Simple Combinatorial Algorithms for the Minimum Dominating Set Problem in Bounded Arboricity Graphs

Adir Morgan*
Tel Aviv University

Shay Solomon†
Tel Aviv University

Nicole Wein‡
MIT

Abstract

We revisit the minimum dominating set problem on graphs with arboricity bounded by α. In the (standard) centralized setting, Bansal and Umboh [BU17] gave an $O(\alpha)$-approximation LP rounding algorithm. Moreover, [BU17] showed that it is NP-hard to achieve an asymptotic improvement. On the other hand, the previous two non-LP-based algorithms, by Lenzen and Wattenhofer [LW10], and Jones et al. [JLR+13], achieve an approximation factor of $O(\alpha^2)$ in linear time.

There is a similar situation in the distributed setting: While there are $\mathrm{poly}\log n$-round LP-based $O(\alpha)$-approximation algorithms [KMW06, DKM19], the best non-LP-based algorithm by Lenzen and Wattenhofer [LW10] is an implementation of their centralized algorithm, providing an $O(\alpha^2)$-approximation within $O(\log n)$ rounds with high probability; they also gave a faster deterministic algorithm but with a worse approximation guarantee.

We address the question of whether one can achieve a simple, elementary $O(\alpha)$-approximation algorithm not based on any LP-based methods, either in the centralized setting or in the distributed setting. We resolve these questions in the affirmative. More specifically, our contribution is two-fold:

1. In the centralized setting, we provide a surprisingly simple combinatorial algorithm that is asymptotically optimal in terms of both approximation factor and running time: an $O(\alpha)$-approximation in *linear* time.
2. Based on our centralized algorithm, we design a distributed combinatorial $O(\alpha)$-approximation algorithm in the CONGEST model that runs in $O(\alpha \log n)$ rounds with high probability. Our round complexity outperforms the best LP-based distributed algorithm for a wide range of parameters.

*adirmorgan@gmail.com
†solo.shay@gmail.com
‡Supported by NSF Grant CCF-1514339. nwein@mit.edu

1 Introduction

1.1 Background

The minimum dominating set (MDS) problem is a classic problem in computer science. Given a graph G we want to find a minimums cardinality set D of vertices, such that every vertex of the graph is either in D or has a neighbor in D. Besides its theoretical implications, solving this basic problem efficiently has many practical applications in domains ranging from wireless networks to text summarizing (see, e.g., [WAF02, NA16, SL10]).

The MDS problem was one of the first problems recognized as NP-complete [Gar79]. It was also one of the first problems for which an approximation algorithm was analyzed: a simple greedy algorithm achieves a $\ln(\Delta + 1)$-approximation [Joh74]. This approximation factor is optimal up to lower order terms unless $\mathsf{P} = \mathsf{NP}$ [DS14].

Distributed MDS The first efficient distributed approximation algorithm for MDS was given by Jia, Rajaraman, and Suel [JRS02], who gave a randomized $O(\log \Delta)$-approximation in $O(\log^2 n)$ rounds in the CONGEST model. This was improved by Kuhn, Moscibroda and Wattenhofer [KMW16], who gave a randomized $(1 + \varepsilon)(1 + \ln(\Delta + 1))$-approximation in $O(\log^2 \Delta/\varepsilon^4)$ rounds in the CONGEST model and in $O(\log n/\varepsilon^2)$ rounds in the LOCAL model. Ghaffari, Kuhn, and Maus [GKM17] showed that by allowing exponential-time local computation, one can get a randomized $(1 + o(1))$-approximation in a polylogarithmic number of rounds in the LOCAL model.

For *deterministic* distributed algorithms, Deurer, Kuhn, and Maus [DKM19] recently gave two algorithms in the CONGEST model with approximation factor $(1 + \varepsilon) \ln(\Delta + 1)$ for $\varepsilon > 1/\text{polylog}\Delta$, running in $2^{O(\sqrt{\log n \log\log n})}$ and $O(\Delta \cdot \text{polylog}\Delta + \text{polylog}\Delta \log^* n)$ rounds, respectively; the runtime of the former algorithm is dominated by the time needed for deterministically computing a network decomposition in the CONGEST model, which, due to [GGR21], is thus reduced to $O(\text{poly}\log n)$.

Graphs of bounded arboricity MDS has been studied on a variety of restricted classes of graphs such as graphs with bounded degree (e.g., [CC08]), planar and bounded genus graphs (e.g., [Bak94, CHW08, ASS19]), and graphs of bounded arboricity, which is the focus of this paper. The class of bounded *arboricity* graphs is a wide family of *uniformly sparse* graphs, defined as follows:

Definition 1.1. The graph G has arboricity bounded by α if for every $S \subseteq V$, it holds that $\frac{m_s}{n_s - 1} \leq \alpha$ where m_s and n_s are the number of edges and vertices in the graph induced by S, respectively.

Thus the arboricity is close to the maximum density $|E(S)|/|S|$ over all induced subgraphs of G. The class of bounded arboricity graphs contains the other graph classes mentioned above as well as bounded treewidth graphs, and in general all graphs excluding a fixed minor. Moreover, many natural and real world graphs, such as the world wide web graph, social networks and transaction networks, are believed to have bounded arboricity. Consequently, this class of graphs has been subject to extensive research, which led to many algorithms for bounded arboricity graphs in both the centralized setting (e.g. [Epp94, GG06, CN85]) and in the distributed setting (e.g. [CHS09, BE10, GS17, SV20]); there are also many algorithms in other settings, such as dynamic graph algorithms, sublinear algorithms and streaming algorithms (see [BF99, HTZ14, PS16, OSSW18, ELR18, ERR19, ERS20, BPS20, MV18, BS20], and the references therein).

Combinatorial algorithms Many problems in theoretical computer science have been efficiently solved using the powerful machinery of algebraic algorithms, and an area of research has emerged where the goal is to find *combinatorial* algorithms that are nearly as fast as their algebraic counterparts. Often such algorithms are simpler and easier to implement, and they typically have smaller constants hidden in their running time expressions. Moreover, in the context of distributed computing, combinatorial algorithms could give rise to much more efficient implementation in terms of *local computation power* as well as *local* memory constraints. Combinatorial algorithms can also provide a deeper understanding of the problem studied. This line of work includes, for example, a combinatorial proof of the PCP theorem [DR06, Mei09], a combinaotiral construction of expanders [RVW00], and combinatorial algorithms for distributed graph coloring [BE14].

While the prior works that we compare our work to do use combinatorial techniques, these techniques are general-purpose LP-based primal/dual methods. Our algorithms are not only combinatorial, but also simple and elementary. We believe that these attributes provide similar benefits over LP-based methods as combinatorial algorithms provide compared to algebraic algorithms.

1.2 Approximating MDS on graphs of arboricity α

Centralized setting In the centralized setting, there are two combinatorial non-LP-based algorithms for MDS for graphs of arboricity (at most) α (for brevity, in what follows we may write graphs of "arboricity α" instead of arboricity *at most* α). One is by Lenzen and Wattenhofer [LW10], the other is by Jones, Lokshtanov, Ramanujan, Saurabh, and Suchý [JLR+13], and both achieve an $O(\alpha^2)$-approximation in deterministic linear time[1]. There is also an LP rounding algorithm by Bansal and Umboh that gives a 3α-approximation [BU17]. The algorithm of [BU17] also translates into a near-linear time algorithm using a general-purpose approximation result for explicit packing and covering LPs [You14, AZO19, Qua20]; specifically, one can get a deterministic $O(\alpha)$-approximation LP-based algorithm within $O(m \log n)$ time. Bansal and Umboh [BU17] also proved that achieving asymptotically better approximation is NP-hard.[2] Perhaps one can get a linear time $O(\alpha)$-approximation algorithm for MDS using LP-based approaches, but such a result has not been stated in the literature.

Distributed setting In the distributed setting, there are two combinatorial non-LP-based algorithms for MDS for graphs of arboricity α, both by Lenzen and Wattenhofer [LW10]. The first is a randomized $O(\alpha^2)$-approximation algorithm in the CONGEST model that runs in $O(\log n)$ rounds with high probability. This algorithm is deterministic except for a subroutine for maximal independent set (MIS), so one can apply the recent network decomposition based MIS algorithm by Ghaffari, Grunau, and Rozhoň [GGR21] to get a deterministic MIS algorithm. This yields an $O(\alpha^2)$-approximation algorithm for MDS in the CONGEST model that runs in $O(\log^5 n)$ rounds. The second algorithm of Lenzen and Wattenhofer is a deterministic $O(\alpha \log \Delta)$-approximation algorithm in the CONGEST model that runs in $O(\log \Delta)$ rounds, where Δ is the maximum degree.

Regarding LP-based algorithms, Kuhn, Moscibroda, and Wattenhofer [KMW06] developed a general-purpose method for solving LPs of a particular structure in the distributed setting. It seems that by applying

[1] Note that the theorem statement of [LW10] has a typo suggesting that the approximation factor is $O(\alpha)$.

[2] More specifically, achieving an $(\alpha-1-\varepsilon)$-approximation is NP-hard for any $\varepsilon > 0$ and any fixed α; achieving an $(\lfloor \alpha/2 \rfloor - \varepsilon)$-approximation is NP-hard for any $\varepsilon > 0$ and any $\alpha = 1, \ldots, \log^{\delta} n$, for some constant δ [BU17, DGKR05]. These hardness of approximation results are achieved by applying a reduction by [BU17] from the k-hypergraph vertex cover (k-HVC) problem (where we need to find a minimum vertex cover of a k-uniform hypergraph) to the MDS problem in arboricity-k graphs, in conjunction with NP-hardness results by [DGKR05] for the k-HVC problem.

their method (specifically, Corollary 4.1 of [KMW06]) to the LP approximation result of Bansal and Umboh in bounded arboricity graphs [BU17], one can get a deterministic $O(\alpha)$-approximation algorithm for MDS in the CONGEST model that runs in $O(\log^2 \Delta)$ rounds, but such a result has not been explicitly claimed in the literature.

A natural question The aforementioned results demonstrate a significant gap for MDS algorithms in bounded arboricity graphs when comparing LP-based methods to elementary combinatorial approaches. It is natural to ask whether this gap can be bridged:

- In the centralized setting, is there any efficient non-LP-based $O(\alpha)$-approximation algorithm for MDS (even one that is slower than the $O(m \log n)$-time LP-based algorithm)? Further, can one achieve an $O(\alpha)$-approximation in *linear time* using *any* (even LP-based) algorithm?
- In the distributed setting, is there any efficient non-LP-based distributed $O(\alpha)$-approximation algorithm for MDS? Further, can one achieve an $O(\alpha)$-approximation in the CONGEST model within $o(\log^2 \Delta)$ rounds using *any* (even LP-based) algorithm?

1.3 Our Contributions

We answer the above question in the affirmative. In particular, we give simple combinatorial algorithms that achieve the asymptotically optimal approximation factor of $O(\alpha)$, and run faster than all known algorithms, including LP-based algorithms.[3]

Our core contribution is a non-LP-based asymptotically optimal algorithm in the centralized setting.

Theorem 1.2. *For graphs of arboricity α, there is an $O(m)$ time $O(\alpha)$-approximation algorithm for MDS.*

Our algorithm is asymptotically optimal in both running time and approximation factor: it runs in linear time, and asymptotically improving the approximation factor it gets is proved to be NP-hard [BU17]. Furthermore, our algorithm is combinatorial and elementary, in contrast to the only other known $O(\alpha)$-approximation, which is LP-based; the improvement in running time over the LP-based algorithm is admittedly minor (only a logarithmic factor), but still getting a truly linear time algorithm seems qualitatively different than an almost-linear time.

We demonstrate the applicability of our simple centralized algorithm, by using its core ideas to develop a distributed algorithm.

Theorem 1.3. *For graphs of arboricity α, there is a randomized distributed algorithm in the* CONGEST *model that gives an $O(\alpha)$-approximation for MDS and runs in $O(\alpha \log n)$ rounds. The bound on the number of rounds holds with high probability (and in expectation).*

This is the first non-LP-based distributed $O(\alpha)$-approximation algorithm. For the "interesting" parameter regime where Δ is polynomial in n, and $\alpha = o(\log n)$, the number of rounds in our algorithm beats even the LP-based algorithm obtained by combining [KMW06] and [BU17] which appears to run in $O(\log^2 \Delta)$ rounds; as noted already, such an algorithm has not been claimed explicitly before. We note the caveat that our algorithm is randomized while their algorithm appears to be deterministic.

In the process of obtaining our distributed algorithm, we also obtain a *deterministic* non-LP-based algorithm in the LOCAL model (with polynomial message sizes) in a polylogarithmic number of rounds, via reduction to the maximal independent set (MIS) problem:

[3] $O(\alpha)$ is the asymptotically optimal approximation factor for polynomial time algorithms in the centralized and distributed settings i.e. assuming processors have polynomially-bounded processing power.

Theorem 1.4. *Suppose there is a deterministic (resp., randomized) distributed algorithm in the* LOCAL *model for computing an MIS on a general graph in R rounds. Then, for graphs of arboricity α, there is a deterministic (resp., randomized) distributed algorithm in the* LOCAL *model that gives an $O(\alpha)$-approximation for MDS in $O(R\alpha^2 \log n)$ rounds.*

While Theorem 1.4 is the first deterministic non-LP-based algorithm to achieve an $O(\alpha)$-approximation, we note that the LP-based approach obtained by combining [KMW06] and [BU17] appears to achieve fewer rounds and work in the CONGEST model. Theorem 1.4 is not our main result and is used as a stepping stone towards our $O(\alpha \log n)$ round algorithm in the CONGEST model.

Wider applicability We demonstrated the applicability of our centralized combinatorial algorithm to the distributed setting. We anticipate that the core idea behind our centralized algorithm could be applied more broadly, to other settings that involve locality. Perhaps the prime example in this context is the standard setting of dynamic graph algorithms, where the graph undergoes a sequence of edge updates (a single edge update per step), and the algorithm should maintain the graph structure of interest ($O(\alpha)$-approximate MDS in our case) with a small *update time* — preferably $\text{poly}\log(n)$ and ideally $O(1)$.

1.4 Technical overview

Centralized algorithm As a starting point, we consider the algorithm of Jones, Lokshtanov, Ramanujan, Saurabh, and Suchỳ [JLR+13], which achieves an $O(\alpha^2)$-approximation in linear time. Their algorithm is as follows. They iteratively build a dominating set D and maintain a partition of the remaining vertices into the dominated vertices B (the vertices that have a neighbor in D), and the undominated vertices W. The basic property of arboricity α graphs used by their algorithm is that every induced subgraph contains a vertex of degree $O(\alpha)$. They begin by choosing a vertex v with degree $O(\alpha)$ and adding v along with v's entire neighborhood $N(v)$ to D. The intuition behind this is that at least one vertex in $\{v\} \cup N(v)$ must be in OPT (an optimal dominating set), since OPT must dominate v. Hence, they add at least one vertex in OPT and use that to pay for adding $O(\alpha)$ vertices not in OPT. We say that a vertex w *witnesses* v and the vertices in $N(v)$ that are added to D, if $w \in OPT \cap (\{v\} \cup N(v))$. Now, the goal of the algorithm is to iteratively choose vertices v to add to D along with $O(\alpha)$ many of v's neighbors so that each vertex in OPT witnesses $O(\alpha)$ vertices v along with $O(\alpha)$ neighbors for each such vertex v. That is, each vertex in OPT witnesses $O(\alpha^2)$ vertices in D, which yields an $O(\alpha^2)$-approximation.

To choose which vertices v and which $O(\alpha)$ of v's neighbors to add to D, they partition the set B into two subsets B_{low} and B_{high}, which are the sets of vertices in B with low and high degree to W, respectively, where the degree threshold is $\delta\alpha$ for some constant δ. Then, they choose a vertex v with degree $O(\alpha)$ in the graph induced by $W \cup B_{high}$, and add v to D along with v's $O(\alpha)$ neighbors that are in $W \cup B_{high}$. In the interest of brevity, we will not motivate why this scheme achieves the desired outcome that each vertex in OPT witnesses $O(\alpha^2)$ vertices in D.

The key innovation in our algorithm that allows us to reduce the approximation factor from $O(\alpha^2)$ to $O(\alpha)$ is a simple but powerful idea. After choosing a vertex v to add to D, we do not *immediately* add $O(\alpha)$ of v's neighbors to D. Instead v casts a "vote" for these $O(\alpha)$ neighbors, and only once a vertex gets $\delta\alpha$ many votes (for a constant δ) is it added to D. With this modification, we can argue that each vertex in OPT still witnesses $O(\alpha)$ such vertices v as in the previous approach, but the catch here is that each such vertex v contributes only $O(1)$ neighbors to D on average, so each vertex in OPT only witnesses a *total* of $O(\alpha)$ vertices in D, rather than $O(\alpha^2)$. Moreover, it is straightforward to implement this algorithm in linear time.

Distributed algorithms using MIS This section concerns the proof of Theorem 1.4: our reduction from MDS to MIS in the LOCAL model. This section also concerns a modification of this reduction that gives an $O(\alpha^2 \log^2 n)$ round algorithm in the CONGEST model. We use this algorithm as a stepping stone towards obtaining our main distributed algorithm (Theorem 1.3) which runs in $O(\alpha \log n)$ rounds in the CONGEST model.

We adapt our centralized algorithm to the distributed setting as follows. Recall that in our centralized algorithm, we repeatedly choose a vertex v that has low degree with respect to the graph induced by $W \cup B_{high}$, add v to D, and cast a vote for each vertex in $N(v) \cap (W \cup B_{high})$. For our distributed algorithms, we would like to choose *many* such vertices v and process them in *parallel*. In fact, a constant fraction of the vertices in $W \cup B_{high}$ could be chosen as our vertex v since a constant fraction of vertices in a graph of arboricity α have degree $O(\alpha)$. However, we cannot simply process all of these vertices in parallel. In particular, if a vertex u has many neighbors being processed in parallel, u might accumulate many votes during a single round. This would invalidate the analysis of the algorithm, which relies on the fact that once a vertex u receives $\delta\alpha$ votes, u enters D.

To overcome this issue, we compute an MIS with respect to a 2-hop graph built from a carefully chosen subgraph of "candidate" vertices, and only process the vertices in this MIS in parallel. This MIS has two useful properties: 1. Its maximality implies that in any 2-hop neighborhood of a candidate vertex there is a vertex in the MIS; this helps to bound the number of rounds, and 2. Its independence implies that every vertex has at most one neighbor in the MIS, which ensures that any vertex can only receive one vote per round. To conclude, this approach gives a reduction from distributed MDS to distributed MIS in the LOCAL model. This approach can be made to work in the CONGEST model by replacing the black-box MIS algorithm with a 2-hop version of Luby's algorithm.

Faster randomized distributed algorithm In the CONGEST model, our distributed algorithm using MIS runs in $O(\alpha^2 \log^2 n)$ rounds with high probability. We devise a new, more nuanced algorithm that decreases the number of rounds to $O(\alpha \log n)$ with high probability. Our new algorithm is based on our previous algorithm, but with two key modifications, which save factors of $\log n$ and α, respectively.

Our first key modification, which shaves a $\log n$ factor from the number of rounds, is that we do not run an MIS algorithm as a black box. Instead, we run only a single phase of a Luby-like MIS algorithm before updating the data structures. Intuitively, this saves a $\log n$ factor because we are running just one phase of a $O(\log n)$-phase algorithm, but it is not clear a priori whether we achieve the same progress as Luby's algorithm in a single phase. We demonstrate that this is indeed the case via more refined treatment of the behavior of each edge.

Our second key modification, which shaves an α factor from the number of rounds, concerns the Luby-like algorithm. Recall that in Luby's algorithm, each vertex v picks a random value $p(v)$ and then joins the MIS if $p(v)$ is the local minimum. In our algorithm, a vertex v instead joins the dominating set if $p(v)$ is an α-minimum, which roughly means that $p(v)$ is among the α smallest values that it is compared to. We show that with this relaxed definition, we still have the desired property that no vertex receives more than $\delta\alpha$ votes in a single round.

The main technical challenge is the analysis of the number of rounds. It is tempting to use an analysis similar to that of Luby's algorithm, where we count the expected number of "removed edges" over time. However, our above modifications introduce several complications that preclude such an analysis. Instead, we use a carefully chosen function to measure our progress. Throughout the algorithm, we add "weight" to particular edges, and our function measures the "total available weight". Specifically, whenever a vertex v is added to the dominating set, v adds *weight* to a particular set of edges in its 2-hop neighborhood. We

show that the total amount of weight added in a single iteration of the algorithm decreases the total available weight substantially, which allows us to bound the total number of iterations.

1.5 Organization

Section 2 is for preliminaries. In Section 3, we present our centralized algorithm (Theorem 1.2). In Section 4, we present our distributed algorithms using MIS: in the LOCAL model we prove Theorem 1.4, and in the CONGEST model we give a randomized algorithm with $O(\alpha^2 \log^2 n)$ rounds that serves as a warm-up for the faster algorithm of Theorem 1.3. In Section 5, we prove Theorem 1.3.

2 Preliminaries

Let $G = (V, E)$ be an unweighted undirected graph. For any $S \subseteq V$, let $G[S]$ be denote the graph induced by S. For any $v \in V$, $N_G(v)$ denotes the neighborhood of v, and $\deg_G(v) = |N_G(v)|$ denotes the degree of v. When the graph G is clear from context, we omit the subscript.

We define the LOCAL and CONGEST models (cf. [Lin87, Lin92, Pel00]):

Definition 2.1. The LOCAL model: given a graph G on n vertices, every vertex is a separate processor running one process. Every vertex starts knowing only n and it's own unique identifier. The algorithm works in synchronous rounds, and in every round each vertex performs some computation based on its own current information, then it sends a message to its neighbors, and finally it receives the messages sent to it by its neighbors in that round.

Definition 2.2. The CONGEST model: given a graph G on n vertices, every vertex is a separate processor running one process. Every vertex starts knowing only n and it's own unique identifier. The algorithm works in synchronous rounds, and in every round each vertex performs some computation based on its own current information, then it sends a message to its neighbors, of at most $B = O(\log n)$ bits on each of its edges, and finally it receives the messages sent to it by its neighbors in that round.

The following two claims about graphs of bounded arboricity will be useful. A simple proof of both can be found in [AMZ97].

Claim 2.3. *In a graph of arboricity α, every induced subgraph contains a vertex of degree at most 2α.*

Claim 2.4. *In a graph G with arboricity α, at least half of the vertices in any induced subgraph have degree at most 4α.*

3 Linear time $O(\alpha)$-approximation for MDS

In this section we will prove Theorem 1.2, which we recall:

Theorem 1.2. *For graphs of arboricity α, there is an $O(m)$ time $O(\alpha)$-approximation algorithm for MDS.*

3.1 Algorithm

A description of our algorithm is as follows. See Algorithm 1 for the pseudocode.

We first introduce some notation. We define a constant δ and let $\delta\alpha$ be our *degree threshold*. We will set $\delta = 2$, but we use the variable δ so that our analysis also applies to our distributed algorithms, where δ is a different constant. Following the terminology of [JLR+13], we maintain a partition of the vertices into three sets: D, B, and W, where initially $D = \emptyset$, $B = \emptyset$, and $W = V$. The set D is our current dominating set, the set B is the vertices not in D with at least one neighbor in D, and the set W is the remaining vertices, i.e. the undominated vertices. The set B is further partitioned into two sets based on the degree of each vertex to W. Let $B_{low} = \{v \in B : |N(v) \cap W| \leq \delta\alpha\}$ and let $B_{high} = B \setminus B_{low}$. Also, each vertex v has a *counter* c_v initialized to 0. (The counter c_v counts the number of "votes" that v receives, for the notion of "votes" introduced in the technical overview.)

The algorithm proceeds as follows. While there still exists an undominated vertex (i.e. while $W \neq \emptyset$), we do the following. First, we find a vertex $w \in W$ such that $|N(w) \cap (W \cup B_{high})| \leq \delta\alpha$. Such a vertex w exists since $\delta = 2$: by Claim 2.3, $G[W \cup B_{high}]$ contains a vertex of degree at most $\delta\alpha$, and this vertex cannot be in B_{high} by the definition of B_{high}, so it must be in W. Then, for all $v \in N(w) \cap (W \cup B_{high})$, we increment c_v, and if $c_v = \delta\alpha$, then we add v to D. Lastly, we add w to D. This concludes the description of the algorithm.

Algorithm 1 Linear time $O(\alpha)$-approximation for MDS

Initialize partition: $D \leftarrow \emptyset$, $B = \emptyset$, $B_{high} \leftarrow \emptyset$, $B_{low} \leftarrow \emptyset$, $W \leftarrow V$
Initialize counters: $\forall v \in V : c_v \leftarrow 0$
while $W \neq \phi$ **do**
    $w \leftarrow$ a vertex in W with $|N(w) \cap (W \cup B_{high})| \leq \delta\alpha$
    for each $v \in N(w) \cap (W \cup B_{high})$ **do**
        $c_v \leftarrow c_v + 1$
        if $c_v = \delta\alpha$ **then**
            $D \leftarrow D \cup v$
    Update partition:
    $D \leftarrow D \cup w$
    $B = \{v : N(v) \cap D \neq \emptyset\}$
    $B_{low} = \{v \in B : |N(v) \cap W| \leq \delta\alpha\}$
    $B_{high} = B \setminus B_{low}$
    $W = V \setminus (D \cup B)$
Return D

3.2 Analysis

First, we note that D is indeed a dominating set because the algorithm only terminates once the set W of vertices that are not dominated, is empty.

3.2.1 Approximation ratio analysis

Let OPT be an optimal MDS. We will prove that the set D returned by Algorithm 1 is of size at most $4\delta\alpha \cdot |OPT|$.

We first make the following simple claim about the behavior of the partition of vertices over time.

Claim 3.1.

1. *No vertex can ever leave D.*
2. *No vertex can ever enter W from another set.*
3. *No vertex can ever leave B_{low}.*

Proof. Item 1 is by definition. Item 2 follows from item 1 combined with the fact that W is defined as the set of vertices with no neighbors in D. Now we prove item 3. A vertex from B_{low} cannot enter W by item 2. A vertex from B_{low} cannot enter B_{high} since the degree partition of B is based on degree to W, and by item 2 the degree of any vertex to W can only decrease over time. A vertex from B_{low} cannot enter D because there are two ways a vertex can enter D: on Line 8 a vertex can only enter D from $W \cup B_{high}$, and on Line 10 a vertex can only enter D from W. □

We partition D into two sets, D_{active} and $D_{passive}$. The set D_{active} consists of the vertices added to D due to being chosen as the vertex w; that is, the vertices added to D in Line 10 of Algorithm 1. The set $D_{passive}$ consists of the vertices added to D as a result of their counters reaching $\delta\alpha$; that is, the vertices added to D in Line 8 of Algorithm 1. To bound $|D|$, we will individually bound $|D_{active}|$ and $|D_{passive}|$. We first bound $|D_{active}|$.

Claim 3.2. $|D_{active}| \leq 2\delta\alpha \cdot |OPT|$.

Proof. For each vertex $v \in D_{active}$, we *assign* v to an arbitrary vertex $u \in N(v) \cap OPT$. Such a vertex u exists since OPT is a dominating set. For each vertex $u \in OPT$, let $D_u \subseteq D_{active}$ be the set of vertices assigned to u. Our goal is to show that for each $u \in OPT$, $|D_u| \leq 2\delta\alpha$.

Fix a vertex $u \in OPT$. We partition the vertices $v \in D_u$ into two sets $D_u[B_{low}]$ and $D_u[B_{high} \cup W]$. Let $D_u[B_{low}] \subseteq D_u$ be the vertices that enter D while u is in B_{low}. Let $D_u[B_{high} \cup W] \subseteq D_u$ be vertices that enter D while u is in $B_{high} \cup W$. We note that no vertex in D_u can enter D while u is in D, because by definition, every vertex in $D_{active} \supseteq D_u$ moves directly from W to D. Therefore, $D_u = D_u[B_{low}] \cup D_u[B_{high} \cup W]$.

We first bound $\left|D_u[B_{low}]\right|$. By definition, while u is in B_{low}, u has at most $\delta\alpha$ neighbors in W. Since no vertex can ever enter W by Claim 3.1, no vertex can ever enter $N(u) \cap W$. Therefore, starting from the time that u first enters B_{low}, the total number of vertices ever in $N(u) \cap W$ is at most $\delta\alpha$. Every vertex $v \in D_u[B_{low}]$ is in $N(u) \cap W$ right before moving to D, so $\left|D_u[B_{low}]\right| \leq \delta\alpha$.

Now, we bound $D_u[B_{high} \cup W]$. By the specification of the algorithm, whenever a vertex $v \in D_u[B_{high} \cup W]$ enters D, the counter x_u is incremented. Once x_u reaches $\delta\alpha$, u is added to D. Therefore, $\left|D_u[B_{high} \cup W]\right| \leq \delta\alpha$.

Putting everything together, we have $|D_u| = \left|D_u[B_{low}]\right| + \left|D_u[B_{high} \cup W]\right| \leq 2\delta\alpha$. □

Now we bound $D_{passive}$.

Claim 3.3. $|D_{passive}| \leq |D_{active}|$.

Proof. We will show that every vertex in $D_{passive}$ has at least $\delta\alpha$ neighbors in D_{active}, while every vertex in D_{active} has at most $\delta\alpha$ neighbors in $D_{passive}$. Then, by the pigeonhole principle, it follows that $|D_{passive}| \leq |D_{active}|$.

First, we will show that every vertex in $D_{passive}$ has at least $\delta\alpha$ neighbors in D_{active}. By definition, every vertex $v \in D_{passive}$ has had its counter c_v incremented $\delta\alpha$ times. Every time c_v is incremented, one

of v's neighbors (the vertex w from Algorithm 1) is added to D, joining D_{active}. Each such neighbor of v that joins D_{active} is distinct since every vertex can be added to D at most once by Claim 3.1. Therefore, every vertex in $D_{passive}$ has at least $\delta\alpha$ neighbors in D_{active}.

Now we will show that every vertex in D_{active} has at most $\delta\alpha$ neighbors in $D_{passive}$. Fix a vertex $w \in D_{active}$. By definition, when w enters D, w is moved straight from W to D. Thus, by Claim 3.1, w is never in B. Therefore, w is added to D before any of its neighbors are added to D, as otherwise w would enter B. Therefore, when w enters D, all of w's neighbors that will enter $D_{passive}$ are in $B \cup W$. By Claim 3.1, no vertex in B_{low} can ever enter D, so actually, when w enters D all of w's neighbors that will enter $D_{passive}$ are in $B_{high} \cup W$. By definition, when w enters D, w has at most $\delta\alpha$ neighbors in $B_{high} \cup W$. Therefore, w has at most $\delta\alpha$ neighbors in $D_{passive}$. □

Combining Claim 3.2 and Claim 3.3, we have that $|D| = |D_{active}| + |D_{passive}| \leq 4\delta\alpha \cdot |OPT|$.

3.2.2 Running time analysis

Our goal is to prove that Algorithm 1 runs in $O(m)$ time.

Throughout the execution of the algorithm, we maintain a data structure that consists of the following:

- The partition of V into D, B_{low}, B_{high}, W
- The induced graph $G[W \cup B_{high}]$ represented as an adjacency list
- For each vertex $v \in W \cup B_{high}$, the quantities $|N(v) \cap W|$ and $|N(v) \cap (W \cup B_{high})|$
- A set $W_{low} = \{v \in W : |N(v) \cap (W \cup B_{high})| \leq \delta\alpha\}$

First, we show that the data structure can be initialized in $O(m)$ time. Initially $D \cup B_{high} \cup B_{low} = \emptyset$, $W = V$, and the induced graph $G[W \cup B_{high}] = G$. For every vertex $v \in V$, initially $|N(v) \cap W| = |N(v) \cap (W \cup B_{high})| = \deg(v)$. Initially $W_{low} = \{v \in V : \deg(v) \leq \delta\alpha\}$.

Now, we show that the data structure can be maintained in $O(m)$ time. In particular, we will show that to maintain this data structure, it suffices to scan the neighborhood of a vertex every time it either leaves $W \cup B_{high}$ (and enters $B_{low} \cup D$), enters D, or leaves W. Note that by Claim 3.1, each of these events only happens once per vertex. As a consequence, the total amount of time spent scanning neighborhoods is $O(m)$.

We assume inductively that we have maintained the data structure so far, and we consider the next iteration of the **for each** loop. First, we consider maintenance of the partition of V into D, B_{low}, B_{high}, and W. During an iteration, the only changes made to the partition are the addition of at least one vertex to D (on Line 8 and/or Line 10), and the resulting update of the rest of the partition. To maintain the partition we do the following. When we add a vertex v to D, we remove v from whichever set it was previously in. Then, we update B by scanning $N(v)$ and adding every vertex $u \in N(v) \setminus D$ to B, removing u from whichever set it was previously in. Updating D and B automatically updates W since $W = V \setminus (D \cup B)$. Before updating B_{low} and B_{high}, we first need to update $|N(v) \cap W|$. To do this, whenever a vertex v leaves W, we scan $N(v)$ and for each $u \in N(v)$, we decrement $|N(u) \cap W|$. Whenever we decrement $|N(u) \cap W|$ down to $\delta\alpha$ for a vertex $u \in B_{high}$, we move u to B_{low}. This concludes the maintenance of the partition of V into D, B_{low}, B_{high}, and W.

It remains to update $G[W \cup B_{high}]$, $|N(v) \cap (W \cup B_{high})|$, and W_{low}. Whenever we remove a vertex v from $W \cup B_{high}$, we scan $N(v)$ and for each vertex $u \in N(v)$, we remove the edge (u, v) from $G[W \cup B_{high}]$ and decrement $|N(u) \cap (W \cup B_{high})|$. Whenever we decrement $|N(u) \cap (W \cup B_{high})|$ down to $\delta\alpha$ for $u \in W$, we add u to W_{low}. This concludes the running time analysis for maintaining the data structure.

Now we will show that maintaining the data structure allows the algorithm to run in time $O(m)$. First, each iteration of the **while** loop adds at least one vertex to D (on Line 10), and by Claim 3.1, each vertex is added to D at most once, so the total number of iterations of the **while** loop is at most n. Now we will go line by line through the body of the **while** loop. On Line 4, we let w be a vertex in W with $|N(w) \cap (W \cup B_{high})| \leq \delta\alpha$. Such a vertex w can be found in constant time since we maintain a set W_{low} of precisely the vertices that satisfy this condition. On Line 5, we loop through every vertex in $|N(w) \cap (W \cup B_{high})|$. The number of iterations of this loop is at most $\delta\alpha$ by choice of w. Furthermore, identifying all of the vertices to loop through takes time $O(\alpha)$ since our data structure explicitly maintains $G[W \cup B_{high}]$. In Line 6 through Line 10, we update counters and then add vertices to D, which takes constant time per iteration of the loop. In Line 11 through Line 14 we update B, B_{low}, B_{high}, and W, which takes constant time since we store these sets in our data structure. Thus, given access to the data structure, the algorithm runs in time $O(n\alpha) = O(m)$.

Previously we showed that maintaining the data structure takes time $O(m)$, so we have that the entire algorithm takes time $O(m)$.

4 Distributed $O(\alpha)$-approximation for MDS using MIS

In this section we will prove Theorem 1.4, which we recall:

Theorem 1.4. *Suppose there is a deterministic (resp., randomized) distributed algorithm in the* LOCAL *model for computing an MIS on a general graph in R rounds. Then, for graphs of arboricity α, there is a deterministic (resp., randomized) distributed algorithm in the* LOCAL *model that gives an $O(\alpha)$-approximation for MDS in $O(R\alpha^2 \log n)$ rounds.*

In this section we also show how to modify of the proof of Theorem 1.4 to get a bound in the CONGEST model:

Theorem 4.1. *For graphs of arboricity α, there is a randomized distributed algorithm in the* CONGEST *model that gives an $O(\alpha)$-approximation for MDS that runs in $O(\alpha^2 \log^2 n)$ rounds with high probability.*

In the next section, we will use the algorithm of Theorem 4.1 as a starting point to get an improved algorithm with $O(\alpha \log n)$ rounds.

4.1 Algorithm

Overview Our algorithm is an adaptation of our centralized algorithm from Theorem 1.2 to the distributed setting. Recall that in our centralized algorithm, we repeatedly choose a vertex w that has low degree with respect to the graph induced by $W \cup B_{high}$, add w to the dominating set, and increment the *counter* of w's neighbors that are in $W \cup B_{high}$. For our distributed algorithms, we would like to choose *many* such vertices w and process them in parallel. There are in fact many vertices that we could choose as our vertex w since Claim 2.4 implies that at least half of the vertices in any induced subgraph has degree at most 4α. However, we cannot simply process all of these vertices at once. In particular, if a vertex v has many neighbors being processed in parallel, v might have its counter incremented once for each of these neighbors. This is undesirable because the analysis of our centralized algorithm relies on the fact that once a vertex has its counter incremented to $\delta\alpha$, it is added to the dominating set. Therefore, we would like to guarantee that only a limited number of v's neighbors are processed in parallel.

This is where the MIS problem becomes relevant: we ensure that no vertex has more than one neighbor being processed in parallel by taking an MIS I with respect to the graph G_{low} defined as follows: the vertex set of G_{low} is the set of candidates for w, that is, the set of vertices v in W with $|N(v) \cap (W \cup B_{high})| \leq 4\alpha$. There is an edge (u, v) in G_{low} if there is a path of length 2 between u and v in $G[W \cup B_{high}]$. Note that because no vertex has more than one neighbor in I, we can process all vertices in I in parallel and only increase the counter of each vertex by at most one.

The algorithms for Theorem 1.4 and Theorem 4.1 are identical except for the MIS subroutine. Theorem 1.4 is for the LOCAL model so we can simply run any distributed MIS algorithm that works in the LOCAL model on G_{low} as a black box. On the other hand, Theorem 4.1 is for the CONGEST model and because G_{low} can have higher degree than G, running an MIS algorithm directly on G_{low} could result in messages that become too large after translating the algorithm to run on G. To bypass this issue, we use a simple modification of Luby's algorithm that computes I using only small messages, without increasing the number of rounds.

Algorithm description We provide a description of the algorithms here, and include the pseudocode in Algorithm 2. The only difference between the algorithms for Theorem 1.4 and Theorem 4.1 is the MIS subroutine, which we will handle separately later.

The sets D, B, W, B_{high}, B_{low}, and W_{low} are defined exactly the same as in our centralized algorithm, except we set $\delta = 4$ instead of $\delta = 2$ so that we can apply Claim 2.4 instead of Claim 2.3. We repeat the definitions here for completeness. The set D is our current dominating set, the set B is the vertices not in D with at least one neighbor in D, and the set W is the remaining vertices, i.e. the undominated vertices. The set B is further partitioned into two sets based on the degree of each vertex to W. Let $B_{low} = \{v \in B : |N(v) \cap W| \leq \delta\alpha\}$ and let $B_{high} = B \setminus B_{low}$. Also, let $W_{low} = \{v \in W : |N(v) \cap (W \cup B_{high})| \leq \delta\alpha\}$. Lastly, each vertex v has a *counter* c_v.

Each vertex v maintains the following information:

- The set(s) among D, B, W, B_{high}, B_{low}, and W_{low} that v is a member of.
- The quantity $|N(v) \cap W|$.
- The quantity $|N(v) \cap (W \cup B_{high})|$.
- The counter c_v.

At initialization, every vertex v is in W (so D and B are empty). Consequently, the quantities $|N(v) \cap W|$ and $|N(v) \cap (W \cup B_{high})|$ are both equal to $\deg(v)$. For each vertex v, if $\deg(v) \leq \delta\alpha$, then $v \in W_{low}$. Each counter c_v is initialized to 0.

It will be useful to define the graph G_{low}, which changes over the execution of the algorithm:

Definition 4.2. Let G_{low} be the graph with vertex set W_{low} such that there is an edge (u, v) in G_{low} if there is a path of length 2 between u and v in $G[W \cup B_{high}]$.

The algorithm proceeds as follows. Repeat the following until W is empty. Compute an MIS I with respect to G_{low}. This step is implemented differently for Theorem 1.4 and Theorem 4.1, and we describe the details of this step later.

Then, each vertex in I adds itself to D and tells its neighbors to increment their counters. Whenever the counter of a vertex reaches $\delta\alpha$, it enters D. (Note that if a vertex u enters D as a result of c_u reaching $\delta\alpha$, c_u does *not* tell its neighbors to increment their counters.)

Whenever a vertex moves from one set of the partition to another, it notifies each of its neighbors v so that v can update the quantities $|N(v) \cap W|$ and $|N(v) \cap (W \cup B_{high})|$, and move to the appropriate set.

When no more vertices are left in W, B_{high} is also empty, and all processors terminate. This concludes the description of the algorithm. See Algorithm 2 for the precise ways that vertices react to the messages that they receive.

MIS subroutine Theorem 1.4 is a reduction from MDS to MIS, while Theorem 4.1 is not, so we need to describe the MIS subroutine (in the CONGEST model) only for Theorem 4.1. Recall that we cannot use a reduction to MIS in the CONGEST model because running an MIS algorithm directly on G_{low} could result in messages that become too large after translating the algorithm to run on G.

Our goal is to compute an MIS with respect to G_{low}, using small messages sent over G. We use a simple adaptation of Luby's algorithm. Recall that Luby's algorithm builds an MIS I as follows. While the graph is non-empty, do the following: Add all singletons to I. Then, each vertex v picks a random value $p(v) \in [0, 1]$. Then, all vertices whose value is less than that of all of their neighbors are added to I. Then, all vertices that are in I or have a neighbor in I are removed from the graph for the next iteration of the loop.

We use the following adaptation of Luby's algorithm. See Algorithm 3 for the pseudocode. Initially, the set L of *live* vertices is the set W_{low}. While $L \neq \emptyset$, do the following: Each vertex $v \in L$ picks a random value $p(v) \in [0, 1]$. In the first round each $v \in L$ sends $p(v)$ to its neighbors. In the second round, each vertex that receives one or more values $p(v)$, forwards to its neighbors the minimum value that it received. Then, for each vertex $v \in W_{low}$, if $p(v)$ is equal to the minimum value that v receives in the second round, v is added to I. When v is added to I, v notifies its neighbors, and each neighbor of v that is in $W \cup B_{high}$ forwards this notification to their neighbors. Note that each vertex has at most one neighbor in I, so forwarding this notification only takes one round. Now, every vertex knows whether it has a neighbor with respect to G_{low} that is in I, and every vertex that does is removed from L for the next iteration of the loop.

The proof that this algorithm runs in $O(\log n)$ rounds with high probability and produces an MIS with respect to G_{low} is the same as the analysis of Luby's algorithm and we will not include it here.

4.2 Analysis

The proof that Algorithm 2 achieves an $O(\alpha)$-approximation is precisely the same as that of the centralized algorithm (see Section 3.2.1) given that no counter c_v ever exceeds $\delta\alpha$. This is true because in a single iteration of the **while** loop each vertex can only have its counter incremented once since only vertices in the MIS I send INCREMENT COUNTER messages, and each vertex in $W \cup B_{high}$ only has at most one neighbor in I. This bound on the number of neighbors in I holds, since otherwise there is a path of length 2 between two vertices in $G[W \cup B_{high}]$, making I not an independent set in G_{low}. Once c_v reaches $\delta\alpha$, the vertex v enters D, which prevents c_v from increasing in the future.

Our goal in this section is to prove that if the MIS subroutine takes R rounds, then Algorithm 2 takes $O(R\alpha^2 \log n)$ rounds. First, we note that the body of the **while** loop besides the MIS subroutine takes a constant number of rounds. Thus, our goal is to show that the number of iterations of the **while** loop is $O(\alpha^2 \log n)$.

We begin with a simple claim about the behavior of the partition of vertices over time:

Claim 4.3.

1. *No vertex can ever enter W from another set.*
2. *No vertex can ever enter W_{low} from another set.*

Algorithm 2 Distributed $O(\alpha)$-approximation for MDS using MIS

Initialize partition: $D \leftarrow \emptyset$, $B_{high} \leftarrow \emptyset$, $B_{low} \leftarrow \emptyset$, $W \leftarrow V$, $W_{low} \leftarrow \{v \in W : |N(v) \cap (W \cup B_{high})| \leq \delta\alpha\}$
Initialize counters: $\forall v \in V : c_v \leftarrow 0$
Initialize degrees: $\forall v \in V : |N(v) \cap W| = \deg(v)$, $|N(v) \cap (W \cup B_{high})| = \deg(v)$
while $W \neq \emptyset$ **do**
    Find an MIS I with respect to the graph G_{low}
    Each vertex v runs the following procedure:
    if $v \in I$ **then**
        Move v to D
        Send INCREMENT COUNTER message to neighbors
        Send MOVED FROM W TO D message to neighbors
    if $v \in W \cup B_{high}$ and v receives INCREMENT COUNTER **then**
        Increment c_v
        if $c_v = \delta\alpha$ **then**
            if $v \in W$ **then**
                Send MOVED FROM W TO D message to neighbors
            if $v \in B_{high}$ **then**
                Send MOVED FROM B_{high} TO D message to neighbors
            Move v to D
    // The rest of the algorithm is bookkeeping
    if v receives MOVED FROM W TO D **then**
        Decrement $|N(v) \cap W|$
        if $v \in B_{high}$ and $|N(v) \cap W| = \delta\alpha$ **then**
            Move v to B_{low}
    if v receives MOVED FROM W TO D or MOVED FROM B_{high} TO D **then**
        Decrement $|N(v) \cap (W \cup B_{high})|$
        if $v \in W$ and $|N(v) \cap W| \leq \delta\alpha$ **then**
            Move v to B_{low}
            Send MOVED FROM W TO B_{low} message to neighbors
        else if $v \in W$ and $|N(v) \cap W| > \delta\alpha$ **then**
            Move v to B_{high}
            Send MOVED FROM W TO B_{high} message to neighbors
    if v receives MOVED FROM W TO B_{low} or MOVED FROM W TO B_{high} **then**
        Decrement $|N(v) \cap W|$
        if $v \in B_{high}$ and $|N(v) \cap W| = \delta\alpha$ **then**
            Move v to B_{low}
    if v receives MOVED FROM W TO B_{low} **then**
        Decrement $|N(v) \cap (W \cup B_{high})|$
        if $v \in W$ and $|N(v) \cap (W \cup B_{high})| = \delta\alpha$ **then**
            Add v to W_{low}

Algorithm 3 Distributed MIS with respect to G_{low} in the CONGEST model

$L = W_{low}$
while $L \neq \emptyset$ **do**
  Each vertex v runs the following procedure:
  if $v \in L$ **then**
    $p(v) \leftarrow$ a value in $[0, 1]$ chosen uniformly at random
    Send $p(v)$ message to neighbors
  Send $m_v = \min_{y \in N(v) \cap L} p(y)$ message to neighbors
  if $p(v) = \min_{y \in N(v)} m_y$ **then**
    Add v to I
    Send ADDED message to neighbors
  if $v \in W \cup B_{high}$ and v receives ADDED **then**
    Send NEIGHBOR ADDED message to neighbors
  if v receives NEIGHBOR ADDED and $v \in L$ **then**
    Remove v from L

Proof. The proof of item 1 is the same as in the proof of Claim 3.1. For item 2, we have that by item 1, no vertex can ever enter W_{low} from any set other than W_{high}. However, it is impossible for a vertex to move from W_{high} to W_{low} since the quantity $N(v) \cap (W \cup B_{high})$ can only decrease over time (in Algorithm 2, this quantity is only decremented). □

We begin with the following claim, which when combined with Claim 4.3, implies that each vertex only spends a limited number of rounds in W_{low}.

Claim 4.4. *For every vertex v that is ever in W_{low}, within $(\delta\alpha)^2$ iterations of the* **while** *loop after v joins W_{low}, v leaves W.*

Proof. First we note that by Claim 4.3 no vertex can ever enter W_{low} from another set. Suppose v is in W_{low} at the beginning of an iteration of the **while** loop. Because I is an MIS with respect to G_{low}, if v does not join I during this iteration, then v has a neighbor $y \in W \cup B_{high}$ such that a neighbor z of y joins I. As a result, z immediately joins D and c_y is incremented. Thus, during every iteration that v remains in W_{low}, a vertex in $N(v) \cap (W \cup B_{high})$ has its counter incremented. Recall that whenever a vertex has its counter incremented $\delta\alpha$ times, it joins D. Because $v \in W_{low}$, we have that $|N(v) \cap (W \cup B_{high})| \leq \delta\alpha$. Therefore, the event that a vertex in $N(v) \cap (W \cup B_{high})$ has its counter incremented can only happen at most $(\delta\alpha)^2$ times. Thus, v can only remain in W_{low} for $(\delta\alpha)^2$ iterations of the **while** loop. □

We will complete the analysis using the fact that many vertices are in W_{low} at any given point in time. In particular, Claim 2.4 implies that at least half of the vertices in $W \cup B_{high}$ are in W_{low}. This implies that at least half of the vertices in W are in W_{low}. Formally, we divide the execution of the algorithm into phases where each phase consists of $(\delta\alpha)^2$ iterations of the **while** loop. At the beginning of any phase, at least half of the vertices in W are in W_{low}. By the end of the phase, all of these vertices have left W by Claim 4.4. Therefore, each phase witnesses at least half of the vertices in W leaving W. By Claim 4.3, no vertex can re-enter W, so there can only be $O(\log n)$ phases.

Putting everything together, there are $O(\log n)$ phases, each consisting of $(\delta\alpha)^2$ iterations of the **while** loop, and one iteration of the **while** loop takes $O(R)$ rounds. Therefore, the total number of rounds is $O(R\alpha^2 \log n)$.

For Theorem 4.1, $R = O(\log n)$, so the number of rounds is $O(\alpha^2 \log^2 n)$.

5 Faster Randomized Distributed $O(\alpha)$-approximation for MDS

In this section we will prove Theorem 1.3, which we recall:

Theorem 1.3. *For graphs of arboricity α, there is a randomized distributed algorithm in the* CONGEST *model that gives an $O(\alpha)$-approximation for MDS and runs in $O(\alpha \log n)$ rounds. The bound on the number of rounds holds with high probability (and in expectation).*

5.1 Algorithm

Overview We use our $O(\alpha^2 \log^2 n)$ round algorithm from Theorem 4.1 as a starting point (though the algorithm description and analysis are self-contained). Our goal is to shave both a $\log n$ factor and an α factor from the number of rounds. To do so, we use a combination of two key modifications, which respectively address the two factors that we wish to shave.

Our first key modification, which shaves a $\log n$ factor from the number of rounds, is that instead of using a Luby-style algorithm as a black box, we open the box and run only one phase of a Luby-style algorithm at a time. Here, one phase means that each participating vertex v picks a single random value $p(v)$ and enters D if $p(v)$ is a local minimum. Between each such phase, we update the dominating set D as well as the information stored by each vertex. This way, we can embed the analysis of the Luby-style algorithm into our analysis instead of repeatedly paying for for all $\log n$ phases of a black-box algorithm.

Our second key modification, which shaves an α factor from the number of rounds, is that instead of adding v to D only when $p(v)$ is the single local minimum, we allow v to be added to D when $p(v)$ is an *α-minimum*. The definition of an *α-minimum* is slightly nuanced due to the fact that we need to be able to compute it using small messages, but it roughly means that $p(v)$ is among the α smallest values that it is compared to. Using this modification we can ensure that during each iteration of our algorithm each vertex only has its counter incremented by $O(\alpha)$. Even though each vertex in our previous $O(\alpha^2 \log^2 n)$-round algorithm only had its counter incremented by at most 1 during each iteration, this change does not asymptotically increase the approximation factor.

The main technical part of the argument is the probabilistic analysis of the number of rounds. We would like to use an analysis similar to that of Luby's algorithm, however there are a few obstacles. Recall that to analyze Luby's algorithm, one can argue that after a single phase, a constant fraction of the edges in the graph are removed in expectation. Our first obstacle is that we are running a phase of a Luby-style algorithm on an auxiliary graph that is different from our original graph; in particular, an edge in the auxiliary graph can represent a 2-hop path in the original graph, and it is not clear how removing an edge from the auxiliary graph translates to the original graph. That is, if a constant fraction of edges are removed in the auxiliary graph, this doesn't necessarily mean that a constant fraction of edges in the original graph are removed. A second obstacle is that we need a more nuanced notion than "removing an edge" as in Luby's algorithm since due to our second modification, up to α vertices could all affect the same edge simultaneously. To address these obstacles, we use a carefully chosen function to measure our progress. Throughout the algorithm, we add "weight" to particular edges, and our function measures the "total available weight". Specifically, whenever a vertex v is added to the dominating set, v adds *weight* to a particular set of edges in its 2-hop neighborhood. We show that the total amount of weight added in a single iteration of the algorithm decreases the expected total available weight substantially, which allows us to bound the total number of iterations.

Algorithm Description We include a description of the algorithm here, and include the pseudocode in Algorithm 4.

The partition of the vertices is exactly the same as in Algorithm 2, with the addition of the set W_{high}. We repeat all of the definitions for completeness. The set D is our current dominating set, the set B is the vertices not in D with at least one neighbor in D, and the set W is the remaining vertices, i.e. the undominated vertices. The set B is further partitioned into two sets based on the degree of each vertex to W. Let $B_{low} = \{v \in B : |N(v) \cap W| \leq \delta\alpha\}$, where $\delta = 4$, and let $B_{high} = B \setminus B_{low}$. Also, let $W_{low} = \{v \in W : |N(v) \cap (W \cup B_{high})| \leq \delta\alpha\}$. We additionally define $W_{high} = W \setminus W_{low}$. Lastly, each vertex v has a *counter* c_v.

Each vertex v maintains the following information:

- The set(s) among D, B, W, B_{high}, B_{low}, W_{high}, and W_{low} that v is a member of.
- The quantity $|N(v) \cap W|$.
- The quantity $|N(v) \cap (W \cup B_{high})|$.
- The counter c_v.

At initialization, every vertex v is in W (so D and B are empty). Consequently, the quantities $|N(v) \cap W|$ and $|N(v) \cap (W \cup B_{high})|$ are both equal to $\deg(v)$. For each vertex v, if $\deg(v) \leq \delta\alpha$, then $v \in W_{low}$. Each counter c_v is initialized to 0.

It will be useful to define the graph $G_{bi} \subseteq G$ that changes over the course of the execution of the algorithm:

Definition 5.1. G_{bi} is a bipartite graph on the vertex set $B_{high} \cup W$. One side of the bipartition is $B_{high} \cup W_{high}$ and the other side is W_{low}. The edge set of G_{bi} is the set of edges in G with one endpoint in each side of the bipartition.

The algorithm proceeds as follows. Repeat the following until W is empty. In the first round, each vertex $v \in W_{low}$ picks a value $p(v) \in [0, 1]$ uniformly at random and sends $p(v)$ to its neighbors. The next step is for v to determine whether $p(v)$ is an α-*minimum*. $p(v)$ is said to be an α-*minimum* if for every $u \in N_{G_{bi}}(v)$, $p(v)$ is among the α smallest values of vertices in $N_{G_{bi}}(u)$. To determine which values are α-minima, in the second round each vertex $u \in B_{high} \cup W_{high}$ sends ACK to each vertex $v \in N_{G_{bi}}(u)$ such that $p(v)$ is among the α smallest values that u received. If $v \in W_{low}$ receives ACK from all $u \in N_{G_{bi}}(v)$, then v is added to D (if $N_{G_{bi}}(v)$ is empty then v is added to D) and v tells its neighbors to increment their counters (in the third round). Whenever the counter c_u of a vertex u reaches $\delta\alpha$, u enters D. (Note that if u enters D as a result of c_u reaching $\delta\alpha$, c_u does *not* tell its neighbors to increment their counters.)

Whenever a vertex v moves from one set of the partition to another, v notifies each vertex $u \in N(v)$ so that u can update the quantities $|N(u) \cap W|$ and $|N(u) \cap (W \cup B_{high})|$, and move to the appropriate set. The bookkeeping for updating this information is identical to that of Algorithm 2. When no more vertices are left in W, B_{high} is also empty, and all processors terminate. This concludes the description of the algorithm.

5.2 Analysis

We begin with a simple claim about the behavior of the partition of vertices over time:

Claim 5.2.

1. *No vertex can ever leave D.*

Algorithm 4 Faster Randomized Distributed $O(\alpha)$-approximation for MDS

1: Initialize partition: $D \leftarrow \emptyset$, $B_{high} \leftarrow \emptyset$, $B_{low} \leftarrow \emptyset$, $W \leftarrow V$, $W_{low} \leftarrow \{v \in W : |N(v) \cap (W \cup B_{high})| \leq \delta\alpha\}$, $W_{high} \leftarrow W \setminus W_{low}$
2: Initialize counters: $\forall v \in V : c_v \leftarrow 0$
3: Initialize degrees: $\forall v \in V : |N(v) \cap W| = \deg(v)$, $|N(v) \cap (W \cup B_{high})| = \deg(v)$
4: **while** $W \neq \emptyset$ **do**
5: Each vertex v runs the following procedure:
6: **if** $v \in W_{low}$ **then**
7: $p(v) \leftarrow$ a value in $[0, 1]$ chosen uniformly at random
8: **Send** $p(v)$ message to neighbors
9: **if** $v \in B_{high} \cup W_{high}$ **then**
10: **Send** ACK to each $u \in N_{G_{bi}}(v)$ such that $p(u)$ is among the α smallest values v received
11: **if** $v \in W_{low}$ and v receives ACK from all $u \in N_{G_{bi}}(u)$ **then**
12: Move v to D
13: **Send** INCREMENT COUNTER message to neighbors
14: **Send** MOVED FROM W TO D message to neighbors
15: Run Algorithm 2 starting from Line 11

2. *No vertex can ever enter $W \cup B_{high}$ from another set.*
3. *No vertex in W_{low} can ever at a later point be in $B_{high} \cup W_{high}$.*

Proof. The proofs of items 1 and 2 follow from the proof of Claim 3.1. Item 3 holds because if a vertex v is in B_{high} then $|N(v) \cap W| > \delta\alpha$ and if v is in W_{high} then $|N(v) \cap (W \cup B_{high})| > \delta\alpha$. For any v, the quantities $|N(v) \cap W|$ and $|N(v) \cap (W \cup B_{high})|$ can only decrease over time (they are only decremented in Algorithm 4). □

Next, we prove a simple claim that upper bounds the counter of each vertex:

Claim 5.3. *For all $v \in V$, at all times $c_v < 2\delta\alpha$.*

Proof. If for any $v \in V$, it happens that $c_v \geq \delta\alpha$, then during the same iteration of the **while** loop, v enters D (on Line 18), after which point v never leaves D (by Claim 5.2) so c_v cannot ever change again. Thus, it suffices to show that for all vertices $v \in V \setminus D$, during a single iteration of the **while** loop, c_v can be incremented at most $\delta\alpha$ times, leading to a maximum value of at most $2\delta\alpha - 1$. This is true for $v \in W_{low}$ because $|N(v) \cap (W \cup B_{high})| \leq \delta\alpha$, and only vertices in $|N(v) \cap (W \cup B_{high})|$ can tell v to increment c_v. On the other hand, if $v \in W \cup B_{high}$, then a neighbor u of v only sends INCREMENT COUNTER if $p(u)$ is among the α smallest values in v's neighborhood, so c_v is only incremented α times during a single iteration. □

The analysis of correctness is the same as that of our centralized algorithm (see Section 3.2.1), with one technicality: By Claim 5.3, the counter of each vertex has maximum value $2\delta\alpha$ instead of $\delta\alpha$, causing an increase in the leading constant in the $O(\alpha)$ approximation factor.

Our goal in the rest of this section is to prove that Algorithm 4 runs in $O(\alpha \log n)$ rounds with high probability. Note that one iteration of the **while** loop takes a constant number of rounds. Thus, our goal is to show that there are $O(\alpha \log n)$ total iterations of the loop.

For any vertex $v \in V(G_{bi})$, let $N^2(v)$ be the set of vertices in the 2-hop neighborhood of v with respect to G_{bi}, and let $E^2(v)$ be the set of edges within 2 hops of v with respect to G_{bi}; that is, $E^2(v)$ contains the edge (v, u) for all $u \in N_{G_{bi}}(v)$, and the edge (u, y) for all $y \in N_{G_{bi}}(u)$.

We divide the iterations of the **while** loop into two types. We say that an iteration is of *type low degree* if at least half of the vertices $v \in W_{low}$ have $|N^2(v)| \leq \alpha$. Otherwise, we say that an iteration is of *type high degree*.

It is simple to deterministically bound the number of iterations of type low degree:

Claim 5.4. *The total number of iterations of type low degree is* $O(\log n)$.

Proof. By the specification of the algorithm, all vertices v with $|N^2(v)| \leq \alpha$ are added to D (on Line 12). Thus, during an iteration of type low degree, at least half of the vertices in W_{low} enter D. By Claim 2.4, at least half of the vertices in $W \cup B_{high}$ are in W_{low}. Thus, during an iteration of type low degree, at least $1/4$ of the vertices in $W \cup B_{high}$ enter D. By Claim 5.2, no vertex can ever enter $W \cup B_{high}$ from another set. Therefore, during every iteration of type low degree, $W \cup B_{high}$ shrinks by a factor of at least 4. Thus, there are only $O(\log n)$ iterations of type low degree. □

It remains to bound the number of iterations of type high degree. Fix an iteration I of type high degree.

For the purpose of analysis, we assign each edge $e \in E$ a *weight* $w(e)$ that increases over the execution of the algorithm. The rule for updating the weight of edges is as follows. Whenever a vertex $v \in W_{low}$ is moved to D on Line 12 (as a result of $p(v)$ being an α-minimum), v increments the weight of every edge in $E^2(v)$.

We will define the *available weight* at iteration I as a function that will capture the total amount of weight that could ever be added over all iterations starting from iteration I. Our goal is to provide:

1. an upper bound of $\alpha \cdot |V(G_{bi})|/4$ for the available weight at iteration I, and
2. a lower bound of $2\delta\alpha^2 \cdot |V(G_{bi})|$ for the expected total weight added to edges during iteration I.

Combining these upper and lower bounds yields the result that in each iteration the expected total amount of weight added is a $1/O(\alpha)$ fraction of the total available weight. This allows us to bound the number of iterations by $O(\alpha \log n)$ with high probability.

Definition 5.5. The *available weight* at iteration I, denoted $A(I)$ is given by

$$A(I) = \sum_{e \in E(G[W \cup B_{high}])} 2\delta\alpha - w(e)$$

where the parameters in the expression are taken to be their values at the beginning of iteration I.

Giving an upper bound on $A(I)$, which is item 1 of our above goal, is simple:

Claim 5.6. $A(I) \leq 2\delta\alpha^2 \cdot |V(G_{bi})|$.

Proof. By definition all edge weights are non-negative, so $A(I) \leq \sum_{e \in E(G[W \cup B_{high}])} 2\delta\alpha$. Since $G[W \cup B_{high}]$ has arboricity at most α, $|E(G[W \cup B_{high}])| \leq \alpha \cdot |W \cup B_{high}| = \alpha \cdot |V(G_{bi})|$. This completes the proof. □

Now, we consider item 2 of our above goal. Let $w(I)$ be a random variable denoting the aggregate total weight added to edges during iteration I. The randomness is over the choice of $p(v)$ for each vertex $v \in W_{low}$. Towards lower bounding $w(I)$, for every vertex $v \in W_{low}$ we define the random variable R_v as the number of edges whose weight is incremented by v during iteration I. That is, $w(I) = \sum_{v \in W_low} R_v$. We now calculate $\mathbb{E}[R_v]$.

Claim 5.7. *For all $v \in W_{low}$ with $|N^2(v)| \geq \alpha$, it holds that $\mathbb{E}[R_v] \geq \alpha$.*

Proof. Note that $E^2(v)$ and $N^2(v)$ are taken to mean these value at the beginning of iteration I. A vertex $v \in W_{low}$ increments the weight of each edge in $E^2(v)$ if v is an α-minimum, and otherwise v does not increment the weight of any edges. A sufficient condition for v to be an α-minimum is that $p(v)$ is among the α smallest values in $N^2(v)$. Since $|N^2(v)| \geq \alpha$, the probability that $p(v)$ is among the α smallest values in $N^2(v)$ is $\alpha/|N^2(v)| \geq \alpha/|E^2(v)|$ since each vertex chooses its value uniformly at random. Therefore, $\mathbb{P}[R_v = |E^2(v)|] \geq \alpha/|E^2(v)|$. Thus, $\mathbb{E}[R_v] \geq \alpha$. □

We now give a lower bound on the expected aggregate weight $\mathbb{E}[w(I)]$:

Claim 5.8. $\mathbb{E}[w(I)] \geq \alpha \cdot |V(G_{bi})|/4$

Proof. Recall that $w(I) = \sum_{v \in W_low} R_v$. Thus,

$$\begin{aligned}
\mathbb{E}[w(I)] &= \mathbb{E}\Big[\sum_{v \in W_{low}} R_v\Big] \\
&= \sum_{v \in W_{low}} \mathbb{E}[R_v] \\
&\geq \sum_{v \in W_{low}, |N^2(v)| \geq \alpha} \mathbb{E}[R_v] \\
&\geq \sum_{v \in W_{low}, |N^2(v)| \geq \alpha} \alpha && \text{(by Claim 5.7)} \\
&\geq \alpha \cdot |W_{low}|/2 && \text{(since iteration } I \text{ is of type high degree)} \\
&\geq \alpha \cdot |B_{high} \cup W|/4 && \text{(by Claim 2.4)} \\
&= \alpha \cdot |V(G_{bi})|/4.
\end{aligned}$$

□

Before combining the above upper and lower bounds, we need to show that our function $A(I)$ accurately measures the progress of our algorithm by proving the following properties:

Claim 5.9.

1. *If I' is the iteration right after I, then $A(I') \leq A(I) - w(I)$.*
2. *$A(I) > 0$.*

Proof. For item 1, it suffices to observe that the weight of an edge can only increase and the set of edges we sum over in the definition of $A(I)$ can only decrease. This is true because by Claim 5.2, no vertex can ever enter $W \cup B_{high}$ from another set.

For item 2, it suffices to show that $2\delta\alpha - w(e)$ is always positive. Suppose for contradiction that there is an edge (u, v) with $w(u, v) \geq 2\delta\alpha$. Consider the point at which $w(u, v)$ was incremented to $\delta\alpha$. Consider G_{bi} at this point in time. Only the weight of edges in G_{bi} can be incremented, so $(u, v) \in E(G_{bi})$. Without loss of generality, $u \in B_{high} \cup W_{high}$ and $v \in W_{low}$. Ever since the edge (u, v) entered G_{bi}, u has been in $B_{high} \cup W_{high}$ and v has been in W_{low}, since no vertex in W_{low} can later be in $B_{high} \cup W_{high}$ by Claim 5.2. Each time $w(u, v)$ is incremented, it is caused by an INCREMENT COUNTER message sent by some vertex y that moved from W_{low} to D, for which $(u, v) \in E^2(y)$. Then since y was in W_{low}, $u \in B_{high} \cup W_{high}$,

and $(u, v) \in E^2(y)$, we have $y \in N(u)$. Thus y sends INCREMENT COUNTER to u. Therefore, every time $w(u, v)$ is incremented, c_u is also incremented. Since $w(u, v) = 2\delta\alpha$, we have $c_u = 2\delta\alpha$, which is a contradiction by Claim 5.3. □

We are now ready to put everything together to complete the analysis. For all j, let I_j denote the j^{th} iteration of type high degree. Then, by Claim 5.6 we have that for all j, $A(I_j) \leq 2\delta\alpha^2 \cdot |V(G_{bi})|$, and by Claim 5.8 we have that for all j, $\mathbb{E}[w(I_j)] \geq \alpha \cdot |V(G_{bi})|/4$ (where G_{bi} is taken to be its value at the beginning of iteration I_j). Thus, $\mathbb{E}[w(I_j)] \geq A(I_j)/(8\delta\alpha)$.

Thus, by item 1 of Claim 5.9, we have that if for all j, $A(I_{j+1}) \leq A(I_j) - w(I_j)$, so for all j, we have

$$\begin{aligned}
\mathbb{E}[A(I_{j+1})] &\leq \mathbb{E}[A(I_j) - w(I_j)] \\
&\leq (1 - \frac{1}{8\delta\alpha})\mathbb{E}[A(I_j)] \\
&\leq (1 - \frac{1}{8\delta\alpha})^j \cdot A(I_1) \\
&= (1 - \frac{1}{8\delta\alpha})^j \cdot m(2\delta\alpha) \\
&= (1 - \frac{1}{32\alpha})^j \cdot 8\alpha^2 n.
\end{aligned}$$

Let $j = 50c\alpha \log n$. For any constant positive integer c, we have $\mathbb{E}[A(I_{j+1})] < 1/n^c$. By Markov's inequality and the fact that the available weight is always integral and non-negative, we have that $A(I_j) = 0$ with high probability. By item 2 of Claim 5.9, the algorithm terminates before it reaches an iteration I^* with $A(I^*) = 0$. Thus, the number of iterations of type high degree is $O(\alpha \log n)$ with high probability.

Acknowledgements The authors would like to thank Quanquan Liu and Yosi Hezi for fruitful discussions.

References

[AMZ97] Srinivasa R Arikati, Anil Maheshwari, and Christos D Zaroliagis. Efficient computation of implicit representations of sparse graphs. *Discrete Applied Mathematics*, 78(1-3):1–16, 1997.

[ASS19] Saeed Akhoondian Amiri, Stefan Schmid, and Sebastian Siebertz. Distributed dominating set approximations beyond planar graphs. *ACM Transactions on Algorithms (TALG)*, 15(3):1–18, 2019.

[AZO19] Zeyuan Allen-Zhu and Lorenzo Orecchia. Nearly linear-time packing and covering lp solvers. *Mathematical Programming*, 175(1):307–353, 2019.

[Bak94] Brenda S Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

[BE10] Leonid Barenboim and Michael Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.

[BE14] Leonid Barenboim and Michael Elkin. Combinatorial algorithms for distributed graph coloring. *Distributed Computing*, 27(2):79–93, 2014.

[BF99] Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representation of sparse graphs. In Frank K. H. A. Dehne, Arvind Gupta, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures, 6th International Workshop, WADS '99, Vancouver, British Columbia, Canada, August 11-14, 1999, Proceedings*, volume 1663 of *Lecture Notes in Computer Science*, pages 342–351. Springer, 1999.

[BPS20] Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[BS20] Suman K. Bera and C. Seshadhri. How the degeneracy helps for triangle counting in graph streams. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 457–467. ACM, 2020.

[BU17] Nikhil Bansal and Seeun William Umboh. Tight approximation bounds for dominating set on graphs of bounded arboricity. *Information Processing Letters*, 122:21–24, 2017.

[CC08] Miroslav Chlebík and Janka Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Information and Computation*, 206(11):1264–1275, 2008.

[CHS09] Andrzej Czygrinow, Michał Hańćkowiak, and Edyta Szymańska. Fast distributed approximation algorithm for the maximum matching problem in bounded arboricity graphs. In *International Symposium on Algorithms and Computation*, pages 668–678. Springer, 2009.

[CHW08] Andrzej Czygrinow, Michal Hańćkowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In *International Symposium on Distributed Computing*, pages 78–92. Springer, 2008.

[CN85] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on computing*, 14(1):210–223, 1985.

[DGKR05] Irit Dinur, Venkatesan Guruswami, Subhash Khot, and Oded Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. *SIAM J. Comput.*, 34(5):1129–1146, 2005.

[DKM19] Janosch Deurer, Fabian Kuhn, and Yannic Maus. Deterministic distributed dominating set approximation in the congest model. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 94–103, 2019.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the pcp theorem. *SIAM Journal on Computing*, 36(4):975–1024, 2006.

[DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 624–633, 2014.

[ELR18] Talya Eden, Reut Levi, and Dana Ron. Testing bounded arboricity. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2081–2092. SIAM, 2018.

[Epp94] David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information processing letters*, 51(4):207–211, 1994.

[ERR19] Talya Eden, Dana Ron, and Will Rosenbaum. The arboricity captures the complexity of sampling edges. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 52:1–52:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[ERS20] Talya Eden, Dana Ron, and C. Seshadhri. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1467–1478. SIAM, 2020.

[Gar79] Michael R Garey. A guide to the theory of np-completeness. *Computers and intractability*, 1979.

[GG06] Gaurav Goel and Jens Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 159–167. Springer, 2006.

[GGR21] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. Improved deterministic network decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2904–2923. SIAM, 2021.

[GKM17] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 784–797, 2017.

[GS17] Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2505–2523. SIAM, 2017.

[HTZ14] Meng He, Ganggui Tang, and Norbert Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2014.

[JLR+13] Mark Jones, Daniel Lokshtanov, MS Ramanujan, Saket Saurabh, and Ondřej Suchỳ. Parameterized complexity of directed steiner tree on sparse graphs. In *European Symposium on Algorithms*, pages 671–682. Springer, 2013.

[Joh74] David S Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

[JRS02] Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

[KMW06] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *SODA*, volume 6, pages 1109557–1109666. Citeseer, 2006.

[KMW16] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *Journal of the ACM (JACM)*, 63(2):1–44, 2016.

[Lin87] Nathan Linial. Distributive graph algorithms global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 331–335. IEEE, 1987.

[Lin92] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on computing*, 21(1):193–201, 1992.

[LW10] Christoph Lenzen and Roger Wattenhofer. Minimum dominating set approximation in graphs of bounded arboricity. In *International symposium on distributed computing*, pages 510–524. Springer, 2010.

[Mei09] Or Meir. Combinatorial pcps with efficient verifiers. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 463–471. IEEE, 2009.

[MV18] Andrew McGregor and Sofya Vorotnikova. A simple, space-efficient, streaming algorithm for matchings in low arboricity graphs. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, volume 61 of *OASICS*, pages 14:1–14:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[NA16] Jose C Nacher and Tatsuya Akutsu. Minimum dominating set-based methods for analyzing biological networks. *Methods*, 102:57–63, 2016.

[OSSW18] Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully dynamic MIS in uniformly sparse graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 92:1–92:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

[PS16] David Peleg and Shay Solomon. Dynamic (1+ϵ)-approximate matchings: A density-sensitive approach. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 712–729. SIAM, 2016.

[Qua20] Kent Quanrud. Nearly linear time approximations for mixed packing and covering problems without data structures or randomization. In *Symposium on Simplicity in Algorithms*, pages 69–80. SIAM, 2020.

[RVW00] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 3–13. IEEE, 2000.

[SL10] Chao Shen and Tao Li. Multi-document summarization via the minimum dominating set. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 984–992, 2010.

[SV20] Hsin-Hao Su and Hoa T. Vu. Distributed dense subgraph detection and low outdegree orientation. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[WAF02] Peng-Jun Wan, Khaled M Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1597–1604. IEEE, 2002.

[You14] Neal E Young. Nearly linear-work algorithms for mixed packing/covering and facility-location linear programs. *arXiv preprint arXiv:1407.3015*, 2014.