Fast Deterministic Massively Parallel Ruling Sets Algorithms

Hongyan Ji
Department of Computer Science
The University of Iowa
Iowa City, Iowa, United States
hongyan-ji@uiowa.edu

Kishore Kothapalli
Center for Security Theory and Algorithmic Research
IIIT Hyderabad
Hyderabad, Telangana, India
kkishore@iiit.ac.in

Sriram V Pemmaraju
Department of Computer Science
The University of Iowa
Iowa City, Iowa, United States
sriram-pemmaraju@uiowa.edu

Ajitanshu Singh
Center for Security Theory and Algorithmic Research
IIIT Hyderabad
Hyderabad, Telangana, India
ajitanshu.singh@research.iiit.ac.in

Abstract

In this paper, we present a deterministic $\tilde{O}(\log^{1/3}\Delta)$-round algorithm for the 2-ruling set problem in the sublinear Massively Parallel Computation (MPC) model. This improves upon the fastest known deterministic 2-ruling set algorithm for this model, which is the $\tilde{O}(\sqrt{\log n})$-round algorithm by Giliberti and Parsaeian (PODC 2024). Our result is obtained by derandomizing the "sample-and-gather" approach of Kothapalli, Pai, and Pemmaraju (FSTTCS 2020). The "sample-and-gather" approach involves making random sampling decisions, not just for the current iteration, but a batch of future iterations. Thus, derandomizing this approach requires the "fixing" of randomness for a batch of future iterations. We further extend our results to show that a β-ruling set for $\beta \geq 2$ can be obtained in $\tilde{O}(\log^{1/2^{\beta}-1}\Delta)$ deterministic rounds in the sublinear MPC model. Additionally, we present a deterministic β-ruling set algorithms for sparse graphs (i.e., bounded arboricity graphs) where $\beta \geq 2$, which runs in $\tilde{O}(\log^{1/2^{\beta}-1}\lambda)$ rounds for arboricity-λ graphs in the sublinear MPC model.

CCS Concepts

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

Keywords

Derandomization, Deterministic Algorithms, Distributed Algorithms, Massively Parallel Computation, Ruling Sets

ACM Reference Format:
Hongyan Ji, Kishore Kothapalli, Sriram V Pemmaraju, and Ajitanshu Singh. 2025. Fast Deterministic Massively Parallel Ruling Sets Algorithms. In *26th International Conference on Distributed Computing and Networking (ICDCN 2025), January 04–07, 2025, Hyderabad, India.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3700838.3700872

ICDCN 2025, January 04–07, 2025, Hyderabad, India

ACM ISBN 979-8-4007-1062-9/25/01
https://doi.org/10.1145/3700838.3700872

1 Introduction

Symmetry breaking problems are some of the most widely studied problems in the context of distributed graph algorithms. Some well-known examples of symmetry breaking problems include maximal independent set (MIS), $(\Delta+1)$-coloring, maximal matching, and the β-ruling set problem, which is the focus of this paper. Randomization has proven to be a key ingredient in designing the fastest known algorithms for all of these problems. For example, for MIS, the fastest algorithm in the classical LOCAL and CONGEST models of distributed computing is an elegant, $O(\log n)$-round, *randomized* algorithm, independently designed by Luby [18] and Alon, Babai, and Itai [1] more than 25 years ago. This algorithm, commonly known as Luby's algorithm, is faster than any known deterministic MIS algorithm.

The success of randomization in solving symmetry breaking problems has motivated both theoretical and practical questions about its inherent power. Specifically, researchers are focused on either designing deterministic symmetry breaking algorithms that match the speed of their randomized counterparts or alternately showing, through lower bound arguments, that achieving this is impossible. In recent years, there has been considerable progress in addressing the first challenge. In particular, within all-to-all communication models such as the CONGESTED–CLIQUE and MPC model, researchers have successfully designed efficient deterministic algorithms for symmetry breaking problems by *derandomizing* existing randomized algorithms (see e.g., [5, 7, 8, 21]).

The MPC model, as introduced in [15], has gained significant attention due to its flexibility and its ability to model several widely used distributed computing frameworks. This model is particularly valued for its ability to model practical applications by simulating frameworks such as MapReduce [9], known for its scalability and efficiency in processing large datasets; Spark [24], which excels in in-memory computations and iterative algorithms; Pregel [19], designed for large-scale graph processing; and Giraph [6], an extension of Pregel, optimized for distributed graph computations. The MPC model (described more precisely in Section 1.1) consists of a collection of memory-constrained machines that are fully interconnected by a bandwidth constrained communication network. The number of machines and the memory per machine are strongly sublinear in the input size, modeling the fact that the problem instances in this setting are extremely large. Of course, the

total memory (across all machines) needs to be sufficient to store the entire input. For graph problems, we distinguish between the *sublinear memory* MPC model, in which the memory per machine is strongly sublinear even in the number of vertices in the input graph, and the *linear memory* MPC model, in which the memory per machine is linear in the number of vertices in the input graph. This distinction seems important because it has been markedly easier to design efficient algorithms for graph problems in the linear memory MPC model compared to the sublinear memory MPC model.

The focus of this paper is the derandomization of randomized algorithms for the *β-ruling set* problem in the sublinear memory MPC model. For any integer $\beta \geq 1$, the *β-ruling set* of a graph $G = (V, E)$ is an independent (i.e., pairwise non-adjacent) set $U \subseteq V$ of vertices such that every vertex in $v \in V$ is either in U or has a vertex in U that is at most β hops in G from it. A 1-ruling set is just a maximal independent set. Randomization has proven to be critical in designing extremely fast, sub-logarithmic, and even constant round algorithms for the β-ruling set problem in the MPC model. In the sublinear memory MPC model, Kothapalli, Pai, and Pemmaraju use a technique that they call "sample and gather" to design a randomized $\tilde{O}(\log^{1/6} \Delta)$-round algorithm for the 2-ruling set problem [16][1]. Here Δ is the maximum degree of the graph. In the linear memory MPC model, Cambus, Kuhn, Pai, and Uitto [4] present a randomized $O(1)$-round algorithm for the 2-ruling set problem.

Now we focus on deterministic algorithms for β-ruling sets in the MPC model; all of the algorithms mentioned here are obtained by derandomizing pre-existing randomized β-ruling set algorithms. Czumaj, Davies, and Parter [7] showed that an MIS (1-ruling set) can be constructed deterministically in $O(\log \Delta + \log \log n)$ rounds in the sublinear memory MPC model. This result is obtained by derandomizing the well-known Luby's MIS algorithm. Pai and Pemmaraju [21] showed that a much faster deterministic 2-ruling set algorithm, requiring only $O(\log \log n)$ rounds, exists in the linear memory MPC model. However, they left unresolved the challenge of efficiently and deterministically constructing 2-ruling sets in the sublinear memory MPC model. The Pai-Pemmaraju deterministic algorithm is obtained by derandomizing a randomized sparsification algorithm due to Kothapalli and Pemmaraju [17]. Very recently, Giliberti and Parsaeian [13] improved the result of Pai and Pemmaraju. They showed that in the linear memory model, a 2-ruling set can be deterministically constructed in just $O(1)$ rounds. This result was obtained by derandomizing the aforementioned $O(1)$-round randomized 2-ruling set algorithm due to Cambus, Kuhn, Pai, and Uitto [4]. In the same paper, Giliberti and Parsaeian [13] also solved the open question from Pai and Pemmaraju [21] by presenting a deterministic $O(\log^{1/2} \Delta)$-round 2-ruling set algorithm in the sublinear memory MPC model. This result was obtained by derandomizing the randomized sparsification algorithm due to Kothapalli and Pemmaraju [17].

Giliberti and Pasaeian [13] leave open the question of whether a faster deterministic 2-ruling set algorithm in the sublinear memory MPC model can be achieved by derandomizing the more recent randomized 2-ruling set algorithm [16], which employs the "sample and gather" technique. Specifically, with regards to derandomizing the "sample and gather" technique, they say

> A key assumption of this technique, however, is that of fixing the randomness of future iterations a priori. Consequently, extending this technique to achieve such a speed up deterministically appears to require a substantially novel approach.

In the present paper we do show a way to *fix* the randomness needed by nodes in future iterations a priori. This technique, coupled with the technique of sparsification [17] and a local simulation based on ball doubling [16] allows us to improve upon the results from [13] for deterministically constructing 2-ruling sets in the sublinear memory MPC model. In particular, we show that a 2-ruling set can be constructed deterministically in $\tilde{O}(\log^{1/3} \Delta)$ rounds. We then extend our results to show that a β-ruling set for $\beta \geq 2$ can be obtained in $\tilde{O}(\log^{1/(2^\beta - 1)} \Delta)$ rounds in the sublinear memory model. Finally, combining our results with those of Fischer, Giliberti, and Grunau [10] for bounded arboricity graphs, we show that a β-ruling set of a graph G with arboricity λ can be computed deterministically in the sublinear memory MPC model in $\tilde{O}(\log^{1/(2^\beta - 1)} \lambda)$ rounds.

1.1 The MPC Model

The MPC model is characterized by having a set of machines, each with a fixed memory of S words. These machines are fully interconnected with communication links. The model assumes synchronous communication and computation. In each round, a machine can receive up to $O(S)$ words via communication links, perform local computations, and send up to $O(S)$ words through the same links. Note that this is not a per edge bandwidth constraint; rather it is bound on the total number of words that a machine can receive or send in a round. The model assumes that for an input size N, both the memory per machine and the number of machines are $O(N^{1-\delta})$ for a constant $\delta \in (0, 1)$. This implies that both the memory per machine and the number of machines are strongly sublinear in the input size.

When focusing on graph problems, notice that $N = \tilde{O}(n + m)$ where the input graph $G = (V, E)$ has n nodes and m edges. Therefore, $S = \tilde{O}((n + m)^{1-\delta})$. However, the relationship between S and the number of nodes n, seems to play an important role in determining how easy it is to design algorithms in the MPC model. There are two popular variants of the MPC model in the context of graph problems: the *sublinear memory* MPC model and the *linear memory* MPC model. In the sublinear memory MPC model, $S = \tilde{O}(n^\epsilon)$ for some constant $0 < \epsilon < 1$, whereas in the linear memory MPC model, $S = \tilde{O}(n)$. Note that for dense graphs, i.e., graphs for which $m = \Omega(n^{1+\epsilon'})$ for constant $\epsilon' > 0$, both the sublinear memory MPC model and the linear memory MPC model are well-defined, whereas for sparse graphs, i.e., graphs for which $m = \tilde{\Theta}(n)$, only the sublinear memory MPC model is well-defined. In other words, the sublinear memory MPC model is defined for all graphs and this model is the focus of this paper.

In the sublinear memory MPC model, node-centric distribution of the input graph simplifies the design and implementation of graph algorithms and is therefore assumed. In this setup, each node

[1] In this paper, we use the $\tilde{O}(f(n))$ notation to hide $O(\text{poly}(\log f(n)))$ terms. We abuse this slightly to allow $\tilde{O}(\log^a \Delta)$ for constant $a > 0$, to hide $O(\text{poly}(\log \log n))$ terms.

v is assigned to a specific machine M_v. This machine M_v knows both the neighbors of v and the machines to which these neighbors are assigned. However, the information about neighbors might exceed the memory of a single machine. To address this, a node v with $\deg(v) > n^\epsilon$ is assumed to be split into copies distributed across different machines. These copies are organized into a virtual balanced tree, where each node has n^ϵ branches and the tree has a depth of $O(1/\epsilon)$. The root of the tree manages communication between v and its neighbors in the input graph, which requires $O(1/\epsilon)$ rounds ([2, 12, 13, 16]).

1.2 Main Results

In this paper, we present an algorithm for the 2-ruling set problem running in $\tilde{O}(\log^{1/3}\Delta)$ rounds deterministically, in the sublinear memory MPC model. Our algorithm uses techniques such as ball doubling and gathering to show that t rounds of the sparsification approach of Kothapalli and Pemmaraju [17] can be compressed to run "locally" in the sublinear memory MPC model in $\tilde{O}(\log t)$ rounds. To this end, we rely on the derandomization technique of Giliberti and Parsaeian. [13] coupled with the deterministic MIS algorithm in the sublinear memory MPC model of Czumaj, Davies and Parter. [7]. Using these techniques, we arrive at Theorem 1.

Theorem 1. *There is a deterministic sublinear memory MPC algorithm that computes a 2-ruling set in $O(\log^{1/3}\Delta\cdot\log\log\Delta+\log\log n)$ rounds using $\tilde{O}(m+n^{1+\epsilon})$ total memory.*

Subsequently, using the repeated sparsification approach from [3], we obtain the following theorem for β–ruling sets.

Theorem 2. *There is a deterministic sublinear memory MPC algorithm that computes a β-ruling set in $O(\log^{1/2^\beta-1}\Delta\cdot\log\log\Delta+\log\log n)$ rounds using $\tilde{O}(m+n^{1+\epsilon})$ total memory.*

Combining our results with those from Fischer, Giliberti, and Grunau [10], we also obtain the following results that show how to deterministically compute 2-ruling sets and β-ruling sets using far fewer rounds for low arboricity graphs. The *arboricity* of a graph is the minimum number of forests into which its edges can be partitioned. The arboricity of a graph is a measure of its sparsity; a graph with low arboricity is uniformly sparse. This can be seen by noting that the arboricity of a graph is closely related (within a factor of 2) to the *degeneracy* of a graph, which is defined as the maximum, over all induced subgraphs of the graph, of the minimum degree of a vertex in the subgraph.

Theorem 3. *There exists a deterministic sublinear memory MPC algorithm that computes a 2-ruling set of a λ-arboricity graph in $O(\log^{1/3}\lambda\cdot\log\log\lambda+\log\log n)$ rounds using $O(m+n^{1+\epsilon})$ total memory.*

Theorem 4. *A β-ruling set of a graph G with arboricity λ can be computed deterministically in the sublinear memory MPC model in $O(\log^{1/2^\beta-1}\lambda\cdot\log\log\lambda+\log\log n)$ rounds with $\tilde{O}(m+n^{1+\epsilon})$ total memory.*

Note: We note that the results in Giliberti and Parsaeian [13] use linear total (global) memory, i.e., $O(m+n)$ memory, over all machines. Our results use slightly more, $O(m+n^{1+\epsilon})$ total memory for the setting where the memory per machine $S=\tilde{O}(n^\epsilon)$ words. Using this amount of total memory keeps the exposition relatively simple. We have not investigated if our current results can be achieved with linear total memory.

1.3 Other Related Work

Besides the work reviewed earlier in the introduction, which is directly related to the present paper, there is other work that is more broadly relevant. Censor-Hillel, Parter, and Schwartzman [5] were the first to take advantage of all-to-all communication in order to derandomize symmetry breaking algorithms using the method of conditional expectations. They applied this approach to derandomize MIS algorithms in the CONGESTED–CLIQUE model. Much of the subsequent work on derandomization in the CONGESTED–CLIQUE and MPC model follows the overall approach of [5]. While this paper focuses on symmetry breaking problems, there has also been significant effort in derandomizing algorithms for other types of problems in all-to-all communication models, such as $(2k-1)$-spanner construction [5, 22].

We remark that our sparsification plus derandomization technique, which combines randomized sparsification algorithms in [16, 17] with derandomization techniques in all-to-all communication settings [5, 13], resembles (at least superficially) the sparsification plus derandomization strategy of Maus, Peltonen and Uitto [20]. Their algorithm [20] computes a β-ruling set in the CONGEST model in a setting where the communication network is a graph G, but the input graph is a power graph G^k of G.

There have also been exciting developments in designing deterministic distributed algorithms for symmetry breaking problems in the classical models of distributed computing, such as LOCAL and CONGEST. This line of work builds on the breakthrough deterministic algorithm for network decomposition, which runs in polylogarithmic rounds, as introduced by Rozhoň and Ghaffari [23].

1.4 Notation

For a node v in graph G, we denote its non-inclusive neighborhood by $\mathtt{Nbr}_G(v)$ and its inclusive neighborhood by $\mathtt{Nbr}^+_G(v) = \mathtt{Nbr}_G(v)\cup\{v\}$. Similarly, for a set S of nodes, $\mathtt{Nbr}_G(S)=\bigcup_{v\in S}\mathtt{Nbr}(v)\setminus S$ represents the non-inclusive neighborhood of all nodes in S, and $\mathtt{Nbr}^+_G(S)=\bigcup_{v\in S}\mathtt{Nbr}^+_G(v)$ represents the inclusive neighborhood. Let $\deg_G(v)$ denote the degree of a node v in graph $G=(V,E)$.

2 Fast Deterministic 2-Ruling Sets

This section demonstrates that for an input graph G with maximum degree Δ, there is a deterministic 2-ruling set algorithm in the MPC model that runs in $O(\log^{1/3}\Delta\cdot\log\log\Delta+\log\log n)$ rounds using $\tilde{O}(m+n^{1+\epsilon})$ total memory.

Our deterministic 2-ruling set algorithm for a graph $G=(V,E)$ consist of two parts. In Part 1, we sparsify the input graph deterministically by identifying a dominating set U of graph G, i.e., a set U satisfying $\mathtt{Nbr}^+_G(U)=V$. Subsequently, in Part 2, the graph $G[U]$ is given as input to a deterministic MIS algorithm in the MPC model. The independent set output by this algorithm serves as a 2-ruling set for G. For Part 2, we directly use the MIS algorithms in [7, 10]. Our main contribution in this section is in showing that a dominating set U can be found efficiently and deterministically

with the property the maximum degree of $G[U]$ is small. Bounding the maximum degree of $G[U]$ is critical because the running time of the MIS algorithms (used in Part 2) is determined by this quantity.

A high-level overview of our algorithm is as follows; details appear in subsequent subsections. We begin with a modified version of the randomized sparsification algorithm of Kothapalli, Pai, and Pemmaraju [16], depicted as Algorithm 1. This algorithm consists of $\lceil \log_f \Delta \rceil$ iterations, for some positive parameter f, which translates into $O(\log_f \Delta)$ MPC rounds. Next, we partition the $\lceil \log_f \Delta \rceil$ iterations of Algorithm 1 into *batches* consisting of t iterations each, where t is a positive integer parameter. We show how to "compress" these t iterations in a batch so that a batch can be executed in $O(\log t)$ MPC rounds, instead of $O(t)$ MPC rounds. This "compressed" randomized sparsification algorithm appears in Algorithm 2. Finally, we apply the derandomization technique of Giliberti and Parsaeian [13] to the "compressed" randomized sparsification algorithm and finish off Part 1.

2.1 The Randomized Sparsification Algorithm

Algorithm 1 is a modified version of DegOrderedSparsify by Kothapalli, Pai, and Pemmaraju [16]. We call it RandSparsify because it no longer processes nodes by degrees and to emphasize the fact that it is randomized. This algorithm will serve as the skeleton of our proposed algorithm.

This algorithm constructs a subset of vertices U that dominates all vertices in the graph (i.e., $\mathtt{Nbr}^+(U) = V$) in $O(\log_f \Delta)$ rounds with the property that the maximum degree in $G[U]$ is $O(f \log n)$ with high probability. Initially, all nodes are active. In iteration k, each active node is sampled with probability $f^k \cdot c \ln n/\Delta$ for some constant c. Sampled nodes are placed in U_k. The set U_k is added to U and all the nodes in U_k and their neighbors are deactivated, i.e., subsequently excluded from consideration in future iterations. Pseudocode is presented in Algorithm 1.

Algorithm 1: RandSparsify [16]

Input : Graph $G = (V, E)$ with maximum degree Δ, parameter f

Output : a dominating set U on G

$U \leftarrow \emptyset, V_0 \leftarrow V$; `// Initially all nodes are active`

for *iteration k = 1 to* $\lceil \log_f \Delta \rceil$ **do**

  $U_k \leftarrow$ sample $v \in V_{k-1}$ with prob. $f^k \cdot c \ln n/\Delta$

  $V_k \leftarrow V_{k-1} \setminus \mathtt{Nbr}^+(U_k)$

  $U \leftarrow U \cup U_k$

end

Notation: For any node $v \in V_{k-1}$, we will use $\sigma_k(v)$ to denote the output at node v after iteration k of Algorithm 1. Thus, $\sigma_k(v)$ can take on 3 possible values because it indicates if v belongs to U_k or to $\mathtt{Nbr}^+(U_k) \setminus U_k$ or to V_k after iteration k. We will use this notation to help in the description of our next algorithm.

2.2 The Compressed Randomized Sparsification Algorithm

Next we propose a *compressed* version of Algorithm 1. We note that this algorithm implicitly appears in [16]. The "sample-and-gather" simulation theorem from [16] describes a collection of conditions that, if satisfied by an algorithm in the CONGEST model, would lead to an efficient simulation of that algorithm in the MPC model. However, due to the general nature of these conditions, it is challenging to determine whether an algorithm satisfying them can be efficiently derandomized.

In this subsection, we explicitly state the "compressed" version of Algorithm 1 which is obtained by applying the "sample-and-gather" simulation theorem to RandSparsify. Once this algorithm is explicitly stated, we will show how to apply the derandomization technique of Giliberti and Parsaeian [13] to convert this into a deterministic algorithm. Specifically, we will show that the "compressed" algorithm can be viewed as executing a batch of iterations in parallel and we can "fix" the randomness of this entire batch in one shot.

2.2.1 ***Parallel Sampling*** . The reader is advised to refer to the pseudocode presented in Algorithm 2 while reading the following description. Firstly, to accelerate RandSparsify, rather than executing it over $\lceil \log_f \Delta \rceil$ iterations, we partition these $\lceil \log_f \Delta \rceil$ iterations into $\lceil \log_f \Delta/t \rceil$ *batches*, where each batch consists of t "compressed" iterations of RandSparsify (steps 3-7 in Algorithm 2). (If $\lceil \log_f \Delta \rceil$ is not divisible by t, the last batch may contain fewer than t iterations.) Here t is a parameter whose value will be specified later.

Specifically, batch i in Algorithm 2 corresponds to t iterations of Algorithm 1, namely iterations $(i-1)\cdot t+1, (i-1)\cdot t+2, \ldots, (i-1)\cdot t+t$. Using $V_{(i-1)\cdot t}$ to denote the set of vertices in G that are still active after batch $i-1$ has been completed, completion of batch $i-1$ corresponds to the completion of $(i-1)\cdot t$ iterations of Algorthm 1. It follows that the beginning of batch i, $G[V_{(i-1)\cdot t}]$ is determined and each node $v \in G[V_{(i-1)\cdot t}]$ knows its neighbors in $G[V_{(i-1)\cdot t}]$. Assuming that batches 1 through $i-1$ have been completed, we execute the t iterations in batch i *in parallel*. This essentially means that in batch i, an iteration $j > 1$ cannot rely on the completion of iteration $j-1$. To execute the iterations in parallel in our compressed version of Algorithm 1, we use exactly the same sampling probability as in Algorithm 1. Thus, assuming both algorithms use the same string of random bits, a node that is sampled in an iteration of Algorithm 1 will also be sampled in the corresponding iteration of Algorithm 2. However, in Algorithm 2, additional nodes may also be sampled. This happens when a node gets deactivated in an iteration of Algorithm 1 because its neighbor has been sampled, but the node is still considered active by Algorithm 2 as the exclusion operation hasn't taken place.

2.2.2 ***Gathering***. After parallel sampling in batch i is completed, the algorithm performs gathering (steps 8-9 in Algorithm 2) towards the simulation of the t iterations of Algorithm 1. For each node $v \in V_{(i-1)\cdot t}$, define $\mathbf{s}_v \in \{0,1\}^t$, as $\mathbf{s}_v[j] = 1$ iff v is sampled in iteration j in the current batch. In other words, the vector $\mathbf{s}_v$ is the indicator vector, indicating the membership of v in the t samples in the current batch. A node v is said to be *sampled* if $\mathbf{s}(v) \neq \mathbf{0}$. In

the gathering step, each node $v \in V_{(i-1)\cdot t}$, gathers the set $S(v,t)$, consisting of labeled, sampled nodes that are reachable within t hops from it, through other sampled nodes.

We precisely define the set $S(v,t)$ now. Let $R_t(v,u)$ denote a function that indicates if it is possible to travel from v to u along a path of length at most t in which every intermediate node is sampled. For any node $v \in V_{(i-1)\cdot t}$, let $\mathsf{SampNbr}(v)$ denote the set of sampled neighbors of node v. Note that all sampled neighbors also belong to $V_{(i-1)\cdot t}$ because only nodes in $V_{(i-1)\cdot t}$ are sampled. Every node $v \in V_{(i-1)\cdot t}$ gives itself the label

$$L_v = (ID_v, \mathbf{s}_v, \{ID_w \mid w \in \mathsf{SampNbr}(v)\}).$$

Then

$$S(v,t) = \{L_u \mid \mathbf{s}_u \neq \mathbf{0}, R_t(v,u) = 1\}.$$

2.2.3 ***Simulation***. Utilizing the labeled set $S(v,t)$ of nodes, node v locally simulates (steps 10-13 in Algorithm 2) iterations $(i-1)\cdot t+1, (i-1)\cdot t+2, \ldots, (i-1)\cdot t+t$ of Algorithm 1 and computes its output $\sigma_{(i-1)\cdot t+j}(v)$ after each iteration $(i-1)\cdot t+j$ of Algorithm 1. Note that the output of a node indicates whether it joins $U_{(i-1)\cdot t+j}$, becomes inactive because a neighbor has joined $U_{(i-1)\cdot t+j}$, or remains active. This completes the final steps corresponding to t iterations of Algorithm 1 (in *compressed* manner). We now present formal argument on adherence to Algorithm 1 using lemma 5 and lemma 6. Additionally, in Lemma 7, we give a result on maximum degree of active graph.

Lemma 5. *Assume both Algorithm 1 and 2 use the same string of random bits. Let $1 \le k \le \lceil \log_f \Delta \rceil$, $1 \le i \le \lceil \log_f \Delta / t \rceil$, $1 \le j \le t$ be integers such that $k = (i-1)\cdot t + j$, then $U_k \subseteq V^{sub}_{(i-1)\cdot t+j}$, where U_k is returned after Step 3 in Algorithm 1 and $V^{sub}_{(i-1)\cdot t+j}$ is returned after Step 5 in Algorithm 2.*

Proof. It is observed that the set of active nodes V_{k-1} in the RandSparsify is a subset of $V_{(i-1)\cdot t}$ in the CompRandSparsify. Since they are sampled with the same probability, $U_k \subseteq V^{sub}_{(i-1)\cdot t+j}$, where $k = (i-1)\cdot t + j$ denote the same iteration in Algorithm 1. □

Lemma 6. *Assume both Algorithm 1 and 2 use the same string of random bits. Let $1 \le k \le \lceil \log_f \Delta \rceil$, $1 \le i \le \lceil \log_f \Delta / t \rceil$, $1 \le j \le t$ be integers. The set U_k computed in Step 3 in Algortihm 1 is identical to the set $U_{(i-1)\cdot t+j}$ computed in Step 12 in Algorithm 2 whenever $k = (i-1)\cdot t + j$.*

Proof. We first claim that starting from the same active graph, within t rounds, U from Algorithm 1 and from Algorithm 2 are identical. Let $U^{(1)}$ denote U from Algorithm 1 and $U^{(2)}$ denote U from Algorithm 2. We prove this claim by induction. In the first round, the active graph is the same, and the active nodes are sampled with the same probability. Since this is the first round, the simulation does not need to consider the future rounds. Thus, U_1 from both algorithms are identical. Assume this claim is true for l rounds, and we will show that this claim is true for $l+1$ round. After $l+1$ rounds, we have $U^{(1)}_{l+1} \subseteq U^{(2)}_{l+1}$ since any node $u \in U^{(1)}_{l+1}$ is also sampled in Algorithm 2, and u must not be the neighbor of previous U's and will not be deleted at simulation. Now, we show $U^{(2)}_{l+1} \subseteq U^{(1)}_{l+1}$. First, we have $U^{(1)}_{l+1} \subseteq V^{sub}_{l+1}$ (Lemma 5). We can show that any node $u \in V^{sub}_{l+1} - U^{(1)}_{l+1}$ is not in $U^{(2)}_{l+1}$. Since u is sampled but not in $U^{(1)}_{l+1}$, such u must be inactive at the start of round $l+1$ and will be deleted in the simulation. Since $U^{(2)}_{l+1} \subseteq V^{sub}_{l+1}$, $U^{(2)}_{l+1} \subseteq U^{(1)}_{l+1}$.

Now, we can prove the lemma by induction. For $i = 1$, we have $k = j$. Since both algorithms start from the original graph, the base case holds according to the above claim. Suppose $i = x$, the lemma still holds. When $i = x+1$, since both algorithms still start from the same active graph, the lemma follows. □

Lemma 7. *The maximum degree of active graph $G[V_{i\cdot t}]$ is $\Delta/f^{i\cdot t}$.*

Proof. Based on lemma 6, we have $U_{(i-1)\cdot t+j} = U_k$, where $k = (i-1)\cdot t + j$. It follows that $V_{(i-1)\cdot t+j} = V_k$. Consequently, when $k = i\cdot t$, the maximum degree of the active graph in both algorithms is identical. Since the maximum degree in Algorithm 1 falls by a factor of f in each iteration, the lemma follows. □

After the simulation steps, nodes join $U_{(i-1)\cdot t+j}$ and deactivate themselves, along with their neighbors (steps 14-15).

Algorithm 2: CompRandSparsify

Input : Graph $G = (V, E)$ with maximum degree Δ, parameters f, t

Output : a dominating set U on G

1 $U \leftarrow \emptyset, V_0 \leftarrow V$; `// Initially all nodes are active`
2 **for** *batches* $i = 1$ *to* $\lceil \frac{\log_f \Delta}{t} \rceil$ **do**
3 **for** $j \in [1,t]$ *in parallel* **do**
  `// Parallel Sampling. See also Section 2.2.1`
4 Sample each node $v \in G[V_{(i-1)\cdot t}]$ with prob. $f^{(i-1)\cdot t+j} \cdot c \ln n/\Delta$
5 $V^{sub}_{(i-1)\cdot t+j} \leftarrow \{v \mid v$ is sampled in Line 4$\}$
6 $\mathbf{s}_v[j] \leftarrow 1$ if node $v \in V^{sub}_{(i-1)\cdot t+j}$
7 **end**
  `// Gathering. See also Section 2.2.2`
8 Each node $v \in V_{(i-1)\cdot t}$ gives itself the label L_v
9 Each node $v \in V_{(i-1)\cdot t}$ gathers $S(v,t)$
10 **for** $j \in [1,t]$ *in parallel* **do**
  `// Simulation. See also Section 2.2.3`
11 Each node $v \in V_{(i-1)\cdot t}$ performs local computation using $S(v,t)$ to simulate Algorithm 1 and compute its output $\sigma_{(i-1)\cdot t+j}(v)$
12 Each node $v \in V_{(i-1)\cdot t}$ joins $U_{(i-1)\cdot t+j}$ if $\sigma_{(i-1)\cdot t+j}(v)$ indicates that it belongs to $U_{(i-1)\cdot t+j}$
13 **end**
14 $V_{i\cdot t} \leftarrow V_{(i-1)\cdot t} \setminus \cup_{j=1}^{t} \mathsf{Nbr}^+(U_{(i-1)\cdot t+j})$
15 $U \leftarrow U \cup (\cup_{j=1}^{t} U_{(i-1)\cdot t+j})$
16 **end**

2.3 MPC Implementation

To keep exposition clear, the description of Algorithm 2 is node-centric, i.e. it specifies actions performed by each node v in G in each iteration. In the MPC model, each node in the graph is assigned to

a specific machine, which is responsible for storing and managing that node as well as handling the associated computations and communication. This machine is aware of the node's neighbors and the machines that store those neighboring nodes. A machine may host multiple nodes, provided it stays within its local memory capacity.

Algorithm 2 involves local sampling (Step 3-7 in Algorithm 2), gathering (Step 8-9 in Algorithm 2), local simulation (Step 10-13 in Algorithm 2) and updating the set of active nodes and the dominating set (Step 14-15 in Algorithm 2) in each batch. We will focus on implementing the gathering step within the MPC model since all other steps can be implemented using local computation and don't need any communication.

There are $O(\log_f \Delta/t)$ batches. Let f and t be fixed as follows: $f := 2^{\epsilon/4\log^{1/3}\Delta}$ and $t := \lceil \log^{1/3}\Delta \rceil$. Thus, there are $O(\log^{1/3}\Delta)$ batches. In each batch, active nodes need to identify whether each neighbor has been sampled in each round of the current batch. Specifically, each active node v needs to gather $S(v,t)$. Here, we do the gathering with the assumption that each machine hosts exactly one node. In this setting, for each vertex, we can gather n^ϵ information. This gathering step can be implemented appropriately because we bound the size of $S(v,t)$. These bounds are proved in the next section in Lemmas [9,12]. Therefore this gathering step can be executed in $O(\log\log\Delta)$ rounds in the sublinear memory MPC model with total memory $\tilde{O}(m+n^{1+\epsilon})$.

2.4 Analysis

Lemma 8. *The number of bits required to represent the set $S(v,t)$ is $O\left(\left(\sum_{j=1}^{t} f^j \log n\right)^t\right)$.*

Proof. In a batch i, consider an arbitrary node $v \in V_{t(i-1)}$ and $u \in S(v,t)$ and an iteration $1 \le j \le t$. Any node in $V_{t(i-1)}$ is sampled with probability $f^{t(i-1)+j} c \ln n/\Delta$. Since the maximum degree in the active graph $G[V_{(i-1)\cdot t}]$ is at most $\Delta/f^{(i-1)\cdot t}$ (Lemma 7), the expected number of sampled neighbors of u in iteration j is at most $c\log n \cdot f^j$. Furthermore, since in any iteration nodes are sampled independently, by Chernoff bounds, the number of neighbors of u sampled in iteration j is $O(\log n \cdot f^j)$ with high probability. By the union bound, the number of sampled neighbors of u over all t iterations in the batch is $O(\log n \cdot \sum_{j=1}^{t} f^j)$ with high probability. Similarly, the neighbors of u that are sampled have $O(\log n \cdot \sum_{j=1}^{t} f^j)$ sampled neighbors whp. Extending this to a t-hop neighborhood from v, the number of sampled neighbors of v within t-hop is $O\left(\left(\sum_{j=1}^{t} f^j \log n\right)^t\right)$. □

Recall that in the MPC implementation (Section 2.3), we have fixed $f = 2^{\epsilon/4\log^{1/3}\Delta}$ and $t = \lceil \log^{1/3}\Delta \rceil$. Then, the number of bits required to represent $S(v,t)$ can be bounded as follows.

Lemma 9. *The number of bits representing $S(v,t)$ is $O(n^{\epsilon/2})$.*

Proof. According to Lemma 8, the number of bits representing $S(v,t)$ is $O\left(\left(\sum_{j=1}^{t} f^j \log n\right)^t\right)$. The sum $\sum_{j=1}^{t} f^j$ is a geometric series with a common ratio f. Simplifying this expression and substituting the value of f and t, we can show that

$$\left(\sum_{j=1}^{t} f^j \log n\right)^t = \left(\log n \cdot \frac{f(1-f^t)}{1-f}\right)^t \le (\log n \cdot f^t)^t = O(n^{\epsilon/2}).$$

□

Lemma 10. *$S(v,t)$ can be gathered at M_v in at most $O(\log t)$ rounds.*

Proof. The assumption of MPC implementation (Section 2.3) is that every machine M_v hosts exactly one node v. Since $S(v,t) = O(n^{\epsilon/2})$ (Lemma 9), $S(v,t)$ contains $O(n^{\epsilon/2})$ nodes. Therefore, M_v needs to send a total of $O(n^{\epsilon/2}) \times O(n^{\epsilon/2}) = O(n^\epsilon)$ words. This communication can be performed in $O(1)$ rounds. Thus, nodes can learn $S(v,t)$ in $O(\log t)$ rounds by graph exponentiation technique. [11, 12, 14, 22]. □

Informally speaking, the reason we gather $S(v,t)$ at each node v is that in order to determine the output of v in the j-th iteration in the future, i.e., $\sigma_{t(i-1)+j}(v)$, we need to know the outputs of the sampled neighbors of v, $w \in \mathsf{SampNbr}(v)$, in $(j-1)$-th iteration of the future, i.e. $\sigma_{t(i-1)+j-1}(w)$. This can be proved precisely, by induction, in a manner very similar to Claim 5 in [16]. This leads to the following lemma.

Lemma 11. *In the batch i, for any node $v \in V_{(i-1)\cdot t}$, the information of $S(v,t)$ is enough to locally compute $\sigma_{t\cdot(i-1)+j}(v)$, where $j \in [1,t]$.*

Lemmas 8, 9, 10, 11 together proved the following lemma.

Lemma 12. *Suppose every machine hosts one node, and given $f = 2^{\epsilon/4\log^{1/3}\Delta}$ and $t = \lceil \log^{1/3}\Delta \rceil$, then each batch can be simulated in $O(\log\log\Delta)$ rounds in the sublinear memory MPC model with a total memory $\tilde{O}(m+n^{1+\epsilon})$.*

Lemma 13. *The maximum degree of U is $O(f \cdot \log n)$.*

Proof. Since the Algorithm 2 CompRandSparsify simulates the Algorithm 1 RandSparsify (Lemma 6), the maximum degree of U is the same. □

Theorem 14. *Given a graph $G = (V,E)$, the algorithm 2 can be implemented in the sublinear memory MPC model in $O(\log^{1/3}\Delta \log\log\Delta)$ rounds with $\tilde{O}(m+n^{1+\epsilon})$ total memory.*

Proof. This algorithm runs in $O(\log^{1/3}\Delta)$ batches (Section 2.3). As shown in Lemmas [12] shown, gathering the sampled subgraphs requires $O(\log\log\Delta)$ rounds in each batch, while the other steps involve only local computations. Thus computing U requires $O(\log^{1/3}\Delta \cdot \log\log\Delta)$ rounds with $\tilde{O}(m+n^{1+\epsilon})$ total memory. □

2.5 Deterministic Compressed Sparsification

In this section, we introduce a deterministic version of Algorithm 2. A key component of our approach is Lemma 15 in Giliberti and Parsaeian [13]. Let $H = (V_H, E_H)$ be a graph with maximum degree Δ_H. Let f be a positive parameter satisfying some mild constraints specified in the lemma statement. This lemma describes a deterministic "sampling" method to identify a subset V^{sub} of nodes adjacent

to all "high degree" nodes in H, i.e., nodes with degree at least Δ_H/f. Furthermore, V^{sub} satisfies the property that the maximum degree of $H[V^{sub}]$ is $2^{O(\log f)}$. Finally, the deterministic "sampling" method the lemma promises runs in $O(\log\log\Delta_H)$ rounds in the sublinear memory MPC model. We state the lemma precisely now. Our statement of the lemma is a slight modification of the main lemma in Giliberti and Parsaeian [13, Lemma 4.3] for degree-based graph sparsification in sublinear MPC regime. It is important to mention that the proof of [13, Lemma 4.3] uses two lemmas: the first lemma, [13, Lemma 4.2] reduces the maximum degree in the input graph to $O(n^\epsilon)$, and the second lemma, [13, Lemma 4.1] process the graph with maximum degree bounded by $O(n^\epsilon)$.

Lemma 15 (Giliberti and Parsaeian [13]). *Let $H = (V_H, E_H)$ be a graph with maximum degree Δ_H. Let f be a parameter satisfying $f \le \frac{\Delta^{0.4}}{\log n}$ and $f \ge$ poly$(\log n)$. Let U denote the set of nodes in H with degree at least $\frac{\Delta}{f}$. There exists a deterministic sublinear memory MPC algorithm that computes, in $O(\log\log\Delta_H)$ rounds, a subset $V^{sub} \subseteq V$ such that for any node $u \in U$ it holds that $|\mathtt{Nbr}(u)\cap V^{sub}| \in [1, 2^{O(\log f)}]$. The algorithm's global space is linear in the input size.*

We utilize this lemma as a black box. Let DetSparsify denote the algorithm promised by Lemma 15. We then modify Algorithm 2 by replacing the randomized sampling steps (Steps 4-5) by an appropriate call to DetSparsify. The modification to Steps 4-5 in Algorithm 2 is highlighted in the box below.

Algorithm 2, Steps 4-5 replaced by the following:

$$V^{sub}_{(i-1)\cdot t+j} \leftarrow \textsc{DetSparsify}\left(H_i = G[V_{t(i-1)}],\ \text{parameter } f^j\right)$$

We now turn to analyzing the deterministic version of Algorithm 2.

Theorem 16. *Given a graph $G = (V, E)$, there is a deterministic sublinear memory MPC algorithm that computes a dominating set $U \subseteq V$, satisfying the property that the maximum degree of $G[U]$ is $2^{O(\log^{1/3}\Delta)}$. This algorithms runs in $O(\log^{1/3}\Delta\cdot\log\log\Delta)$ rounds using $\tilde{O}(m + n^{1+\epsilon})$ total memory.*

Proof. There are $O(\log^{1/3}\Delta)$ batches (Section 2.3). In each batch, the deterministic sampling step requires $O(\log\log\Delta)$ rounds (see Lemma 15), and the gathering step also requires $O(\log\log\Delta)$ rounds (see Lemma 12). All other steps are local. Thus, the theorem follows. □

2.6 Finishing off the 2-Ruling Set Computation

As Theorem 16 shown, we obtain a dominating set U deterministically. We can then get a 2-ruling set of G by executing the deterministic MIS algorithm from [7] on $G[U]$. The fact that the maximum degree of $G[U]$ is bounded by $2^{O(\log^{1/3}\Delta)}$ implies that the MIS algorithm runs in $O(\log^{1/3}\Delta + \log\log n)$ rounds. The theorem below follows.

Theorem 1. *There is a deterministic sublinear memory MPC algorithm that computes a 2-ruling set in $O(\log^{1/3}\Delta\cdot\log\log\Delta + \log\log n)$ rounds using $O(m + n^{1+\epsilon})$ total memory.*

3 Fast β-Ruling Set Algorithms

In the previous section, we developed an efficient deterministic sublinear memory MPC algorithm for 2-ruling sets. Let Det2RulingSet denote the algorithm promised by Theorem 1. Now, we generalize this algorithm to create a sublinear memory MPC algorithm suitable for any integer $\beta \ge 2$.

The strategy involves iterating Det2RulingSet for β-1 times, following the approach outlined in [3, 16]. Initially, we execute Det2RulingSet with parameters f_1 and t_1, yielding a dominating set U_1 of $G[U_0]$, i.e., $\mathtt{Nbr}^+(U_1) = U_0$. After this step, the subgraph $G[U_1]$ has a maximum degree of at most $O(f_1\cdot\log n)$, as indicated by Lemma 13. Subsequently, we apply Det2RulingSet to $G[U_1]$ with parameters f_2 and t_2, resulting in a dominating set U_2 of $G[U_1]$, i.e., $\mathtt{Nbr}^+(U_2) = U_1$. Thus, all the nodes $U_0 = V$ are at most 2 hops away from the nodes in U_2. This process continues for β-1 phases, concluding with $G[U_{\beta-1}]$ having a maximum degree of $O(f_{\beta-1}\cdot\log n)$. It is clear that all nodes $U_0 = V$ are at most β-1 hops away from the nodes in $U_{\beta-1}$. Next, we utilize a sublinear memory MPC implementation of a MIS algorithm on $G[U_{\beta-1}]$. The outcome of this step effectively serves as a β-ruling set for the original graph G. For the pseudocode, refer to Algorithm 3.

Algorithm 3: β-RulingSet

Input : Graph $G = (V, E)$ with maximum degree Δ, parameters $f_i, t_i, i \in [1, \beta\text{-}1]$
Output : a β-ruling set on G

$U_0 \leftarrow V$
for *i = 1 to β - 1* **do**
  $U_i \leftarrow$ Det2RulingSet$(G[U_{i-1}], f_i, t_i)$
end

Theorem 2. *A β-ruling set of a graph $G = (V, E)$ can be computed deterministically in the sublinear memory MPC model within $O(\log^{1/2^\beta-1}\Delta\log\log\Delta + \log\log n)$ rounds with $O(m + n^{1+\epsilon})$ total memory.*

Proof. Let $f_0 = \Delta$. Let $f_i = 2^{\epsilon/4\log^{\frac{2^\beta-2^{i+1}+1}{2^\beta-1}}\Delta}$, and $t_i = \log^{\frac{2^i-1}{2^\beta-1}}\Delta$, where $i \in [1, \beta-1]$. According to Theorem 16 and Theorem 1, the running time of Algorithm 3 should be:

$$\sum_{i=1}^{\beta-1}\frac{\log f_{i-1}}{t_i\cdot\log f_i}\cdot\log t_i\log\log\Delta + \log f_i + \log\log n$$
$$= O(\log^{\frac{1}{2^\beta-1}}\Delta\log\log\Delta + \log\log n).$$

□

4 Fast β-Ruling Set on Bounded Arboricity Graphs

Note that Fischer, Giliberti, and Grunau [10] present a deterministic sublinear MPC algorithm with linear total memory that computes

an independent set I of a graph G with λ-arboricity in $O(\log\log n)$ rounds. The resulting graph $G \setminus I$ satisfies $\deg(G \setminus I) \leq \text{poly}(\lambda)$. It is clear that $\deg(G \setminus \mathtt{Nbr}^+(I)) \leq \text{poly}(\lambda)$. By applying Theorem 1 to the graph $G \setminus \mathtt{Nbr}^+(I)$, we achieve a faster deterministic sublinear MPC algorithm for computing a 2-ruling set.

Theorem 3. *There exists a deterministic sublinear MPC algorithm that computes a 2-ruling set of a graph with arboricity λ in $O(\log^{1/3}\lambda \cdot \log\log\lambda + \log\log n)$ rounds using $\tilde{O}(m + n^{1+\epsilon})$ total memory.*

Similarly, by applying Theorem 2 to the graph $G \setminus \mathtt{Nbr}^+(I)$, we achieve a faster deterministic sublinear MPC algorithm for computing a β-ruling set.

Theorem 4. *A β-ruling set of a graph with arboricity λ can be computed deterministically in $O(\log^{1/2^\beta - 1}\lambda \log\log\lambda + \log\log n)$ rounds in the sublinear memory MPC model with $\tilde{O}(m + n^{1+\epsilon})$ total memory.*

5 Conclusions and Future Work

In this paper, we improve upon the 2-ruling set result of Giliberti and Parsaeian [13] by presenting a deterministic 2-ruling set algorithm in the sublinear memory MPC model that runs in $\tilde{O}(\log^{1/3}\Delta)$ rounds. By combining this result with the recent result of Fischer, Giliberti, and Grunau [10], we also obtain faster, deterministic 2-ruling set algorithms for sparse (i.e., bounded arboricity) graphs. Additionally, we present results for β-ruling sets for $\beta > 2$ in both general graphs and sparse graphs. In our view, our technique is more important than the specific results we obtained. We have successfully derandomized the "sample-and-gather" approach from [16], which, as noted by Giliberti and Parsaeian [13], requires fixing the randomness in future iterations.

This raises an intriguing question: can we apply these ideas to achieve a faster derandomization of MIS algorithms in the MPC model? Currently, the fastest (randomized) MIS algorithm in the sublinear memory MPC model, running in $\tilde{O}(\sqrt{\log\Delta})$ rounds, is due to Ghaffari and Uitto [12]. In contrast, the fastest, deterministic MIS algorithm runs in $\tilde{O}(\log\Delta)$ rounds [7]. As observed in [16], the Ghaffari-Uitto algorithm can also be viewed as an instance of the "sample-and-gather" approach. Could the ideas presented in this paper be applied to the Ghaffari-Uitto algorithm to obtain a deterministic MIS algorithm in the sublinear MPC model that runs in $o(\log\Delta)$ rounds?

Acknowledgments

Hongyan Ji and Sriram V. Pemmaraju were partially supported during this project by NSF grant 1955939. The work of Ajitanshu Singh and Kishore Kothapalli is partially supported by a grant from the Ministry of Education, Govt. of India, under the National Mission on Education through Information and Communication Technology, Project titled Virtual Labs (Ph III) vide Ref. No: F.16-13/2017-TEL/2022/29.

References

[1] Noga Alon, László Babai, and Alon Itai. 1986. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *J. Algorithms* 7, 4 (1986), 567–583.

[2] Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M Karp, and Jara Uitto. 2019. Massively parallel computation of matching and MIS in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. Association for Computing Machinery, New York, NY, United States, 481–490.

[3] Tushar Bisht, Kishore Kothapalli, and Sriram Pemmaraju. 2014. Brief announcement: Super-fast t-ruling sets. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing* (Paris, France) *(PODC '14)*. Association for Computing Machinery, New York, NY, USA, 379–381. https://doi.org/10.1145/2611462.2611512

[4] Mélanie Cambus, Fabian Kuhn, Shreyas Pai, and Jara Uitto. 2023. Time and Space Optimal Massively Parallel Algorithm for the 2-Ruling Set Problem. In *37th International Symposium on Distributed Computing (DISC 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 281)*, Rotem Oshman (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 11:1–11:12. https://doi.org/10.4230/LIPIcs.DISC.2023.11

[5] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. 2020. Derandomizing local distributed algorithms under bandwidth restrictions. *Distributed Comput.* 33, 3-4 (2020), 349–366. https://doi.org/10.1007/S00446-020-00376-1

[6] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan. 2015. One trillion edges: Graph processing at facebook-scale. *Proc. VLDB Endow.* 8 (2015), 1804–1815.

[7] Artur Czumaj, Peter Davies, and Merav Parter. 2020. Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*. Association for Computing Machinery, New York, NY, United States, 175–185.

[8] Artur Czumaj, Peter Davies, and Merav Parter. 2021. Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space. *ACM Trans. Algorithms* 17, 2 (2021), 16:1–16:27.

[9] J. Dean and S. Ghemawat. 2008. Mapreduce: Simplified data processing on large clusters. *Commun. ACM* 51 (2008), 107–113.

[10] Manuela Fischer, Jeff Giliberti, and Christoph Grunau. 2023. Deterministic massively parallel symmetry breaking for sparse graphs. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*. Association for Computing Machinery, New York, NY, United States, 89–100.

[11] Mohsen Ghaffari. 2017. Distributed MIS via all-to-all communication. In *Proceedings of the ACM symposium on principles of distributed computing*. Association for Computing Machinery, New York, NY, United States, 141–149.

[12] Mohsen Ghaffari and Jara Uitto. 2019. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Society for Industrial and Applied Mathematics, USA, 1636–1653.

[13] Jeff Giliberti and Zahra Parsaeian. 2024. Massively Parallel Ruling Set Made Deterministic (See also https://arxiv.org/abs/2406.12727). In *Proceedings of the 2024 ACM Symposium on Principles of Distributed Computing*. Association for Computing Machinery, New York, NY, United States, 523–526.

[14] James W Hegeman, Sriram V Pemmaraju, and Vivek B Sardeshmukh. 2014. Near-constant-time distributed algorithms on a congested clique. In *International Symposium on Distributed Computing*. Springer, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 514–530.

[15] H. Karloff, S. Suri, and S. Vassilvitskii. 2010. A model of computation for mapreduce. In *in Proc. ACM SODA*. Society for Industrial and Applied Mathematics3600 University City Science Center, Philadelphia, PA, United States, 938–948.

[16] Kishore Kothapalli, Shreyas Pai, and Sriram V Pemmaraju. 2020. Sample-and-Gather: Fast Ruling Set Algorithms in the Low-Memory MPC Model. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, Vol. 182. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Oktavie-Allee, Wadern, Germany, 28–1.

[17] Kishore Kothapalli and Sriram Pemmaraju. 2012. Super-Fast 3-Ruling Sets. In *32nd International Conference on Foundations of Software Technology and Theoretical Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Oktavie-Allee, Wadern, Germany, 136.

[18] M Luby. 1985. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, New York, NY, USA, 1–10.

[19] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proc. of ACM SIGMOD*. Association for Computing Machinery, New York, NY, United States, 135–146.

[20] Yannic Maus, Saku Peltonen, and Jara Uitto. 2023. Distributed Symmetry Breaking on Power Graphs via Sparsification. In *Proceedings of the ACM symposium on principles of distributed computing*. Association for Computing Machinery, New York, NY, United States, 157–167.

[21] Shreyas Pai and Sriram V Pemmaraju. 2022. Brief Announcement: Deterministic Massively Parallel Algorithms for Ruling Sets. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. Association for Computing Machinery, New York, NY, United States, 366–368.

[22] Merav Parter and Eylon Yogev. 2018. Congested Clique Algorithms for Graph Spanners. In *32nd International Symposium on Distributed Computing*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 3.

[23] Václav Rozhoň and Mohsen Ghaffari. 2020. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing* (Chicago, IL, USA) *(STOC 2020)*. Association for Computing Machinery, New York, NY, USA, 350–363. https://doi.org/10.1145/3357713.3384298

[24] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, M. J. Franklin S. Venkataraman, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. 2016. Apache spark: A unified engine for big data processing. *Commun. ACM* 59 (2016), 56–65.