singular-optimality has been given recently by Haeupler et al. [21], who provide a new deterministic distributed algorithm for MST which is simultaneously time- and message-optimal, and also achieve the same goal for some other problems.

# 6 Time and Message Lower Bounds

In this section, we provide some intuition behind the complexity lower bounds that have been shown for the distributed MST problem.

**The Peleg-Rubinovich Lower Bound.** In a breakthrough result, Peleg and Rubinovich [48] showed that $\Omega(D + \sqrt{n}/B)$ time is required by any distributed algorithm for constructing an MST, even on networks of small diameter (at least $\Omega(\log n)$), when assuming that at most $B$ bits can be sent across an edge in a round. As a consequence, this result established the asymptotic (near) time-optimality of the algorithm of [36]. Peleg and Rubinovich introduced a novel graph construction that forces any algorithm to trade-off dilation with communication bottlenecks. While the original lower bound of Peleg and Rubinovich applies to deterministic algorithms for exact MST computation, the more recent work of [11, 7] extend

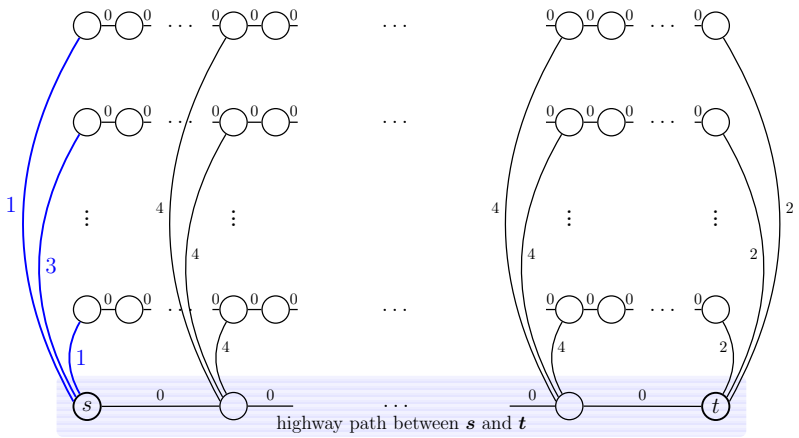


Figure 1: The simplified Peleg-Rubinovich lower bound graph of [48]. The shaded bottom path between vertices $s$ and $t$ represents the *highway path* and the rest of the other horizontal paths between the two nodes represent the vertex-disjoint *long paths*. The weight of each blue edge incident to $s$ is chosen to be either 1 or 3. The *spoke edges* connect each highway vertex to the appropriate vertices of each long path and have weight 4.

this result by showing a lower bound of $\tilde{\Omega}(D + \sqrt{n})$ for randomized (Monte Carlo) and approximation algorithms as well.

The (simplified) lower bound graph construction considers two designated nodes, $s$ and $t$, which are connected by $\Theta(\sqrt{n})$ vertex-disjoint *long paths*, each of length $\Theta(\sqrt{n})$, and one *highway path* consisting of $D = \Theta(n^{1/4})$ *highway vertices*, which determines the graph diameter. To ensure that the resulting graph does indeed have diameter $O(D)$, *spoke edges* are added from each $i$-th highway vertex to the respective $(iD)$-th vertex of each long path. Figure 1 depicts the weighted variant of the above graph; the assignment of the weights is explained below.

So far, the constructed graph $\bar{G}$ is unweighted and, as an intermediate step, [48] consider the *mailing problem* on $\bar{G}$, where node $s$ is given as input a $\Theta(\sqrt{n})$-length bit string. An algorithm solves the mailing problem if eventually node $t$ outputs the same bit string. The way $\bar{G}$ is constructed, one strategy is to send the input bits held by $s$ in parallel along the $\Theta(\sqrt{n})$ long paths; another strategy is to send them along the (single) highway path of length $\Theta(n^{1/4})$ towards $t$, which means the bits are essentially forced to travel in sequence since each (highway) edge can carry at most $B$ bits per round. Of course, we cannot make any assumptions on how exactly these bits are being sent from $s$ to $t$: In fact, an algorithm is not bound to send these bits explicitly (i.e. one by one) along the edges but might attempt to somehow compress the bit string to speed up the process. To

rule out such shortcuts, [48] show that any algorithm that correctly solves the mailing problem, must ensure that the size of the possible state space of node $t$ is large enough upon termination. That is, since there are $2^{\Theta(\sqrt{n})}$ choices for the input bit string given to $s$, the number of distinct states that node $t$ can be in upon termination must also be at least $2^{\Theta(\sqrt{n})}$. Otherwise, there are at least 2 distinct input bit strings of $s$ for which $t$ is in the same state upon termination and hence $t$ (wrongly) outputs the same bit string for both inputs. Initially, only $s$ holds the input string and hence all other nodes (including $t$) have a state space of size 1. Taking into account the possibility of remaining silent, there are $2^B + 1$ possible ways that any given edge can be used in any given round. It follows that, after $r$ rounds, the "information" delivered by the highway edges can increase the state space by a factor of at most $(2^B + 1)^r$. Given that $r \geq \Theta(\sqrt{n}/B)$ must hold upon termination, [48] obtain an $\Omega(\sqrt{n}/B)$ lower bound for the mailing problem. They also extend the above idea to graphs of diameter as small as $\Theta(\log n)$.

To obtain a time lower bound for the MST problem, Peleg and Rubinovich consider a weighted version of the graph, where all path edges have weight 0 and all spoke edges are given weight 4. The edges connecting $t$ to the paths have weight 2, whereas the weights of edges incident to $s$ have weight either 1 or 3. The crucial property of this construction is that node $t$ must add to the MST edges its edge incident to the $j$-th slow path if and only if the weight of the edge that connects $s$ to the $j$-th slow path is 3. See Figure 1 for an example. Intuitively speaking, this means that node $t$ can only add the correct MST edges if there is an algorithm that solves the corresponding mailing problem, hence implying the sought $\Omega(D + \sqrt{n}/B)$ time lower bound.

**The Das Sarma et al. [7] Lower Bound.** The work of Das Sarma et al.[7] established that $\tilde{\Omega}(D + \sqrt{n})$ rounds is a fundamental time lower bound for several important distributed network problems including MST, shortest paths, minimum cut etc., and showed that this lower bound holds even for Monte-Carlo randomized algorithms. More importantly, they show that the bound holds for approximation algorithms as well, i.e., those that compute a solution that is within any non-trivial approximation factor to the optimal.

The approach of Das Sarma et al is to prove lower bounds by establishing an interesting connection between *communication complexity* and distributed computing. A key concept that facilitates this connection is the notion of *distributed verification*. In distributed verification, we are given a network $G$ and a subgraph $H$ of $G$ where each vertex of $G$ knows which edges incident on it are in $H$. The goal is to verify whether $H$ has some properties, e.g., if it is a tree or if it is connected (every node knows at the end of the process whether $H$ has the specified property or not). Das Sarma et al. [7] initiate a systematic study of dis-

tributed verification, and give almost tight lower bounds on the running time of distributed verification algorithms for many fundamental problems such as connectivity, spanning connected subgraph, and $s - t$ cut verification. They show applications of these results in deriving time lower bounds on the *hardness of distributed approximation* for many classical optimization problems including MST, shortest paths, and minimum cut.

The lower bound proofs consider the family of graphs (similar to the one in Figure 1) that was first used in [48, 10, 38]. However, while previous results [48, 10, 38, 31] rely on counting the number of states needed to solve the *mailing problem* (along with some sophisticated techniques for its variant, called *corrupted mailing problem*, in the case of approximation algorithm lower bounds), and use Yao's method [51] to derive lower bounds for randomized algorithms, the results of [7] used three main steps of simple reductions, starting from problems in communication complexity, as follows:

In the first step, they reduce the lower bounds of problems in the standard communication complexity model [33] to the lower bounds of the equivalent problems in the "distributed version" of communication complexity. Specifically, they prove the *Simulation Theorem* [7] which relates the *communication* lower bound from the standard communication complexity model [33] to compute some appropriately chosen function $f$, to the distributed *time* complexity lower bound for computing the same function in a specially chosen graph $G$. In the standard two-party communication complexity model[33], Alice and Bob can communicate directly (via a bidirectional edge of bandwidth one). In the distributed model, we assume that Alice and Bob are some vertices of $G$ and they together wish to compute the function $f$ using the communication graph $G$. The choice of graph $G$ is critical and is the same as the graph used by Peleg and Rubinovich [48] (see Figure 1).

The connection established in the first step allows one to bypass the state counting argument and Yao's method, and reduces the task in proving lower bounds of *distributed verification problems* (see below) to merely picking the right function $f$ to reduce from. The function $f$ that is useful in showing randomized lower bounds for many problems (including MST) is the *set disjointness function* [33], which is the quintessential problem in the world of communication complexity with applications to diverse areas. Following a result well known in communication complexity [33], they show that the distributed version of the set disjointness problem has an $\tilde{\Omega}(\sqrt{n})$ lower bound on graphs of small diameter.

The second step is to reduce the concerned problem (such as MST or shortest paths) to an appropriate *verification problem* using simple reductions similar to those used in data streams [26]. The verification problem that is appropriate for MST is the *spanning connected subgraph verification* problem, where the goal is to verify if a given subgraph $H$ of a graph $G$ is spanning and connected.

Finally, in the third step, they reduce the verification problem to hardness of

distributed approximation for a variety of problems to show that the same lower bounds hold for approximation algorithms as well. For this, they use a reduction whose idea is similar to the one used to prove hardness of approximating TSP (Traveling Salesman Problem) on general graphs (see, e.g., [50]): Convert a verification problem to an optimization problem by introducing edge weights in such a way that there is a large gap between the optimal values for the cases where $H$ satisfies (or does not satisfy) a certain property. This technique is surprisingly simple, yet yields strong unconditional hardness bounds — many hitherto unknown, left open (e.g., minimum cut) [9] and some that improve over known ones (e.g., MST and shortest path tree) [10]. As mentioned earlier, this approach shows that approximating MST by *any* factor needs $\tilde{\Omega}(\sqrt{n})$ time, while the previous result due to Elkin gave a bound that depends on $\alpha$ (the approximation factor), i.e. $\tilde{\Omega}(\sqrt{n/\alpha})$, using more sophisticated techniques.

To summarize, the lower bound proof for MST that applies to Monte Carlo randomized and approximation algorithms proceeds as follows. It starts by reducing the set disjointness problem in the standard two party communication complexity setting to the distributed version of the set disjointness problem which has to solved by communicating in a graph (the one in Figure 1). Then the distributed version is in turn reduced to the spanning connected subgraph verification problem which is then finally reduced to MST or approximate MST. Hence starting with the lower bound of the set disjointness problem (which is already known in communication complexity) we obtain a lower bound for MST.

The lower bound proof technique via this approach is quite general and conceptually straightforward to apply as it circumvents many technical difficulties by using well-studied communication complexity problems. Nevertheless, this approach yields tight lower bounds for many problems. More specifically, a significant advantage of this technique over previous approaches is that it avoids starting from scratch every time we want to prove a lower bound for a new problem. For example, extending the result from the mailing problem in [48] to the corrupted mailing problem in [10] requires some sophisticated techniques. The new technique, on the other hand, allows one to use known lower bounds in communication complexity to streamline such tasks. Another advantage of the approach by [7] is that extending a deterministic lower bound to a randomized one is sometimes technically difficult, when all that is available is a deterministic bound. By using connection to communication complexity, the technique of [7] allows one to obtain lower bounds for *Monte Carlo* randomized algorithms, while previous lower bounds hold only for Las Vegas randomized algorithms. The general approach of Das Sarma et al has been subsequently used to show many new lower bounds for other distributed graph algorithms.

**A Simultaneous Lower Bound.** The prior time and message lower bounds for MST are derived using two completely different graph constructions; the existing lower bound construction that shows one lower bound does not work for the other. Specifically, the existing graph constructions of [11, 7] used to establish the lower bound of $\tilde{\Omega}(D + \sqrt{n})$ rounds does not simultaneously yield the message lower bound of $\Omega(m)$; similarly, the existing lower bound graph construction of [35] that shows the message lower bound of $\Omega(m)$ (cf. Section 3.1) does not simultaneously yield the time lower bound of $\tilde{\Omega}(D + \sqrt{n})$.

In [45], the authors combine the two lower bound techniques — hardness of distributed symmetry breaking, used to show the message lower bound of $\Omega(m)$ for the $KT_0$ model [35], and communication complexity, used to show the lower bound on time complexity [7] — to show that, for any algorithm, there is some graph where *both* the message and time lower bounds hold.

# 7 Conclusion and Open Problems

This article surveyed the distributed MST problem, a cornerstone problem in distributed network algorithms. It has received a rich and continuing interest since the very first distributed MST algorithm was published more than 35 years ago.

On the algorithmic side, there has been a lot of work in trying to improve the time complexity and more recently both time and message complexity with the goal of obtaining singularly optimal algorithms. An important conceptual contribution that research in distributed MST introduced is the notion of efficiency with respect to specific graph parameters such as $D$, the network diameter. This notion led to development of new algorithms — starting with that of [16] and leading to the more recent singularly-optimal algorithms of [45, 12, 21]. More importantly, it also led to a deeper understanding of lower bounds in distributed network algorithms. The time lower bound of $\tilde{\Omega}(D + \sqrt{n})$ for MST first shown by Peleg and Rubinovich [48] led to similar lower bounds for distributed network algorithms for various other problems, including for randomized algorithms and approximation algorithms [7].

An open question is whether there exists a distributed MST algorithm with near-optimal time and message complexities in the $KT_1$ variant of the model, i.e., using $\tilde{O}(n)$ messages and $\tilde{O}(D + \sqrt{n})$ time. Very recently there has been some progress on this question [20]. Another interesting question is improving the performance of MST in special classes of graphs, e.g., in graphs of diameter two (where it is not known if one can get $o(\log n)$-round algorithms). There has been recent progress in this direction (cf. Section 3.1).

Currently, it is not known whether other important problems such as shortest paths, minimum cut, and random walks, enjoy singular optimality. These prob-

lems admit distributed algorithms which are (essentially) time-optimal but not message-optimal [41, 25, 8, 42]. Some work in this direction recently started to appear [21], but further research is needed to address these questions.

# References


[1] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012.

[2] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM Symposium on Principles of Database Systems (PODS)*, pages 5–14, 2012.

[3] 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, 1987.

[4] B. Awerbuch, O. Goldreich, D. Peleg, and R. Vainish. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, 1990.

[5] O. Borůvka. O jistém problému minimálním (about a certain minimal problem). *Práce Mor. Prírodoved. Spol. v Brne III*, 3, 1926.

[6] F. Chin and H. 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.

[7] 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. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

[8] A. Das Sarma, D. Nanongkai, G. Pandurangan, and P. Tetali. Distributed random walks. *J. ACM*, 60(1), 2013.

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

[10] M. Elkin. A faster distributed protocol for constructing a minimum spanning tree. *J. Comput. Syst. Sci.*, 72(8):1282–1308, 2006.

[11] M. Elkin. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM J. Comput.*, 36(2):433–456, 2006.

[12] M. Elkin. A simple deterministic distributed MST algorithm, with near-optimal time and message complexities. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 157–163, 2017.

[13] M. Faloutsos and M. Molle. A linear-time optimal-message distributed algorithm for minimum spanning trees. *Distributed Computing*, 17(2):151–170, 2004.

[14] 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.

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

[16] J. A. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302–316, 1998.

[17] M. Ghaffari and B. Haeupler. Distributed algorithms for planar networks II: low-congestion shortcuts, MST, and min-cut. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 202–219, 2016.

[18] M. Ghaffari, F. Kuhn, and H.-H. Su. Distributed MST and routing in almost mixing time. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 131–140, 2017.

[19] M. Ghaffari and M. Parter. MST in log-star rounds of congested clique. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 19–28, 2016.

[20] R. Gmyr and G. Pandurangan. Time-message tradeoffs in distributed algorithms. In *Manuscript*, 2018.

[21] B. Haeupler, D. E. Hershkowitz, and D. Wajc. Round- and message-optimal distributed graph algorithms. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, 2018. To appear.

[22] B. Haeupler, T. Izumi, and G. Zuzic. Low-congestion shortcuts without embedding. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 451–460, 2016.

[23] B. Haeupler, T. Izumi, and G. Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, pages 158–172, 2016.

[24] J. W. Hegeman, G. Pandurangan, S. V. Pemmaraju, V. B. Sardeshmukh, and M. Scquizzato. Toward optimal bounds in the congested clique: graph connectivity and MST. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 91–100, 2015.

[25] M. Henzinger, S. Krinninger, and D. Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC)*, pages 489–498, 2016.

[26] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In J. M. Abello and J. S. Vitter, editors, *External memory algorithms*, pages 107–118. American Mathematical Society, Boston, MA, USA, 1999.

[27] T. Jurdzinski and K. Nowicki. MST in $O(1)$ rounds of congested clique. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2620–2632, 2018.

[28] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. *Distributed Computing*, 20(6):391–402, 2008.

[29] V. King, S. Kutten, and M. Thorup. Construction and impromptu repair of an MST in a distributed network with $o(m)$ communication. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 71–80, 2015.

[30] H. Klauck, D. Nanongkai, G. Pandurangan, and P. Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–410, 2015.

[31] L. Kor, A. Korman, and D. Peleg. Tight Bounds For Distributed MST Verification. In *STACS*, pages 69–80, 2011.

[32] L. Kor, A. Korman, and D. Peleg. Tight bounds for distributed minimum-weight spanning tree verification. *Theory Comput. Syst.*, 53(2):318–340, 2013.

[33] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, New York, NY, USA, 1997.

[34] S. Kutten, D. Nanongkai, G. Pandurangan, and P. Robinson. Distributed symmetry breaking in hypergraphs. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 469–483, 2014.

[35] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. On the complexity of universal leader election. *J. ACM*, 62(1), 2015.

[36] S. Kutten and D. Peleg. Fast distributed construction of small $k$-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

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

[38] Z. Lotker, B. Patt-Shamir, and D. Peleg. Distributed MST for constant diameter graphs. *Distributed Computing*, 18(6):453–460, 2006.

[39] A. Mashreghi and V. King. Time-communication trade-offs for minimum spanning tree construction. In *Proceedings of the 18th International Conference on Distributed Computing and Networking (ICDCN)*, 2017.

[40] A. McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

[41] D. Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*, pages 565–573, 2014.

[42] D. Nanongkai, A. D. Sarma, and G. Pandurangan. A tight unconditional lower bound on distributed randomwalk computation. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 257–266, 2011.

[43] G. Pandurangan. *Distributed Network Algorithms*. 2018. `https://sites.google.com/site/gopalpandurangan/dna`.

[44] G. Pandurangan, P. Robinson, and M. Scquizzato. Fast distributed algorithms for connectivity and MST in large graphs. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 429–438, 2016.

[45] G. Pandurangan, P. Robinson, and M. Scquizzato. A time- and message-optimal distributed algorithm for minimum spanning trees. In *Proceedings of the 49th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 743–756, 2017.

[46] D. Peleg. Distributed matroid basis completion via elimination upcast and distributed correction of minimum-weight spanning trees. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 164–175, 1998.

[47] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

[48] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.

[49] R. E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[50] V. V. Vazirani. *Approximation Algorithms*. Springer, July 2001.

[51] A. C.-C. Yao. Probabilistic Computations: Toward a Unified Measure of Complexity. In *FOCS*, pages 222–227, 1977.