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

Christian Konrad
University of Bristol
christian.konrad@bristol.ac.uk

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, which improves 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 or improves on the following prior work: $O(\log^2\log n)$-round $(1+\varepsilon)$-approximation algorithm of Czumaj et al. [STOC'18] and $O(\log\log n)$-round $(1+\varepsilon)$-approximation algorithm of Assadi et al. [arXiv'17].
- 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].

CCS CONCEPTS

• **Theory of computation → Massively parallel algorithms**; **Distributed algorithms**;

KEYWORDS

Maximal Independent Set; Maximum Matching; Vertex Cover; Massively Parallel Computation; Congested Clique

ACM Reference Format:
Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. 2018. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *PODC '18: ACM Symposium on Principles of Distributed Computing, July 23–27, 2018, Egham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3212734.3212743

PODC '18, July 23–27, 2018, Egham, United Kingdom

ACM ISBN 978-1-4503-5795-1/18/07…$15.00
https://doi.org/10.1145/3212734.3212743

1 INTRODUCTION

A growing need to process massive data led to development of a number of frameworks for large-scale computation, such as MapReduce [16], Hadoop [43], Spark [44], or Dryad [28]. 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 [32] 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 [39] gives a simple randomized algorithm for constructing MIS in $O(\log n)$ PRAM rounds. When this algorithm is applied to the line graph of input graph G, it outputs a maximal matching of G, and hence a 2-approximate maximum matching of G. The output maximal matching also provides a 2-approximate minimum vertex cover. Similar results, also in the context of PRAM algorithms, were obtained in [3, 29, 30]. 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 conceptual contribution of this paper to (further) exhibit the proximity of these two models. We next review these models.

1.1.1 The MPC model. The MPC model was first introduced in [32] and later refined in [4, 9, 23]. 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 the data and each machine has S words of space, the typical settings that are of interest are when S is sublinear in N

and $S \cdot m = \Theta(N)$. 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 $S \in \Theta(n/\text{polylog}\, n)$ or $S \in \Theta(n)$.

Communication vs. computational complexity: Our main focus is the number of rounds required to finish the computation, which is essentially the complexity of the communication needed to solve the problem. Although we do not explicitly state the computational 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. A second model that we consider is the CONGESTED-CLIQUE model of distributed computing, which was introduced by Lotker, Pavlov, Patt-Shamir, and Peleg[38] and has been studied extensively since then, see e.g.,[10, 11, 13, 14, 17, 18, 20–22, 24–27, 31, 33, 35, 41, 42]. In this model, we have n players which can communicate in synchronous rounds. In each round, every player can send $O(\log n)$ bits to every 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 [36], 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. [34] 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 [2] provide a method for constructing a $(1+\varepsilon)$-approximation of weighted maximum matching in $O(1/(\delta\varepsilon))$ rounds while, similarly to [34], 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 [7] 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. [15] designed an algorithm for constructing a $(1+\varepsilon)$-approximate maximum matching in $O((\log\log n)^2)$ MPC rounds of computation and $O(n/\text{polylog}\, n)$ 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 computing $O(1)$-approximate minimum vertex cover.

Building on [15] and [7], Assadi [5] 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 $O(n/\text{polylog}\, n)$. The work by Assadi et al. [6] 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 [39] and Alon, Babai, and Itai [3] 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.[8] and consequently by Ghaffari [20], 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 [21] gave another algorithm 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. [14].

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. [12] showed that one can implement this algorithm in $O(\log^2 n)$ parallel/distributed rounds, and recently Fischer and Noever [19] 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 the CONGESTED-CLIQUE model.

1.3 Our Contributions

As our first result, in Section 3 we present an algorithm for constructing MIS.

Theorem 1.1. *There is an algorithm that with high probability 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 4, we first design an algorithm that returns a $(2+\varepsilon)$-approximate fractional maximum matching and a $(2+\varepsilon)$-approximate integral minimum vertex cover in $O(\log\log n)$ MPC rounds. In the full version of this paper we show how to round this fractional matching to a $(2+\varepsilon)$-approximate integral maximum matching. In comparison to previous work: our result has somewhat better round-complexity than [15], provides a stronger approximation guarantee than [6], and appears to be simpler than both. After applying vertex-based random partitioning (that was proposed in this context in [15]), 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 integral maximum matching and a $(2+\varepsilon)$-approximate integral minimum vertex cover in $O(\log\log n)$ rounds of the MPC model, with $\tilde{O}(n)$-bits of memory per machine.*

Following similar observations as Assadi et al. [6], it is possible to apply the techniques of [40] on Theorem 1.2 to obtain the following result.

Corollary 1.3. *There exists an algorithm that with high probability constructs a $(1+\varepsilon)$-approximate integral 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. [15], the result of Lotker et al. [37] 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 to [15], 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 $O(n/\text{polylog}\, n)$.

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: In Section 4.1, we start from a sequential algorithm that outputs a $(2+\varepsilon)$-approximate fractional maximum matching and a $(2+\varepsilon)$-approximate integral minimum 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 is frozen, and its incident edges do not change their weights afterward. The vertices that are 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 Section 4.3, we show how to 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 [15], we apply vertex-based sampling. Namely, the vertex-set is randomly partitioned across the machines into disjoint sets, and each machine considers only the induced graph on its local copy of vertices. Then, during each MPC round, every machine simulates several iterations of the sequential algorithm on its local subgraph. During this simulation, each machine estimates weights of the vertices that it maintains locally 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 their 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 $[1-4\varepsilon, 1-2\varepsilon]$. For most vertices, this prevents slight errors in estimates from having large effects. Then, vertices are frozen only if their estimated weight is above their randomly chosen threshold. Intuitively, this significantly reduces the chance of these decisions (on whether to freeze a vertex or not) deviating from the true ones

As our final component, in the full version of the paper, 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 decision. Thus it is straightforward to parallelize the rounding procedure.

2 PRELIMINARIES

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.

Independent Sets: An *independent set* $I\subseteq V$ is a subset of non-adjacent vertices. An independent set I is *maximal* if for every $v\in V\setminus I$, $I\cup\{v\}$ is not an independent set. Given an independent set I, we call the graph $G'=G[V\setminus\Gamma_G[I]]$ the residual graph with respect to I. If clear from the context, we may simple call G' the residual graph. We say that a vertex $u\in V$ is *uncovered* with respect to I, if u is not adjacent to a vertex in I, i.e., $u\in V\setminus\Gamma_G[I]$. Again, if clear from the context, we simply say u is uncovered without specifying I explicitly.

Ghaffari gave the following result that we will reuse in this paper:

Theorem 2.1 (Ghaffari [21]). *Let G be an n-vertex graph with $\Delta(G)=\text{poly}\log(n)$. Then, there exists a distributed algorithm that runs in the* CONGESTED-CLIQUE *model and computes an MIS on G in* $O(\log\log\Delta)$ *rounds.*

Routing: As a subroutine, our algorithm needs to solve the following simple routing task: Let $u\in V$ be an arbitrary vertex. Suppose that every other vertex $v\in V\setminus\{u\}$ holds $0\le n_v\le n$ messages each of size $O(\log n)$ that it wants to deliver to u. We are guaranteed that $\sum_{v\in V} n_v\le n$. Lenzen proved that in the CONGESTED-CLIQUE model there is a deterministic routing scheme that achieves this task in $O(1)$ rounds [35]. In the following, we will refer to this scheme as Lenzen's routing scheme.

3 MAXIMAL INDEPENDENT SET

The Greedy algorithm for maximal independent set processes the vertices of the input graph in arbitrary order. It adds the current vertex under consideration to an initially empty independent set I if none of its neighbors are already in I.

This algorithm progressively thins out the input graph, and the rate at which the graph loses edges depends heavily on the order in which the vertices are considered. 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. As it was observed in [1] in the context of correlation clustering in the streaming model, the number of edges in the

residual graph decreases relatively quickly with high probability. 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 with high probability 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.*

3.1 Randomized Greedy Algorithm for MIS

Let us first consider a randomized variant of the sequential greedy MIS algorithm described below, that we show how to implement in the CONGESTED-CLIQUE and the MPC model. We remark that this algorithm has been studied before in the literature of parallel algorithms[12, 19].

Greedy Randomized Maximal Independent Set:

- Initially, choose a permutation $\pi : [n] \to [n]$ uniformly at random.
- Repeat until the next rank is at least $n/\log^{10} n$ and the maximum degree is at most $\log^{10} n$:
  - (A) Mark the vertex v which has the smallest rank among the remaining vertices according to π, and add v to the MIS.
  - (B) Remove all the neighbors of v.
- Run $O(\log\log\Delta)$ rounds of the Sparsified MIS Algorithm of [21] 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.

3.2 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 3.1). Note that clearly also $\Delta' \leq \Delta$. At that point, we apply the MIS Algorithm of [21] 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 [21]). After $O(\log\log\Delta')$ rounds, w.h.p., that algorithm finds 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 round.

We note that the Algorithm of [21] 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 CONGESTED-CLIQUE: We now argue that each iteration can be implemented in $O(1)$ rounds of CONGESTED-CLIQUE. To simulate the first step of the algorithm, all vertices agree on a uniform random order as follows: the vertex with the smallest ID choses a uniform random order locally and informs all other vertices about their positions within the order. Then, all vertices broadcast their positions to all other vertices. As a result, all vertices know the entire order. Also, 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 vertex, 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[35]. Then, the leader can compute the MIS among the vertices with ranks in the selected range. It then reports the result to all the vertices in a single round, by telling each vertex whether it is in the computed independent set or not. A single round of computation, in which the vertices in the independent set report to all their neighbors, is then used to remove all the vertices that have a neighbor in the independent set (or are in the set). After these steps, the algorithm proceeds to the next iteration.

Regarding the round-complexity of the algorithm once the rank becomes $n/\log^{10} n$: The work [21] 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 3.1, that $\Delta' \leq \text{polylog}\, n \ll 2^{O(\sqrt{\log n})}$. Hence, the overall round complexity is again $O(\log\log\Delta)$ rounds.

3.3 Analysis

Since by the i-th iteration the algorithm has processed the ranks up to n/Δ^{α^i}, the rank $n/\log^{10} n$ is processed within $O(\log\log\Delta)$ iterations. In the proof of Theorem 1.1 presented below, we prove that with high probability the number of edges sent to one machine per phase is $O(n)$. Before that, we present a lemma that will aid in bounding the degrees and the number of edges in our analysis. A variant of this lemma was proved in [1].

Lemma 3.1. *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 G_r contains a vertex of degree at least d. Then, we conclude that the degree of every 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 3.2. 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 $i^\star \leq \log_{4/3} \log \Delta$, 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 3.1 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 ranks 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 at most

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

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., where we used that $\alpha = 3/4$. 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 3.1). 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 [21] 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, using Theorem 2.1, we conclude that the MIS will be computed after $O(\log \log \Delta)$ rounds in the MPC or the CONGESTED-CLIQUE model.

□

4 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 Section 4.4 we prove our main technical lemma, while the complete proof of correctness of the algorithm is deferred to the full version of this paper. Also in the full version, we design an algorithm that efficiently constructs an integral $(2 + \varepsilon)$-approximate maximum matching from a fractional one. That result along with standard techniques underlined in Section 1.3 provides $1 + \varepsilon$ approximation of maximum matching.

In Section 4.1, we first present the advertized algorithm that runs in $O(\log n)$ rounds. Then, in Section 4.2 and Section 4.3, we explain how to simulate this algorithm in $O(\log \log n)$ rounds of the MPC model.

4.1 Basic $O(\log n)$-iteration 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, until each edge is frozen, in iteration t:
  - (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 4.1. *For any constants ε such that $0 < \varepsilon \leq 1/10$, the algorithm Central terminates after $O(\log n)$ 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 that has size within a $(2 + 5\varepsilon)$ 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. 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 $\varepsilon \leq 1/10$. Note that the algorithm ensures that at every step $y_v \leq 1$. Hence, 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 $W_M^\star$ be the weight of a fractional maximum matching. Then, it holds $|M^\star| \leq W_M^\star \leq |C|$. From our analysis above and the last chain of inequalities we have $W_M \geq |M^\star|/(2(1+5\varepsilon))$. □

4.2 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 these shortcomings.

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 randomly partition the vertex-set of G', that consists only of active edges, among $m = \sqrt{d}$ machines; let G'_i be the graph given to machine i. In this way, each machine receives $O(n)$ edges w.h.p. Machine i for the next $\log_{1/(1-\varepsilon)} d/10$ rounds simulates the basic algorithm on G'_i. For that, in each round the machine which received a vertex v estimates $y_v = \sum_{e \ni v} x_e$ by $\tilde{y}_v$ defined as

$$\tilde{y}_v = m \cdot \sum_{e \ni v;\, e \in G'_i} x_e + \sum_{e \ni v;\, e \in G \setminus G'} x_e.$$

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. In each round and for every vertex v, if $\tilde{y}_v \geq 1-2\varepsilon$, then the machine freezes v and the edges incident to v. After this step, 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 be 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 4.4.

4.3 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 $\mathcal{T}_{v,t}$ 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, until each edge is frozen, in iteration t:

(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-Simul, 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-Simul 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 the full version we prove that the maximum degree in G' is at most d. Also at the beginning of a phase, the algorithm defines y_v^{old} (see line (b)). This is part of the vertex-weight that remains the same throughout the execution of the phase. It corresponds to the sum of weights of the edges incident to v that were frozen in prior phases. 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 the full version we prove that each of these induced graphs consists of $O(n)$ edges.

Each phase executes the steps under line (e), which simulates 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 last active.

Every vertex v that has weight more than 1, i.e., $y_v^{MPC} > 1$, is along with its incident edges 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. Note that after the removal of such v, the edges incident to it are not considered anymore while computing y^{MPC} or $\tilde{y}$. 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 final vertex cover.

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

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

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

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

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

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

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

(d) Partition V' into m sets $V_1, \ldots, V_m$ by assigning each vertex to a machine independently and uniformly at random.

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

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

(B) For each active edge of $G'[V_i]$, set $x_e^{MPC} \leftarrow \frac{x_e^{MPC}}{1-\varepsilon}$.

(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 $y_v^{MPC} > 1$: 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 simply 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-Simul and Central-Rand behave identically.

4.4 The Main Technical Lemma

In this section, we show that $|y_v - \tilde{y}_v|$ remains small for most of the vertices (this claim is formalized in Lemma 4.8). Before we provide an outline of the analysis, we state some definition and describe the notation we use.

Definition 4.2 (Bad and good vertex)**.** *We say that vertex is* bad *if it gets frozen in Central-Rand and not in MPC-Simul (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 4.3 (Local neighbor)**.** *If a neighbor u of vertex v is in the given iteration of MPC-Simul 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-Simul 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 $\left|y_{v,t} - \tilde{y}_{v,t}\right|$ inductively as a function of t. Suppose that we already have an upper bound on $\left|y_{v,t-1} - \tilde{y}_{v,t-1}\right|$; 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-Simul and Central-Rand start from the same fractional matching. Namely, we compare MPC-Simul 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-Simul 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 4.4. *Consider the t^{th} iteration of a phase. Let $\left|y_{v,t} - \tilde{y}_{v,t}\right| \le \sigma$ for every vertex v that is active in both Central-Rand and MPC-Simul. Then, v becomes bad in the t^{th} iteration with probability at most ε/σ and independently of other vertices.*

Proof. If $\left|\tilde{y}_{v,t} - \mathcal{T}_{v,t}\right| > \sigma$, then the algorithms MPC-Simul 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-Simul 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-Simul 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-Simul 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 4.5 (Weight-difference)**.** *We use $\mathrm{diff}(v,t)$ 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-Simul (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,*

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

Notice that $\left|y_{v,t} - y_{v,t}^{MPC}\right| \le \mathrm{diff}(v,t)$. In general it might be the case that $\left|y_{v,t} - y_{v,t}^{MPC}\right| < \mathrm{diff}(v,t)$. 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-Simul 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 $\mathrm{diff}(v,t)$.

As a first step, we state the following lemma that shows that y_v is close to y_v^{MPC} in the first iteration of some phase. Its proof is deferred to the full version.

Lemma 4.6. *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.*

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

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

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 4.7 (Evolution of weight-estimates)**.** *Let v be an active vertex in iteration $t-1$ in both Central-Rand and MPC-Simul. Then, if $\left|y_{v,t-1} - \tilde{y}_{v,t-1}\right| \le \sigma$ and $\mathrm{diff}(v,t) \le \sigma$, the following holds w.h.p.:*

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

Proof. We proceed by upper-bounding the effect of three different kinds of vertices on $\left|y_{v,t} - \tilde{y}_{v,t}\right|$: 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 4.4, 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.

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

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.

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 η defined as

$$\eta := \max\{m^{-0.2}|N_A^{central}(v,t)|, m^{1.6}\}/m.$$

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

$$\left|y_{v,t-1} - \tilde{y}_{v,t-1}\right| \leq \sigma$$

and

$$\left|y_{v,t-1} - y_{v,t-1}^{MPC}\right| \leq \sigma,$$

then we have

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

and similarly

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

as desired. □

Now, combining Lemma 4.6 and Lemma 4.7, it is not hard to show our main technical result.

Lemma 4.8. *Let v be an active vertex in iteration $t-1$ in both MPC-Simul and Central-Rand. If a phase consists of at most $I := (\log m)/(10 \log 5)$ iterations, then it holds $\left|y_{v,t} - \tilde{y}_{v,t}\right| \leq m^{-0.1}$ and $\text{diff}(v,t) \leq m^{-0.1}$ w.h.p.*

Acknowledgments

We thank anonymous reviewers for their valuable feedback. S.M. is grateful to his co-authors for the previous collaboration in [15] that was the starting point of this project. R.R. was supported by NSF award numbers CCF-1650733, CCF-1733808, CCF-1740751, IIS-1741137 and Israel Science Foundation Grant 1147/09. Most of the work on this paper has been carried out while C.K. was at the University of Warwick, where he was supported by the Centre for Discrete Mathematics and its Applications (DIMAP) and by EPSRC award EP/N011163/1. Part of this work has been carried out while S.M. was visiting MIT.

REFERENCES

[1] Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. 2015. Correlation Clustering in Data Streams. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15)*. JMLR.org, 2237–2246. http://dl.acm.org/citation.cfm?id=3045118.3045356

[2] Kook Jin Ahn and Sudipto Guha. 2015. 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*. 202–211. https://doi.org/10.1145/2755573.2755586

[3] Noga Alon, László Babai, and Alon Itai. 1986. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *Journal of Algorithms* 7, 4 (1986), 567–583. https://doi.org/10.1016/0196-6774(86)90019-2

[4] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. 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*. 574–583. https://doi.org/10.1145/2591796.2591805

[5] Sepehr Assadi. 2017. Simple Round Compression for Parallel Vertex Cover. *CoRR* abs/1709.04599 (September 2017). https://arxiv.org/abs/1709.04599

[6] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. 2017. Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs. *CoRR* abs/1711.03076 (2017). arXiv:1711.03076 http://arxiv.org/abs/1711.03076

[7] Sepehr Assadi and Sanjeev Khanna. 2017. 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*. 3–12. https://doi.org/10.1145/3087556.3087581

[8] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2012. The locality of distributed symmetry breaking. In *Foundations of Computer Science (FOCS) 2012*. IEEE, 321–330.

[9] Paul Beame, Paraschos Koutris, and Dan Suciu. 2013. 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*. 273–284. https://doi.org/10.1145/2463664.2465224

[10] Florent Becker, Antonio Fernandez Anta, Ivan Rapaport, and Eric Reémila. 2015. 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)*. ACM, 167–169.

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

[12] Guy E Blelloch, Jeremy T Fineman, and Julian Shun. 2012. 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*. ACM, 308–317.

[13] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. 2015. Algebraic Methods in the Congested Clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*. ACM, 143–152.

[14] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. 2017. Derandomizing Local Distributed Algorithms under Bandwidth Restrictions. In *31 International Symposium on Distributed Computing*.

[15] Artur Czumaj, Jakub Łacki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. 2018. Round Compression for Parallel Matching Algorithms. *STOC* (2018).

[16] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, Volume 6 (OSDI'04)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1251254.1251264

[17] Danny Dolev, Christoph Lenzen, and Shir Peled. 2012. "Tri, Tri Again": Finding Triangles and Small Subgraphs in a Distributed Setting. In *Distributed Computing*. Springer, 195–209.

[18] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. 2014. On the Power of the Congested Clique Model. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*. ACM, 367–376.

[19] Manuela Fischer and Andreas Noever. 2018. Tight Analysis of Parallel Randomized Greedy MIS. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2152–2160.

[20] Mohsen Ghaffari. 2016. An Improved Distributed Algorithm for Maximal Independent Set. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*.

[21] Mohsen Ghaffari. 2017. Distributed MIS via All-to-All Communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 141–149.

[22] Mohsen Ghaffari and Merav Parter. 2016. MST in Log-Star Rounds of Congested Clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*.

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

[24] James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. 2015. 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)*. ACM, 91–100.

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

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

[27] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2016. 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*. ACM, 489–498.

[28] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. *SIGOPS Operating Systems Review* 41, 3 (March 2007), 59–72. https://doi.org/10.1145/1272998.1273005

[29] Amos Israeli and Alon Itai. 1986. A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Inform. Process. Lett.* 22, 2 (1986), 77–80. https://doi.org/10.1016/0020-0190(86)90144-4

[30] Amos Israeli and Yossi Shiloach. 1986. An Improved Parallel Algorithm for Maximal Matching. *Inform. Process. Lett.* 22, 2 (1986), 57–60. https://doi.org/10.1016/0020-0190(86)90141-9

[31] Tomasz Jurdziński and Krzysztof Nowicki. 2018. MST in O (1) Rounds of Congested Clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2620–2632.

[32] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. 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*. 938–948. https://doi.org/10.1137/1.9781611973075.76

[33] Janne H Korhonen. 2016. Deterministic MST Sparsification in the Congested Clique. *arXiv preprint arXiv:1605.02022* (2016).

[34] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. 2011. 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*. 85–94. https://doi.org/10.1145/1989493.1989505

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

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

[37] Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. 2009. Distributed approximate matching. *SIAM J. Comput.* 39, 2 (2009), 445–460.

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

[39] Michael Luby. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM J. Comput.* 15, 4 (1986), 1036–1053. https://doi.org/10.1137/0215074

[40] Andrew McGregor. 2005. 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*. 170–181. https://doi.org/10.1007/11538462_15

[41] Danupon Nanongkai. 2014. Distributed Approximation Algorithms for Weighted Shortest Paths. In *Proc. of the Symp. on Theory of Comp. (STOC)*.

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

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

[44] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22*. https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets