Distributed Minimum Degree Spanning Trees

Michael Dinitz
Johns Hopkins University
Baltimore, Maryland
mdinitz@cs.jhu.edu

Magnús M. Halldórsson
Reykjavik University
Reykjavik, Iceland
mmh@ru.is

Taisuke Izumi
Nagoya Institute of Technology
Nagoya, Japan
t-izumi@nitech.ac.jp

Calvin Newport
Georgetown University
Washington, D.C.
cnewport@cs.georgetown.edu

ABSTRACT

The *minimum degree spanning tree* (MDST) problem requires the construction of a spanning tree T for graph G, such that the maximum degree of T is the smallest among all spanning trees of G. Let d be this MDST degree for a given graph. In this paper, we present a randomized distributed approximation algorithm for the MDST problem that constructs a spanning tree with maximum degree in $O(d + \log n)$. With high probability in n, the algorithm runs in $O((D + \sqrt{n}) \log^4 n)$ rounds, in the broadcast-CONGEST model, where D is the graph diameter and n is the graph size. We then show how to derandomize this algorithm, obtaining the same asymptotic guarantees on degree and time complexity, but now requiring the standard CONGEST model. Although efficient approximation algorithms for the MDST problem have been known in the sequential setting since the 1990's (finding an exact solution is NP-hard), our algorithms are the first efficient distributed solutions. We conclude by proving a lower bound that establishes that any randomized MDST algorithm that guarantees a maximum degree in $\tilde{\Omega}(d)$ requires $\tilde{\Omega}(n^{1/3})$ rounds, and any deterministic solution requires $\tilde{\Omega}(n^{1/2})$ rounds. These bounds proves our deterministic algorithm to be asymptotically optimal, and eliminates the possibility of significantly more efficient randomized solutions.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**; *Graph algorithms analysis.*

KEYWORDS

CONGEST, spanning tree, minimum degree

ACM Reference Format:
Michael Dinitz, Magnús M. Halldórsson, Taisuke Izumi, and Calvin Newport. 2019. Distributed Minimum Degree Spanning Trees. In *2019 ACM Symposium on Principles of Distributed Computing (PODC '19), July 29-August 2, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3293611.3331604

PODC '19, July 29-August 2, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6217-7/19/07…$15.00
https://doi.org/10.1145/3293611.3331604

1 INTRODUCTION

We present a distributed approximation algorithms for the *minimum degree spanning tree* (MDST) problem. In this problem, we are given a graph $G = (V, E)$, and are asked to construct a spanning tree T of V such that the maximum degree of T is the smallest among all spanning trees of G.

As argued in [14, 15], in addition to their theoretical interest, these trees are particularly useful in network communication scenarios in which low-degree backbones reduce overhead. In designing optical networks, for example, high-degree switches are expensive, prioritizing the identification of low-degree spanning structures. These trees can also be used to maximize throughput in network settings in which devices can only transmit or receive messages along a single link at a time, such as when aggregating data in a sensor network, or flooding information in peer-to-peer scenarios where devices connect in a pairwise manner.

In the *centralized* setting, the problem is easily shown to be NP-hard by reduction from the Hamiltonian path problem. The best known approximation is due to Fürer and Raghavachari [15], who provide a polynomial-time algorithm that constructs a spanning tree with maximum degree $d + 1$, where d is the minimum maximum degree over all spanning trees in the graph. As we elaborate below, to the best of our knowledge, there exists no efficient *distributed* approximation algorithm for the MDST problem, despite its importance to network communication. This paper addresses this gap.

1.1 Results

We assume the CONGEST model of distributed computation. In this model, the network is described as an n-node graph $G = (V, E)$, with a computational process assigned to each node, and the edges representing communication channels. Time proceeds in synchronous rounds. In each round, each node can send a $O(\log n)$-bit message to each of its neighbors in the graph. Many of our results (including our main algorithm) hold in the harder *broadcast* variation of the CONGEST model (broadcast-CONGEST) in which nodes must broadcast the *same* message to all of their neighbors in a given round, whereas our lower bounds hold for the easier standard CONGEST model.

In the statement of our results, we let d denote the degree of an MDST (i.e., OPT) and let D denote the graph diameter. Throughout this paper we omit some details due to space constraints, but all missing details can be found in the full version [9].

Main Algorithm. Our main result is the following.

Theorem 1.1. *There exists a (randomized) distributed algorithm in the broadcast-CONGEST model that constructs a spanning tree of degree $O(d + \log n)$, running in time $O((D + \sqrt{n}) \log^4 n)$ (in expectation).*

A nice property of this solution is that it is an iterative improvement algorithm, meaning that it strictly improves the degree of the current spanning tree with each phase until terminating with a degree at most $O(d + \log n)$. This means the process can be stopped once the degree is sufficiently small for a given application, potentially saving time for applications with looser degree requirements.

The basic idea of this new algorithm is to parallelize a large number of the improvement steps used in one of the original sequential solutions [14]. This alone, however, is not enough, as the number of such improvements is bounded by $O(n^2)$ [14, 22], meaning that a straightforward parallel approach might still be too slow. Our method instead enables many nodes to simultaneously make *large* improvements to their degrees in a short period of time, progressively lowering the threshold for a viable improvement size.

Derandomization. We also show that it is possible to derandomize this algorithm, at the cost of working in the CONGEST model (rather than broadcast-CONGEST) and a slightly worse logarithmic factor. The main idea is to generalize the derandomization techniques of Fischer [11] from ordinary matchings to the more complicated versions of matchings that we use in our randomized algorithm. This allows us to prove the following theorem:

Theorem 1.2. *There is a deterministic distributed algorithm in the CONGEST model that finds a spanning tree of degree $O(d + \log n)$, running in time $O((D + \sqrt{n}) \log^5 n)$.*

Lower Bound. We show that the polynomial term in the time complexity is necessary. Specifically, there exists a family of instances of diameter $O(\log n)$ which incurs $\tilde{\Omega}(n^{1/3})$ rounds for any (randomized) MDST algorithm achieving a *polylogarithmic multiplicative approximation*. That lower bound is improved to $\tilde{\Omega}(n^{1/2})$ rounds for any deterministic algorithm, and even the randomized bound can be made arbitrarily close to the bound of $\tilde{\Omega}(n^{1/2})$ rounds if we consider instances of larger diameters. We prove the following theorem:

Theorem 1.3. *For any $\epsilon < 1/6$, there exists a family of instances of diameter $D = \Theta(n^{1/2-3\epsilon} + \log n)$ where any MDST algorithm with a polylogarithmic multiplicative approximation factor needs $\tilde{\Omega}(n^{1/2-\epsilon} + D)$ rounds. There also exists a family of the instances of diameter $D = O(\log n)$ where any deterministic MDST algorithm with a polylogarithmic multiplicative approximation factor needs $\tilde{\Omega}(n^{1/2})$ rounds.*

Note that a more precise version of the lower-bound theorem (Theorem 5.1) also includes the tradeoffs betweeen time and approximation factors.

1.2 Background & Related Work.

The construction of spanning trees with useful properties is a primary topic in distributed graph algorithms. The most well-studied problem in this area is the *minimum spanning tree* (MST) problem, which requires the construction of a spanning tree that minimizes the sum of edge weights. Gallager, Humblet, and Spira [17] helped instigate this area with a distributed algorithm that constructs an MST in $O(n \log n)$ rounds (similar ideas appeared in a 1926 paper by Boruvka [23] that was not translated into English until more recently). A series of follow up papers [2, 8, 16] improved this complexity to $O(n)$ rounds, which is worst-case optimal in the sense that $\Omega(n)$ rounds are required in certain graphs with diameter $D = \Theta(n)$.

Garay, Kutten and Peleg [18] isolated the graph diameter D as a distinct parameter, enabling further progress. They described a distributed MST algorithm that solves the problem in $O(D + n^{0.61})$ rounds, which is sub-linear for graphs with sub-linear diameters. This result was subsequently improved to $O(D + \sqrt{n} \log^* n)$ rounds [21]. A series of lower bound results [10, 24, 25] established that any non-trivial approximation of an MST requires $\Omega(D + \sqrt{n/\log n})$ rounds, even in graphs with small diameters.

Turning our attention from MST to MDST, we note that prior to this paper the MDST problem has been primarily studied in the context of sequential algorithms. In 1990, Fürer and Raghavachari [13] gave an *NC* algorithm (and thus a polynomial-time sequential algorithm) that constructs a tree with a maximum degree $O(d \log n)$ (recall that d is the maximum degree of the optimal tree). Agrawal, Klein and Ravi [1] subsequently generalized this result by providing a sequential polynomial time algorithm for the Steiner tree variation of the MDST problem. Fürer and Raghavachari improved both results by presenting sequential algorithms that guarantee a maximum degree of $d + 1$ for both the standard [14] and Steiner tree [15] versions of the problem. Given that finding a spanning tree with maximum degree exactly d is NP-hard, these approximations are likely the best possible that can be achieved in polynomial time.

The $(d + 1)$-approximation algorithm of [14] operates by performing repeated improvements on an arbitrary initial tree. Each improvement involves a recursive application, making any parallel or distributed implementation highly challenging, if not impossible. We refer to these recursive improvements as *semi-local*. In the same paper, Fürer and Raghavachari [14] also gave a simpler algorithm involving *local* improvements. They showed that this algorithm terminates with a tree of maximum degree $2d + \log n$. They proved an upper bound of $O(n^3)$ on the number of improvement steps, which implies polynomial time complexity. Our algorithm leverages these local improvements, but chooses them more selectively to enable more parallelism and to significantly reduce the total number of improvements needed. It remains a key open question whether an efficient distributed solution exists that can provide a tree with degree in $O(d)$. Given the difficulty of parallelizing the semi-local improvements of [14], resolving this question would likely require novel techniques.

To the best of our knowledge, the first discussion of the MDST problem in the distributed context was by Blin and Butelle [3, 4], who observed that the search for improvements in [14] can be implemented in distributed models using global broadcast and aggregations. The time complexity of their solution, however, is in

$\Omega(n^2)$, leaving open the problem of identifying efficient distributed approximation algorithms for the problem.

Followup work along these lines [5, 6] focus on self-stabilization and/or space complexity, which is important but orthogonal to time complexity considerations. Indeed, the time complexity claimed for these self-stabilizing MDST algorithms is $O(mn^2 \log n)$ [6] and polynomial [5]. Another direction is the more general problem of finding a *minimum* spanning tree whose maximum degree is minimized. Lavault and Valencia-Pabon [22] give an algorithm with time complexity $O(\Delta n^{2+1/\ln b})$ that outputs a tree of degree $bd + \log_b n$, for any $b > 1$. In summary, none of this existing work provides a distributed MDST algorithm running in time less than n^2 (which is the complexity boundary beyond which it is possible to simply gather the entire graph topology at each node and run a centralized algorithm locally), supporting our claim that our $\tilde{O}(D + \sqrt{n})$-time algorithm is the first efficient distributed solution for this problem.

2 ITERATIVE IMPROVEMENT ALGORITHM

We give an iterative improvement algorithm in broadcast-CONGEST (see Section 1 for model definitions) that produces a spanning tree of degree at most $4(1+\epsilon)d + O(\log n)$. Recall that d is the degree of an optimal MDST.

Overview. A natural approach for finding low-degree spanning trees is to incrementally reduce the number of high degree nodes in an arbitrary initial spanning tree T. Adding a graph edge $e = (u, v)$ to T introduces a cycle C, and removing any edge in C results again in a spanning tree. If C contains a degree-k vertex w while the degrees of both u and v are at most $k-2$, then replacing one of the incident edges on w with e reduces the number of nodes of degree k (or more) in the tree.

The first step towards a distributed algorithm is to *parallelize* the search for improvements. We then seek a set M of graph edges so that there is a subset M' of tree edges for which $(T - M') \cup M$ is again a tree. This can be a tricky process in full generality, both in identifying the edges M' to replace and in maximizing $|M|$, the number of parallel improvements. We tackle this by restricting the set of improvements that can be identified, without reducing significantly the extent of the improvements. In particular, we restrict the improvement set so that each *improving* edge in M has an easily identifiable corresponding *replaced* edge in T.

The other major issue is to find sufficiently significant improvements in each parallel phase. The $O(n^3)$ bound on the number of improvement steps derived by Fürer and Raghavachari [14] suggests that the basic improvements alone form an insufficient basis for a distributed algorithm. Instead, we want to replace edges to high degree nodes by edges between nodes of quite low degree. We also want to allow high degree nodes to shed multiple edges in a single parallel phase. As part of this, we develop a strategy of *progressive* improvements, where nodes initially reduce their degree rapidly, while in later stages the degrees decrease more slowly. By defining a sequence of potential functions, we can prove a polylogarithmic bound on the number of phases.

In the next subsection, we give a matching-based algorithm to find parallel improvements, and show that under appropriate assumptions, there will be many improvements. We apply this algorithm systematically in Section 2.2 to gradually apply all significant improvements in a polylogarithmic number of calls. Distributed and parallel implementations are given in Section 2.4, but the main technical difficulty, of finding "constrained" matchings, is deferred to Section 3.

2.1 Parallel Improvements

We argue in this subsection that many improvements can be made in parallel, under the right conditions. Let γ and q be numbers, and let $\gamma_0 = \gamma - 2q$. We aim to reduce the number of vertices of degree γ or more, but only by increasing the degrees of nodes of degree less than γ_0. As mentioned, we need to be careful about how we define and find the possible parallel improvements.

Definitions. Let T be the current spanning tree and let $d_T(v)$ denote the degree of node v in T. Let X_t denote the set of nodes of degree at least t, for integer t. Removing the nodes in X_γ from T results in a collection of subtrees that we call *branches*. A branch is a *leaf branch* if only one edge in T has a single endpoint in the branch (i.e., the other endpoint has degree γ or more). The *head* of a leaf branch is the node in the branch that is incident on the edge from T that was removed. The *parent* of a leaf branch is the unique node outside the branch that is adjacent to its head. Leaf branches with the same parent are collectively called a *bundle*.

An edge in $G \setminus T$ is *good* if its endpoints are: of degree less than γ_0, in different branches, and with at least one in a leaf branch.

Definition 2.1. The *imp graph* $H = H_{T,\gamma,q}$ is a structured bipartite graph $H = (U, \mathcal{U}, Q, E')$ with sides U and Q, where U is the set of leaf branches, Q consists of the low-degree nodes, and $\mathcal{U}$ is the partition of U into bundles. There is an edge uq in E', for $q \in Q$, if the leaf branch $u \in U$ contains a node w such that wq is a good edge (in G). If both endpoints of a good edge are in leaf branches, then the edge contributes two edges to H.

Consider Figure 1 for illustration on a single bundle (with f as parent). There are four leaf branches given by $U = \{c, e, g, h\}$, while node i is in a non-leaf branch. All the nodes except f are of low degree (defined here as less than 4). The corresponding imp graph on the right has U on the left, and the low degree nodes (excluding the zero-degree nodes a, c) on the right. The edge hi is the only good edge that appears only once in H; for instance, the good edge gd in G leads to the edges $\mathbf{g}d$ and $\mathbf{e}g$ in H.

Notice that we overload notation by speaking of edges of G being contained in H; formally speaking, of course, they have a natural correspondence to edges in H.

Definition 2.2. A *constrained q-matching* M is a subgraph of H such that each bundle in $\mathcal{U}$ is incident on at most q edges in M. Namely, M satisfies: a) Each node in U is incident on at most one edge in M, b) Each node of Q is incident on at most q edges in M, and c) Each bundle $U_i \in \mathcal{U}$ is incident on at most q edges in M.

The corresponding optimization problem is to maximize the number of edges in the constrained matching found. In Section 3, we describe and analyze a distributed algorithm that efficiently approximates a constrained q-matching. This algorithm is a key subroutine in the MDST algorithm described and analyzed here. Formally, in Section 3 we prove:

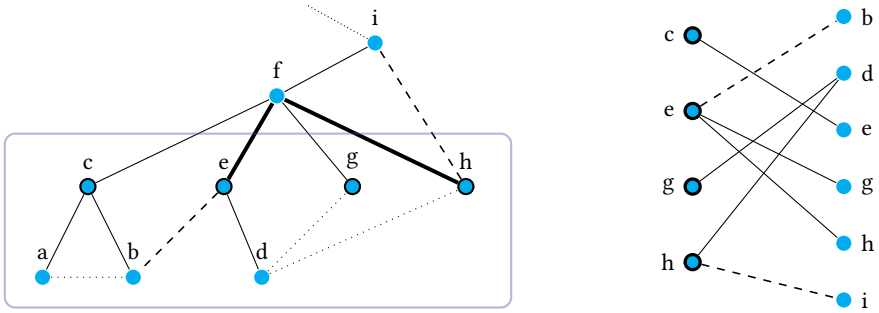

Figure 1: On left, a graph fragment with two parallel improvements is identified, formatted such that: tree edges are solid, replaced edges are bold, improvement edges are dashed, and other non-tree graph edges are dotted. The heads of the four leaf branches (within rectangle) are emphasized with a black border. On right, the corresponding imp graph, with zero-degree nodes (a, c) omitted. The dashed edge form a maximal 1-constrained matching.

Theorem 2.3. *There exists a distributed algorithm that finds a c-approximate constrained q-matching in expected $O((D+\sqrt{n})\log n)$ rounds in the broadcast-CONGEST mode, for some $c \leq 128$.*

We now continue with describing and analyzing the MDST algorithm that leverages the above theorem. We begin with some useful definitions and observations.

We say that an edge e *departs* a leaf branch B if the node in U corresponding to B is incident on e, and an edge *enters* a branch B if some node in Q contained in the branch B is incident on e. Our MDST algorithm transforms constrained matchings into those with the following useful property:

Definition 2.4. An *improvement subgraph* is a constrained q-matching with additional property that no leaf branch has both departing and entering edges.

Given an improvement subgraph $\bar{M}$, we can readily identify the edges $R_{\bar{M}}$ that it replaces. For *improving* edge e in $\bar{M}$, let B_e be the leaf branch that e departs from and let $e(B_e)$ denote the edge between the head and the parent of B_e, separating B_e from the rest of the tree $(T - B_e)$. Then, the *replaced* edges of T are given by $R_{\bar{M}} = \{e(B_e) : e \in \bar{M}\}$.

Observation 1. *If $\bar{M}$ is an improvement subgraph (of T), then $T_{\bar{M}} = (T \setminus R_{\bar{M}}) \cup \bar{M}$ is a (spanning) tree.*

Proof. Let $\hat{T} = T - \cup_{e\in\bar{M}} B_e$ be the subtree obtained by removing all the vertices and incident edges of all leaf branches B_e from which an edge $e \in \bar{M}$ departs. Observe that each $e \in \bar{M}$ enters (a branch in) $\hat{T}$, since no branch has edges both entering and departing. To restate, each edge $e \in \bar{M}$ has one endpoint in a distinct leaf branch B_e and the other in $\hat{T}$. Each edge $e(B_e) \in R_{\bar{M}}$ also has one endpoint in B_e and the other in $\hat{T}$. □

The resulting degrees in $T_{\bar{M}}$ should also be "better" than before, but not too much better (in order for our analysis to work), and the nodes which are worse are not too much worse.

Observation 2. *For each edge e of an improvement subgraph $\bar{M}$ it holds that: a) the endpoints of $e = (u, v)$ are not too much worse off: $d_{T_{\bar{M}}}(u), d_{T_{\bar{M}}}(v) < \gamma_0 + q$, and b) the parent p of $e(B_e)$ is not improved too much: $d_{T_{\bar{M}}}(p) \geq \gamma - q$.*

Intuitively, decreasing the degree of a higher degree vertex should be more valuable than the increase in the degree of two lower degree nodes.

2.1.1 Improvement Algorithm. We encode our observations in an algorithm Improve with parameters γ and q, which takes the tree T, finds an improvement subgraph $\bar{M}$, and produces a modified tree $T_{\bar{M}} = (T \setminus R_{\bar{M}}) \cup \bar{M}$.

The algorithm first computes an approximate *constrained q-matching* M in the imp graph H with the procedure Constrained-Matching presented and analyzed in Section 3. The algorithm then identifies an improvement subgraph $\bar{M} \subseteq M$ of size at least $\bar{M} \geq M/8$ using the following randomized selection procedure: Independently flip an unbiased coin for each leaf branch B to either mark its departing edge or its entering edges. If the set $\bar{M}$ of unmarked edges contains fewer than $|M|/8$ edges, redo the random marking step until $|\bar{M}| \geq |M|/8$.

Analysis. We first argue that every maximal constrained q-matching must have many edges. We first need an accounting of the adjacencies of nodes in X_γ that do not contribute to that count.

Lemma 2.5. *$2(|X_\gamma| - 1)$ adjacencies of nodes in X_γ are to non-leaf branches or to other nodes in X_γ. More formally, for each vertex $v \in X_\gamma$, let $d'(v)$ denote the number of neighbors of v in T (or G) that are either in X_γ or in a non-leaf branch of T. Then, $\sum_{v\in X_\gamma} d'(v) = 2(|X_\gamma| - 1)$.*

Proof. Form a tree T' on the nodes of X_γ as follows. Starting with T, remove all leaf branches and incident edges and contract every non-leaf branch into one of its adjacent nodes in X_γ. The adjacencies of nodes in X_γ, other than those to leaf branches, are unchanged. Since the tree T' has $X_\gamma - 1$ edges, it has $2(X_\gamma - 1)$ adjacencies. □

Lemma 2.6. *H contains a constrained q-matching with at least $\frac{q}{\gamma}((\gamma-2)|X_\gamma| - d|X_{\gamma_0}|)$ edges.*

Proof. We restrict our attention to a certain subgraph of H. From each bundle with s leaf branches, retain an arbitrary set of $\min(s, \gamma)$ leaf branches, and let L denote the resulting combined set of leaf branches. As each leaf branch has exactly one adjacent node

in X_γ, the set contributes exactly L to the adjacencies of nodes in X_γ. By Lemma 2.5, then, $|L| \geq \sum_{v \in X_\gamma} \min(d_T(v), \gamma) - 2(|X_\gamma| - 1) = (\gamma - 2)|X_\gamma| + 2$.

Let T^* be a spanning tree of maximum degree d. T^* must connect the branches of L to the rest of the graph. Root T^* at some vertex in a non-leaf branch and direct the edges towards the root. Select an arbitrary edge entering each leaf branch in L, to form a set R of edges. Note that R corresponds to a subgraph of H with one edge incident on each node of $U \cap L$. At most $d|X_{\gamma_0}|$ edges in R have one or more endpoints in X_{γ_0}, since T^* has maximum degree d. Let $R' \subseteq R$ be the subset of good edges, i.e., those with both endpoints of degree less than γ_0 in T. Then,

$$|R'| \geq |R| - d|X_{\gamma_0}| \geq (\gamma - 2)|X_\gamma| - d|X_{\gamma_0}| \,.$$

Now form the bipartite graph $H' = (\mathcal{U}, Q, R')$ on the edge set R' but with nodes on one side now representing the bundles (rather than the leaf branches). Observe that the maximum degree of H' is at most γ (since we constrained L to involve at most γ leaf branches from each bundle), each leaf branch is incident on at most one edge in R, and each node in Q is of degree less than γ in G. Since H' is bipartite, its edges can be colored with γ colors [20]. Let M be the union of the q largest color classes, whose size is then at least $q|R'|/\gamma$. We claim that M is a q-constrained matching of H. Namely, by design, each bundle and each node of Q is incident on at most one edge of each color, and thus at most q edges of M; and since $M \subseteq R$, each node in U is incident on at most one edge in M. □

The following is key to finding large parallel improvements. It is conditional on the parameter γ being large enough (roughly $> 4d$) and that there are not too many more intermediate degree nodes (between γ_0 and γ) than high degree nodes ($> \gamma$). Let c be the approximation factor of the Constrained-Matching algorithm.

Theorem 2.7. *Let $\delta > 0$, $\tau = 2/(1-\delta)$, $h = h_\delta = (d\tau^2+2)/(1-\delta)$, and $\Pi = \Pi_{\gamma,q,\delta} = \delta q|X_\gamma|/(8c)$. Suppose $\gamma \geq h$ and $|X_{\gamma_0}| \leq \tau^2 \cdot |X_\gamma|$. Then, Improve (γ, q) finds an improvement subgraph with at least Π edges.*

Proof. By Lemma 2.6, the hypothesis, and the assumed lower bound on γ, we have that H contains a constrained q-matching M^* with at least $q((1 - 2/\gamma)|X_\gamma| - d|X_{\gamma_0}|/\gamma) \geq q|X_\gamma|\left(1 - \frac{2+d\tau^2}{\gamma}\right) \geq \delta q|X_\gamma|$ edges. Thus, the improvement subgraph $\bar{M}$ output satisfies $|\bar{M}| \geq |M^*|/(8c) \geq \delta q|X_\gamma|/(8c) = \Pi$. □

2.2 Progressive Improvements

Theorem 2.7 allows us to find large improvements under certain assumptions (that $|X_{\gamma_0}| \leq \tau^2 \cdot |X_\gamma|$ and the lower bound on γ). We now want to show how to repeatedly find improvements in a smart way, so as to make significant progress on decreasing the degrees in the tree.

Notation. Our algorithm Prog takes a parameter q that specifies the "extent" of the improvements found, i.e., the required separation between the degrees of nodes that have their degrees increased/decreased. Recall also the parameter $\delta > 0$.

Let $k = \Delta(T)$ be the maximum degree spanning tree T at the start of Prog. Recall the notation $\tau = 2/(1-\delta)$, $h = h_\delta = (d\tau^2+2)/(1-\delta)$, X_γ, $\Pi = \Pi_{\gamma,q,\delta} = \delta q|X_\gamma|/(8c)$. Let $b_j = k + 1 - j \cdot q$, for $j \geq 0$; these will play the role of the parameter γ. Let $C_j = X_{b_j}$ denote the set of nodes of degree at least b_j, for $j \geq 0$, and note that $C_0 = \emptyset$. We refer to each non-empty set $C_j \setminus C_{j-1}$ as a *block*.

The algorithms also maintain a persistent variable $\hat{h}$ (which we later shows satisfies $\hat{h} \leq h$). Let t be the smallest value such that $b_{t-1} \geq \hat{h}$ at the start of Prog. The analysis of this subsection will be in terms of the parameters $\hat{h}, t, \tau$, which will be instantiated in the following subsection.

We say that a call to Improve (b_j, q) *fails* if the size of the improvement subgraph found is less than $\Pi_{b_j,q,\delta}$.

Algorithm 1 Prog (q)

```
Let t be the smallest number s.t. b_{t-1} ≥ ĥ
repeat
    j := arg max_{s=1}^{t} |C_s| τ^s
    Improve (b_j, q).
    if the call to Improve failed then
        ĥ = max(ĥ, b_j)
until b_j ≤ ĥ
```

We argue the termination of the algorithm using a potential function Φ. Define the potential $\Phi(v)$ of a node $v \in C_j \setminus C_{j-1}$, for $j \geq 1$ as

$$\Phi(v) = (d_T(v) - b_j)\tau^{-j} + \sum_{s=j+1}^{t} (b_{s-1} - b_s)\tau^{-s}$$

$$= (d_T(v) - b_j)\tau^{-j} + q \sum_{s=j+1}^{t} \tau^{-s}.$$

Each adjacency of v contributes a term to the potential, with the terms increasing by a factor of τ as we move past each threshold b_s. Observe that if nodes v and v' are in $C_j \setminus C_{j+1}$, then $\Phi(v) = \Theta(\Phi(v')) = \Theta(q\tau^j) = \Theta(q\sum_{s=0}^{j} \tau^s)$. The potential of the whole tree T is $\Phi(T) = \sum_{v \in V} \Phi(j)$.

Lemma 2.8. *Each call by Prog to Improve, except the last one, reduces the potential of the tree by an $\Omega(1/(\tau t))$-factor.*

Proof. Since the call was not the last one, it neither failed nor was $j \geq t$.

We first claim that each edge $e = (u, u')$ of the subgraph $\bar{M}$ contributes a drop of $\Omega(\tau^j)$ in the total potential. Namely, it decreased the degree of a node in C_j, but its degree did not drop below b_{j+1}. Thus, it decreased the potential by at least $1/\tau^{j+1}$, while the nodes u and u' that increased in degree experienced a potential increase of at most $2/\tau^{j+2}$. The net decrease is then $1/t^{j+1} - 2/\tau^{j+2} = \delta/\tau^{j+1}$. Since $\bar{M}$ contained at least Π edges, the total decrease in potential, $\Phi(T) - \Phi(T_{\bar{M}})$, is then at least $\Omega(\Pi/\tau^{j+1}) = \Omega(q|C_j|\tau^{-j-1})$.

Observe that

$$\Phi(T) = \sum_v \Phi(v) = \Theta\left(\sum_{j'} (|C_{j'}| - |C_{j'-1}|) \cdot q \sum_{s=j'}^{t} \tau^{-s}\right)$$

$$= \Theta\left(\sum_{j'} |C_{j'}| \cdot q\tau^{-j'}\right).$$

Since j maximized $|C_{j'}|/\tau^{j'}$, it follows that $\Phi(T) \leq t \cdot (|C_j|q\tau^{-j})$. Thus, the call to Improve decreases the potential by $\Phi(T) - \Phi(T_{\breve{M}}) = \Omega(\Phi(T)/(\tau t))$. □

This now lets us bound the number of improvements required.

Lemma 2.9. *Prog runs in $O(\tau t(\log n + t \log \tau))$ phases (or calls to Improve).*

Proof. Consider here only phases (or calls to Improve) that exclude the last one. The initial potential of each vertex (when running Prog) is at most $q \sum_{s=0} \tau^{-s} = O(q)$. Thus, the initial potential of the tree is $O(qn) = O(n^2)$. Since $C_t \neq \emptyset$ (before the last phase), the potential of the tree is at least τ^{-t}. Each phase reduces the potential by a fraction $\Omega(1/(\tau t))$, by Lemma 2.8. Hence, there are at most $O(\tau t \log(n^2\tau^t)) = O(\tau t(\log n + t \log \tau))$ phases. □

We bound the degree of the tree in terms of our variable $\hat{h}$.

Lemma 2.10. *When Prog terminates, the degree k of the resulting tree is less than $\hat{h} + q(1 + \lfloor \log_\tau n \rfloor)$.*

Proof. When Prog terminates, $\hat{h} \geq b_j$. Since the initial degree of the tree was $b_0 - 1$, it follows that $k < b_0 = b_j + jq \leq \hat{h} + jq$. The claim then follows when $j \leq \tau' := 1 + \lfloor \log_\tau n \rfloor$.

Suppose then that $j > \tau'$. Since j maximized $|C_s|/\tau^s$, we have that $|C_j|/\tau^j \geq |C_{j-\tau'}|/\tau^{j-\tau'}$, or $n \geq |C_j| \geq |C_{j-\tau'}|\tau^{\tau'} > |C_{j-\tau'}|n$. Namely, $|C_{j-\tau'}| < 1$, and hence it cannot contain any node. Thus, the largest degree in the current tree is less than $b_{j-\tau'} = b_j + \tau' q \leq \hat{h} + \tau' q$. □

2.3 Main algorithm

Our top-level algorithm calls the Prog algorithm with progressively finer block-sizes.

Algorithm 2 Mds

```
1: q_0 := 2^⌊lg Δ⌋−2
2: ĥ := 1
3: i := 0
4: while q_i ≥ 1 do
5:     Prog(q_i)
6:     i := i + 1
7:     q_i := q_{i−1}/2
```

The variable $\hat{h}$ is a proper lower bound on h, so indirectly on the optimum degree, d.

Lemma 2.11. *The invariant $\hat{h} \leq h_\delta$ is maintained.*

Proof. The claim holds initially, since $d \geq 1$. Consider a call to Prog where $\hat{h}$ is increased. Then the call to Improve (b_j, q) failed to find an improvement subgraph with at least $\Pi_{\gamma,q,\delta}$ edges. Since j was maximized over $|C_s|/\tau^s$, it follows that $|X_{b_j}| = |C_j| \geq |C_{j+2}|/\tau^2 = |X_{b_{j+2}}|/\tau^2$. The contrapositive of Theorem 2.7 (where $\gamma = b_j$ and $\gamma_0 = b_{j+2}$) then implies that $b_j < h_\delta$. Since $\hat{h}$ was increased, it was assigned the value b_j. Thus, $\hat{h} \leq b_j < h_\delta$, maintaining the invariant. □

We are led to our main result.

Theorem 2.12. *For any given $\epsilon > 0$, Mds outputs a spanning tree of degree $4d(1+\epsilon)+O(\log n)$. The number of phases is $O(\log \Delta \log^2 n)$.*

Proof. In the final iteration of Mds, $q_i = 1$. By Lemma 2.10, $X_{b_j} = \emptyset$ for $b_j \geq \log_\tau n + 1 \geq \lg n + 1$. Setting $\tau = 2/(1-\delta)$, we have by Lemmas 2.10 and 2.11 that the degree of the spanning tree T is at most $\hat{h} + \lg_\tau n \leq h + \lg n = 4d/(1-\delta)^3 + 2/(1-\delta) + \lg n$. Choosing $\delta = 1 - 1/(1+\epsilon)^{1/3}$ yields the degree bound.

It follows by induction and Lemma 2.10 that the number of nonempty blocks is bounded by $t \leq 2 \log_\tau n$. The bound on the number of phases then follows from Lemma 2.9. □

If we want a purely multiplicative guarantee, we can obtain a bound that is slightly stronger than the $O(\log n)$ which is immediate from the previous theorem. However, note that for $d \geq \log \log n$, our mixed approximation gives a stronger bound.

Theorem 2.13. *There is a δ such that Mds outputs a spanning tree of degree $O(d \cdot \log n / \log \log n)$.*

Proof. Set $\delta = 1 - 1/(\log n)^{1/4}$ so $\tau = (\log n)^{1/4}$ and $h = O((\log n)^{3/4} d)$. Then, by Lemmas 2.11 and 2.10, the maximum degree of the resulting tree is at most $\hat{h} + \log_\tau n = O((\log n)^{3/4} d + \log n / \log \log n) = O(d \log n / \log \log n)$. □

2.4 Distributed Implementation

Up to this point, the Mds algorithm (and its constituent subroutines) has largely been described and analyzed as a centralized graph algorithm, with the exception being the construction of constrained matchings, for which we assumed from the beginning a distributed solution exists (see Section 3.1). Here we obtain a final time complexity by accounting for the time required to implement the centralized operations previously discussed in a distributed manner in the broadcast-CONGEST model.

There are three different types of standard distributed coordination strategies required for this implementation. The first type of strategies require a given value to be aggregated and/or disseminated globally in the network. For example, when Prog aggregates the total count of edges in the improvement subgraph produced by the preceding call to Improve. These global operations are easily implemented in $O(D)$ rounds using a global BFS tree initialized at the beginning of the algorithm execution.

The second type of strategies require the aggregation and/or dissemination of information within well-defined connected components (that is, components in which all nodes in the same component share the same component id). For example, when the nodes in Improve agree on a random coin flip for their component, and then count the total number of remaining unmarked edges. It is straightforward to implement these internal component aggregations and dissemination in $O(D + \sqrt{n})$ rounds (see [9] for details).[1]

[1] The standard trick here, which we summarize for the sake of completeness, and which we elaborate in [9], is to treat small components (less than $\sqrt{n}$ nodes) and large components (at least $\sqrt{n}$ nodes) differently. Communication within small components can be implemented with BFS trees in order of the component diameter time, which is no more than $\sqrt{n}$. Because there are at most $\sqrt{n}$ total large components, the aggregation/dissemination values for all components can be pipelined through a global BFS tree in $O(D + \sqrt{n})$ rounds.

The third type of distributed coordination strategies require nodes to discover the name of the component in which they now belong after edges are removed from a spanning tree. In our algorithm, this occurs at the beginning of Improve where in order to construct the improvement graph needed by the constrained matching, nodes first remove high degree nodes from T, creating a forest T', and then agree upon names for the components in T'.

Consider the following strategy to accomplish this goal. First, each node marks itself with probability $\log n/\sqrt{n}$. Marked nodes initiate a flood of their id within T' up to distance $\alpha\sqrt{n}$, for a sufficiently large constant α. A node only accepts and passes on such a token if it has not accepted and broadcast an id token previously.

With high probability, there are $O(\sqrt{n}\log n)$ marked nodes and their floods cover all n nodes in the graph. In this case, consider the component graph that results where we treat for each marked node u, the set of nodes that accepted u's token as a component that we call u's *flood component*. It is not hard to show that the component graph defined over these flood components contains $O(\sqrt{n}\log n)$ total edges.[2] We can therefore broadcast the full topology of this component graph to the whole network in $O(D+\sqrt{n}\log n)$ rounds by pipelining these edges through the global BFS tree.

At this point, every node v knows its flood component and the full flood component graph. This allows v to determine the set of flood components that cover its tree in forest T'. Each such flood component is described by its marked node's id, so v can choose the largest such id as the name for its T' tree. All nodes in this same tree will choose the same name.

We now combine these distributed coordination strategies with our preceding results on the number of calls to the Mds subroutines to obtain the following bound. We defer the proof to the full version [9].

Theorem 2.14. *There exists a distributed implementation of Mds that outputs a spanning tree of degree $O(d+\log n)$ in $O((D+\sqrt{n})\log^4 n)$ rounds, in expectation, in the broadcast-CONGEST model.*

3 ALGORITHM FOR CONSTRAINED MATCHINGS

We now prove Theorem 2.3 by designing a randomized distributed algorithm (in the broadcast-CONGEST model) for finding near-maximum constrained q-matchings, which runs in $O((D+\sqrt{n})\log n)$ rounds. The algorithm, which we call Constrained-Matching, is based on finding approximate flow in a shallow network.

What distinguishes constrained matchings from other variations of bipartite matching is the constraint on the bundles, which is a constraint on *sets* of vertices rather than just on individual vertices. It appears to preclude a simple reduction to a standard matching computation. To overcome this extra difficulty, we extend in the obvious way the standard relationships between bipartite matchings and flows on 4-layer graphs (a source s, the two layers of the bipartite graph, and the sink t) to add in a fifth layer which will constrain the bundles appropriately. We then find maximal flows in this graph in $O(\log n)$ phases, and show that these flows give an $O(1)$-approximation to the maximum constrained q-matching. At a high level, Constrained-Matching computes a maximal *fractional* solution and then rounds this fractional solution (all in a distributed fashion).

There is one additional difficulty, which is that one side of the bipartite graph we are working in (the imp graph H) consists of leaf branches, not just nodes. Fortunately, by using the communication primitives discussed in Section 2.4, each phase of the algorithm can be executed in $O(D+\sqrt{n})$ distributed rounds, thus giving a total running time of $O((D+\sqrt{n})\log n)$.

3.1 Algorithm

Constrained matchings correspond to flows in a related flow graph F. The vertices of the flow graph F consist of the set B of bundles, the extreme nodes s and t, as well as the sets U and Q from the graph H. There is a directed edge from s to each bundle node in B, from each bundle to its constituent leaf branches in U, from leaf branches to the nodes in Q they are adjacent to in H, and finally from each node in Q to t. Edges from s to bundles, and those from Q to t, are of capacity q, while the rest are of unit capacity.

Observe that there is a one-one correspondence between constrained q-matchings in H and integral flows in F. Moreover, each edge in H has a unique flow path in F and vice versa. Thus we may specify a flow in F by giving only the flow on edges in H.

For a flow f, let $f(e)$ denote the flow through edge e, $f(v)$ denote the flow going out of v, and $v(f)=f(s)$ be the *value* of the flow. The *size* of a constrained q-matching equals the value of the corresponding flow. We say that a node is *full* if it has incoming flow at least 1/8-th of its capacity (where the capacity of a vertex is the maximum of its total incoming and total outgoing capacities).

The algorithm initially assigns each flow path a flow of $1/m$, where m is the number of edges. In each phase, every non-full node in Q doubles the flow on its incident paths from non-full nodes in U (through its parents in B) by sending a "double the flow" message to its neighbors (full neighbors will ignore this message). It takes one round to send these "double" messages, and then in each leaf bundle we can use our intra-component aggregation primitive (the second type of strategy in Section 2.4) to make sure that in each leaf bundle C, every node knows how much flow goes through the node representing C in F before this doubling (so in particular knows if C is full already, and so should ignore the message) and after the doubling. After this knowledge has been disseminated, each node in a leaf bundle that is adjacent to at least one node in Q sends the new flow value to its neighbors in Q, which will then know which of its incident edges did actually double the flow. After at most $\log m$ doubling phases there will be no way of sending more flow using only non-full nodes (as we show in the next subsection), so we move to the next part of the algorithm, where we use randomized rounding to find a large constrained q-matching.

In particular, we would like to do the following (from a centralized perspective). First add each edge e in H to a set S independently with probability $f(e)$. Then any leaf branch in U with more than one incident edge in S removes all such edges from S, any bundle with more than q incident edges in S removes all such edges from S, and any node in Q with more than q incident edges in S removes all

[2] For the sake of analysis, orient all edges in T toward a common root before removing high degree nodes to create T'. In the resulting directed forest, each flood component has at most one *outgoing* edge from a node in the flood component to a parent outside the flood component. Each edge in the flood component graph is an outgoing edge for some component, so if there are $O(\sqrt{n}\log n)$ total components there are $O(\sqrt{n}\log n)$ total edges.

such edges from S. This would (by definition) result in a constrained q-matching, which we call S'.

In order to implement this in broadcast-CONGEST, we first have every vertex v in each leaf branch C in U make the appropriate randomized decisions for the edges from H that are incident on v, so every vertex v in each leaf branch known which edges of S are incident on it. Note that this results in the same S as in the centralized algorithm. Now if v added more than one incident edge to S, then it removes all of these edges from S and uses our intra-component communication primitive to send a message to all other nodes in C about this fact, which will then cause all other nodes in C to remove all of their incident edges from S (if there were any). Otherwise, if v added exactly one edge to S, it broadcasts the identity of this edge to all of its neighbors (and in particular the other endpoint of the added edge) as well as using our intra-component primitives to send the identity of the edge to the rest of the nodes in C. It is easy to see that if multiple such messages are being sent in some leaf branch, then we can detect that fact and send a "failure" message to all nodes in the branch so that they all remove their incident edges from S.

Now each leaf branch has either 0 or 1 incident edge in S, and all nodes in the branch know the identity of this edge (if it exists). The root of each leaf branch sends to its parent the identity of this edge. So now the root of each leaf *bundle* knows the edges incident on the bundle which are in S. If there are more than q such edges, then this bundle root removes them all by sending a message to all of the nodes in all of the leaf branches in the bundle (again using our primitives from Section 2.4). Similarly, each node in Q knows all of its incident edges that are in S, and if there are more than q of them then it removes all of them from S by broadcasting a failure message to its neighbors. The edges which survive this process are S', and it is easy to see that it is precisely the same set as would have been computed in the centralized version.

3.2 Analysis

We omit the proof of the following claim.

Lemma 3.1. *Constrained-Matching requires $O((D+\sqrt{n})\log n)$ rounds in broadcast-CONGEST.*

The flow f that the algorithm computed has the property that there is no way to send more flow using only non-full nodes (or else the algorithm would send more flow). Thus the value of f is at least $1/8$ of some *maximal* flow, which is where every $s-t$ path has some node that is *saturated*, i.e., whose flow is at full capacity. We now claim that any maximal flow is close to a maximum flow. The *depth* of a flow network is the length of the longest $s-t$ path, so in our flow network the depth is 4.

Lemma 3.2. *The value of a maximal flow in a depth-d flow network is at least $1/d$-fraction of the value of the maximum flow.*

Proof. The depth constraint implies that $v(f)\cdot d \geq \sum_{e\in F} f(e)$, for any flow f. Namely, since each flow path is of length at most d, each unit of flow is counted at most d times in the sum. Maximality means that there is an $s-t$ cut $(Z, V-Z)$ such that all edges in F that go from Z to $V-Z$ are at full capacity (in f). This implies that $\sum_{v\in Z} f(v) \geq cap(Z, V-Z)$, where $cap(Z, V-Z)$ is the sum of the edge capacities across the cut. Observe that $\sum_{v\in Z} f(v) \leq \sum_{v\in V\setminus\{t\}} f(v) = \sum_{e\in F} f(e)$. The capacity constraints imply that $cap(Z, V-Z) \geq v(f^*)$, where f^* is a maximum flow. Combined, we have that $v(f) \geq v(f^*)/d$. □

Corollary 3.3. *Let f be the flow computed by our algorithm. Then, $v(f) \geq \frac{1}{32}|M|$ for every constrained q-matching M of H.*

Proof. Since in f every $s-t$ path contains at least one full node, $v(f)$ is at least $1/8$ of the value of any maximal flow, and so by Lemma 3.2 we know that $v(f)$ is at least $1/32$ of the value of the maximum flow (which will be integral since alll capacities are integral). As discussed, there is a bijection between the integral flows in F and constrained q-matchings in H, so this implies that $v(f)$ is at least $1/32$ times the size of the maximum constrained q-matching in F. □

By construction S' is a feasible constrained q-matching, so we just need to show that it has large value.

Lemma 3.4. *Let f be the (fractional) flow and S' be the edge set computed by our algorithm. Then $\mathbb{E}[|S'|] \geq v(f)/4$.*

Proof. Consider an edge $e=(u,v)$ in H (i.e., from U to Q). Let A_e be the event that some other edge incident on u is added to S. Let B_e be the event that q or more edges are added that have endpoints in the same bundle as e but not u. Finally, let C_e be the event that q or more other edges incident on v were added to S. Observe that if e was added to S, then it will remain in S' if none of the three events (A_e, B_e and C_e) take place. Using that the flow is at most one-fourth of capacity,

$$\mathbb{P}[A_e] = 1 - \prod_{e'\ni u, e'\neq e} (1-f(e')) \leq \sum_{e'\ni u} f(e') \leq 1/4\,.$$

Let Y be the number of edges in S incident on the same bundle as e. Then, $\mathbb{E}[Y] \leq q/4$. So, by Markov's inequality, $\mathbb{P}[B_e] \leq \mathbb{P}[Y \geq 4\,\mathbb{E}[Y]] \leq 1/4$. Similarly, $\mathbb{P}[C_e] \leq 1/4$. By the union bound, the probability of a bad event is at most $\mathbb{P}[A_e \cup B_e \cup C_e] \leq 3/4$. Thus, the event X_e that edge e is contained in S' has probability

$$\mathbb{P}[X_e] \geq f(e)\cdot(1-\mathbb{P}[A_e\cup B_e\cup C_e]) = f(e)/4\,,$$

since the event of e being chosen in S is independent from the three bad events. Thus, by linearity of expectation,

$$\mathbb{E}[|S'|] = \sum_{e\in E(H)} \mathbb{P}[X_e] \geq \frac{1}{4}\sum_{e\in E(H)} f(e) = v(f)/4\,. \qquad \square$$

We can now combine these pieces to prove our main result regarding this algorithm (presented in Theorem 2.3):

Proof of Theorem 2.3. The running time is direct from Lemma 3.1. Lemma 3.4 and Corollary 3.3 imply that $\mathbb{E}[|S'|] \geq v(f)/4 \geq M^*/128$, where M^* is the size of an optimal q-constrained matching. So the algorithm returns a 128-approximation. □

4 DERANDOMIZATION

We sketch here how the algorithm can be derandomized in the CONGEST model.

The main effort is on constrained matching, for which adapt a method of Fischer [11] for deterministic matchings. We initially compute fractional matching deterministically with repeated doubling, and then round the flow values down to the nearest power

of 2. The $O(\log \Delta)$ weight classes are then eliminated in sequence, starting with the lowest value. Each node has an number of incident edges with the lowest value and by arbitrarily coloring them, we obtain a partition into degree-2 subgraphs, i.e., paths and cycles. The idea is to shift the weight from even numbered edges on each path/cycle to the odd numbered edges, which obeys the capacity constraints while eliminating the lowest weight class.

The algorithm cannot be applied directly to the case of matching branches, since the communication paths involved may now induce heavy loads. We show that we can actually maintain unit load on the edges by leveraging the flexibility available in pairing the incident edges to a node, i.e., the edges that should belong to the same path/cycle. We still need to process the outgoing edges from each node in parallel, which limits the application to the the full CONGEST model. We refer to the full version for the complete details [9].

The other randomized step is in ensuring that a component does not both have an incoming and outgoing edges in improvement subgraph. This can be replaced by the following simple deterministic rule: A component with two or more incoming edges removes its outgoing edge. Then components with a single incoming and outgoing edges induce a degree-2 graph on which we compute a maximal matching.

5 LOWER BOUNDS

Our main lower bound is the following:

Theorem 5.1. *For any $n \in \mathbb{N}$, $\alpha > 4$, and $h \geq 1$, there exist two families $\mathcal{G}^{\mathrm{Rand}}_{n,h,\alpha}$ and $\mathcal{G}^{\mathrm{Det}}_{n,\alpha}$ of $\Theta(n)$-node graphs, which have diameters $O(h + \log n)$ and $O(\log n)$ respectively and give the following lower bounds for the computation of $(\alpha - 4)$-additive or $\alpha/4$-multiplicative approximate MDSTs. 1) For graphs in $\mathcal{G}^{\mathrm{Rand}}_{n,h,\alpha}$ such that $h/\alpha = o(\sqrt{n}\log^2 n)$ holds, any randomized CONGEST algorithm with success probability at least $2/3$ requires $\tilde{\Omega}((nh/\alpha)^{1/3})$ rounds in the worst case. 2) For graphs in $\mathcal{G}^{\mathrm{Det}}_{n,h,\alpha}$, any deterministic CONGEST algorithm requires $\tilde{\Omega}(\sqrt{n}/\alpha)$ rounds in the worst case.*

Note that this theorem applies not only to broadcast-CONGEST but also to the standard CONGEST model.

Before discussing the proof of Theorem 5.1, we explain how this theorem matches our upper bound. Consider the randomized lower bound first. Let $h = \Theta(n^{1/2-3\epsilon})$ for an arbitrarily small positive constant $\epsilon < 1/6$, and $\alpha = \mathrm{polylog}(n)$. Then the diameter D of the instances from $\mathcal{G}^{\mathrm{Rand}}_{n,h,\alpha}$ is $\Theta(n^{1/2-3\epsilon})$, and Theorem 5.1 yields a lower bound of $\tilde{\Omega}(n^{1/2-\epsilon}/\mathrm{polylog(n)})$ rounds. Since $D = o(n^{1/2-\epsilon}/\mathrm{polylog(n)})$ holds, it implies a $\tilde{\Omega}(n^{1/2-\epsilon} + D)$-round lower bound for any polylog(n)-multiplicative approximation algorithms. That is, our algorithm is round-optimal up to subpolynomial (i.e., $O(n^{o(1)})$) factors. The $\tilde{\Omega}(n^{1/3})$-round lower bound for the logarithmic diameter case is derived from choosing $h = O(\log n)$. In the deterministic case, we have $D = O(\log n)$. Then any $\tilde{O}(\sqrt{n} + D)$-round algorithm with polylog(n)-multiplicative approximation is round-optimal up to polylogarithmic factors. The deterministic lower bound also implies that there is no efficient deterministic algorithm with multiplicative dependency on D. That is, if there exists an algorithm running in $\tilde{O}(n^{1/3}D)$ rounds (the randomized lower bound does not exclude the possibility of such an algorithm), it must be randomized.

The proof of Theorem 5.1 follows the seminal framework by Das Sarma et al. [25] of reducing from two-party communication complexity. Due to space constraints, some details are deferred to the full version [9]. A key idea is to utilize the *gap-k disjointness* function $\mathrm{gdisj}_{N,k}$ [7, 12], which is a generalized version of the standard set-disjointness. In this problem, Alice has some vector $\boldsymbol{x} \in \{0,1\}^N$ and Bob has some vector $\boldsymbol{y} \in \{0,1\}^N$, and their goal is to determine whether $\boldsymbol{x} \cdot \boldsymbol{y} \geq k$ or $\boldsymbol{x} \cdot \boldsymbol{y} = 0$, where $\cdot$ is the (bitwise) inner product of two vectors. Clearly the case of $k = 1$ corresponds to the standard set-disjointness. We will use the following lower bound:

Lemma 5.2 ([7, 12]). *The randomized communication complexity of $\mathrm{gdisj}_{N,k}$ with error probability at most $1/3$ is $\Omega(N/k)$, and the deterministic communication complexity of $\mathrm{gdisj}_{N,k}$ for $k \leq N/4\log N$ is $\Omega(N)$.*

The construction of the two families $\mathcal{G}^{\mathrm{Rand}}_{n,h,\alpha}$ and $\mathcal{G}^{\mathrm{Det}}_{n,\alpha}$ follows the reduction technique by Izumi [19] on top of the framework by Das Sarma et al. [25]. We first introduce a graph $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b})$ associated with two parameters N and M, and a pair of two N-bit bianary vectors $(\boldsymbol{a},\boldsymbol{b})$. Each family is defined as the sets of $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b})$ over all $(\boldsymbol{a},\boldsymbol{b})$ with an appropriate choice of N and M. For simplicity, we assume that N, M, and M/h are all integer values. The graph $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b})$ is constructed as follows:

(1) We first create $4N + 1$ paths of length M, each of which is denoted by P_i $(0 \leq i \leq 4N)$.The nodes constituting P_i are identified by $v_{(i,0)}, v_{(i,1)}, \ldots, v_{(i,M-1)}$ from left to right.
(2) We then create a complete binary tree T with M leaves $u_0, u_1, \ldots u_{M-1}$. For every $i \in [0, M/h]$, the node u_{ih} is connected with all the nodes $v_{ih,j}$ for $j \in [0, 4N-1]$.
(3) For every $i \in [0, N-1]$, we add two edges $(v_{4i+1,0}, v_{4i+2+\boldsymbol{a}[i],0})$ and $(v_{4i+3-\boldsymbol{a}[i],0}, v_{4(i+1),0})$. Similarly, for every $i \in [0, N-1]$, we add two edges $(v_{4i,M-1}, v_{4i+1+\boldsymbol{b}[i],M-1})$ and $(v_{4i+2-\boldsymbol{b}[i],M-1}, v_{4i+3,M-1})$.

It is not difficult to check that $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b})$ has diameter $O(h + \log n)$. Let $T_{\boldsymbol{a},\boldsymbol{b}}$ be the minimum-degree spanning tree of $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b})$, and $\Delta(U)$ be the maximum degree of a spanning tree U. The key property of $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b})$ is the following lemma:

Lemma 5.3. *$\Delta(T_{\boldsymbol{a},\boldsymbol{b}}) \geq 1 + (\boldsymbol{a}\cdot\boldsymbol{b}+1)h/M$ for every $(\boldsymbol{a},\boldsymbol{b}) \in (\{0,1\}^N)^2$. In addition, if $\boldsymbol{a}\cdot\boldsymbol{b} = 0$ then $d_{\boldsymbol{a},\boldsymbol{b}} \leq 3$.*

Proof. Let Z_i be the subgraph of $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b})$ induced by $\{v_{i,j} \mid k \in [4i, 4i+3], 0 \leq j \leq M-1\} \cup \{v_{4(i+1),0}\}$, and $Z = (\cup_i Z_i) \cup P_{4N}$. The graph Z_i forms a simple path Q_i from $v_{4i,0}$ to $v_{4(i+1),0}$ if $\boldsymbol{a}[i]\cdot\boldsymbol{b}[i] \neq 1$. Otherwise, it forms a simple path Q_i from $v_{4i,0}$ to $v_{4(i+1),0}$ and a cycle C_i disconnected in Z. The simple paths Q_i for all $i \in [0, N-1]$ and P_{4N} are concatenated into a long simple path, and the number of disconnected cycles is equal to $\boldsymbol{a}\cdot\boldsymbol{b}$. Consequently, Z consists of $\boldsymbol{a}\cdot\boldsymbol{b}+1$ connected components, which must be connected in $T_{\boldsymbol{a},\boldsymbol{b}}$ using the edges incident to leaves of T. Since at most M/h nodes has those edges, at least one node must have a degree $(\boldsymbol{a}\cdot\boldsymbol{b}+1)h/M$ or more. If $\boldsymbol{a}\cdot\boldsymbol{b} = 0$, Z forms a single path, and thus adding an edge $(u_0, v_{0,0})$ to $T \cup Z$ results in a spanning tree of the maximum degree three. □

The reduction utilizes the simulation theorem by Das Sarma et al., which allows Alice and Bob to simulate any $o(M)$-round CONGEST algorithm running on $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b})$ with relatively few communication bits.

Theorem 5.4 (Das Sarma et al. [25]). *Assume that Alice and Bob respectively knows two N-bit strings $\boldsymbol{a}$ and $\boldsymbol{b}$. Let X be any CONGEST algorithm running in $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b})$ within $r(n) = o(M)$ rounds. Then there exists a two-party protocol which uses $O(r(n)\log^2 n)$-bit communication and provides the outputs (i.e. the internal states at the beginning of round $r(n)+1$) of X at the nodes $v_{0,0}$ and $v_{0,M-1}$ respectively to Alice and Bob.*

Now we are ready to prove Theorem 5.1.

Proof. We mainly consider the case of randomized algorithms. The deterministic case can be proved similarly with a different setting of N and M. We set $N = \left(\frac{\alpha n^2 \log^2 n}{h}\right)^{1/3}$ and $M = \left(\frac{hn}{\alpha \log^2 n}\right)^{1/3}$. We have already shown that $\mathcal{G}_{N,M,h}$ is the set of graphs of $\Theta(n)$ nodes and diameter $O(h+\log n)$. Suppose for contradiction a CONGEST algorithm X computing an α-approximate min-degree spanning tree, which runs in $o(M) = o\left(\frac{nh}{\alpha \log^2 n}\right)^{1/3}$ rounds for any instance $\Gamma_{N,M,h}(\boldsymbol{a},\boldsymbol{b}) \in \mathcal{G}_{n,h,\alpha}$. Without loss of generality, this algorithm informs all the nodes of the value $\Delta(U)$ for the constructed spanning tree U, which takes extra $O(h+\log n)$ rounds for the execution of X but does not affect the asymptotic time complexity of X. (recall that $\Omega(D) = \Omega(h+\log n)$ is the trivial lower bound). Let $k = \frac{\alpha M}{h} = \left(\frac{\alpha^2 n}{h^2 \log^2 n}\right)^{1/3}$. By Lemma 5.3, if $\boldsymbol{a}\cdot\boldsymbol{b} \geq k$, $\Delta(T_{\boldsymbol{a},\boldsymbol{b}})$ is at least $kh/M \geq \alpha$ and thus $\Delta(U) \geq \alpha$ holds. If $\boldsymbol{a}\cdot\boldsymbol{b} = 0$ holds, $\Delta(T\{\boldsymbol{a},\boldsymbol{b}\})$ is at most three, and thus $\Delta(U)$ is at most $3 + (\alpha - 4) \leq \alpha - 1$ in the additive approximation case, and is $3 \cdot \alpha/4 < \alpha$ in the multiplicative approximation case. In any case, the value of $gdistj_{N,k}(\boldsymbol{a},\boldsymbol{b})$ can be correctly decided from the output value of X. By Theorem 5.4, it yields a two-party protocol for the N-bit gap-k disjointness problem, which consumes $o(M\log^2 n)$-bit communication. Since $N/k = \left(\frac{hn\log^4 n}{\alpha}\right) = M\log^2 n$ holds, the communication complexity of that protocol is $o(M\log^2 n) = o(N/k)$ bits. It contradicts Theorem 5.2. For the proof of the deterministic case, we set $N = \left(\alpha n \log^2 n\right)^{1/2}$, $M = \left(\frac{n}{\alpha \log^2 n}\right)^{1/2}$, $h = 1$, and $k = \alpha M$. Then, the same argument results in a deterministic two-party protocol for $\mathrm{gdisj}_{N,k}$ consuming $o(M\log^2 n)$-bit communication. Since k satisfies $k \leq N/4\log N$ for sufficiently large n, the deterministic lower bound is $\Omega(N)$ bits by Lemma 5.2, but $M\log^2 N \geq N$ holds and thus we have a contradiction. □

ACKNOWLEDGMENTS

This work was supported in part by NSF Awards #7773087 and #1535887, JST SICORP, KAKENHI No.16H02878 and 19K11824, and Icelandic Research Fund grant 174484.

REFERENCES

[1] A. Agrawal, P. Klein, and R. Ravi. 1991. *How tough is the minimum-degree Steiner tree? A new approximate Min-Max equality.* Technical Report CS-91-94. Brown University.

[2] B. Awerbuch. 1987. Optimal distributed algorithms for minimum-weight spanning tree, counting, leader election and related problems. In *Proceedings of the Symposium on Theory of Computing*. 230–240.

[3] Lélia Blin and Franck Butelle. 2003. The first approximated distributed algorithm for the minimum degree spanning tree problem on general graphs. In *Proceedings of the International Parallel and Distributed Processing Symposium*.

[4] Lélia Blin and Franck Butelle. 2004. The First Approximated Distributed Algorithm For The Minimum Degree Spanning Tree Problem On General Graphs. *Int. J. Found. Comput. Sci.* 15, 3 (2004), 507–516.

[5] Lélia Blin and Pierre Fraigniaud. 2015. Space-Optimal Time-Efficient Silent Self-Stabilizing Constructions of Constrained Spanning Trees. In *35th IEEE International Conference on Distributed Computing Systems, ICDCS 2015, Columbus, OH, USA, June 29 - July 2, 2015*. 589–598.

[6] Lélia Blin, Maria Gradinariu Potop-Butucaru, and Stephane Rovedakis. 2011. Self-stabilizing minimum degree spanning tree within one from the optimal degree. *J. Parallel Distrib. Comput.* 71, 3 (2011), 438–449.

[7] Keren Censor-Hillel and Michal Dory. 2018. Distributed Spanner Approximation. In *Proc. of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*. 139–148.

[8] F. Chin and H.F. Ting. 1985. An almost linear time and $O(n\log n + E)$ message distributed algorithm for minimum-weight spanning trees. In *Proceedings of the Symposium on Foundations of Computer Science*. 257–266.

[9] Michael Dinitz, Magnús M Halldórsson, Taisuke Izumi, and Calvin Newport. 2018. Distributed Algorithms for Minimum Degree Spanning Trees. *arXiv preprint arXiv:1806.03365* (2018).

[10] M. Elkin. 2004. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proceedings of the Symposium on Theory of Computing*. 331–340.

[11] Manuela Fischer. 2018. Improved deterministic distributed matching via rounding. *Distributed Computing* (2018).

[12] Orr Fischer, Tzlil Gonen, and Rotem Oshman. 2017. Distributed Property Testing for Subgraph-Freeness Revisited. *CoRR* abs/1705.04033 (2017).

[13] Martin Fürer and Balaji Raghavachari. 1990. An NC approximation for the minimum degree spanning tree problem. In *Proceedings of the Annual Allerton Conference on Communication, Control and Computing*. 274–281.

[14] Martin Fürer and Balaji Raghavachari. 1992. Approximating the minimum degree spanning tree to within one from the optimal degree. In *Proceedings of the Symposium on Discrete Algorithms*. 317–324.

[15] Martin Fürer and Balaji Raghavachari. 1994. Approximating the Minimum-Degree Steiner Tree to within One of Optimal. *Journal of Algorithms* 17, 3 (1994), 409 – 423.

[16] E. Gafni. 1985. Improvements in the time complexity of two message-optimal election algorithms. In *Proceedings of the Symposium on the Principles of Distributed Computation*. 175–185.

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

[18] J. Garay, S. Kutten, and D. Peleg. 1998. A sub-linear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal of Computing* 27 (1998), 302–316.

[19] Taisuke Izumi. 2014. Randomized Lower Bound for Distributed Spanning-Tree Verification. In *Proc. of the 21st International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. 137–148.

[20] Dénes König. 1916. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Math. Ann.* 77, 4 (1916), 453–465.

[21] S. Kutten and D. Peleg. 1995. Fast distributed construction of k-dominating sets and applications. In *Proceedings of the Symposium on the Principles of Distributed Computation*. 238–251.

[22] Christian Lavault and Mario Valencia-Pabon. 2008. A distributed approximation algorithm for the minimum degree minimum weight spanning trees. *J. Parallel and Distrib. Comput.* 68, 2 (2008), 200–208.

[23] J. Nešetřil, E. Milková, and H. Nešetřilová. 2001. Otakar Boruvka on minimum spanning tree problem. Translation of both the 1926 papers, comments, history. *Discrete Mathematics* 233, 1 (2001), 3–36.

[24] D. Peleg and V. Rubinovich. 2001. A near-tight lower bound on the time complexity of distributed MST construction. *SIAM Journal of Computing* 30, 5 (2001), 1427–1442.

[25] A. Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. 2012. Distributed Verification and Hardness of Distributed Approximation. *SIAM J. Comput.* 41, 5 (2012), 1235–1265.