Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover

Mohsen Ghaffari
ETH Zurich
ghaffari@inf.ethz.ch

Themis Gouleakis
MIT
tgoule@mit.edu

Slobodan Mitrović
EPFL
slobodan.mitrovic@epfl.ch

Ronitt Rubinfeld
MIT and Tel Aviv University
ronitt@csail.mit.edu

Abstract

We present $O(\log\log n)$-round algorithms in the Massively Parallel Computation (MPC) model, with $\tilde{O}(n)$ memory per machine, that compute a maximal independent set, a $1+\varepsilon$ approximation of maximum matching, and a $2+\varepsilon$ approximation of minimum vertex cover, for any n-vertex graph and any constant $\varepsilon > 0$. These improve the state of the art as follows:

- Our MIS algorithm leads to a simple $O(\log\log\Delta)$-round MIS algorithm in the Congested Clique model of distributed computing. This result improves exponentially on the $\tilde{O}(\sqrt{\log\Delta})$-round algorithm of Ghaffari [PODC'17].
- Our $O(\log\log n)$-round $(1+\varepsilon)$-approximate maximum matching algorithm simplifies and improves on a rather complex $O(\log^2\log n)$-round $(1+\varepsilon)$-approximation algorithm of Czumaj et al. [STOC'18].
- Our $O(\log\log n)$-round $(2+\varepsilon)$-approximate minimum vertex cover algorithm improves on an $O(\log\log n)$-round $O(1)$-approximation of Assadi et al. [arXiv'17].

1 Introduction

A growing need to process massive data led to development of a number of frameworks for large-scale computation, such as MapReduce [DG04], Hadoop [Whi12], Spark [ZCF+10], or Dryad [IBY+07]. Thanks to their natural approach to processing massive data, these frameworks have gained great popularity. In this work, we consider the *Massively Parallel Computation* (MPC) model that is abstracted out of the capabilities of these frameworks.

In our work, we study some of the most fundamental problems in algorithmic graph theory: maximal independent set (MIS), maximum matching and minimum vertex cover. The study of these problems in the models of parallel computation dates back to PRAM algorithm. A seminal work of Luby [Lub86] gives a simple randomized algorithm for constructing MIS in $O(\log n)$ PRAM rounds. When this algorithm is applied to a line graph of graph G, it outputs a maximal matching of G, and hence a 2-approximate maximum matching. This consequently provides a method for constructing a 2-approximate minimum vertex cover. Similar results, also in the context of PRAM algorithm, were obtained in [ABI86, II86, IS86]. Since then, the aforementioned problems were studied quite extensively in various models of computation. In the context of MPC, we design simple randomized algorithms that construct (approximate) instances for all the three problems.

1.1 The models

We consider two closely related models: *Massively Parallel Computation* (MPC), and the CONGESTED-CLIQUE model of distributed computing. Indeed, we consider it as a (non-technical) contribution of this paper to (further) exhibit the proximity of these two models and we are hopeful that it will bring the related research communities closer. We next review these models.

1.1.1 MPC

The MPC model was first introduced in [KSV10] and later refined in [GSZ11, BKS13, ANOY14]. The computation in this model proceeds in synchronous *rounds* carried out by m machines. At the beginning of every round, the data (e.g. vertices and edges) is distributed across the machines. During a round, each machine performs computation locally without communicating to other machines. At the end of the round, the machines exchange messages which are used to guide the computation in the next round. In every round, each machine receives and outputs messages that fit into its local memory.

Space: In this model, each machine has S words of space. If N is the total size of data, then one usually wants that S is sublinear in N and that $S \cdot m = \Theta(N)$. That is, the total memory across all the machines suffices to fit all the data, but is not much larger than that. If we are given a graph on n vertices, in our work we consider the regimes in which $S \in \Theta(n/\operatorname{polylog} n)$ or $S \in \Theta(n)$.

Communication vs. computation complexity: Our main focus is the communication complexity, i.e. the number of rounds required to finish computation. Although we do not explicitly state the computation complexity in our results, it will be apparent from the description of our algorithms that the total computation time across all the machines is nearly-linear in the input size.

1.1.2 CONGESTED-CLIQUE

The other model that we consider is the CONGESTED-CLIQUE model of distributed computing, which was introduced by Lotker, Pavlov, Patt-Shamir, and Peleg [LPPSP03] and has been stud-

ied extensively since then, see e.g., [PST11, DLP12, BHP12, Len13, DKO14, Nan14, HPS14, HP14, CHKK+15, HPP+15, BFARR15, Gha16, GP16, Kor16, HKN16, CHPS16, Gha17, JN18]. In this model, we have n-players which can communicate in synchronous rounds. Per round each player can send $O(\log n)$ bits to each other player. Besides this communication restriction, the model does not limit the players, e.g., they can use large space and arbitrary computations; though, in our algorithms, both of these will be small. Furthermore, in studying graph problems in this model, the standard setting is that we have an n-vertex graph $G = (V, E)$, and each player is associated with one vertex of this graph. Initially, each player knows only the edges incident on its own vertex. At the end, each player should know the part of the output related to its own vertex, e.g., whether its vertex is in the computed maximal independent set or not, or whether some of its edges is in the matching or not.

We emphasize that CONGESTED-CLIQUE provides an all-to-all communication model. It is worth contrasting this with the more classical models of distributed computing. For instance, the LOCAL model, first introduced by Linial [Lin87], allows the players to communicate only along the edges of the graph problem G (with unbounded size messages).

1.2 Related work

Maximum Matching and Minimum Vertex Cover: If the space per machine is $O(n^{1+\delta})$, for any $\delta > 0$, Lattanzi et al. [LMSV11] show how to construct a maximal matching, and hence a 2-approximate minimum vertex cover, in $O(1/\delta)$ MPC rounds. Furthermore, in case the machine-space is $\Theta(n)$, their algorithm requires $O(\log n)$ many rounds to output a maximal matching. In their work, they apply *filtering* techniques to gradually sparsify the graph. Ahn and Guha [AG15] provide a method for constructing a $(1 + \varepsilon)$-approximation of weighted maximum matching in $O(\delta/\varepsilon)$ rounds while, similarly to [LMSV11], requiring that the space per machine is $O(n^{1+\delta})$.

If the space per machine is $\tilde{O}(n\sqrt{n})$, Assadi and Khanna [AK17] show how construct an $O(1)$-approximate maximum matching and an $O(\log n)$-approximate minimum vertex cover in two rounds. Their approach is based on designing randomized composable coresets.

Recently, Czumaj et al. [CŁM+18] designed an algorithm for constructing a $(1+\varepsilon)$-approximate maximum matching in $O((\log\log n)^2)$ MPC rounds of computation and $O(n/\operatorname{polylog} n)$ memory per machine. To obtain this result, they start from a variant of a PRAM algorithm that requires $O(\log n)$ parallel iterations, and showed how to compress many of those iterations (on average, $O(\log n/(\log\log n)^2)$ many of them) into $O(1)$ MPC rounds. Their result does not transfer to an algorithm for $O(1)$-approximate minimum vertex cover.

Building on [CŁM+18] and [AK17], Assadi [Ass17] shows how to produce an $O(\log n)$-approximate minimum vertex cover in $O(\log\log n)$ MPC rounds when the space per machine is $O(n/\operatorname{polylog} n)$. The work by Assadi et al. [ABB+17] also addresses these two problems, and provides a way to construct a $(1 + \varepsilon)$-approximate maximum matching and an $O(1)$-approximate minimum vertex cover in $O(\log\log n)$ rounds when the space per machine is $\tilde{O}(n)$. Their result builds on techniques originally developed in the context of dynamic matching algorithms and composable coresets.

Maximal Independent Set: Maximal independent set has been central in the study of graph algorithms in both the parallel and the distributed models. The seminal work of Luby [Lub86] and Alon, Babai, and Itai [ABI86] provide $O(\log n)$-round parallel and distributed algorithms for constructing MIS. The distributed complexity in the LOCAL model was first improved by Barenboim et al. [BEPSv3] and consequently by Ghaffari [Gha16], which led to the current best round complexity of $O(\log \Delta) + 2^{O(\sqrt{\log\log n})}$. In the CONGESTED-CLIQUE model of distributed computing, Ghaffari [Gha17] gave another algorithms which computes an MIS in $\tilde{O}(\sqrt{\log \Delta})$ rounds. A

deterministic $O(\log n \log \Delta)$-round CONGESTED-CLIQUE algorithm was given by Censor-Hillel et al. [CHPS17].

It is also worth referring to the literature on one particular MIS algorithm, known as the *randomized greedy MIS*, which is relevant to what we do for MIS. In this algorithm, we permute the vertices uniformly at random and then add them to the MIS greedily. Blelloch et al. [BFS12] showed that one can implement this algorithm in $O(\log^2 n)$ parallel/distributed rounds, and recently Fischer and Noever [FN18] improved that to a tight bound of $\Theta(\log n)$. We will show a $O(\log \log \Delta)$-round simulation of the randomized greedy MIS algorithm in the MPC and CONGESTED-CLIQUE models.

1.3 Our contributions

As our first result, in Section 2 we present an algorithm for constructing MIS.

Theorem 1.1. *There is an algorithm that computes an MIS in $O(\log \log \Delta)$ rounds of the MPC model, with $\tilde{O}(n)$-bits of memory per machine. Moreover, the same algorithm can be adapted to compute an MIS in $O(\log \log \Delta)$ rounds of the* CONGESTED-CLIQUE *model.*

As our second result, in Section 3, we design an algorithm that returns a $(2+\varepsilon)$-approximate maximum matching and a $(2+\varepsilon)$-approximate minimum vertex cover in $O(\log \log n)$ MPC rounds. Although, compared to the prior work, our result has better round-complexity (compared to [CŁM+18]) or provides stronger approximation-guarantee (compared to the result for vertex cover in [ABB+17]), we find that the main advantage of our algorithm is its simplicity. After applying random partitioning, the algorithm repeats only a couple of simple steps to perform all its decisions.

Theorem 1.2. *There is an algorithm that with high probability computes a $(2+\varepsilon)$-approximate maximum matching and a $(2+\varepsilon)$-approximate minimum vertex cover in $O(\log \log n)$ rounds of the MPC model, with $\tilde{O}(n)$-bits of memory per machine.*

As noted by Assadi et al. [ABB+17], applying the techniques of [McG05] on Theorem 1.2 leads to following result.

Corollary 1.3. *There exists an algorithm that with high probability constructs a $(1+\varepsilon)$-approximate maximum matching in $O(\log \log n) \cdot (1/\varepsilon)^{O(1/\varepsilon)}$ MPC rounds, with $\tilde{O}(n)$-bits of memory per machine.*

As noted by Czumaj et al. [CŁM+18], the result of Lotker et al. [LPSR09] can be used to obtain the following result.

Corollary 1.4. *There exists an algorithm that outputs a $(2+\varepsilon)$-approximation to maximum weighted matching in $O(\log \log n \cdot (1/\varepsilon))$ MPC rounds and $\tilde{O}(n)$-bits of memory per machine.*

For the sake of clarity, we present our algorithms for the case in which each machine has $\tilde{O}(n)$-bits of memory (or $O(n)$ words of memory). However, similarly as in [CŁM+18], our algorithm for matching and vertex cover can be adjusted to still run in $O(\log \log n)$ MPC rounds even when the memory per machine is $O(n/\operatorname{polylog} n)$.

1.4 Our techniques

Maximal independent set: Our MPC algorithm for MIS is based on the randomized greedy MIS algorithm. We show how to efficiently implement this algorithm in only $O(\log \log n)$ MPC and CONGESTED-CLIQUE rounds.

Maximum matching and vertex cover: We start from a sequential algorithm that outputs a $(2+\varepsilon)$-approximate fractional maximum matching and a $(2+\varepsilon)$-approximate minimum vertex cover. This algorithm constructs a fractional matching, and in the process also obtains a vertex cover. The algorithm maintains edge-weights. Initially, every edge-weight is set to $1/n$. Then, gradually, at each iteration the edge-weights are simultaneously increased by a multiplicative factor of $1/(1-\varepsilon)$. Each vertex whose sum of the incident edges becomes $1-2\varepsilon$ or larger gets frozen, and its incident edges do not change their weights afterward. The vertices that got frozen in this process constitute the desired vertex cover. It is not hard to see that after $O(\log n/\varepsilon)$ iterations every edge will be incident to at least one frozen vertex, and at this point the algorithm terminates.

In our work, we simulate this sequential algorithm in the MPC model, by on average simulating $\Theta(\log n/\log\log n)$ iterations in $O(1)$ MPC rounds. As the first step, motivated by [CŁM+18], we apply vertex-based sampling. Namely, the vertex-set is randomly partitioned across the machines, and each machine considers only the induced graph on its local copy of vertices. Then, during one MPC round, each machine simulates several iterations of the sequential algorithm on the subgraph it has. During this simulation, each machine estimates weights of its local vertcies in order to decide which vertices should be frozen. However, even if the estimates are sharp, only a slight error could potentially cause many vertices to largely deviate from its true behavior. To alleviate this issue, instead of having a fixed threshold $1-2\varepsilon$, for each vertex and in every iteration we choose a random threshold from the interval $[1-4\varepsilon, 1-2\varepsilon]$. Then, a vertex gets frozen only if its estimated weight is above this randomly chosen threshold. Intuitively, this significantly reduces the chance of these decisions (on whether to freeze a vertex of not) deviating from the true ones.

As our final component, we provide a rounding procedure that for a given fractional matching produces an integral one of size only a constant-factor smaller than the size of the fractional matching. Furthermore, every vertex in that rounding method chooses edges based only on its neighborhood, i.e. makes local decisions, so it is easy to parallelize this procedure.

1.5 Notation

For a graph $G=(V,E)$ and a set $V'\subseteq V$, $G[V']$ denotes the subgraph of G *induced* on the set V', i.e. $G[V']=(V', E\cap(V'\times V'))$. We use $N(v)$ to refer to the neighborhood of v in G. Throughout the paper, we use $n:=|V|$ to denote the number of vertices in the input graph.

2 Maximal Independent Set

Consider a sequential random greedy algorithm that ranks/permutes vertices 1 to n randomly and then greedily adds vertices to the MIS, while walking through this permutation. In this section, we simulate this algorithm in $O(\log\log\Delta)$ rounds of the congested clique model, thus proving the following result:

Theorem 1.1. *There is an algorithm that computes an MIS in $O(\log\log\Delta)$ rounds of the MPC model, with $\tilde{O}(n)$-bits of memory per machine. Moreover, the same algorithm can be adapted to compute an MIS in $O(\log\log\Delta)$ rounds of the* CONGESTED-CLIQUE *model.*

2.1 Randomized Greedy Algorithm for MIS

Let us first consider a randomized variant of the sequential greedy MIS algorithm described below. We remark that this algorithm has been studied before in the literature of parallel algorithms [FN18, BFS12].

Greedy Randomized Maximal Independent Set:

- Initially, choose a permutation $\pi : [n] \to [n]$ uniformly at random.
- Repeat until the next rank is $n/\log^{10} n$ and the maximum degree is at least $\log^{10} n$:
  - (A) Mark the vertex v which has the smallest rank among the remaining vertices according to π, and v to the MIS.
  - (B) Remove all the neighbors of v.
- Run $O(\log\log\Delta)$ rounds of the Sparsified MIS Algorithm of [Gha17] in the remaining graph. Remove from the graph the constructed MIS and its neighborhood.
- Deliver the remaining graph on a single machine and find its MIS.
- At the end, output the constructed MIS sets.

2.2 Simulation in $O(\log\log\Delta)$ rounds of MPC and Congested Clique

Simulation in the MPC model: We now explain how to simulate the above algorithm in the MPC model with $O(n\log n)$-bits of memory per machine, and also in the congested clique model. In each iteration, we take an induced subgraph of G that is guaranteed to have $\tilde{O}(n)$ edges and simulate the above algorithm on that graph. We show that the total number of edges drops fast enough, so that $O(\log\log\Delta)$ rounds will suffice. More concretely, we first consider the subgraph induced by vertices with ranks 1 to n/Δ^{α}, for $\alpha = 3/4$. This subgraph has $O(n)$ edges, with high probability. So we can deliver it to one machine, and have it simulate the algorithm up to this rank. Now, this machine sends the resulting MIS to all other machines. Then, each machine removes its vertices that are in MIS or neighboring MIS. In the second phase, we take the subgraph induced by remaining vertices with ranks n/Δ^{α} to n/Δ^{α^2}. Again, we can see that this subgraph has $O(n)$ edges (a proof is given below), so we can simulate it in $O(1)$ rounds. More generally, in the i-th iteration, we will go up to rank n/Δ^{α^i}. Once the next rank becomes $n/\log^{10} n$, which as we show happens after $O(\log\log\Delta)$ rounds, the maximum degree of the graph is some value $\Delta' \leq O(\log^{11} n)$ (see Lemma 2.1). Note that clearly also $\Delta' \leq \Delta$. At that point, we apply the MIS Algorithm of [Gha17] for sparse graphs to the remaining graph. This algorithm is applicable whenever the maximum degree is at most $2^{O(\sqrt{\log n})}$ (see Theorem 1.1 of [Gha17]). After $O(\log\log\Delta')$ rounds, w.h.p., that algorithm find an MIS which after removed along with its neighborhood results in the graph having $O(n)$ edges. Now we deliver the whole remaining graph to one machine where it is processed in a single MPC rounds.

We note that the Algorithm of [Gha17] performs only simple local decisions with low communication, and hence every iteration of the algorithm can be implemented in $O(1)$ MPC rounds, with $\tilde{O}(n)$ memory per machine, by using standard techniques.

Simulation in the Congested Clique Model: We now argue that each iteration can be implemented in $O(1)$ rounds of the congested clique model. For that, in each iteration, we make all vertices with permutation rank in the selected range send their edges to the leader vertex. Here, the leader is an arbitrarily chosen one of the vertices, e.g., the one with the minimum identifier. As we show below, the number of these edges per iteration is $O(n)$ with high probability, and thus we can deliver all the messages to the leader in $O(1)$ rounds using Lenzen's routing method [Len13]. Then, the leader can compute the MIS follow among the vertices with rank in the selected range. It then reports the result to all vertices in a single round, by telling each vertex whether it is in the computed independent set or not. A single round of computation, where vertices in the MIS report

to all their neighbors, can then let us remove all vertices that have a neighbor in the independent set (or are in the set) from the problem. We can then continue to the next iteration.

Regarding the round-complexity of the algorithm once the rank becomes $n/\log^{10} n$: The work [Gha17] already provides a way to solve MIS in $O(\log\log\Delta')$ congested-clique rounds for any $\Delta' = 2^{O(\sqrt{\log n})}$. Here, Δ' is the maximum degree of the graph remained after processing the vertices up to rank $n/\log^{10} n$, and, as we show by Lemma 2.1, that $\Delta' \leq \operatorname{polylog} n \ll 2^{O(\sqrt{\log n})}$. Hence, the overall round complexity is again $O(\log\log\Delta)$ rounds.

2.3 Analysis

Since in the i-th iteration, we go up to rank n/Δ^{α^i}, we reach rank $n/\log^{10} n$ in $O(\log\log\Delta)$ iterations. In the proof of Theorem 1.1 presented below, we prove that the number of edges sent to one machine per phase is $O(n)$, with high probability. Before that, we present lemma that will aid in bounding the degrees and the number of edges throughout the process.

Lemma 2.1. *Suppose that we have simulated the algorithm up to rank r. Let G_r be the remaining graph. Then, the maximum degree in G_r is $O(n\log n/r)$ with high probability.*

Proof. We first upper-bound the probability that a vertex of degree d appears in G_r. Then, we conclude that the degree of a vertex in G_r is $O(n\log n/r)$ with high probability.

Consider a vertex whose degree is still d. When the sequential algorithm considers one more vertex, which is like choosing a random one among the remaining vertices, one of this vertex or its neighbors gets hit with probability at least d/n. If that happens, this vertex would be removed. The probability that this does not happen throughout ranks 1 to r is at most $(1-d/n)^r \leq \exp(-rd/n)$. Now, the probability that a vertex in G_r has degree more than $20n\log n/r$ is at most $1/n^5$, which implies that, the maximum degree of G_r is at most $20n\log n/r$ with probability at least $1-n^{-4}$. ☐

We are now ready to prove the main theorem of this section.

Proof of Theorem 1.1. We first argue about the MPC round-complexity of the algorithm, and then show that it requires $\tilde{O}(n)$ memory.

Round complexity: Recall that the algorithm considers ranks of the form $r_i := n/\Delta^{\alpha^i}$, until the rank becomes $n/\log^{10} n$ or greater. When that occurs, it applies other algorithms for $O(\log\log\Delta)$ iterations, as described in Section 2.2. Hence, the algorithm runs for at most $i^\star + \log\log\Delta$ iterations, where $i^\star$ is the smallest integer such that rank $r_{i^\star} := n/\Delta^{\alpha^{i^\star}} \geq n/\log^{10} n$. A simple calculation gives $i^\star = \log\log_{4/3}\Delta$, for $\alpha = 3/4$. Furthermore, every iteration can be implemented in $O(1)$ rounds as discussed above.

Memory requirement: We first discuss the memory required to implement the process until the rank becomes $O(n/\log^{10} n)$. By Lemma 2.1 we have that after the graph up to rank r_i is simulated, the maximum degree in the remaining graph is $O(n\log n/r_i)$ w.h.p. Observe that it also trivially holds in the first iteration, i.e. the initial graph has maximum degree $O(n)$. Let G_i be the graph induced by the rank between r_i and r_{i+1}. Then, a neighbor u of vertex v appears in G_i with probability $(r_{i+1}-r_i)/(n-r_i) \leq r_{i+1}/n$. Hence, the expected degree of every vertex in this graph is

$$\mu := \Theta(n\log n/r_i \cdot r_{i+1}/n) = \Theta\left(\Delta^{(1-\alpha)\alpha^i}\log n\right).$$

Since $\mu \geq \log n$, by Chernoff bound we have that every vertex in G_i has degree $O(\mu)$ w.h.p. Now, since there are $O(r_{i+1})$ vertices in G_i, we have that G_i contains

$$O\left(r_{i+1}\Delta^{(1-\alpha)\alpha^i}\log n\right) = O\left(n\Delta^{-\alpha^i/2}\log n\right) \tag{1}$$

many edges w.h.p. Recall that the algorithm iterates over the ranks until the maximum degree becomes less than $\log^{10} n$. Also, $\Theta(n \log n / r_i)$ upper-bounds the maximum degree (see Lemma 2.1). Hence, we have

$$\Theta(n \log n / r_i) \geq \log^{10} n \implies \Delta^{\alpha^i} \geq \Omega\left(\log^9 n\right).$$

Combining the last implication with Eq. (1) provides that G_i contains $O(n)$ edges w.h.p.

After the rank becomes $n/\log^{10} n$ or greater, we run the CONGESTED-CLIQUE algorithm of [Gha17] for $O(\log \log \Delta)$ iterations. Since that algorithm performs only simple local decisions with low communication, every iteration of the algorithm can be implemented in $O(1)$ MPC rounds, with $\tilde{O}(n)$ memory per machine, by using standard techniques. Finally, the graph remaining after running $O(\log \log \Delta)$ iterations of that algorithm contains $O(n)$ edges w.h.p. (see Lemma 2.11 of [Gha17]). Hence, we deliver the remaining graph to one machine and construct its MIS. □

3 Matching and Vertex Cover, Simple Approximations

In this section, we describe a simple algorithm that leads to a fractional matching of weight within a $(2+\varepsilon)$-factor of (integral) maximum matching and, the same algorithm, leads to a $2+\varepsilon$ approximation of minimum vertex cover, for any small constant $\varepsilon > 0$. In the next section (Section 4), we explain how to obtain an integral $(2+\varepsilon)$-approximate maximum matching from the described fractional one. That result along with standard techniques underlined in Section 1.3 provides $1+\varepsilon$ approximation of maximum matching.

In Section 3.1, we first present the advertized algorithm that runs in $O(\log n)$ rounds. Then, in Section 3.2 and Section 3.3, we explain how to simulate this algorithm in $O(\log \log n)$ rounds of the MPC model. In Section 3.4 we provide the analysis of this simulation.

3.1 Basic $O(\log n)$-round Centralized Algorithm

We now provide a simple centralized algorithm for obtaining the described fractional matching and minimum vertex cover. We refer to this algorithm as CENTRAL.

Central: Centralized $O(\log n)$-round Fractional Matching and Vertex Cover:

- Initially, for each edge $e \in E$, set $x_e = 1/n$.
- Then, in each iteration $t \in [L]$:
  - (A) Freeze each vertex v for which $y_v = \sum_{e \ni v} x_e \geq 1 - 2\varepsilon$ and freeze all its edges.
  - (B) For each active edge, set $x_e \leftarrow x_e/(1-\varepsilon)$.
- At the end, once all edges are frozen, output the set of values x_e as a fractional matching and the set of frozen vertices as a vertex cover.

Lemma 3.1. *For any $0 < \varepsilon \leq 1/10$, the algorithm* CENTRAL *terminates after $O(\log n/\varepsilon)$ iterations, at which point all edges are frozen. Moreover, we have two properties:*

(A) The set of frozen vertices—i.e., those v for which $y_{v,t} = \sum_{e \ni v} x_e \geq 1 - 2\varepsilon$—is a vertex cover with size within $(2+5\varepsilon)$ factor of the minimum vertex cover.

(B) $\sum_{e \in E} x_e \geq |M^\star|/(2+5\varepsilon)$, that is, the computed fractional matching has size within $(2+5\varepsilon)$-factor of the maximum matching

Proof. For the sake of completeness, we provide a simple proof to this claim. We first prove the claim about vertex cover, and then about maximum matching.

Vertex cover: Let C be the vertex cover obtained by the algorithm. Every vertex added to C has weight at least $1 - 2\varepsilon$. Furthermore, an edge can be incident to at most 2 vertices of C. Let W_M be the weight of the fractional matching the algorithm constructs. Then, we have $|C| \leq 2W_M/(1-2\varepsilon) \leq 2(1+5\varepsilon)W_M$, for $\varepsilon \leq 1/10$. From the strong duality we have that the weight of fractional minimum vertex covers is at least W_M. Therefore, the minimum (integral) vertex cover has size at least W_M as well. This now implies that $|C|$ is a $2(1+5\varepsilon)$-approximate minimum vertex cover.

Maximum matching: Let $W^\star_M$ be the weight of a fractional maximum matching. Then, it holds $|M^\star| \leq W^\star_M \leq |C|$. From our analysis above and the last chain of inequalities we have $W_M \geq |M^\star|/(2(1+5\varepsilon))$. □

3.2 An Attempt for Simulation in $O(\log\log n)$ rounds of MPC

An Idealized MPC Simulation: Next, we explain an attempt toward simulating the algorithm CENTRAL in the MPC model. Once we discuss this, we will point out some shortcomings and then explain how we plan to adjust the algorithm to address this shortcoming.

The algorithm starts with every vertex and every edge being active. If not active, an edge/vertex is frozen. Throughout the algorithm, the minimum active fractional edge value increases and consequently, the degree of each vertex with respect to active edges decreases gradually. We break the simulation into phases, where the i^{th} phase ensures to simulate enough of the algorithm until the minimum active fractional edge value is $1/\Delta^{-(0.9)^i}$, which implies that the active degree is at most $\Delta^{(0.9)^i}$. Hence, we finish within $O(\log\log n)$ phases. **Remark:** In our final implementation, the number of iteration one phase simulates is slightly different than presented here. However, that final implementation, that we precisely define in the sequel, follows the exact same behavior as presented here.

Let us focus on one phase. Suppose that G' is the remaining graph on the active edges, the minimum active fractional edge value is $1/d$, and thus G' has degree at most d. In this phase, we simulate the algorithm until the minimum active fractional edge value reaches $1/d^{0.9}$, which implies that the active degree is at most $d^{0.9}$.

We partition the vertex-set of the graph G' with active edges among $m = \sqrt{d}$ machines; let G'_i be the graph given to machine i. Each machine receives $O(n)$ active edges, with high probability. Each machine simulates the basic algorithm for the next $\log_{1/(1-\varepsilon)} d/10$ rounds. For that, the machine i who received a vertex v uses $\tilde{y}_v$ as an estimate of $y_v = \sum_{e \ni v} x_e$—to compare with threshold $1-2\varepsilon$, for deciding whether to freeze v's edges or not—, defined as follows: $\tilde{y}_v = m \cdot \sum_{e \ni v;\, e \in G'_i} x_e + \sum_{e \ni v;\, e \in G \setminus G'} x_e$. That is, $\tilde{y}_v$ is the summation of edge-values of G'-edges incident on v whose other endpoint is in the same machine, multiplied by m (to normalize for the partitioning), plus the value of all edges remaining from $G \setminus G'$, i.e., edges that were frozen before this phase. If $\tilde{y}_v \geq 1-2\varepsilon$, then the machine freezes vertex v in this round and freezes its edges. After deciding that for all vertices in the machine, for any active edge $e \in G'_i$, the machine sets $x_e \leftarrow x_e \cdot 1/(1-\varepsilon)$. The phase ends after $\log_{1/(1-\varepsilon)} \Delta/10$ rounds. At the end, the round in which different vertices were frozen determines when the corresponding edges got frozen (if they did). So, it suffices to spread the information about the frozen vertices and the related timing to deduce the edge-values of all edges. Since per iteration each active edge increases by a factor of $1/(1-\varepsilon)$, after $\log_{1/(1-\varepsilon)} \Delta/10$ rounds, the minimum active edge value reaches $1/d^{0.9}$ and we are done with this phase.

The Issue with the Direct Simulation: Consider first the following wishful-thinking scenario. Assume for a moment that in *every iteration* it holds $|y_v - (1-2\varepsilon)| > |y_v - \tilde{y}_v|$, that is, y_v and $\tilde{y}_v$ are "on the same side" of the threshold. Then, the algorithm CENTRAL and the MPC simulation of

it make the same decision on whether a vertex v gets frozen or not. Moreover, this happens in every iteration, as can formalized by a simple induction. This in turn implies that the MPC algorithm performs the exact same computations as the CENTRAL algorithm and thus it provides the same approximation as CENTRAL. However, in general case, even if y_v and $\tilde{y}_v$ are almost equal, e.g. $|y_v - \tilde{y}_v| \ll \varepsilon$, it might happen that $y_v \geq 1 - 2\varepsilon$ and $\tilde{y}_v < 1 - 2\varepsilon$, resulting in the two algorithms making different decisions with respect to v. Furthermore, this situation could occur for many vertices simultaneously, and this deviation of the two algorithms might grow as we go through the round; these complicate the task of analyzing the behavior of the MPC algorithm.

Random Thresholding to the Rescue:

Observe that if $|y_v - \tilde{y}_v|$ is small then there is only a "small range" of values of y_v around the threshold $1-2\varepsilon$ which could potentially lead to the two algorithms behaving differently with respect to v. Motivated by this observation, instead of having one fixed threshold throughout the whole algorithm, in each iteration t and for each vertex v the algorithm will uniformly at random choose a fresh threshold $\mathcal{T}_{v,t}$ from the interval $[1 - 4\varepsilon, 1 - 2\varepsilon]$. We call this algorithm CENTRAL-RAND, and state it below. Then, if v is not frozen until the t^{th} iteration, v gets frozen by CENTRAL-RAND if $y_{v,t} \geq \mathcal{T}_{v,t}$ (and similarly, v get frozen by the MPC simulation if $\tilde{y}_{v,t} \geq \mathcal{T}_{v,t}$). In that case, if $|y_v - \tilde{y}_v| \ll \varepsilon$, then most of the time y_v would be far from the threshold and the two algorithms would behave similarly. We make this intuition formal in the next section by Lemma 3.11.

3.3 Our Actual Simulation in $O(\log \log n)$ rounds of MPC

We now present the modified CENTRAL-RAND algorithm with the random thresholding and then discuss how we simulate it in the MPC model.

Central-Rand: Centralized $O(\log n)$-round Fractional Matching and Vertex Cover with Random Thresholding:

- Each vertex v chooses a list of thresholds $\mathcal{T}_v$ such that: the thresholds are chosen independently; each threshold is chosen uniformly at random from $[1 - 4\varepsilon, 1 - 2\varepsilon]$.
- Initially, for each edge $e \in E$, set $x_e = 1/n$.
- Then, in each iteration $t \in [L]$:
  - (A) Freeze each vertex v for which $y_{v,t} = \sum_{e \ni v} x_e \geq \mathcal{T}_{v,t}$ and freeze all its edges.
  - (B) For each active edge, set $x_e \leftarrow x_e/(1-\varepsilon)$.
- At the end, once all edges are frozen, output the set of values x_e as a fractional matching and the set of frozen vertices as a vertex cover.

Our Actual MPC Simulation: We now provide an MPC simulation of CENTRAL-RAND, that we will refer to by MPC-SIMULATION, and discuss it below.

Our algorithm begins by selecting a collection of random thresholds $\mathcal{T}$. In the actual implementation, since these thresholds are chosen independently and each from the same interval, threshold $\mathcal{T}_{v,t}$ can be sampled when needed ("'on the fly"'). During the simulation, we maintain a vertex set $V' \subseteq V$ that consists of vertices that we consider for the rest of the simulation. The algorithm defines the initial weight of the edges to be $w_0 = (1-2\varepsilon)/n$. Also, it maintains variable d representing the upper-bound on the maximum degree in the remaining graph (in principle, the maximum degree can be smaller than d).

MPC-SIMULATION is divided into phases. At the beginning of a phase, we consider a subgraph G' of $G[V']$ that consists only of the active edges. In Lemma 3.6 we prove that the maximum degree in G' is at most d. Then, the vertex set V' is distributed across $m = \sqrt{d}$ machines. Each machines

collects the induced graph of G' on the vertex set assigned to it. In Lemma 3.7 we prove that each of these induced graphs consists of $O(n)$ edges. Also at the beginning of a phase, the algorithm defines y_v^{old} (see line (d)). This is part of the vertex-weight that remains same throughout the execution of the phase. It corresponds to the sum of weights of the edges incident to v there were assigned in prior phases.

Each phase executes the steps under line (e), which simulate I iterations of Central-Rand. During a phase, we maintain the iteration-counter t. The value of t counts all the iterations since the beginning of the algorithm, and not only from the beginning of a phase. After this simulation is over, the weight x_e^{MPC} of each edge e is properly set/updated. For instance, if e was not assigned to any of the machines (i.e. its endpoints were assigned to distinct machines), then x_e^{MPC} was not changing during the simulation of Central-Rand in this phase even if both of its endpoints were active. To account for that, at line (g) the value x_e^{MPC} is set to $w_0 \frac{1}{(1-\varepsilon)^{t'}}$, where t' is the last iteration when both endpoints of e were active. To implement this step, each vertex will also keep a variable corresponding to the iteration when it was the last active.

Every vertex v that has weight more than 1 is removed from the consideration, e.g. removed from V' at line (i), but v is added to the vertex cover that is reported at the end of the algorithm. This step ensures that throughout the algorithm the fractional matching on $G[V']$ will be valid. But it also ensures that all the edges that are in $G[V \setminus V']$, in particular those incident to v, will be covered by the by the final vertex cover.

MPC-Simulation: MPC Simulation of algorithm Central-Rand:

(1) Each vertex v chooses a list of thresholds $\mathcal{T}_v$ such that: the thresholds are chosen independently; each threshold is chosen uniformly at random from $[1-4\varepsilon, 1-2\varepsilon]$.

(2) Init: $V' = V$; for each edge $e \in E$, set $x_e^{MPC} = w_0 = (1-2\varepsilon)/n$; $d = n$; $t = 0$.

(3) While $d > \log^{20} n$:

  (a) Set: # machines $m = \sqrt{d}$; # iterations $I = (\log m)/(10 \log 5)$.

  (b) Partition V' into m sets $V_1, \ldots, V_m$ by assigning each vertex independently uniformly at random.

  (c) Let G' be a graph on V' consisting only of the active edges of $G[V']$.

  (d) For each $v \in V'$, define $y_v^{old} = \sum_{e \ni v;\, e \in G[V']} x_e^{MPC}$.

  (e) For each $i \in \{1, \ldots, m\}$ in parallel execute I iterations

    (A) Freeze each $v \in V_i$ for which $\tilde{y}_{v,t} = m \cdot \sum_{e \ni v;\, e \in G'[V_i]} x_e^{MPC} + y_v^{old} \geq \mathcal{T}_{v,t}$ and freeze all its edges.

    (B) For each active edge, set $x_e^{MPC} \leftarrow x_e^{MPC}/(1-\varepsilon)$.

    (C) Increment the total iteration count: $t \leftarrow t+1$.

  (f) Update $d \leftarrow d(1-\varepsilon)^I$.

  (g) For every edge $e = \{u, v\}$: set $x_e^{MPC} = w_0 \frac{1}{(1-\varepsilon)^{t'}}$, where t' is the last iteration in which both u and v were active.

  (h) For each $v \in V'$ let $y_v^{MPC} = \sum_{e \ni v;\, e \in G[V']} x_e^{MPC}$.

  (i) For each $v \in V'$ such that $y_v^{MPC} > 1$: freeze v and remove v from V'.

  (j) For each $v \in V'$ such that $y_v^{MPC} > 1-2\varepsilon$: freeze v and freeze all its edges.

(4) Directly simulate $\log_{1/(1-\varepsilon)} \log^{20} n$ iterations of Central-Rand.

(5) Output the vector x^{MPC} as a fractional matching and the set of frozen vertices as a vertex cover.

If some vertex has weight between $1-2\varepsilon$ and 1, it has sufficiently large fractional weight, so we simple freeze it (line (j)) before the next phase.

Once the upper-bound d becomes less than $\log^{20} n$, the algorithm exits from the main while loop, and the rest of the iterations needed to simulate Central-Rand are executed one by one. During this part of the simulation, MPC-Simulation and Central-Rand behave identically.

3.4 Analysis

We prove that the set of frozen vertices forms a $2 + O(\varepsilon)$ approximation of the minimum vertex cover, and the computed fractional matching is a $2 + O(\varepsilon)$ approximation of maximum matching.

Lemma 3.2. MPC-Simulation *with high probability outputs a* $(2+50\cdot\varepsilon)$*-approximate minimum vertex cover and a fractional matching which is a* $(2+50\cdot\varepsilon)$ *approximation of maximum matching. Moreover, there is an implementation of* MPC-Simulation *that with high probability has* $O(\log\log n)$ *MPC-round complexity and requires* $O(n)$ *space per machine.*

Furthermore, the algorithm outputs fractional matching x *and a vertex cover* C *such that the fractional weight of at least* $|C|/3$ *vertices of* C *is at least* $1-5\varepsilon$.

Remark: For technical reasons and for the sake of clarity of our exposition, in our analysis we

assume that $\varepsilon < 1/50$. (If the input $\varepsilon \geq 1/50$, we simply reduce its value and deliver even better approximation than required.) Also, as ε is a constant, we assume that $\varepsilon > 1/\log n$.

Roadmap: We split the proof of Lemma 3.2 into three parts. We start by, in Section 3.4.1, showing some properties of the edge-weights and the maximum degree of vertices during the course of MPC-Simulation. Then, in Section 3.4.2 we prove that $O(n)$ space per machine suffices for the execution of MPC-Simulation, and that the algorithm can be executed in $O(\log\log n)$ MPC-rounds. Next, in Section 3.4.3 we relate the vertex-weights in MPC-Simulation (i.e. the vectors $\tilde{y}$ and y^{MPC}) to the corresponding weights in the algorithm Central-Rand (i.e. to the vector y). namely, we trace $|\tilde{y}_v - y_v|$ over the course of one phase, and show that for most of the vertices this difference remains small. We put forth those results in Section 3.4.4 and prove Lemma 3.2.

3.4.1 Weight and degree properties

We now state several properties of edge-weights that are easily derived from the algorithm, and provide an upper-bound on the maximal active degree of any vertex. These properties will be used throughout our proofs in the coming sections.

Define $w_t = w_0 \frac{1}{(1-\varepsilon)^t}$. Observe that at the t^{th} iteration, the weight of all the active edges that are on some of the machines equals w_t. Furthermore, if for an edge $e = \{u, v\}$ such that u and v are on different machines, vertices u and v are both active in the t^{th} iteration, then after the phase ends the weight x_e^{MPC} will be set to at least w_t (see line (g)). We next state two observations.

Observation 3.3 (Degree — active-weight invariant)**.** *Consider an iteration t at which is updated d at line (f) of* MPC-Simulation*. Then, just after the degree d is updated, it holds $d \cdot w_t = 1 - 2\varepsilon$.*

Proof. At the beginning of the algorithm, it holds $w_0 \cdot d = 1 - 2\varepsilon$. Over a phase, weights of the active edges increase by $1/(1-\varepsilon)^I$. On the other hand, the degree d decreases by $1/(1-\varepsilon)^I$. Hence, their product remains the same after every phase. □

Observation 3.4 (Maximum weight of active-edge)**.** *The weight of any active edge at the beginning of a phase is $(1-2\varepsilon)/m^2$, where m is the number of machines used in that phase. During that phase, the weight of any edge is at most $1/m^{1.8}$.*

Proof. Let $w_{t^\star}$ be the weight of any active edge at the beginning of a phase. As defined at line (a), we have $m^2 = d$. From Observation 3.3 we hence conclude that $w_{t^\star} = (1-2\varepsilon)/m^2$.

Also at line (a), I is defined to be $(\log m)(10\log 5) < (\log m)/5$. On the other hand, for at most I iterations the weight of any active edge is increased by at most $1/(1-\varepsilon) \leq 2$ at line (eC). Hence

$$w_{t^\star+I} \leq 2^I \cdot (1-2\varepsilon)/m^2 \leq (1-2\varepsilon)m^{0.2}/m^2 \leq m^{1.8}.$$

□

3.4.2 Memory requirement and round complexity

In this section, we first show that $O(n)$ space per machines suffices to store the induced graphs $G'[V_i]$ considered by MPC-Simulation (see Lemma 3.7). After, in Lemma 3.8, we upper-bound the number of phases of MPC-Simulation. At the end of the section, we combine these together to prove the following lemma.

Lemma 3.5. *There is an implementation of* MPC-Simulation *that requires $O(n)$ memory per machine and executes $O(\log\log n)$ MPC rounds w.h.p.*

We start by upper-bounding the number of active edges incident to a vertex of V'.

Lemma 3.6. *Let V', G' and d be as defined in* MPC-Simulation *at the beginning of the same phase. Then, the degree of every vertex in $G'[V']$ is at most d.*

Proof. In the beginning of the algorithm, we have that $d = n$, and hence the statement holds for the very first phase.

Towards a contradiction, assume that there exists a phase and a vertex v such that the degree of v in $G'[V']$ is more than d. Let d_v denote its degree. Let $t^\star$ be the first iteration of that phase. Notice that $w_{t^\star}$ was the weight of active edges at the end of the previous phase. Now by Observation 3.3 we have

$$w_{t^\star} \cdot d_v > w_{t^\star} \cdot d = 1 - 2\varepsilon.$$

But this now contradicts the step at line (j) of MPC-Simulation after which all the edges incident to v would become frozen. □

Now we prove that every induced graph processed on machine has $O(n)$ edges.

Lemma 3.7 (Size of induced graphs)**.** *Let G' and V_i be as defined at line (c) and line (b) of* MPC-Simulation*, respectively. Than, $|E(G'[V_i])| \in O(n)$ w.h.p.*

Proof. We split the proof into two parts. First, we argue that the size of V_i is $O(n/m)$ w.h.p. After, we argue that the degree of each vertex in $G'[V_i]$ is $O(d/m)$ w.h.p., after which the proof will follow by union bound.

Expected size of V_i: Now, $\mathbb{E}[|V_i|] = |V'|/m \leq n/m$. Observe that we have $m \leq \sqrt{n}$ at any phase, and hence $n/m \geq \sqrt{n}$. Now Chernoff bound implies that

$$|V_i| \leq |V'|/m + n/m \in O(n/m) \tag{2}$$

hold w.h.p.

Degree bound: Consider a vertex $v \in V_i$, and let d_v be its degree in V_i. Lemma 3.6 implies $\mathbb{E}[d_v] \leq d/m$. By the definition it holds $d/m = m$, and also $m \geq \log^{10} n$. Now again by applying Chernoff bound, we conclude that

$$\mathbb{E}[d_v] \leq d/m + m \in O(m) \tag{3}$$

holds w.h.p.

Combining the bounds: Since Eq. (2) and Eq. (3) hold independently and w.h.p., by taking union bound over all the vertices we conclude that the number of the edges in $G'[V_i]$ is bounded by $O((n/m) \cdot m)$ w.h.p. This concludes the proof. □

Lemma 3.8 (Number of phases upper-bound)**.** MPC-Simulation *executes $O(\log\log n)$ phases.*

Proof. Let d_i be the degree d of MPC-Simulation at the beginning of a phase, and d_{i+1} the degree updated at line (f) after the phase ends. Let $I = (\log m)/(10\log 5) = (\log d_i)/(20\log 5)$. Then, by the definition of the algorithm, we have the following relation

$$d_{i+1} = d_i(1-\varepsilon)^I = d_i \left(\frac{1}{2}\right)^{\log(1/(1-\varepsilon)\frac{\log d_i}{20\log 5}} = d_i^{1-\frac{\log(1/(1-\varepsilon)}{20\log 5}}. \tag{4}$$

For the sake of brevity, define $\gamma := \frac{\log(1/(1-\varepsilon))}{20\log 5}$. Observe that for a constant ε such that $0 < \varepsilon < 1/2$ it implies that γ is a constant and $\gamma < 1$. Then, from Eq. (4) we have

$$d_i = d_0^{(1-\gamma)^i} = n^{(1-\gamma)^i}.$$

MPC-Simulation is executed for $i^\star$ phases, where $i^\star$ is the smallest integer such that $d_{i^\star} \leq \log^{20} n$. This implies

$$n^{(1-\gamma)^{i^\star}} \leq \log^{20} n.$$

Taking log on the both sides of the last inequality, we obtain

$$(1-\gamma)^{i^\star} \log n \leq 20 \log\log n.$$

Now a simple calculation shows that $i^\star \in O\left(\frac{\log\log n}{\log(1/(1-\gamma))}\right) \in O(\log\log n)$. □

Proof of Lemma 3.5. By Lemma 3.8, MPC-Simulation executes $O(\log\log n)$ phases. Also, for constant ε, line (3.3) requires $O(\log\log n)$ iterations. Furthermore, by Lemma 3.7, each induced graph $G'[V_i]$ that is processed on a single machine during a phase has $O(n)$ size w.h.p.

There is an implementation of MPC-Simulation such that, when every machine has space $O(n)$, each phase and each of the operations the algorithm performs are executed in $O(1)$ MPC-rounds. For more details on such implementation, we refer the reader to [CŁM+18], section *MPC Implementation Details*, and to [GSZ11]. □

3.4.3 Properties of vertex- and edge-weights in MPC-Simulation

In this section, we show that $|y_v - \tilde{y}_v|$ remains small for most of the vertices (this claim is formalized in Lemma 3.15). Before we provide an outline of the analysis, we state some definition and describe the notation we use.

Definition 3.9 (Bad and good vertex)**.** *We say that vertex is* bad *if it gets frozen in* Central-Rand *and not in* MPC-Simulation *(or the other way around). Once bad, the vertex remains bad throughout the whole phase. If a vertex is not bad, we say it is* good*.*

Definition 3.10 (Local neighbor)**.** *If a neighbor u of vertex v is in the given iteration of* MPC-Simulation *on the same machine as v, then we say that u is a* local *neighbor of v.*

Notation: We use w_t to refer to the weight of active edge in the t^{th} iteration. Let $N_A^{central}(v,t)$ (resp. $N_A^{local}(v,t)$) denote the active neighbors of v at the beginning of the t^{th} iteration of the ideal (resp. MPC) algorithm. Similarly, we use $N^{local}(v,t)$ to denote the local neighbors of v in iteration t. If it is clear from the context which iteration we are referring to, sometimes we omit t from the notation. Throughout our proofs, we will be making claims of the following form $a = b \pm c$, which should be read as $a \in [b-c, b+c]$.

Analysis Outline: Recall that $\tilde{y}_{v,t}$ and $y_{v,t}$ represent the fractional weight of vertex v in the t^{th} iteration of MPC-Simulation and Central-Rand, respectively. From the definition, we have $y_{v,t} = y_{v,t-1} + \varepsilon\, w_{t-1}\, |N_A^{central}(v,t)|$, and similarly $\tilde{y}_{v,t} = \tilde{y}_{v,t-1} + \varepsilon\, w_{t-1}\, m\, |N_A^{local}(v,t)|$. To say that the algorithms stay close to each other, we upper-bound $|y_{v,t} - \tilde{y}_{v,t}|$ inductively as a function of t. Suppose that we already have an upper bound on $|y_{v,t-1} - \tilde{y}_{v,t-1}|$; we focus on upper-bounding the difference between $|N_A^{central}(v,t)|$ and $m|N_A^{local}(v,t)|$. There are two sources of difference between $|N_A^{central}(v,t)|$ and $m|N_A^{local}(v,t)|$:

(1) Some of the neighbors of v might be bad. Notice that this also implies that in general the set $N_A^{local}(v,t)$ might not even be a subset of $N_A^{central}(v,t)$.

(2) Even in the very first iteration (or more generally even if there is no bad vertex in $N_A^{local}(v,t)$), the set $N_A^{local}(v,t)$ is a random sample of $N_A^{central}(v,t)$ and hence $|N_A^{local}(v,t)|$ might deviate from its expectation $|N_A^{central}(v,t)|/m$.

Furthermore, in our analysis, we assume that at the beginning of each phase MPC-Simulation and Central-Rand start from the same fractional matching. Namely, we compare MPC-Simulation to the behavior of Central-Rand assuming that initially x equals x^{MPC}, for the value of x^{MPC} at the beginning of a given phase. Since we ensure that x^{MPC} is at the beginning of a phase always a valid fractional matching, Central-Rand in our approach will also maintain a valid fractional matching.

Also, we assume that the thresholds, i.e. $\mathcal{T}_{v,t}$ for each $v \in V$ and each iteration t, are the same for both MPC-Simulation and Central-Rand. Note that the latter algorithm is only a hypothetical one, whose purpose is to compare our simulation to a process that constructs a fractional matching, so this assumption is made without loss of generality.

Analysis:

The following claim is a direct consequence of choosing the thresholds randomly at each iteration.

Lemma 3.11. *Consider the t^{th} iteration of a phase. Let $|y_{v,t} - \tilde{y}_{v,t}| \leq \sigma$ for every vertex v that is active in both* Central-Rand *and* MPC-Simulation*. Then, v becomes bad in the t^{th} iteration with probability at most ε/σ and independently of other vertices.*

Proof. If $|\tilde{y}_{v,t} - \mathcal{T}_{v,t}| > \sigma$, then MPC-Simulation and Central-Rand would behave the same with respect to vertex v. Since $\mathcal{T}_{v,t}$ is chosen uniformly at random within interval of size 2ε, MPC-Simulation and Central-Rand would differ in iteration t with respect to v with probability at most $2\sigma/(2\varepsilon) = \sigma/\varepsilon$. Furthermore, as $\mathcal{T}_{v,t}$ is chosen independently of other vertices, v becomes bad independently of other vertices. □

There are two distinct steps where MPC-Simulation directly or indirectly estimates y. The first one is computing $\tilde{y}$, which is used to deduce whether a vertex should be frozen or not. The second one corresponds to the actual weight that MPC-Simulation assigns to the vertices. Namely, at the end of a phase, weight is assigned to each edge (line (g)) – for edge $e = \{u, v\}$, if u or v is frozen, then it is set $x_e^{MPC} = w_t$, where t is the iteration when the first of the two vertices got frozen; otherwise, $x_e^{MPC} = w_t$ for t being the most recent simulated iteration. Then, the weight of a vertex v, that we denote by y_v^{MPC}, is simply the sum of all x_e^{MPC} incident to v. This can be seen as an indirect estimate of y_v.

Our next goal is to understand how does the estimate $\tilde{y}_v$ and simulated vertex weight y_v^{MPC} relate to y_v. To that end, we define the notion to capture the difference in how the weights y_v and y_v^{MPC} are composed.

Definition 3.12 (Weight-difference)**.** *We use* $\text{diff}(v,t)$ *to denote the total weight of the edges that contributed to the weight of $y_{v,t}$ and not to $y_{v,t}^{MPC}$, and the other way around. Formally, let $x_{e,t}^{MPC}$ be the updated weight of edge e in iteration t in* MPC-Simulation *(updated in the sense as given by line (g) of the algorithm). Let $x_{e,t}$ be the weight of edge e in iteration t in* Central-Rand*. Then,*

$$\text{diff}(v,t) := \sum_{e \in N(v)} \left| x_{e,t} - x_{e,t}^{MPC} \right|.$$

Notice that $\left|y_{v,t} - y_{v,t}^{MPC}\right| \leq \text{diff}(v,t)$. In general it might be the case that $\left|y_{v,t} - y_{v,t}^{MPC}\right| < \text{diff}(v,t)$. For instance, consider two edges e_1 and e_2 both incident to v. Assume that in CENTRAL-RAND e_1 is active, while e_2 is frozen. On the other hand, assume that in MPC-SIMULATION it is the case that e_1 is frozen while e_2 active. So, these two edges alone do not make any difference in the change of the weight of $y_{v,t}$ and $y_{v,t}^{MPC}$ – their effect cancels out. However, their effect does not cancel each other in the definition of $\text{diff}(v,t)$.

As a first step, we show that y_v is close to y_v^{MPC} in the first iteration of some phase.

Lemma 3.13. *Let iteration $t^\star$ be the first iteration of some phase, and let v be an active vertex by iteration $t^\star$. Then, w.h.p.*

$$|y_{v,t^\star} - \tilde{y}_{v,t^\star}| \leq m^{-0.2}.$$

Furthermore, $\text{diff}(v,t^\star) = 0$ *with certainty.*

Proof. To argue that $\text{diff}(v,t^\star) = 0$ it suffices to observe that, at the beginning of a phase, edge-weights in MPC-SIMULATION and CENTRAL-RAND coincide.

We upper-bound $|y_{v,t^\star} - \tilde{y}_{v,t^\star}|$ as follows. At the beginning of the phase, no vertex is bad. Hence, the only difference between $y_{v,t^\star}$ and $\tilde{y}_{v,t^\star}$ comes from the random partitioning of the vertices.

Observe that $N_A^{local}(v,t^\star)$ is a random sample of $N_A^{central}(v,t^\star)$, and hence $\mu := \mathbb{E}\left[m \cdot |N_A^{local}(v,t^\star)|\right] = |N_A^{central}(v,t^\star)|$. We now consider two cases, based on μ.

Case $\mu \leq m^{1.6}$: Then, from Chernoff bound we have

$$P\left(\left|m \cdot |N_A^{local}(v,t^\star)| - \mu\right| \geq \frac{m^{1.6}}{\mu}\mu\right) \leq 2\exp\left(-m^{3.2}/(3\mu)\right) \leq 2\exp\left(-m^{1.6}/3\right), \tag{5}$$

which is high probability as $m \geq \log^{10} n$ during every phase.

Case $\mu > m^{1.6}$: Now again by Chernoff bound, we have

$$P\left(\left|m \cdot |N_A^{local}(v,t^\star)| - \mu\right| \geq m^{-0.2}\mu\right) \leq 2\exp\left(m^{-0.4}\mu/3\right) \leq 2\exp\left(m^{1.2}\mu/3\right). \tag{6}$$

This probability is again high, as $m \gg \log n$.

Combining the two cases: From Eq. (5) and Eq. (6) we derive that w.h.p. that

$$m \cdot |N_A^{local}(v,t^\star)| = |N_A^{central}(v,t^\star)| \pm \max\{m^{-0.2}|N_A^{central}(v,t^\star)|, m^{1.6}\}.$$

Hence, w.h.p.

$$|y_{v,t^\star} - \tilde{y}_{v,t^\star}| \leq w_{t^\star} \max\{m^{-0.2}|N_A^{central}(v,t^\star)|, m^{1.6}\}.$$

Now, from Observation 3.3 and Observation 3.4 we have that $|N_A^{central}(v,t^\star)|w_{t^\star} \leq 1$ and $w_{t^\star} \leq m^{-1.8}$. This implies that w.h.p. it holds

$$|y_{v,t^\star} - \tilde{y}_{v,t^\star}| \leq m^{-0.2},$$

as desired. ∎

We now prove our main technical lemma, which quantifies the increase in the difference between y_v and its estimates $\tilde{y}_v$ and y_v^{MPC} over the course of one phase.

Lemma 3.14 (Evolution of weight-estimates)**.** *Let v be an active vertex in iteration $t-1$ in both* CENTRAL-RAND *and* MPC-SIMULATION*. Then, if $|y_{v,t-1} - \tilde{y}_{v,t-1}| \leq \sigma$ and $\text{diff}(v,t) \leq \sigma$, the following holds w.h.p.:*

- $|y_{v,t} - \tilde{y}_{v,t}| \leq 4(\sigma + \varepsilon m^{-0.2})$, *and*
- $\text{diff}(v,t) \leq 4(\sigma + \varepsilon m^{-0.2})$.

Proof. We proceed by upper-bounding the effect of three different kinds of vertices on $|y_{v,t} - \tilde{y}_{v,t}|$: bad vertices prior to the t^{th} iteration; vertices becoming bad in the t^{th} iteration; and, the effect of the random partitioning. Observe that $\text{diff}(v,t)$ is *not directly* affected by the random partitioning.

Old bad vertices.: Let $B_{v,t-1}$ be the set of bad vertices prior to the beginning of the t^{th} iteration that are also local neighbors of v. The set $B_{v,t-1}$ accounts for the vertices $N_A^{local}(v,t-1) \setminus N_A^{central}(v,t-1)$, and for the local neighbors of v that are in $N_A^{central}(v,t-1)$ but not in $N_A^{local}(v,t-1)$. Since their weight was bounded by σ in the $(t-1)^{th}$, then their weight is bounded by $(1+\varepsilon)\sigma$ in the t^{th} iteration. Hence, from iteration $t-1$ to iteration t, the effect of the old bad vertices increased by $\varepsilon\sigma$.

In a similar way, by defining $B_{v,t-1}$ to be the set of bad neighbors of v across *all* the machines, we get that from iteration $t-1$ to iteration t the effect of the old bad vertices on $\text{diff}(v,t)$ increased by $\varepsilon\sigma$.

New bad vertices.: In addition to the bad vertices in $B_{v,t-1}$, there might be new bad vertices in the beginning of the t^{th} iteration – the vertices of $N_A^{local}(v,t-1) \cap N_A^{central}(v,t-1)$ that are not in $N_A^{local}(v,t) \cap N_A^{central}(v,t)$. To bound the weight of those bad vertices, we first upper-bound the cardinality of $N_A^{local}(v,t-1) \cap N_A^{central}(v,t-1)$. For the sake of brevity, define $n_{v,t-1}^{local} := |N^{local}(v,t-1) \cap N_A^{central}(v,t-1)|$ where, as a reminder, the set $N^{local}(v,t-1)$ refers to the local neighbors (both frozen and active) of v. We trivially have

$$|N_A^{local}(v,t-1) \cap N_A^{central}(v,t-1)| \leq n_{v,t-1}^{local}.$$

Then, by Lemma 3.11, the number of new bad vertices is in expectation at most $n_{v,t-1}^{local}\sigma/\varepsilon$. We now proceed by providing a sharp concentration around this expected value. To that end, we provide an upper-bound on $n_{v,t-1}^{local}$ that holds w.h.p.
Observe that $N_A^{central}(v,t-1)$ is defined deterministically and independently of the MPC algorithm. Then, if $|N_A^{central}(v,t-1)| \geq m^{1.6}$, we have that w.h.p. $n_{v,t-1}^{local} \leq (1+m^{-0.2})|N_A^{central}(v,t-1)|/m$. Otherwise, if $|N_A^{central}(v,t-1)| < m^{1.6}$, then w.h.p. $n_{v,t-1}^{local} \leq 2m^{0.6}$. Therefore, for $\gamma := \max\{(1+m^{-0.2})|N_A^{central}(v,t-1)|, 2m^{1.6}\}$, we have the w.h.p.

$$m \cdot n_{v,t-1}^{local} \leq \gamma.^{1}$$

Applying similar reasoning about $n_{v,t-1}^{local}\sigma/\varepsilon$, i.e. considering cases $n_{v,t-1}^{local}\sigma/\varepsilon \geq m^{0.6}$ and $n_{v,t-1}^{local}\sigma/\varepsilon < m^{0.6}$, we obtain that w.h.p. the number of new bad vertices is upper-bounded by $\max\{(1+m^{-0.2})n_{v,-1}^{local}\sigma/\varepsilon, 2m^{0.6}\}$. So, putting all together, we have that the weight coming from new bad vertices that affects the local estimate of $y_{v,t}$ is at most

$$\sigma_2 := \varepsilon w_{t-1} \cdot \max\{(1+m^{-0.2})\gamma\sigma/\varepsilon, 2m^{1.6}\}.$$

But now, using that $w_{t-1} \leq m^{-1.8}$ and also that $|N_A^{central}(v,t-1)|w_{t-1} \leq 1$ as v is a active vertex in Central-Rand, we derive $\sigma_2 \leq 2(\sigma + \varepsilon m^{-0.2})$.

It remains to comment about the effect of new bad vertices on $\text{diff}(v,t)$. Note that the expected number of new bad vertices affecting $\text{diff}(v,t)$ is at most $|N_A^{central}(v,t-1)|\sigma/\varepsilon$. So, applying the same arguments as above, the weight of new bad vertices affects $\text{diff}(v,t)$ by at most σ_2 w.h.p.

[1] A more detailed proof for this type of claim is given in the proof of Lemma 3.13.

Effect of random partitioning.: Finally, we upper-bound the effect of the random partitioning on the estimate $\tilde{y}_{v,t}$. Similarly to our arguments given earlier, we have that w.h.p. the number of vertices of $N_A^{central}(v,t)$ that are local neighbors of v deviates from $|N_A^{central}(v,t)|/m$ by at most $\eta := \max\{m^{-0.2}|N_A^{central}(v,t)|, m^{1.6}\}/m$. The total weight of these vertices scaled by m is at most $\varepsilon w_{t-1} m\eta \leq \varepsilon m^{-0.2}$.

Final step: Putting altogether, if $|y_{v,t-1} - \tilde{y}_{v,t-1}| \leq \sigma$ and $\left|y_{v,t-1} - y_{v,t-1}^{MPC}\right| \leq \sigma$, then

$$|y_{v,t} - \tilde{y}_{v,t}| \leq (1+\varepsilon)\sigma + 2(\sigma + \varepsilon m^{-0.2}) + \varepsilon m^{-0.2} \leq 4(\sigma + \varepsilon m^{-0.2}),$$

and similarly

$$\mathrm{diff}(v,t) \leq (1+\varepsilon)\sigma + 2(\sigma + \varepsilon m^{-0.2}) \leq 4(\sigma + \varepsilon m^{-0.2}),$$

as desired. □

Lemma 3.15. *Let v be an active vertex in iteration $t-1$ in both* MPC-Simulation *and* Central-Rand. *If a phase consists of at most $I := (\log m)/(10 \log 5)$ iterations, then it holds $|y_{v,t} - \tilde{y}_{v,t}| \leq m^{-0.1}$ and $\mathrm{diff}(v,t) \leq m^{-0.1}$ w.h.p.*

Proof. Let iteration $t^\star$ be the first iteration of the i^{th} phase. Combining Lemma 3.13 and Lemma 3.14, for any $t^\star \leq t \leq t^\star + I$ in which v is not bad, it holds

$$|y_{v,t} - \tilde{y}_{v,t}| \leq 5^I m^{-0.2} \leq m^{-0.1},$$

and

$$\mathrm{diff}(v,t) \leq 5^I m^{-0.2} \leq m^{-0.1}.$$

□

We are now ready to prove the main result of this section.

3.4.4 Proof of Lemma 3.2

We start the proof by recalling that Lemma 3.5 shows the desired bound on the space- and round-complexity of MPC-Simulation. The rest of the proof is divided into two parts. First, we prove the statement for vertex cover, and then for matching.

Throughout the proof we consider only those rounds of the MPC algorithm that execute at least two iterations. The rounds in which is executed only one iteration coincide with the ideal algorithm, and for them the claims in the rest of the proof follow directly. In this section, we assume that a maximum matching and a minimum vertex cover is of size at least $\log^{10} n$. In Section 3.4.5 we show how to handle that case when the maximum matching has size less than $\log^{10} n$.

Part I — Vertex Cover:

Let $\tilde{C}$ be the vertex cover constructed by MPC-Simulation. First, $\tilde{C}$ is indeed a vertex cover as by the end of the algorithm every edge is incident to at least one frozen vertex, and every frozen vertex is included to $\tilde{C}$. This follows from the following facts: by the end of the algorithm the weight of active edges is at least $1 - 2\varepsilon$; and, the last iterations of MPC-Simulation directly simulate Central-Rand. Since Central-Rand freezes any vertex (and its incident edges) having incident edge of weight at least $1 - 2\varepsilon$, MPC-Simulation freezes such vertices as well.

Informally, our goal is to show that $|\tilde{C}|$ is roughly at most twice larger than $W_M := \sum_{v \in V'} y_v^{MPC}$, where V' is the set of vertices after removing those of weight more than 1. Our proof consists of two main parts. First, we consider the contribution to W_M of the vertices that remained active

in Central-Rand for at least as many iterations in MPC-Simulation (and at first we ignore the other vertices). After that, we take into account the remaining vertices, and in the same time account for the vertices having weight more than 1.

Central-Rand freezing last: Let t be the last iteration of a phase. We first consider only those vertices added to $\tilde{C}$ which remained active in Central-Rand for at least as many iterations as in MPC-Simulation, and claim that for every such vertex v it holds $y_v^{MPC} \geq 1 - 5\varepsilon$. In the analysis we give, we ignore that some vertices u such that $y_u^{MPC} > 1$ got removed along with their incident edges. We analyze two types of vertices: good vertices; and, bad vertices that got frozen by MPC-Simulation first.

(1) If v is good, then it was active in MPC-Simulation in the same iterations as in Central-Rand. Hence, by Lemma 3.15, we have that $y_{v,t}^{MPC} \geq y_{v,t} - m^{-0.1} \geq 1 - 4\varepsilon - \varepsilon = 1 - 5\varepsilon$.

(2) Assume that v is bad, but got frozen by MPC-Simulation first. Let t' be the iteration v got frozen by MPC-Simulation. This directly implies that $\tilde{y}_{v,t} \geq 1 - 4\varepsilon$. Since v was active in the both algorithms in iteration $t' - 1$, by Lemma 3.15 we have $y_{v,t'} \geq 1 - 4\varepsilon - m^{-0.1}$. But now again by Lemma 3.15 we conclude that $y_{v,t'}^{MPC} \geq 1 - 4\varepsilon - m^{-0.1} - m^{-0.1} \geq 1 - 5\varepsilon$.

Informally (again), this analysis can be stated as: for every vertex of the two considered types which is added to $\tilde{C}$, and while disregarding the vertices whose incident edges got removed, there is at least $(1 - 5\varepsilon)/2$ weight in W_M. The weight is scaled by 2 as every edge is incident to at most 2 vertices of $\tilde{C}$.

MPC-Simulation freezing last: We now consider the vertices that got frozen by MPC-Simulation in later iteration that by Central-Rand. We call such vertices *late-bad*, and use n^{late} to denote their number. Let C denote the vertex cover constructed by Central-Rand. Observe that the late-bad vertices are a subset of C — if vertex is not active in Central-Rand anymore, it means it has been frozen and added to C. Late-bad have another important property – every vertex v such that $y_v^{MPC} > 1$ is late-bad, as we argue in the sequel. In our next step, we upper-bound n^{late} by the number of the vertices of C that are bad.

Let C_t denote the vertices that join the vertex cover C in the t^{th} iteration of Central-Rand. From Lemma 3.11 and Lemma 3.15, a vertex is bad with probability at most $m^{-0.1}/\varepsilon$. Hence, the expected number of bad vertices in the t^{th} iteration is at most $m^{-0.1}|C_t|/\varepsilon$. Notice that C_t is a deterministic set, defined independently of MPC-Simulation. Furthermore, at the t^{th} iteration, every vertex of C_t becomes bad independently of other vertices. So, the number of bad, and also heavy-bad, vertices throughout all the phases is with high probability upper-bounded by $O(\max\{\log^2 n, m^{-0.1}|C|/\varepsilon\})$. Recall that we assume $m^{-0.1} \leq \varepsilon^2$, and also that a minimum vertex cover of the graph has size at least $\log^{10} n$. This now implies

$$|\tilde{C}| \geq \left(1 - \frac{m^{-0.1}}{\varepsilon}\right)|C| \geq (1 - \varepsilon)|C|,$$

and hence

$$n^{late} \leq \varepsilon|C| \leq \frac{\varepsilon}{1 - \varepsilon}|\tilde{C}|. \tag{7}$$

Vertices v such that $y_v^{MPC} > 1$: We say that a vertex v is *heavy-bad* if $y_v^{MPC} > 1$. The analysis we performed above on relating $|\tilde{C}|$ and W_M does not take into account heavy-bad vertices. Recall that heavy-bad vertices are removed from the graph along with their incident edges, which we did not account for while lower-bounding y_u^{MPC} for $u \in \tilde{C}$. Next, we discuss how much heavy-bad vertices affect y_u^{MPC} for any vertex $u \in V' \cap \tilde{C}$.

Observe that v is heavy-bad only if: v belongs to some set C_t; v was active in the $(t-1)^{st}$ iteration by MPC-Simulation; and, v remained active (by MPC-Simulation) throughout the t^{th} iteration. Hence, every heavy-bad vertex is also late-bad (but there can be a late-bad vertex that is not heavy-bad).

Let v be late-bad. Observe that by the time it holds $y_v^{MPC} > 1$, vertex v is already bad w.h.p. — as long as v is not bad from Lemma 3.15 we have $y_v^{MPC} \leq y_v + m^{-0.1} \leq 1 - \varepsilon + m^{-0.1} < 1$. On the other hand, from the iteration v got frozen in Central-Rand, along with its incident edges, the increase in y_v^{MPC} is accounted to $\text{diff}(\cdot,\cdot)$, which we have already analyzed. So, to account for the removal of heavy-bad vertices and their incident edges, it suffices to upper-bound the total weight of heavy-bad vertices while they were still active in Central-Rand. That weight is trivially upper-bounded by n^{late}. Furthermore, the weight n^{late} takes into account those vertices that are late-bad but not heavy-bad, so we do not consider separately such vertices (as we did for the other kind of bad vertices and for the good ones).

Finalizing: Let $\tilde{C}^{late}$ be the subset of $\tilde{C}$ consists of late-bad vertices. Our analysis shows

$$\frac{W_M + n^{late}}{|\tilde{C}| - |\tilde{C}^{late}|} \geq \frac{1-5\varepsilon}{2}.$$

For the sake of brevity, define $\alpha := (1-5\varepsilon)/2$. Using that $|\tilde{C}^{late}| \leq n^{late}$ and upper-bound Eq. (7), we derive

$$\begin{aligned} W_M &\geq \alpha|\tilde{C}| - n^{late}(1+\alpha) \\ &\geq \left(\alpha - \frac{\varepsilon}{1-\varepsilon}(1+\alpha)\right)|\tilde{C}| \end{aligned} \tag{8}$$

Next, observe that $\alpha < 1$ and hence $1+\alpha < 2$. Also, we assume $\varepsilon < 1/2$. Then, (8) further implies

$$W_M \geq (\alpha - 4\varepsilon)\,|\tilde{C}|,$$

and hence

$$|\tilde{C}| \leq \frac{2}{1-13\varepsilon} W_M \leq 2(1+50\varepsilon) W_M. \tag{9}$$

Finally, from strong duality it implies that $|\tilde{C}| \leq 2(1+50\varepsilon)W_C^\star$, where $W_C^\star$ is the minimum fractional vertex cover weight. Since the minimum integral vertex cover has size at least $W_C^\star$, the lemma follows.

Part II — Maximum Matching:

After we provided an upper-bound for $\tilde{C}$ by Eq. (9), the analysis of the weight of fractional maximum matching our algorithm MPC-Simulation designs follows almost directly. First, recall that W_M denotes the weight of the fractional matching MPC-Simulation designs. Also recall that W_M does not include the vertices v that got removed due to having $y_v^{MPC} > 1$. Therefore, by the design of the algorithm, the vertex-weights y_v^{MPC} satisfy the matching constraint, i.e. $y_v^{MPC} \leq 1$. Furthermore, from Eq. (9) and from the fact $|\tilde{C}| \geq W_C^\star$ we have

$$W_M \geq \frac{1}{2(1+50\varepsilon)}|\tilde{C}| \geq \frac{1}{2(1+50\varepsilon)} W_C^\star.$$

Now by strong duality, W_M is a $(2(1+50\varepsilon))$-approximation of fractional maximum matching.

3.4.5 Finding small matchings and vertex covers

In the proof of Section 3.4.4 we made an assumption that the maximum matching size is at least $\log^{10} n$. If the maximum matching size is less than $\log^{10} n$, in this section we show how to find a maximal matching and a 2-approximate minimum vertex cover in $O(\log \log n)$ rounds when the memory per machine is $\Theta(n)$.

First, observe that if the size of a minimum vertex cover is $O(\log^{10} n)$, then the underlying graph has $O(n \log^{10} n)$ edges – each vertex can cover at most n edges. If our graph has more than $100 \cdot n \log^{10} n$ edges we do not invoke the method described in this section. Instead, if our graph has $O(n \log^{10} n)$ edges, we apply the result of [LMSV11] to find a maximal matching of the graph in $O(\log \log n)$ MPC rounds. Namely, in [LMSV11] in the proof of Lemma 3.2 it is shown that their algorithm w.h.p. halves the number of the edges in each MPC round. Hence, after $O(\log \log n)$ the algorithm will produce some matching, and the induced graph on the unmatched vertices will have $O(n)$ edges. After that, we gather all the edges on one machine and find the remaining matching. The endpoints of this maximal matching give a 2-approximate vertex cover. We point out that it is crucial that their method outputs a maximal matching, so it is easy to turn it into a 2-approximate minimum vertex cover.

4 Integral Matching and Improved Approximation

In this section we prove the following theorem.

Theorem 1.2. *There is an algorithm that with high probability computes a $(2+\varepsilon)$-approximate maximum matching and a $(2+\varepsilon)$-approximate minimum vertex cover in $O(\log \log n)$ rounds of the MPC model, with $\tilde{O}(n)$-bits of memory per machine.*

Before we provide a proof, recall that Lemma 3.2 shows how to construct *a fractional* matching of large size. In the following lemma we show how to round that matching (i.e. to obtain an integral one), while still retaining large size of the fractional matching. This lemma is the main ingredient of the proof of Theorem 1.2.

Lemma 4.1 (Randomized rounding)**.** *Let $G = (V, E)$ be a graph. Let $x : E \to [0, 1]$ be a fractional matching of G, i.e. for each $v \in V$ it holds $\sum_{e \ni v} x_e \leq 1$. Let $C \subseteq V$ be a set of vertices such that for each $v \in C$ it holds $\sum_{e \ni v} x_e \geq 1 - \beta$, for some constant $\beta \leq 1/2$. Then, there exists an algorithm that with probability at least $2 \exp(-k)$ outputs a matching in G of size at least $k/50$.*

In our proof of this lemma we use McDiarmid's inequality that we review first.

Theorem 4.2 (McDiarmid's inequality)**.** *Suppose that $X_1, \ldots, X_k$ are independent random variables and assume that f is a function that satisfies*

$$\sup_{x_1,\ldots,x_k,t x_i} |f(x_1, \ldots, x_k) - f(x_1, \ldots, x_{i-1}, \tilde{x}_i, x_{i+1}, \ldots, x_k)| \leq c, \text{ for all } 1 \leq i \leq k.$$

(The inequality above states that if one coordinate of the function is changed, then the value of the function changes by at most c.)

Then, for any $\delta > 0$ it holds

$$P\left(|f(X_1, \ldots, X_k) - \mathbb{E}\left[f(X_1, \ldots, X_k)\right]| \geq \delta\right) \leq 2 \exp\left(-\frac{2\delta^2}{kc^2}\right).$$

Proof of Lemma 4.1. Our goal is to apply Theorem 4.2 in order to prove this lemma. So we will design a randomized process that will correspond to the setup of the theorem, but also round the fractional matching x.

Setup and the rounding algorithm: For every vertex $v \in C$ we define a random variable X_v as follows. X_v takes value from the set $\{N(v) \cup \{\star\}\}$. So, X_v is either a neighbor of v or a special symbol $\star$. Intuitively, X_v will correspond to v (randomly) choosing some neighbor, and if X_v equals $\star$, then it would mean v have not chosen any of the neighbors. The probability space for each X_v is defined as follows: for every $u \in N(v)$, we define $P(X_v = u) = x_{\{u,v\}}/10$, and $P(X_v = \star) = 1 - (\sum_{e \ni v} x_e)/10$. Observe that $P(X_v = \star) \geq 9/10$. For any two vertices $u, v \in C$, the random variables X_u and X_v are chosen independently.

Now we define function f. First, given a set of edges H we say that edge $e \in H$ is *good* if $H \setminus \{e\}$ does not contain edge incident to e. For a set of variables $\{X_v\}_{v \in C}$ we construct a set of edges H_X as follows: if $X_v \neq \star$ we add edge $\{v, X_v\}$ to H_X; otherwise X_v does not contribute to H_X. Let $C = \{v_1, \ldots, v_k\}$. Finally, we set $f(X_{v_1}, \ldots, X_{v_k})$ to be the number of good edges in H_X.

The number of good edges obtained in this random process represent our rounded matching. Next, we lower-bound the size of the integral matching obtained in this way. To that end, we derive the upper-bound on c for f as defined in Theorem 4.2 and lower-bound the expectation of f.

Upper-bound on c: Fix a vertex $v \in C$. If $X_v = \star$, then X_v does not contribute any edge to H_X. If X_v would change to some neighbor of v, then it would result in adding edge $e = \{X_v, v\}$ to H_X. But now, if there were good edges incident to v or X_v, they will not be good anymore. So, changing X_v from $\star$ to a neighbor of v could increase f by at most 2. On the other hand, if there was no edge incident to $\{X_v, v\}$ in H_X, then changing X_v in the described way would increase f by 1.

Assume now that $X_v \neq \star$. Then, similarly to the analysis above, changing $X_v = u$ to another neighbor u' of v could increase f by two at most if u initially had two incident edges while u' had none, so by changing X_v to u' there are two more good edges. In the opposite way, the number of good edges could be decreased by 2 at most. Finally, changing X_v to $\star$ could increase f by at most 2 at decrease by at most 1.

From this case analysis, we conclude $c = 2$.

Lower-bounding the expectation of f: Consider an edge $e = \{u, v\}$ incident to a vertex $v \in C$. Now we will analyze when $\{X_v = u, v\}$ is a good edge. If $X_u = \star$, and for every neighbor $w \in N(v) \cap C$ we have $X_w \neq v$, the variable $X_v = u$ will contribute 1 to f. First, $P(X_u = \star) \geq 9/10$. On the other hand

$$P(X_w \neq v \text{ for all } w \in N(v) \cap C) = \prod_{e \ni v} \left(1 - \frac{x_e}{10}\right) \geq \exp\left(-\sum_{e \ni v} \frac{x_e}{10} - \sum_{e \ni v} \frac{x_e^2}{100}\right), \tag{10}$$

where we used inequality $-\ln(1-y) \leq y + y^2$ that holds for $|y| \leq 1/2$. Now using $y^2 \leq y$ for $0 \leq y < 1$ and $\sum_{e \ni v} x_e \leq 1$, from Eq. (10) we further have

$$P(X_w \neq v \text{ for all } w \in N(v) \cap C) \geq \exp\left(-\frac{11}{100} \sum_{e \ni v} x_e\right) \geq \exp\left(-\frac{11}{100}\right) \geq \frac{89}{100},$$

where the last inequality follows from $1 - y \leq \exp(-y)$.

So, $X_v = u$ contributes 1 to f with probability at least $\frac{x_{\{v,u\}}}{10} \cdot 9/10 \cdot 89/100 \geq 4x_{\{v,u\}}/5$. Since for each vertex $v \in C$ it holds $\sum_{e \ni v} \frac{x_e}{10} \geq \frac{1-\beta}{10}$, and $\beta \leq 1/2$, from linearity of expectation we get

$$\mathbb{E}[f(X_{v_1}, \ldots, X_{v_k})] \geq 4k(1-\beta)/50 \geq k/25. \tag{11}$$

Applying Theorem 4.2: We are now ready to conclude the proof. Let $\delta = k/50$. By applying Theorem 4.2 to the function f and the random variables we defined, using that $c = 2$ and the lower-bound Eq. (11) on the expectation of f, we conclude that $f(X_{v_1}, \ldots, X_{v_k}) \geq k/50$ with probability at least $2\exp(-k)$. □

We are now ready to prove the main theorem.

Theorem 1.2. *There is an algorithm that with high probability computes a $(2+\varepsilon)$-approximate maximum matching and a $(2+\varepsilon)$-approximate minimum vertex cover in $O(\log\log n)$ rounds of the MPC model, with $\tilde{O}(n)$-bits of memory per machine.*

Proof. Invoking Lemma 3.2 for the approximation parameter $\varepsilon/50$ we obtain the desired approximation of the minimum vertex cover. To obtain a $(2+\varepsilon)$-approximate (integral) maximum matching, we alternatively apply the results of Lemma 3.2 and Lemma 4.1, as we describe in the sequel. We proceed with the proof as follows. First, we describe how to handle the case when the input graph has small matching. Second, we define an algorithm that iteratively extracts matching of a constant size from our graph. Finally, we analyze the designed algorithm – the probability of success and the number of required iteration to produce a $(2+\varepsilon)$-approximate maximum matching.

Small degree: We invoke two methods separately, each of them providing a matching, and we output the larger of them as the final result. The first method is described Section 3.4.5, and performs well when the matching size if $O(n\log^{10} n)$. Hence, from now on we assume that the maximum matching is of size at least $\log^{10} n$.

Algorithm: Now, define algorithm $\mathcal{A}$ that as input gets a graph $G = (V, E)$ and consists of the following steps:

- Invoke MPC-Simulation to obtain a fractional matching x.
- Apply the rounding method described by Lemma 4.1 on x. Let M be the produced integral matching.
- Update V by removing from it all the vertices in M.

Analysis of the algorithm: Consider one execution of $\mathcal{A}$. Let x be the fractional matching returned by MPC-Simulation for the approximation parameter set to $\varepsilon/50$, and let $W(x)$ denote its weight. By Lemma 3.2, there are at least $W(x)/3$ vertices that have fractional weight at least $1 - 5\varepsilon$ (observe that $W(x) \leq |C|$). Hence, as long as x has weight at least $\log^9 n$, the rounding method described by Lemma 4.1 w.h.p. produces an integral matching M of size at least $W(x)/150$.

Consider now multiples executions of $\mathcal{A}$. Once it holds $W(x) < \log^9 n$, it means that we have already collected a large fraction of any maximal matching, i.e. $(1 - 1/\log n)$ fraction. On the other hand, as long as $W(x) \geq \log^9 n$ algorithm $\mathcal{A}$ will produce an integral matching of size at least $1/150$ fraction of the size of the current maximum matching. This discussion motivates our final algorithm which is as follows: run $\mathcal{A}$ for $\log_{150/149}(1/\varepsilon)$ many iterations and output the union of integral matching it produces. Our discussion implies that the final returned matching is a $(2+\varepsilon)$-approximate maximum matching of the input graph. Furthermore, for constant ε, this algorithm can be implemented in $O(\log\log n)$ MPC-rounds. □

References

[ABB+17] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. *CoRR*, abs/1711.03076, 2017.

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.

[AG15] Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13–15, 2015*, pages 202–211, 2015.

[AK17] Sepehr Assadi and Sanjeev Khanna. Randomized composable coresets for matching and vertex cover. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24–26, 2017*, pages 3–12, 2017.

[ANOY14] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th ACM Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31–June 3, 2014*, pages 574–583, 2014.

[Ass17] Sepehr Assadi. Simple round compression for parallel vertex cover. *CoRR*, abs/1709.04599, September 2017.

[BEPSv3] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *Foundations of Computer Science (FOCS) 2012*, pages 321–330. IEEE, 2012, also coRR abs/1202.1983v3.

[BFARR15] Florent Becker, Antonio Fernandez Anta, Ivan Rapaport, and Eric Reémila. Brief announcement: A hierarchy of congested clique models, from broadcast to unicast. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, PODC '15, pages 167–169. ACM, 2015.

[BFS12] Guy E Blelloch, Jeremy T Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 308–317. ACM, 2012.

[BHP12] Andrew Berns, James Hegeman, and Sriram V Pemmaraju. Super-fast distributed algorithms for metric facility location. In *the Pro. of the Int'l Colloquium on Automata, Languages and Programming (ICALP)*, pages 428–439. 2012.

[BKS13] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA, June 22–27, 2013*, pages 273–284, 2013.

[CHKK+15] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 143–152. ACM, 2015.

[CHPS16] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *arXiv preprint arXiv:1608.01689*, 2016.

[CHPS17] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. In *31 International Symposium on Distributed Computing*, 2017.

[CŁM+18] Artur Czumaj, Jakub Łacki, Aleksander Madry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. *STOC*, 2018.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[DKO14] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 367–376. ACM, 2014.

[DLP12] Danny Dolev, Christoph Lenzen, and Shir Peled. "Tri, Tri again": Finding triangles and small subgraphs in a distributed setting. In *Distributed Computing*, pages 195–209. Springer, 2012.

[FN18] Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy mis. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2152–2160. SIAM, 2018.

[Gha16] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, 2016.

[Gha17] Mohsen Ghaffari. Distributed mis via all-to-all communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 141–149. ACM, 2017.

[GP16] Mohsen Ghaffari and Merav Parter. Mst in log-star rounds of congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, 2016.

[GSZ11] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the MapReduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer, 2011.

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 489–498. ACM, 2016.

[HP14] James W Hegeman and Sriram V Pemmaraju. Lessons from the congested clique applied to MapReduce. In *the Proceedings of the International Colloquium on Structural Information and Communication Complexity*, pages 149–164. Springer, 2014.

[HPP+15] James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 91–100. ACM, 2015.

[HPS14] James W Hegeman, Sriram V Pemmaraju, and Vivek B Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 514–530. Springer, 2014.

[IBY+07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Operating Systems Review*, 41(3):59–72, March 2007.

[II86] Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986.

[IS86] Amos Israeli and Yossi Shiloach. An improved parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):57–60, 1986.

[JN18] Tomasz Jurdziński and Krzysztof Nowicki. Mst in O (1) rounds of congested clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2620–2632. SIAM, 2018.

[Kor16] Janne H Korhonen. Deterministic mst sparsification in the congested clique. *arXiv preprint arXiv:1605.02022*, 2016.

[KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17–19, 2010*, pages 938–948, 2010.

[Len13] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 42–50, 2013.

[Lin87] Nathan Linial. Distributive graph algorithms global solutions from local data. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 331–335. IEEE, 1987.

[LMSV11] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2011, San Jose, CA, USA, June 4–6, 2011*, pages 85–94, 2011.

[LPPSP03] Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in $O(\log \log n)$ communication rounds. In *the Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 94–100. ACM, 2003.

[LPSR09] Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. Distributed approximate matching. *SIAM Journal on Computing*, 39(2):445–460, 2009.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[McG05] Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, pages 170–181, 2005.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. of the Symp. on Theory of Comp. (STOC)*, 2014.

[PST11] Boaz Patt-Shamir and Marat Teplitsky. The round complexity of distributed sorting. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 249–256, 2011.

[Whi12] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22*, 2010.