Sample-And-Gather: Fast Ruling Set Algorithms in the Low-Memory MPC Model

Kishore Kothapalli
IIIT Hyderabad, India
kkishore@iiit.ac.in

Shreyas Pai
The University of Iowa, Iowa City, IA, USA
shreyas-pai@uiowa.edu

Sriram V. Pemmaraju
The University of Iowa, Iowa City, IA, USA
sriram-pemmaraju@uiowa.edu

Abstract

Motivated by recent progress on symmetry breaking problems such as maximal independent set (MIS) and maximal matching in the low-memory Massively Parallel Computation (MPC) model (e.g., Behnezhad et al. PODC 2019; Ghaffari-Uitto SODA 2019), we investigate the complexity of ruling set problems in this model. The MPC model has become very popular as a model for large-scale distributed computing and it comes with the constraint that the memory-per-machine is strongly sublinear in the input size. For graph problems, extremely fast MPC algorithms have been designed assuming $\tilde{\Omega}(n)$ memory-per-machine, where n is the number of nodes in the graph (e.g., the $O(\log \log n)$ MIS algorithm of Ghaffari et al., PODC 2018). However, it has proven much more difficult to design fast MPC algorithms for graph problems in the *low-memory* MPC model, where the memory-per-machine is restricted to being strongly sublinear in the number of nodes, i.e., $O(n^{\varepsilon})$ for constant $0 < \varepsilon < 1$.

In this paper, we present an algorithm for the 2-ruling set problem, running in $\tilde{O}(\log^{1/6} \Delta)$ rounds whp, in the low-memory MPC model. Here Δ is the maximum degree of the graph. We then extend this result to β-ruling sets for any integer $\beta > 1$. Specifically, we show that a β-ruling set can be computed in the low-memory MPC model with $O(n^{\varepsilon})$ memory-per-machine in $\tilde{O}(\beta \cdot \log^{1/(2^{\beta+1}-2)} \Delta)$ rounds, whp. From this it immediately follows that a β-ruling set for $\beta = \Omega(\log \log \log \Delta)$-ruling set can be computed in in just $O(\beta \log \log n)$ rounds whp. The above results assume a total memory of $\tilde{O}(m + n^{1+\varepsilon})$. We also present algorithms for β-ruling sets in the low-memory MPC model assuming that the total memory over all machines is restricted to $\tilde{O}(m)$. For $\beta > 1$, these algorithms are all substantially faster than the Ghaffari-Uitto $\tilde{O}(\sqrt{\log \Delta})$-round MIS algorithm in the low-memory MPC model.

All our results follow from a *Sample-and-Gather Simulation Theorem* that shows how random-sampling-based CONGEST algorithms can be efficiently simulated in the low-memory MPC model. We expect this simulation theorem to be of independent interest beyond the ruling set algorithms derived here.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Massively Parallel Computation, Ruling Set, Simulation Theorems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2020.28

Related Version A full version of this paper is available at: `https://arxiv.org/abs/2009.12477`.

Funding This work is funded in part by the Department of Science and Technology, Government of India, via Grant number MTR/2017/000849.

1 Introduction

There has been considerable recent progress in the design and study of large-scale distributed computing models that are closer to reality, yet mathematically tractable. Of these, the *Massively Parallel Computing (MPC)* model [24, 37] has gained significant attention due to its flexibility and its ability to closely model existing distributed computing frameworks used in practice such as MapReduce [14], Spark [38], Pregel [32], and Giraph [11].

The MPC model is defined by a set of machines, each having at most S words of memory. The machines are connected to each other via an all-to-all communication network. Communication and computation in this model are synchronous. In each round, each machine receives up to S words from other machines, performs local computation, and sends up to S words to other machines. The key characteristic of the MPC model is that both the memory upper bound S and the number of machines used are assumed to be strongly sublinear in the input size N, i.e., bounded by $O(N^{1-\varepsilon})$ for some constant ε, $0 < \varepsilon < 1$. This characteristic models the fact that in modern large-scale computational problems the input is too large to fit in a single machine and is much larger than the number of available machines.

Even though the MPC model is relatively new, a wide variety of classical graph problems have been studied in this model. This stream of research includes the design of fast algorithms [4, 6, 13, 12, 21] as well as lower bound constructions [10, 20, 35]. A particular, though not exclusive, focus of this research has been on *symmetry breaking* problems such as maximal independent set (MIS) [6, 21, 18], maximal matching [7], and $(\Delta+1)$-coloring [9, 3], along with related graph optimization problems such as minimum vertex cover and maximum matching.

For graph problems, the input size is $\tilde{O}(m+n)$ where m is the number of edges and n is the number of nodes of the input graph. Thus, $O((m+n)^{1-\varepsilon})$, for some constant ε, $0 < \varepsilon < 1$, is an upper bound on both the number of machines that can be used and the size S of memory per machine. It turns out that the difficulty of graph problems varies significantly based on how S relates to the number of nodes (n) of the input graph. Specifically, three regimes for S have been considered in the literature.

- **Strongly superlinear memory ($S = O(n^{1+\varepsilon})$):** For this regime to make sense in the MPC model, the input graph needs to be highly dense, i.e., $m \gg S \gg n$ such that S is strongly sublinear in m. Even though the input graph is dense, the fact that each machine has $O(n^{1+\varepsilon})$ local memory makes this model quite powerful. For example, in this model, problems such as minimum spanning tree, MIS, and 2-approximate minimum vertex cover, all have $O(1)$-round algorithms [24, 22].
- **Near-linear memory ($S = \tilde{O}(n)$):** Problems become harder in this regime, but symmetry breaking problems such as MIS, approximate minimum vertex cover, and maximal matching can still be solved in $O(\log\log n)$ rounds [13, 2, 17, 19]. Furthermore, recently Assadi, Chen, and Khanna [3] presented an $O(1)$-round algorithm for $(\Delta+1)$-vertex coloring.
- **Strongly sublinear memory ($S = O(n^{\varepsilon})$):** Problems seem to get much harder in this regime and whether there are sublogarithmic-round algorithms for certain graph problems in this regime is an important research direction. For example, it is conjectured that the problem of distinguishing if the input graph is a single cycle vs two disjoint cycles of length $n/2$ requires $\Omega(\log n)$ rounds [37, 20]. However, even in this regime, Ghaffari and Uitto [21] have recently shown that MIS does have a sublogarithmic-round algorithm, running in $\tilde{O}(\sqrt{\log\Delta})$ rounds, where Δ is the maximum degree of the input graph. This particular result serves as a launching point for the results in this paper.

The MIS problem has been called "a central problem in the area of locality in distributed computing" (2016 Dijkstra award citation). Starting with the elegant, randomized MIS algorithms from the mid-1980s by Luby [31] and by Alon et al. [1], several decades of research has now been devoted to designing MIS algorithms in various models of parallel and distributed computing (e.g., PRAM, LOCAL, CONGEST, CONGESTED-CLIQUE, and MPC). A *ruling set* is a natural relaxation of an MIS and considerable research has been devoted to solving the ruling set problem in different models of distributed computation as well [5, 26, 8, 15]. An *(α, β)-ruling set* of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that (i) every pair of nodes in S are at distance at least α from each other and (ii) every node in V is at distance at most β from some node in S. An MIS is just a $(2, 1)$-ruling set. Research on the ruling set problem has focused on the question of how much faster distributed ruling set algorithms can be relative to MIS algorithms and whether there is a provable separation in the distributed complexity of these problems in different models of distributed computing. For example, in the LOCAL model[1], Kuhn, Moscibroda, and Wattenhofer [27, 28] show an $\Omega\left(\min\left\{\frac{\log\Delta}{\log\log\Delta}, \sqrt{\frac{\log n}{\log\log n}}\right\}\right)$ lower bound for MIS, even for randomized algorithms. However, combining the recursive sparsification procedure of Bisht et al. [8] with the improved MIS algorithm of Ghaffari [15] and the recent deterministic network decomposition algorithm of Rozhon and Ghaffari [36], it is possible to compute β-ruling sets in $O(\beta \log^{1/\beta}\Delta + \text{polyloglog}(n))$ rounds, thus establishing a separation between these problems, even for $\beta = 2$, in the LOCAL model. In this paper, we are interested only in $(2, \beta)$-ruling sets and so as a short hand, we drop the first parameter "2" and call these objects β-ruling sets. Also as a short hand, we will use *low-memory MPC model* to refer to the *strongly sublinear memory MPC model.* As mentioned earlier, Ghaffari and Uitto [21] recently presented an algorithm that solves MIS in the low-memory MPC model in $\tilde{O}(\sqrt{\log\Delta})$ rounds. However, nothing more is known about the 2-ruling set problem in this model and the fastest 2-ruling set algorithm in the low-memory MPC model is just the above-mentioned MIS algorithm. This is in contrast to the situation in the linear-memory MPC model. In this model, the fastest algorithm for solving MIS runs in $O(\log\log n)$ rounds [17], whereas the fastest 2-ruling set algorithm runs in $O(\log\log\log n)$ rounds [23]. This distinction between the status of MIS and 2-ruling sets in the linear-memory MPC model prompts the following related questions.

> *Is it possible to design an $o(\sqrt{\log\Delta})$-round, 2-ruling set algorithm in the low-memory MPC model? Could we in fact design 2-ruling set algorithms in the low-memory MPC model that run in $O(\text{polyloglog}(n))$ rounds?*

1.1 Main Results

We make progress on the above question via the following results proved in this paper.

1. We show (in Theorem 19 part (i)) that a 2-ruling set of a graph G can be computed in $\tilde{O}(\log^{1/6}\Delta)$ rounds in the low-memory MPC model. We generalize this result to β-ruling sets, for $\beta \geq 2$ (in Theorem 23 part (i)), and show that a β-ruling set of a graph G can be computed in $\tilde{O}(\log^{1/(2^{\beta+1}-2)}\Delta)$ rounds in the low-memory MPC model. These algorithms are substantially faster than the MIS algorithm [21] for the low-memory MPC model. The inverse exponential dependency on β in the running time of the β-ruling set algorithm is

[1] The LOCAL model is a synchronous, message passing model of distributed computation [29, 34] with unbounded messages. See Section 1.2 for definitions of related models of computation.

worth noting. This dependency implies that for any $\beta = \Omega(\log\log\log\Delta)$, we can compute a β-ruling set in only $O(\beta\cdot\text{polyloglog}(n))$ rounds. This is in contrast to the situation in the Local model; using the $O(\beta\cdot\log^{1/\beta}\Delta + \text{polyloglog}(n))$-round β-ruling set algorithm in the Local model mentioned earlier, one can obtain an $O(\text{polyloglog}(n))$-round algorithm only for $\beta = \Omega(\log\log\Delta)$.

2. Even though the above-mentioned results are in the low-memory MPC model, they assume no restrictions on the total memory used by all the machines put together. Specifically, we obtain the above results allowing a total of $\tilde{O}(m + n^{1+\varepsilon})$ memory. Note that the input uses $\tilde{O}(m)$ memory and thus these algorithms make use of $\tilde{O}(n^{1+\varepsilon})$ extra total memory. If we place the restriction that the total memory cannot exceed the input size, i.e., $\tilde{O}(m)$, then we get slightly weaker results. Specifically, we show (in Theorem 19 part (ii)) that a 2-ruling set can be computed in $\tilde{O}(\log^{1/4}\Delta)$ rounds in the low-memory MPC model using $\tilde{O}(m)$ total memory. Additionally, we show (in Theorem 23 part (ii)) that a β-ruling set, for any $\beta \geq 1$, can be computed in $\tilde{O}(\log^{1/2\beta}\Delta)$ rounds in the low-memory MPC model using $\tilde{O}(m)$ total memory. Note that even though these results are weaker than those we obtain in the setting where total memory is unrestricted, for $\beta > 1$, these algorithms are much faster than the $\tilde{O}(\sqrt{\log\Delta})$-round, low-memory MPC model algorithm for MIS by Ghaffari and Uitto [21] that uses $\tilde{O}(m)$ total memory. Also note that by plugging in $\beta = 1$, we recover the Ghaffari-Uitto MIS algorithm.

Technical Contributions. We obtain all of these results by applying new Simulation Theorems (Theorems 9 and 12) that we develop and prove. These Simulation Theorems provide a general method for deriving fast MPC algorithms from known distributed algorithms in the Congest model[2] and they form the main technical contribution of this paper.

A well-known technique [16, 21, 23, 33] for designing fast algorithms in "all-to-all" communication models such as MPC is the following "ball doubling" technique. Informally speaking, if every node v knows the state of the k-neighborhood around v, then by exchanging this information with all nodes, ideally in $O(1)$ rounds, it is possible to learn the state of the $2k$-neighborhood around each node. In this manner, nodes can learn the state their ℓ-neighborhood in $O(\log\ell)$ rounds. Then it is possible to simply use local computation at each node to "fast forward" the algorithm by ℓ rounds, without any further communication. In this manner, a phase consisting of ℓ rounds in the Congest model can be compressed into $O(\log\ell)$ rounds in the MPC model. This description of the "ball doubling" technique completely ignores the main obstacle to using this technique: the k-neighborhoods around nodes may be so large that bandwidth constraints of the communication network may disallow rapid exchange of these k-neighborhoods.

Our main contribution is to note that in many randomized, distributed algorithms in the Congest model, there is a natural *sparsification* that occurs, i.e., in each round a randomly sampled subset of the nodes are active, and the rest are silent. This implies that the k-neighborhoods that are exchanged only need to involve sparse subgraphs induced by the sampled nodes. A technical challenge we need to overcome is that the subgraph induced by sampled nodes is not just from the next round, but from the ℓ future rounds; so we need to be able to estimate which nodes will be sampled in the future. On the basis of this idea, we introduce the notion of *α-sparsity* of a randomized Congest algorithm, for a parameter α; basically smaller the α greater the sparsification induced by random sampling. We present

[2] The Congest model [34] is similar to the Local model except that in the Congest model there is an $O(\log n)$ bound on the size of each message.

Sample-and-Gather Simulation Theorems in which, roughly speaking, an R-round CONGEST algorithm is simulated in $\tilde{O}(R/\sqrt{\log_\alpha n})$ rounds (respectively, $\tilde{O}(R/\sqrt{\log_\alpha \Delta})$ rounds) in the low-memory MPC model, where the total memory is $\tilde{O}(m + n^{1+\varepsilon})$ (respectively, $\tilde{O}(m)$).

Our Simulations Theorems are inspired by a Simulation Theorem due to Behnehzhad et al. [6, Lemma 5.5]. Using their Simulation Theorem, an R-round state-congested algorithm can be simulated in (roughly) $R/\log_\Delta n$ low-memory MPC rounds. In contrast, our Simulation Theorem (Theorem 9) yields a running time of (roughly) $R/\sqrt{\log_\alpha n}$, where α is a sparsity parameter. When the input graph has high maximum degree, but the state-congested algorithm samples a very sparse subgraph (i.e., α is small) then our Simulation Theorems provide a huge advantage over the Behnehzhad et al. Simulation Theorems.

To obtain our results for ruling sets, we apply the Sample-and-Gather Simulation Theorems to the sparsification procedure of Kothapalli and Pemmaraju [26] and Bisht et al. [8] and to the sparsified MIS algorithm of Ghaffari [16]. We note that by applying the Sample-and-Gather Simulation Theorems to the sparsified MIS algorithm of Ghaffari [16], we recover the Ghaffari-Uitto low-memory MPC algorithm for MIS [21], built from scratch. We believe that the Sample-and-Gather Theorems will be of independent interest because they simplify the design of fast MPC algorithms.

1.2 Technical Preliminaries

Notation. For a node $v \in V$ we denote its non-inclusive neighborhood in G by $\mathrm{Nbr}(v)$. Moreover, we define $\mathrm{Nbr}^+(v) = \mathrm{Nbr}(v) \cup \{v\}$, $\mathrm{Nbr}(S) = \bigcup_{v\in S} \mathrm{Nbr}(v)$, $\mathrm{Nbr}^+(S) = \bigcup_{v\in S} \mathrm{Nbr}^+(v)$. The standard usage of the $\tilde{O}(f(n))$ notation is to denote $O(\mathrm{poly}\log(f(n)) \cdot f(n))$. But, because our round and memory complexity bounds involve multiple parameters (e.g., n, m, and Δ), we abuse notation and use the $\tilde{O}(\cdot)$ notation to hide $\mathrm{poly}\log n$ or $\mathrm{poly}\log\log n$ factors, as appropriate (e.g., $\tilde{O}(\log^{1/6}\Delta)$ denotes $O(\log^{1/6}\Delta \cdot \mathrm{poly}\log\log n)$). Moreover, we consider ε to be a constant in $(0, 1)$ and hence, we don't explicit mention ε dependency in the run time results. However the dependency on epsilon is of the form $1/\varepsilon^c$ for some small constant $c \geq 1$ (and not 2^{-eps}).

Distributed Computing Models. In the CONGEST model [34] a communication network is abstracted as an n-node graph. In synchronous rounds each node can send a $O(\log n)$ bit message to each of its neighbors. The CONGESTED-CLIQUE model is similar to the CONGEST model, but nodes can send $O(\log n)$-bits messages to all other nodes, not only to its neighbors in the input graph G [30]. The LOCAL model [29] is the same as the CONGEST model, except the message sizes can be unbounded.

MPC model. Typically, in the MPC model, it is assumed that the input graph is distributed in a node-centric manner among the machines. In other words, for each node v, there is a machine M_v that hosts it and M_v knows all the neighbors of v and the machines that host these neighbors. However, this scheme cannot be implemented as-is in the low-memory MPC model because the degree of a node could be larger than the memory volume n^ε of a machine. To deal with this issue, we first assume that a node v with $\deg(v) > n^\varepsilon$ is split into copies that are distributed among different machines and we have a virtual $O(1/\varepsilon)$-depth balanced tree on these copies of v. The root of this tree coordinates communication between v and its neighbors in the input graph. By itself, this is insufficient because information from v's neighbors cannot travel up v's tree without running into a memory bottleneck. However, if computation at each node can be described by a *separable* function, then this is possible.

The following definition of separable functions captures functions such as max, min, sum, etc. This issue and the proposed solution have been discussed in [21, 6].

▶ **Definition 1.** *Let $f : 2^{\mathbb{R}} \to \mathbb{R}$ denote a set function. We call f* separable *iff for any set of reals A and for any $B \subseteq A$, we have $f(A) = f\big(f(B), f(A \setminus B)\big)$.*

The following lemma [6] shows that it is possible to compute the value of a separable function f on each of the nodes in merely $O(1/\varepsilon)$ rounds. The bigger implication of this lemma is that a single round of a Congest algorithm can be simulated in $O(1/\varepsilon)$ low-memory MPC rounds.

▶ **Lemma 2** ([6])**.** *Suppose that on each node $v \in V$, we have a number x_v of size $O(\log n)$ bits and let f be a separable function. There exists an algorithm that in $O(1/\varepsilon)$ rounds of MPC, for every node v, computes $f(\{x_u \mid u \in \text{Nbr}(v)\})$ whp in the low-memory MPC model with $\tilde{O}(m)$ total memory.*

Note about proofs. Due to space constraints, we only include two proofs of the main Sample-and-Gather Simulation Theorem in the paper; a full version of the paper, with all proofs, is available at `https://arxiv.org/abs/2009.12477`, [25].

2 The Sample-and-Gather Simulation

Our simulation theorems apply to a subclass of Congest model algorithms called *state-congested* algorithms [6].

▶ **Definition 3.** *An algorithm in the Congest model is said to be state-congested if*

(i) *by the end of round r, for any r, at each node v, the algorithm stores a state $\sigma_r(v)$ of size $O(\deg(v) polylog(n))$ bits, i.e., an average of $O(polylog(n))$ bits per neighbor. The initial state $\sigma_0(v)$ of each node v is its `ID`. Furthermore, the computation performed by each node v in each round r uses an additional temporary space of size $O(\deg(v) \cdot polylog(n))$ bits.*

(ii) *The states of the nodes after the last round of the algorithm are sufficient in determining, collectively, the output of the algorithm.*

A key feature of a state-congested algorithm is that the local state at each node stays bounded in size throughout the execution of the algorithm.

We inductively design a fast low-memory MPC algorithm that simulates a given state-congested algorithm. For this purpose, we start by assuming that we have a state-congested, possibly randomized, algorithm *Alg*, whose first t rounds have been correctly simulated in the low-memory MPC model. Our goal now is to simulate a *phase* consisting of the next ℓ rounds of *Alg*, i.e., rounds $t+1, t+2, \ldots, t+\ell$, in just $O(\log \ell)$ low-memory MPC rounds. We categorize each node u in a round τ, $t+1 \leq \tau \leq t+\ell$, based on its activity in round τ. Specifically, a node u is a *sending node* in round τ if sends at least one message in round τ. Moreover, a node is called a *oblivious node* if it does not update its state in round τ. In other words, an oblivious node ignores any messages it receives in round τ.

Consider a node u at the start of the phase we want to compress. Since this is immediately after round t, node u knows its local state $\sigma_t(u)$. Let $p_{t+1}(u)$ denote the probability that node u is a sending node in round $t+1$. We call this the *activation probability* of node u in round $t+1$. Also, for any node v, let $A_{t+1}(v) := \sum_{u \in Nbr(v)} p_{t+1}(u)$ denote the *activity level* in v's neighborhood in round $t+1$. Note that $p_{t+1}(u)$ is completely determined by $\sigma_t(u)$ and so node u can locally calculate $p_{t+1}(u)$ after round t. In order to simulate rounds

$t+1, t+2, \ldots, t+\ell$ in a compressed fashion in the MPC model, every node u needs to know the probability of it being a sending node in each of these rounds. But, rounds $t+2, t+3, \ldots, t+\ell$ are in the future and so node u, using current knowledge, can only *estimate* an upper bound $\tilde{p}_\tau(u)$ on the probability that it will be a sending node in round τ, $t+2 \leq \tau \leq t+\ell$.

In general, doing this estimation can be difficult because sampling probabilities of a node u in the ℓ future rounds depend on current states of nodes in an ℓ-radius neighborhood around node u. The volume of such a neighborhood may be too high to fit in the memory of any machine in the low-memory MPC model. In fact, our Sample-and-Gather Simulation Theorems are designed to avoid exactly this type of ball gathering of potentially dense neighborhoods. It turns out that this estimation is essentially trivial for the two applications of our Simulation Theorem described in Section 3.1. This is because for these algorithms, sampling probabilities for all active nodes increase by a known multiplicative factor in each round. Thus independent of a node u's future state (e.g., whether it is active), it is possible to obtain an upper bound, denoted $\tilde{p}_\tau(u)$, that node u will be a sending node in round τ, for rounds $\tau = t+2, t+3, \ldots, t+\ell$. For round $\tau = t+1$, we simply set $\tilde{p}_{t+1}(u) := p_{t+1}(u)$, i.e., the estimated activation probability in round $t+1$ is the actual activation probability.

For any τ, $t+1 \leq \tau \leq t+\ell$, for any node v, let $\tilde{A}_\tau(v) := \sum_{u \in Nbr(v)} \tilde{p}_\tau(u)$ denote the *estimated activity level* in node v's neighborhood in round τ. Note that for the first round in the phase, $\tau = t+1$, the estimated and actual activity levels are identical. Finally, let $\tilde{A}_\tau$ be the maximum $\tilde{A}_\tau(v)$, where the maximum is taken over all nodes v that are not oblivious nodes. The maximum being taken over all non-oblivious nodes is motivated by the fact that if a node is oblivious, it does not update its state and therefore the activity level in its neighborhood is not relevant.

▶ **Lemma 4.** *Suppose ℓ is such that*

$$\left(\sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n\right)^\ell \leq O(n^{\varepsilon/2}). \tag{1}$$

Then the next phase of the algorithm Alg consisting of rounds $t+1, t+2, \ldots, t+\ell$ can be simulated in $O(\log \ell)$ rounds in the low-memory MPC model with $\tilde{O}(m + n^{1+\varepsilon})$ total memory.

Proof. Simulating rounds $t+1, t+2, \ldots, t+\ell$ of algorithm *Alg* is equivalent to computing the state $\sigma_{t+\ell}(v)$ for every node $v \in V$. We use the 2-step algorithm below to do this computation. First, we introduce some notation. Let $B_G(v, \ell)$ denote the labeled subgraph of G, induced by nodes that are at most ℓ hops from v in G and in which each node u is labeled with its local state $\sigma_t(u)$ after round t.

Step 1: For each node $v \in V$, designate a distinct machine M_v at which we gather a "sampled" subgraph $S_G(v, \ell)$ of $B_G(v, \ell)$. The definition of $S_G(v, \ell)$ is provided below.

Step 2: Using the subgraph $S_G(v, \ell)$, machine M_v locally simulates rounds $t+1, t+2, \ldots, t+\ell$ of *Alg* and computes $\sigma_{t+\ell}(v)$.

In the rest of the proof, we will first define the subgraph $S_G(v, \ell)$. We will then show in Claim 5 that using this subgraph, it is possible for machine M_v to locally simulate rounds $t+1, t+2, \ldots, t+\ell$ of *Alg*. We then show in Claim 6 that assuming ℓ satisfies (1), the size of $S_G(v, \ell)$ is $O(n^\varepsilon)$ whp. Finally, in Claim 7, we show that the subgraph $S_G(v, \ell)$ can be gathered at each machine M_v in parallel in $O(\log \ell)$ rounds. These claims together complete the proof of the lemma.

Each node $u \in V$ generates a sequence of uniformly distributed random bits $r_\tau^1(u)$, $r_\tau^2(u)$, …, $r_\tau^{c \cdot \log n}(u)$ for a large enough constant c. These bits are designated for round τ, $t+1 \leq \tau \leq t+\ell$ and they serve two purposes: (i) they are used to randomly sample u based on the estimate $\tilde{p}_\tau(u)$ that u will be a sending node in round τ, and (ii) they are used to simulate u's actions in round τ. It is important that the same bits be used for both purposes so that there is consistency in u's random actions. Specifically, u constructs a real number $R_\tau(u)$ that is uniformly distributed over $\{i/2^{c \log n} \mid 0 \leq i < c \log n\}$ using these bits. Node u adds these $O(\ell \cdot \log n)$ bits to its local state after round t, $\sigma_t(u)$. Node u then *marks itself for round* τ if $R_\tau(u) \leq p_\tau(u)$. If a node u marks itself for a round τ it means that in u's estimate after round t, u will be a sending node in round τ. Further, node u is *marked* if it is marked for round τ for any τ, $t+1 \leq \tau \leq t+\ell$. The "sampled" subgraph $S_G(v,\ell)$ is the subgraph of $B_G(v,\ell)$ induced by v along with all nodes u in $B_G(v,\ell)$ that are marked.

▷ Claim 5. For any node $v \in V$, information in $S_G(v,\ell)$ is enough to locally compute $\sigma_{t+\ell}(v)$.

Proof. We prove this claim inductively. Specifically, we prove the following:

For any i, $0 < i \leq \ell$, in addition to knowing $S_G(v,\ell)$, if we know the states $\sigma_{t+\ell-i}(u)$ for all $u \in S_G(v,i)$ then we can compute the states $\sigma_{t+\ell-i+1}(u)$ for all $u \in S_G(v,i-1)$.

The premise of this statement is true for $i = \ell$ because $S_G(v,\ell)$ contains the round-t local states $\sigma_t(u)$ for all $u \in S_G(v,\ell)$. For $i = 1$ this claim is equivalent to saying that in addition to $S_G(v,\ell)$, if we know $\sigma_{t+\ell-1}(u)$ for all neighbors of v in $S_G(v,\ell)$ then we can compute $\sigma_{t+\ell}(v)$. This is what we need to show.

To be able to compute $\sigma_{t+\ell-i+1}(u)$ for any u in $S_G(v,i-1)$, we need to know the round-$(t+\ell-i)$ local states $\sigma_{t+\ell-i}(w)$ for all neighbors w of u that are sending nodes in round $t+\ell-i$. With high probability, the probability p_w that a neighbor w of u sends messages in round $t+\ell-i$ is upper bounded by the estimate $\tilde{p}_{t+\ell-i}(w)$ that w computed after round t. Node w sends messages in round $t+\ell-i$ if $R_{t+\ell-i}(w) \leq p_w$. Since $p_w \leq \tilde{p}_{t+\ell-i}(w)$, we know that $R_{t+\ell-i}(w) \leq \tilde{p}_{t+\ell-i}(w)$ and therefore w is marked and included in $S_G(v,\ell)$. Also note that since $u \in S_G(v,i-1)$ and w is a neighbor of u, we see that $w \in S_G(v,i)$. Thus any node w that sends a message to node u in round $t+\ell-i$ belongs to $S_G(v,i)$ and by the hypothesis of the inductive claim we know $\sigma_{t+\ell-i}(w)$. With the knowledge of $\sigma_{t+\ell-i}(w)$, we can simulate round $t+\ell-i+1$ at each node w, using the random real $R_{t+\ell-i+1}(w)$ to execute any random actions w may take. Then using the message received by u from all such neighbors w in round $t+\ell-i+1$, we can update u's local state, thus computing $\sigma_{t+\ell-i+1}(u)$. ◁

▷ Claim 6. For any node $v \in V$, the size of $S_G(v,\ell)$ is at most $\left(\sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n\right)^\ell$ whp.

Proof. Consider an arbitrary $v \in V$ and $u \in B_G(v,\ell)$ and a round $t+1 \leq \tau \leq t+\ell$. Node u is marked for round τ with probability $p_\tau(u)$. Recalling that $Nbr(u)$ denotes the set of neighbors of u in G, we see that expected number of neighbors of u marked for round $t+1 \leq \tau \leq t+\ell$ is at most

$$\sum_{w \in Nbr(u)} p_\tau(w) \leq \tilde{A}_\tau(u) \leq \tilde{A}_\tau.$$

Furthermore, since neighbors of u are marked for round τ independently, by Chernoff bounds we see that the number of neighbors that u has in $S_G(v,\ell)$ that are marked for round τ is $\tilde{A}_\tau \log n$ whp. By the union bound this means that the number of neighbors that u has in $S_G(v,\ell)$ is $\sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n$ whp. From this it follows that the size of $S_G(v,\ell)$ is $\left(\sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n\right)^\ell$. ◁

▷ Claim 7. For every node $v \in V$, the graph $S_G(v, \ell)$ can be gathered at M_v in at most $O(\log \ell)$ rounds.

Proof. Here we use the "ball doubling" technique that appears in a number of papers on algorithms in "all-to-all" communication models (e.g., [16, 21, 23, 33]). Suppose that each machine M_v knows $S_G(v, i)$ for some $0 \leq i \leq \ell/2$. Each machine M_v then sends $S_G(v, i)$ to machine M_u for every node u in $S_G(v, i)$. After this communication is completed, each machine M_v can construct $S_G(v, 2i)$ from the information it has received because $S_G(v, 2i)$ is contained in the union of $S_G(u, i)$ for all u in $S_G(v, i)$.

We now argue that this communication can be performed in $O(1)$ rounds. First, note that the size of $S_G(v, i)$ is bounded above by $O(n^{\varepsilon/2})$. This also means that $S_G(v, i)$ contains $O(n^{\varepsilon/2})$ nodes. Therefore, M_v needs to send a total of $O(n^{\varepsilon/2}) \times O(n^{\varepsilon/2}) = O(n^{\varepsilon})$ words. A symmetric argument shows an $O(n^{\varepsilon})$ bound on the number of words M_v receives. Since $O(n^{\varepsilon})$ words can be sent and received in each communication round, this communication can be completed in $O(1)$ rounds. ◁

With the claims proven, we finish the proof of the Lemma. ◀

The biggest benefit from using this "sample-and-gather" simulation approach is for state-congested algorithms that sample a sparse subgraph and all activity occurs on this subgraph. We formalize this sparse sampling property as follows.

▶ **Definition 8.** *Consider a state-congested algorithm Alg that completes in R rounds. For a parameter $\alpha \geq 2$, we say that Alg is α-sparse if for all positive integers, t and ℓ satisfying $t + \ell \leq R$, for a length-ℓ phase of Alg starting at round $t + 1$ the following two properties hold.*

(a) ***Bounded activity level:*** *The activity level in the first round of the phase, A_{t+1}, satisfies the property: $A_{t+1} = O(\alpha^{\ell} \cdot \log n)$.*

(b) ***Bounded growth of estimated activity level:*** *The estimated activity level $\tilde{A}_{\tau}$, $t+1 \leq \tau \leq t+\ell$, shows bounded growth. Specifically, $\tilde{A}_{\tau+1} \leq \alpha \tilde{A}_{\tau}$ for for all $t+1 \leq \tau \leq t+\ell-1$.*

Together these properties require the activity level in each neighborhood to be low (Property (a)), but also that the *estimated* activity level of each node does not grow too fast in future rounds (Property (b)). When these two properties hold, Lemma 4 can be applied inductively to obtain the following theorem. The fact that we use a single parameter α as an upper bound for both Properties (a) and (b) is just a matter of convenience and leads to an easy-to-state bound on number of rounds in this theorem.

▶ **Theorem 9** (Sample-and-Gather Theorem v1). *Let Alg be an α-sparse state-congested algorithm that completes in R rounds. Then Alg can be simulated in the low-memory MPC model with $\tilde{O}(m + n^{1+\varepsilon})$ total memory, for constant $0 < \varepsilon < 1$, in $O\left(R \log\log n / \sqrt{\log_{\alpha} n}\right)$ rounds.*

Proof. Let $\ell = \lfloor \sqrt{\frac{\varepsilon}{8} \cdot \log_{\alpha} n} \rfloor$. Partition the R rounds of *Alg* into $\lceil R/\ell \rceil$ phases, where Phase i, $1 \leq i < \lceil R/\ell \rceil$, consists of the ℓ rounds $(i-1) \cdot \ell + 1, (i-1) \cdot \ell + 2, \ldots, i \cdot \ell$ and Phase $\lceil R/\ell \rceil$ consists of at most ℓ rounds $(\lceil R/\ell \rceil - 1) \cdot \ell + 1, (\lceil R/\ell \rceil - 1) \cdot \ell + 2, \ldots, R$.

We now use the fact that *Alg* is α-sparse to show, via series of inequalities, that ℓ satisfies Inequality (1).

$$\begin{aligned}
\left(\sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \cdot \log n\right)^\ell &\le \left(\tilde{A}_{t+1} \cdot \log n \cdot \sum_{i=0}^{\ell-1} \alpha^i\right)^\ell && \text{(by Property (b) of being } \alpha\text{-sparse)}\\
&\le \left(A_{t+1} \cdot \log n \cdot \sum_{i=0}^{\ell-1} \alpha^i\right)^\ell && \text{(by } \tilde{A}_{t+1} = A_{t+1}\text{)}\\
&\le \left(\alpha^\ell \cdot \log^2 n \cdot \sum_{i=0}^{\ell-1} \alpha^i\right)^\ell && \text{(by Property (a) of being } \alpha\text{-sparse)}\\
&= \left(\alpha^\ell \cdot \log^2 n \cdot \frac{\alpha^\ell - 1}{\alpha - 1}\right)^\ell && \text{(by geometric series)}\\
&\le \alpha^{2\ell^2} \cdot (\log^2 n)^\ell && \text{(by } \ell \ge 1,\ \alpha \ge 2\text{)}\\
&\le n^{\varepsilon/4} \cdot n^{o(1)} && \text{(by } \ell = \left\lfloor \sqrt{\tfrac{\varepsilon}{8} \cdot \log_\alpha n} \right\rfloor\text{)}\\
&\le n^{\varepsilon/2}.
\end{aligned}$$

By using Lemma 4, this implies that each phase can be simulated in the MPC models with $O(n^\varepsilon)$ memory per machine in $O(\log \ell) = O(\log\log n)$ rounds. Given that the R rounds of *Alg* are partitioned into $\lceil R/\ell \rceil$ phases, we see that *Alg* can be implemented in the MPC model with $O(n^\varepsilon)$ memory per machine in $O(R \log\log n/\sqrt{\varepsilon \log_\alpha n})$ rounds. ◀

Theorem 9 provides a Simulation Theorem for the MPC model in which machines use $O(n^\varepsilon)$ memory per machine. However, the total memory used by MPC algorithms that result from this theorem is $\tilde{O}(m + n^{1+\varepsilon})$. We now show that under fairly general circumstances, it is possible to obtain a Simulation Theorem yielding low-memory MPC algorithms that use only $\tilde{O}(m)$ total memory, while taking slightly more time.

▶ **Definition 10.** *A* CONGEST *algorithm Alg is said to be degree-ordered if it satisfies two properties.*

(a) *The execution of Alg can be partitioned into Stages* $1, 2, \ldots$ *such that in Stage* i *the only active nodes are those whose degree is greater than* $\Delta^{1/2^i}$ *and other nodes that are neighbors of these "high degree" nodes.*

(b) *Let* R_i *be the number of rounds in Stage* i*. Then* $R_i \le R_{i-1}/2$*.*

A lot of symmetry breaking algorithms are either inherently degree-ordered or can be made so with small modifications – this can be seen in the applications of the Sample-and-Gather Theorems in Section 3.1. The fact that our definition permits activity in a stage not just at nodes that are "high degree" for that stage, but even at other nodes that are neighbors of high degree nodes, provides the flexibility we need for our applications. In fact, it is possible to further relax this definition and allow all nodes within $O(1)$ hops of "high degree" nodes to be active in a stage; for ease of exposition we just work with the current definition. For algorithms that are degree-ordered, we can gather balls centered at active nodes, whose volume is at most the degree threshold for the current stage. This allows us to use a simple charging scheme to charge the sizes of the balls to the memory already allocated for the neighborhoods of the active nodes. This in turn yields the $\tilde{O}(m)$ total memory bound. Property (b) holds for algorithms whose running time is dominated by $O(\log \Delta)$. Given that the degree threshold in Property (a) falls as $\Delta^{1/2^i}$, the running time of each stage falls by a factor of 2.

▶ **Lemma 11.** *Suppose that Alg is a state-congested, degree-ordered algorithm. Consider a phase of $\ell - 1$ rounds $t+1, t+2, \ldots, t+\ell-1$ with a Stage i. If ℓ satisfies*

$$\left(\sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n\right)^\ell \leq \min\left\{n^{\varepsilon/2}, \Delta^{1/2^i}\right\}, \tag{2}$$

then this phase can be simulated in $O(\log \ell)$ rounds in the low-memory MPC model with a total of $\tilde{O}(m)$ memory over all the machines.

Finally, if *Alg* is a state-congested algorithm that is α-sparse and degree-ordered, we obtain the following Simulation Theorem that guarantees an $\tilde{O}(m)$ total memory usage.

▶ **Theorem 12** (Sample-and-Gather Theorem v2)**.** *Let Alg be a state-congested, α-sparse, degree-ordered algorithm that completes in R rounds. Let $\alpha' = \alpha \cdot \log^2 n$. Then Alg can be simulated in the MPC model with $O(n^\varepsilon)$ memory per machine, for constant $0 < \varepsilon < 1$ and $\tilde{O}(m)$ total memory, in $O\left(R \log\log\Delta / \sqrt{\log_{\alpha'}\Delta}\right)$ rounds.*

3 Fast 2-Ruling Set Algorithms

Our 2-ruling set algorithms consist of 3 parts. In Part 1, we sparsify the input graph, in Part 2 we "shatter" the graph still active after Part 1, and in Part 3 we deterministically finish off the computation. Part 1 is a modification of Sparsify, a Congest model algorithm due to Kothapalli and Pemmaraju [26]; Part 2 is a sparsified MIS algorithm, also in the Congest model, due to Ghaffari [15, 16]. Our main contribution in this section is to show that these algorithms are state-congested, α-sparse for small α, and degree-ordered. As a result, we can apply the Sample-and-Gather Simulation Theorems (Theorems 9 and 12) to these algorithms to obtain fast low-memory MPC algorithms. Part 3 – in which we finish off the computation – is easy to directly implement in the MPC model.

3.1 Simulating Sparsify in low-memory MPC

Algorithm 1 is a modified version of the Sparsify algorithm of Kothapalli and Pemmaraju [26]. The algorithm computes a "sparse" set of vertices U that dominates all the vertices in the graph (i.e. $\text{Nbr}^+(U) = V$, see Lemma 13). In each iteration, "high degree" nodes and their neighbors are sampled and the sampled nodes are added to U. In successive iterations, the threshold for being a high degree node falls by a factor f and the sampling probability grows by a factor f. The neighbors of the nodes that successfully join U are deactivated. The parameter f takes on different values in different instantiations of this algorithm, though always satisfying $\log f = \log^\delta \Delta$ for some constant $0 < \delta < 1$. For example, f is set to $2^{(\log\Delta)^{2/3}}$ (respectively, $2^{(\log\Delta)^{1/2}}$) to obtain Theorem 19 part (i) (respectively, part (ii)).

DegOrderedSparsify fits nicely within the framework of the Sample-and-Gather Simulation Theorems from Section 2. The state of each vertex stays small throughout the algorithm (just `ID` plus $O(1)$ bits), making DegOrderedSparsify state-congested. The activity level in any iteration is bounded by $O(f \log n)$, because we show in Lemma 13 that in any neighborhood only $O(f \log n)$ vertices are sampled whp and only these sampled vertices need be active in that iteration. Furthermore, since the sampling probability grows by a factor f in each iteration, the estimated neighborhood activity levels also grow by a factor f, as we consider future iterations of DegOrderedSparsify. As shown in Lemma 13, this makes DegOrderedSparsify f-sparse. In the Sparsify algorithm [26] *all* nodes,

Algorithm 1 DegOrderedSparsify(G, f).

```
1 $U \leftarrow \emptyset$
2 $V_0 \leftarrow V$ // Initially all nodes are active
3 for $i = 1$ to $\lceil \log_f \Delta \rceil$ do
4     Let $H_i$ be the nodes in $V_{i-1}$ with degree at least $\Delta/f^i$ in $G[V_{i-1}]$
5     Each node in $\mathrm{Nbr}^+(H_i) \cap V_{i-1}$ joins $U_i$ with probability $f^i \cdot c \ln n/\Delta$, where $c$ is a
        fixed constant
6     $V_i \leftarrow V_{i-1} \setminus \mathrm{Nbr}^+(U_i)$ // Nodes with at least one neighbor in $U_i$
          deactivate themselves
7     $U \leftarrow U \cup U_i$
8 end
9 return $U$
```

independent of their degrees, sample themselves (as in Line 5). Here, in order to make Algorithm DegOrderedSparsify degree-ordered, we make a small modification and permit only high degree nodes and their neighbors to sample themselves. As we show in Lemma 13, the algorithm continues to behave as before, but is now degree-ordered.

▶ **Lemma 13.** *Given a graph $G = (V, E)$ and a parameter $f > 3$, a subset $U \subseteq V$ can be computed in $O(\log_f \Delta)$ rounds such that for every $v \in V$, $N^+(v) \cap U \neq \emptyset$, and for every $v \in U$, $\deg_U(v) \leq 2cf \ln n$, with probability at least $1 - n^{-c+2}$.*

It is easy to see that Algorithm DegOrderedSparsify(G, f) can be implemented in the Congest model in $O(\log_f \Delta)$ rounds because each iteration of the **for**-loop takes $O(1)$ rounds in Congest. Furthermore, since each node can update its state by simply knowing if it or a neighbor has joined set U_i, the update function at each node is separable (see Definition 1). Therefore, DegOrderedSparsify(G, f) can be faithfully simulated in the low-memory MPC model in $O(\log_f \Delta)$ rounds.

We now show that Algorithm DegOrderedSparsify has the three properties needed for round compression via our Simulation Theorems and this leads to a substantial speedup.

▶ **Lemma 14.** *Algorithm DegOrderedSparsify(G, f) is a state-congested, f-sparse, degree-ordered algorithm.*

Using Theorem 9 and Theorem 12, we obtain the following theorem.

▶ **Theorem 15.** *Algorithm DegOrderedSparsify(G, f) can be implemented in the low-memory MPC model in (i) $O\left(\frac{\log_f \Delta}{\sqrt{\log_f n}} \log \log n\right)$ rounds whp using $\tilde{O}(m + n^{1+\varepsilon})$ total memory and (ii) $O\left(\sqrt{\log_f \Delta} \cdot \log \log \Delta\right)$ rounds whp using $\tilde{O}(m)$ total memory.*

3.2 Simulating Sparsified Graph Shattering in low-memory MPC

Distributed graph shattering has become an important algorithmic technique for symmetry breaking problems [5, 16, 20]. In this section, we use a sparsified graph shattering algorithm due to Ghaffari [16] to process the graph $G[U]$ returned by DegOrderedSparsify. The output of the shattering algorithm consists of an independent set $I \subseteq U$ such that the graph induced by the remaining set of vertices $S = U \setminus \mathrm{Nbr}^+(I)$ contains only small connected components.

Ghaffari's sparsified shattering algorithm [16] is shown in Algorithm 2. At the start of each round t, each node v has a *desire-level* $p_t(v)$ for joining the independent set I, and initially this is set to $p_1(v) = 1/2$. The independent set I is also initialized to the empty set. The algorithm runs in phases, with each phase having $\ell := \sqrt{\delta \log n}/10$ rounds for a small constant δ.

Several aspects of the algorithm make it nicely fit the Sample-and-Gather framework from Section 2. We now point these out. (i) The desire-level $p_\tau(u)$ for $t+1 \leq \tau \leq t+\ell$ can be viewed the probability of sampling u; after the initial communication amongst neighbors (Line 1), only sampled nodes send messages (beeps) and all other nodes remain silent. (ii) The quantity $d_{t+1}(u)$ is identical to the activity level $A_{t+1}(u)$ in u's neighborhood, defined in Section 2. (iii) Nodes with a high activity level, i.e., $d_{t+1}(u) \geq 2^{\sqrt{\log n}/5}$ (aka super-heavy nodes), are oblivious nodes and are therefore excluded in the definition of A_{t+1}. As a result $A_{t+1} \leq 2^{\sqrt{\log n}/5}$. (iv) In each iteration in a phase, the sampling probability grows by a factor of at most 2 (Line 8). This implies that the estimated activity levels grow by a factor of 2 in future rounds.

Algorithm 2 SHATTER(G): (one phase, starting at iteration $t+1$).

```
1  Each node $u$ sends its current desire-level $p_{t+1}(u)$ to all its neighbors
2  Each node $u$ computes $d_{t+1}(u) = \sum_{v \in \mathrm{Nbr}(u)} p_{t+1}(v)$
3  If node $u$ has $d_{t+1}(u) \geq 2^{\sqrt{\log n}/5}$ then $u$ is called a *super-heavy* node
4  $\ell = \sqrt{\delta \log n}/10$ ;                              // $\delta$ is a small constant
5  **for** $\tau = t+1, t+2, \ldots, t+\ell$ *iterations* **do**
       // Round 1
6      Each node $u$ beeps with probability $p_\tau(u)$ and remains silent otherwise.
7      Node $u$ is added to $I$ if it is not super-heavy, it beeps, and none of its neighbors
        beep
8      Node $u$ sets $p_{\tau+1}(u)$ as follows:
         $$p_{\tau+1}(u) = \begin{cases} p_\tau(u)/2 & \text{if } u \text{ is super-heavy, or a neighbor of } u \text{ beeps} \\ \min\{1/2, 2 \cdot p_\tau(u)\} & \text{otherwise} \end{cases}$$
          // Round 2
9      Node $u$ beeps if it joins $I$ in this iteration.
10     Neighbors of node $u$ that are not in $I$ become inactive on hearing the beep from $u$
11 **end**
```

The first four steps of Algorithm 2 do not fit into the Sample-and-Gather framework since each node needs to send its p_{t+1} value to its neighbors. But the nodes are computing $d_{t+1}(u) = \sum_{v \in \mathrm{Nbr}(u)} p_{t+1}(v)$ which is a separable function (sum). Therefore, we can implement the first two steps in $O(1/\varepsilon)$ rounds using Lemma 2, and use the Sample-and-Gather framework to simulate the **for**-loop of the algorithm. These observations are formalized in the lemma below to show that SHATTER is 2-sparse. Additionally, the lemma shows that the algorithm is state-congested.

▶ **Lemma 16.** *Algorithm 2 is a state-congested algorithm whose* ***for****-loop is* 2*-sparse.*

A total of $O(\log \Delta/\sqrt{\log n})$ repeated applications of SHATTER (i.e. a total of $O(\log \Delta)$ iterations) suffice to shatter the graph into small-sized components [16, 21].

Using Lemma 16 and Theorem 9, we obtain the following lemma that shows that Shatter can be simulated efficiently in the low-memory MPC model.

▶ **Lemma 17.** *We can simulate a total $O(\log \Delta)$ iterations of Algorithm* Shatter *in the low-memory MPC model with $\tilde{O}(m + n^{1+\varepsilon})$ total memory in $O\left(\frac{\log \Delta \cdot \log \log n}{\sqrt{\log n}}\right)$ rounds whp.*

Ghaffari and Uitto [21] present a variant of Algorithm Shatter and show (in Theorem 3.7) that this variant can be simulated in $O(\sqrt{\log \Delta} \cdot \log \log \Delta)$ rounds in the low-memory MPC model, while using only $\tilde{O}(m)$ total memory. While they describe their MPC implementation from scratch, this MPC implementation can also be obtained by applying our Sample-and-Gather Theorem (specifically, Theorem 12). It can be shown that this variant is state-congested and has the same sparsity property as Algorithm Shatter, i.e., it is 2-sparse. Furthermore, it can also be made degree-ordered by simply processing nodes in degree buckets $(\Delta^{1/2^i}, \Delta^{1/2^{i-1}}]$, in the order $i = 1, 2, \ldots, O(\log \log \Delta)$.

▶ **Lemma 18** (Ghaffari-Uitto [21])**.** *There is a variant of Algorithm* Shatter *can be simulated in the low-memory MPC model with $\tilde{O}(m)$ total memory in $O(\sqrt{\log \Delta} \cdot \log \log \Delta)$ rounds whp.*

3.3 Finishing off the 2-ruling set computation

After applying DegOrderedSparsify to the input graph $G = (V, E)$ and then Shatter to the subgraph $G[U]$, induced by the subset $U \subseteq V$ output by DegOrderedSparsify, we are left with a number of small-sized components. Ghaffari and Uitto [21, Theorem 3.7] show that given the properties that the remaining graph has after Shatter, it is possible to simply (and deterministically) gather each component at a machine and find an MIS of the component locally in $O(\sqrt{\log \log n})$ rounds in the low-memory MPC model using $\tilde{O}(m)$ memory. Applying this "finishing off" computation completes our 2-ruling set algorithm. The output of the algorithm is the union of the independent set output by Shatter and the independent set output by the "finishing off" computation.

▶ **Theorem 19.** *A 2-ruling set can be computed whp in the low-memory MPC model in*
(i) $O((\log \Delta)^{1/6} \log \log n)$ *rounds using $\tilde{O}(m + n^{1+\varepsilon})$ total memory and in*
(ii) $O((\log \Delta)^{1/4} \log \log \Delta + \sqrt{\log \log n} \log \log \Delta)$ *rounds using $\tilde{O}(m)$ total memory.*

▶ Remark. We note that by just running Shatter on an input graph followed by the "finishing off" computation, we get an MIS of the input graph. So our approach yields MIS algorithms in the low-memory MPC model via the Sample-and-Gather Simulation Theorems.

▶ **Theorem 20.** *An MIS of a graph G can be found in the low-memory MPC model in:*
(i) $O\left(\frac{\log \Delta \cdot \log \log n}{\sqrt{\log n}} + \sqrt{\log \log n}\right)$ *rounds whp using $\tilde{O}(m + n^{1+\varepsilon})$ total memory and*
(ii) $O(\sqrt{\log \Delta} \log \log \Delta + \sqrt{\log \log n})$ *rounds whp using $\tilde{O}(m)$ total memory.*
As far as we know, the MIS result for the $\tilde{O}(m + n^{1+\varepsilon})$ total memory setting is new, but the result for the $\tilde{O}(m)$ total memory setting simply recovers the result from [21].

4 Fast β-ruling Set Algorithms

We now extend the 2-ruling set low-memory MPC algorithm in the previous section to obtain a β-ruling set low-memory MPC algorithm for any integer $\beta \geq 2$. The overall idea is to repeatedly use Algorithm DegOrderedSparsify, as in [8]. We start by running a low-memory MPC implementation of DEGORDEREDSPARSIFY with a parameter f_1; this call returns a set of nodes S_1. Once this phase ends, the remaining graph $G[S_1]$ has degree at most $O(f_1 \cdot \log n)$, by Lemma 13. We then run DEGORDEREDSPARSIFY on the graph $G[S_1]$ with a parameter f_2 and this yields a set of nodes S_2. This process continues for $\beta - 1$ phases at the end of which the graph $G[S_{\beta-1}]$ has a maximum degree $O(f_{\beta-1} \cdot \log n)$. We now proceed to run a low-memory MPC implementation of an MIS algorithm on $G[S_{\beta-1}]$.

The correctness of the β-ruling set algorithm can be noted from Lemma 13. The set S_i covers all the nodes in S_{i-1} which means that all the nodes in $S_0 = V$ are at most $\beta - 1$ hops away from the nodes in $S_{\beta-1}$. Therefore all the nodes of V are at most β hops away from the MIS C of $G[S_{\beta-1}]$. This means that the set C that the above technique returns is a β-ruling set of G. In the following, we analyze the round complexity of the β-ruling set algorithm in the low-memory MPC model.

▶ **Lemma 21.** *Let $f_0 = \Delta$. The β-ruling set algorithm runs in*

$$O\left(\left(\sum_{i=1}^{\beta-1} \frac{\log(f_{i-1}\log n)}{\sqrt{\log f_i \cdot \log n}} + \frac{\log(f_{\beta-1}\log n)}{\sqrt{\log n}}\right)\log\log n\right) \tag{3}$$

rounds whp in the low-memory MPC model with $\tilde{O}(m + n^{1+\varepsilon})$ total memory.

▶ **Lemma 22.** *Let $f_0 = \Delta$. The β-ruling set algorithm runs in*

$$O\left(\left(\sum_{i=1}^{\beta-1} \sqrt{\frac{\log(f_{i-1}\log n)}{\log f_i}} + \sqrt{\log(f_{\beta-1}\log n)}\right)\log\log\Delta + \sqrt{\log\log n}\right) \tag{4}$$

rounds whp in the low-memory MPC model with $\tilde{O}(m)$ total memory.

We now instantiate the parameters $f_1, f_2, \ldots, f_{\beta-1}$ so as to minimize the running times in Lemmas 21 and 22. This leads to the following corollaries.

▶ **Theorem 23.** *A β-ruling set of a graph G can be found whp in the low-memory MPC model in*

(i) $O\left(\beta \cdot \log^{1/(2^{\beta+1}-2)}\Delta \cdot \log\log n\right)$ *rounds with $\tilde{O}(m + n^{1+\varepsilon})$ total memory and in*

(ii) $O\left(\beta \cdot \left(\log^{1/2^{\beta}}\Delta \cdot \log\log\Delta + \sqrt{\log\log n}\right) \cdot \log\log\Delta\right)$ *rounds with $\tilde{O}(m)$ total memory.*

4.1 β-ruling sets in $O(\text{polyloglog}(n))$ rounds

As mentioned in the Introduction, this research is partly motivated by the question of whether ruling set problems can be solved in the low-memory MPC model in $O(\text{polyloglog}(n))$ rounds. Using our results we identify two interesting circumstances under which β-ruling sets can be computed in the low-memory MPC model in $O(\text{polyloglog}(n))$ rounds. First, because the running time in Theorem 23 part (i) has an inverse exponential dependency on β, we get the following corollary.

▶ **Corollary 24.** *For $\beta \in \Omega(\log\log\log\Delta)$, a β-ruling set of a graph G can be computed in $O(\beta\log\log n)$ rounds whp in the low-memory MPC model with $\tilde{O}(m + n^{1+\varepsilon})$ total memory.*

Second, we can also show that for graphs with bounded Δ, we can compute a β-ruling set in $O(\beta\log\log n)$ rounds, however this bound increases quickly with β, giving us the following corollary.

▶ **Corollary 25.** *If we have that $\Delta = O\left(2^{\log^{1-\frac{1}{2^{\beta}}} n}\right)$, then a β-ruling set can be computed in $O(\beta\log\log n)$ rounds whp in the low-memory MPC model with $\tilde{O}(m + n^{1+\varepsilon})$ total memory.*

References

1 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, December 1986. doi:10.1016/0196-6774(86)90019-2.

2 Sepehr Assadi. Simple round compression for parallel vertex cover. *CoRR*, abs/1709.04599, 2017. arXiv:1709.04599.

3 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta+1)$ vertex coloring. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 767–786, 2019. doi:10.1137/1.9781611975482.48.

4 Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pages 461–470, 2019. doi:10.1145/3293611.3331596.

5 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *J. ACM*, 63(3):20:1–20:45, 2016. doi:10.1145/2903137.

6 Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M. Karp, and Jara Uitto. Massively parallel computation of matching and mis in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 481–490, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331609.

7 Soheil Behnezhad, Mohammadtaghi Hajiaghayi, and David G. Harris. Exponentially Faster Massively Parallel Maximal Matching. In *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, volume 2019-November, pages 1637–1649, 2019. doi:10.1109/FOCS.2019.00096.

8 Tushar Bisht, Kishore Kothapalli, and Sriram V. Pemmaraju. Brief announcement: Super-fast t-ruling sets. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 379–381. ACM, 2014. doi:10.1145/2611462.2611512.

9 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of $(\delta + 1)$ coloring in congested clique, massively parallel computation, and centralized local computation. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 471–480, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331607.

10 Moses Charikar, Weiyun Ma, and Li-Yang Tan. Unconditional lower bounds for adaptive massively parallel computation. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '20, page 141–151, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3350755.3400230.

11 Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proc. VLDB Endow.*, 8(12):1804–1815, August 2015. doi:10.14778/2824032.2824077.

12 Artur Czumaj, Peter Davies, and Merav Parter. Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space, 2019. arXiv:1912.05390.

13 Artur Czumaj, Slobodan Mitrovic, Jakub Łącki, Krzysztof Onak, Aleksander Mądry, and Piotr Sankowski. Round compression for parallel matching algorithms. *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 471–484, 2018. doi:10.1145/3188745.3188764.

14 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. doi:10.1145/1327452.1327492.

15 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, page 270–277, USA, 2016. Society for Industrial and Applied Mathematics.

16 Mohsen Ghaffari. Distributed MIS via all-to-all communication. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, Part F129314:141–150, 2017. doi:10.1145/3087801.3087830.

17 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for MIS, matching, and vertex cover. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pages 129–138, 2018. doi:10.1145/3212734.3212743.

18 Mohsen Ghaffari, Christoph Grunau, and Ce Jin. Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond. *CoRR*, 2020. arXiv:2002.09610.

19 Mohsen Ghaffari, Ce Jin, and Daan Nilis. A massively parallel algorithm for minimum weight vertex cover. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 259–268. ACM, 2020. doi:10.1145/3350755.3400260.

20 Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, volume 2019-November, pages 1650–1663, 2019. doi:10.1109/FOCS.2019.00097.

21 Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1636–1653, July 2019. doi:10.1137/1.9781611975482.99.

22 Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. Greedy and local ratio algorithms in the mapreduce model. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, page 43–52, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3210377.3210386.

23 James W. Hegeman, Sriram V. Pemmaraju, and Vivek Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. *CoRR*, abs/1408.2071, 2014. arXiv:1408.2071.

24 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 938–948, Philadelphia, PA, January 2010. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973075.76.

25 Kishore Kothapalli, Shreyas Pai, and Sriram V. Pemmaraju. Sample-and-gather: Fast ruling set algorithms in the low-memory MPC model, 2020. arXiv:2009.12477.

26 Kishore Kothapalli and Sriram V. Pemmaraju. Super-fast 3-ruling sets. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 136–147. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.136.

27 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '04, page 300–309, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1011767.1011811.

28 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *J. ACM*, 63(2), March 2016. doi:10.1145/2742012.

29 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, February 1992. doi:10.1137/0221015.

30 Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in o(log log n) communication rounds. In *SPAA*, pages 94–100, 2003. doi:10.1145/777412.777428.

31 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986. doi:10.1137/0215074.

32 Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In

Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10, page 135–146, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1807167.1807184.

33 Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.40.

34 D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.

35 Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits (on lower bounds for modern parallel computation). *J. ACM*, 65(6), November 2018. doi:10.1145/3232536.

36 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 350–363, 2020. doi:10.1145/3357713.3384298.

37 Grigory Yaroslavtsev and Adithya Vadapalli. Massively parallel algorithms and hardness for single-linkage clustering under ℓ_{p} distances. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5596–5605. PMLR, 2018. URL: http://proceedings.mlr.press/v80/yaroslavtsev18a.html.

38 Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016. doi:10.1145/2934664.