

Deterministic Massively Parallel Algorithms for Graph Problems

Master Thesis

Author(s):

Giliberti, Jeff

Publication date:

2022

Permanent link:

<https://doi.org/https://doi.org/10.3929/ethz-b-000573443>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Deterministic Massively Parallel Algorithms for Graph Problems

Master Thesis

Jeff Giliberti

September 14, 2022

Advisors: Prof. Dr. Mohsen Ghaffari, Dr. Manuela Fischer, Christoph Grunau

Department of Computer Science, ETH Zürich

Abstract

The model of Massively Parallel Computation (MPC) has emerged as a standard theoretical abstraction for the algorithmic study of large-scale graph processing. A long line of work in this setting has shown that many fundamental graph problems admit fast randomized solutions. Yet, only recently, has a derandomization technique been developed to reduce the gap between randomized and deterministic algorithms.

One of the main problems now in this area is to understand which graph problems admit fast deterministic solutions. A key challenge is not only to obtain round complexities that match randomized ones but also to bring global memory and total computation guarantees closer to those of randomized algorithms.

This thesis studies the deterministic complexity of several fundamental graph problems in the sublinear and linear regimes of Massively Parallel Computation: *Connectivity*, *Maximal Independent Set (MIS)*, *Maximal Matching (MM)*, and *Vertex Coloring*.

Connectivity: We present the first computation-efficient deterministic algorithm for general graphs in the strongly sublinear memory regime of MPC. Our algorithm simplifies the best-known deterministic algorithm and achieves nearly-optimal local computation time.

MIS and MM: We develop improved deterministic fully-scalable MPC algorithms parameterized by the sparsity of a graph, which is measured by its *arboricity*, running in $O(\log \log n)$ time on bounded arboricity graphs. Moreover, these algorithms are the first to work also using linear global memory, albeit with a slight loss in the round complexity.

Coloring: We settle the complexity of $O(\lambda)$ -coloring λ -arboricity graphs in the linear-memory MPC model by providing a strikingly simple deterministic algorithm running in a constant number of rounds. Our algorithm works in the relaxed setting where the global memory allowed is $n \cdot \text{poly}(\lambda)$. In the other setting, we derive a $O(\log \lambda)$ bound based on the problem of $(\Delta + 1)$ -list coloring.

As a key technical contribution, this paper provides tailored applications of the MPC derandomization framework based on limited independence. These applications demonstrate in a number of cases how to match randomized round complexities while providing global memory and local computation bounds resembling those of randomized approaches.

Acknowledgements

It is the trust, understanding, and help of a network of people that has made this thesis possible.

I would like to thank Mohsen Ghaffari for his guidance, helpful feedback, and several valuable discussions. I benefited greatly from his Advanced Algorithms and Principles of Distributed Computing courses at ETH Zurich. Not only were they extremely entertaining and thorough, but they also sparked my interest in the subject.

I owe an enormous thanks to my immediate supervisors Manuela Fischer and Christoph Grunau for their immense support and endless encouragement. Thank you for many insightful and inspiring conversations. Without you, this wonderful learning experience would not have been as much fun!

Further, I would like to thank Andreas Karrenbauer, who hosted me for a research internship during my studies and from whom I learned so much.

Last but not least, a special thanks to my family and friends for all the unconditional support in these intense academic years.

Contents

Contents	v
1 Introduction	1
1.1 Distributed Graph Algorithms	1
1.1.1 Graph Algorithms	3
1.1.2 Deterministic Algorithms and Derandomization	5
1.2 Our Contribution	6
1.3 Structure of the Thesis	8
2 Tools and Techniques	9
2.1 Notation and Preliminaries	9
2.2 Local Decomposition Algorithms	10
2.3 Local Simulation via Global Communication	12
2.4 Derandomization in All-to-All Models	14
2.4.1 Example of Pairwise Independent Analysis	17
3 Deterministic Work-Efficient Massively Parallel Connectivity	21
3.1 Randomized Connectivity Algorithms in a Nutshell	22
3.2 Comparison with the State-of-the-Art	23
3.3 Derandomization of Algorithmic Primitives	24
3.3.1 Approximation of Maximum Matching	24
3.3.2 Hitting Set	26
3.4 Connectivity Algorithm	29
4 Deterministic MPC Algorithms for MIS and MM	33
4.1 Arboricity-Based Degree Reduction	33
4.1.1 LOCAL Algorithmic Primitive	34
4.1.2 Derandomization and MPC Implementation	37
4.2 Pipelining of Arboricity-Based Degree Reduction	41
4.3 Near-Linear Global Space Algorithms	47

CONTENTS

4.3.1	High-Arboricity Case	48
4.3.2	Low-Arboricity Case	53
4.4	Superlinear Global Space Algorithm	57
5	Deterministic Arb-Coloring via All-to-All Communication	59
5.1	Connection with $(\Delta + 1)$ List Coloring	60
5.2	Constant-Round $O(\lambda)$ Coloring	61
6	Conclusion	65
	Bibliography	67

Introduction

1.1 Distributed Graph Algorithms

Theory of Distributed Computing is a fast-growing research field within theoretical computer science concerned with the design and analysis of algorithms for solving computational tasks in massive distributed networks. Distributed solutions are needed in a broad variety of modern computational settings such as computer and wireless networks, high-performance, cloud, and grid computing, blockchain technology, internet of things, mobile agents, swarm robotics, and many more. Besides technological systems, distributed computing has a far wider range of applications, for example, systems based on biological structures, nature-inspired models, and social groupings. One of the most fundamental and difficult tasks in these *decentralized networks* are symmetry breaking problems such as electing a leader or resolving conflicts for the allocation of shared resources.

To meaningfully abstract the operations that computing entities need to perform to break the symmetry, many parallel and distributed models have been proposed. A distributed network is usually represented by an n -vertex graph, where n is the number of entities in the network, and edges of the graph model the network's topology. In the classic synchronous message-passing model, known as the LOCAL model of computation [53], each vertex of the graph hosts an autonomous processor and each edge connects two vertices if there is a communication channel between them. The computation proceeds in synchronous rounds, and, per-round, each processor can send unbounded messages to all its neighbors. The running time of an algorithm is measured by the number of rounds of distributed communication that this algorithm requires; local computation is for free as the focus lies on communication. One also assumes that vertices have unique identifiers (IDs). At the end of the computation each node outputs its local part of the solution.

One aspect that modern algorithms need to deal with is limited memory. In recent years, in fact, memory has become a major bottleneck due to the ever-increasing amount of data. This renders many traditional graph algorithms inefficient, or even inapplicable, and demands novel distributed computational paradigms, as opposed to the unbounded space and bandwidth of the LOCAL model. Inspired by popular large-scale data-intensive applications such as MapReduce [28] and Spark [72], the Massively Parallel Computation (MPC) model [31, 45] emerged as a clean, theoretical abstraction of these computing frameworks and serves as a basis for their algorithmic study.

The MPC Model In the MPC model, the distributed network consists of M machines with memory S each. The input is distributed across the machines and the computation proceeds in synchronous rounds. In each round, each machine performs an arbitrary *local computation* and then communicates up to S data. All messages sent and received by each machine in each round have to fit into the machine’s local space. The main complexity measure of an algorithm is its *round complexity*, that is, the number of rounds needed by the algorithm to solve the problem. Secondary complexity measures are the *global memory usage*—i.e., the number of machines times the memory per machine required—as well as the *total computation* required by the algorithm, i.e., the (asymptotic) sum of the local computation performed by each machine.

We primarily focus on the design of fully scalable graph algorithms in the *low-memory* MPC model, where each machine has strongly sublinear memory. More precisely, an input graph $G = (V, E)$, with n vertices and m edges, is distributed arbitrarily across machines with local memory $S = O(n^\delta)$ each, for some constant $0 < \delta < 1$, so that the global space is $S_{Global} = \Omega(n + m)$. Secondly, we consider the *linear-memory* MPC model, in which each machine has local memory $S = O(n)$. This model is closely linked to the CONGESTED-CLIQUE one, first introduced by Lotker et al. [56]. There, the communication graph is complete but the size of messages is restricted to $O(\log n)$ bits. By a beautiful result of Lenzen [51], linear-space MPC results apply also to CONGESTED-CLIQUE under certain mild conditions [12].

Sparsity as a Complexity Measure As many real-world graphs are believed to be sparse, it would be desirable to develop algorithms that improve with the sparsity of graphs [27]. A well-received measure of sparsity is the *arboricity* λ of a graph: It captures a *global* notion of sparsity as it does not impose any *local* constraints to the graph like bounded maximum degree. Bounded arboricity graphs include many natural graph families like trees, planar graphs, graphs avoiding a fixed minor, bounded genus and bounded treewidth graphs. The study of *arboricity-dependent* algorithms was initiated by Barenboim and Elkin [6] and has attracted a lot of attention ever since in models of both distributed and parallel computing [7, 9, 10, 16, 35, 38, 39, 50].

These results show that the concept of arboricity covers a wide range of graphs for which far more efficient algorithms than for general graphs exist.

1.1.1 Graph Algorithms

In the sublinear regime of MPC, fundamental graph combinatorial and optimization problems have recently gained a lot of attention. There is a plethora of work on the problems of connectivity, matching, maximal independent set, coloring, vertex cover, and many more (see, e.g., [11, 13, 22, 25, 26, 35, 40]). We study the problem of connectivity in general graphs and the problems of maximal independent set (MIS), maximal matching (MM), and vertex coloring parameterized by the *arboricity*, although without imposing any upper bound on the value of λ . Usually algorithmic approaches as well as round complexities may differ considerably depending on whether their success guarantee is with high probability or deterministic. Our focus is on deterministic algorithms for these four problems, which are among the most well-studied. We define the problems and survey algorithms most relevant to our work below. Henceforth, let $G = (V, E)$ be the undirected input graph.

Connectivity One particularly important graph problem that has received increasing attention over the past few years in the sublinear regime of MPC is that of connectivity. This is not only a problem of independent interest, but it serves as a subroutine for many algorithms such as finding a rooted spanning forest, a minimum spanning forest, a bottleneck spanning forest, and a 4-coloring of trees [22, 35]. Moreover, it is closely linked to the widely believed 1-vs-2 cycles conjecture. This conjecture states that distinguishing whether an input graph is an n -vertex cycle or consists of two $\frac{n}{2}$ -vertex cycles requires $\Omega(\log n)$ rounds and plays a major role in conditional lower-bounds for this setting [13, 22, 24, 37, 62, 70]. The problem is defined as follows.

Definition 1.1 (Connectivity Problem) *Find a function $cc: V \rightarrow \mathbb{N}$ such that every vertex $u \in V$ knows $cc(u)$ and for any pair of vertices $u, v \in V$, u and v are connected in G if and only if $cc(u) = cc(v)$.*

A sequence of works [3, 4, 13, 20, 49, 55] on this problem culminated in a randomized algorithm by Behnezhad et al. [13] that finds all connected components of a graph with diameter D in $O(\log D + \log \log_{\frac{m}{n}} n)$ rounds. This time complexity is nearly-optimal due to a $\Omega(\log D)$ conditional lower bound [13, 22]. In a very recent breakthrough, Coy and Czumaj [22] obtained the same round complexity with a deterministic algorithm. Their derandomization approach, however, comes at a cost of (polynomially) heavy local computation, which makes it impractical for large-scale applications.

Maximal Independent Set and Maximal Matching The complexity of finding an MIS and an MM is one of the most fundamental problems in the area of distributed computing. They have been extensively studied since the very beginning of the area [1, 9, 34, 44, 57] and are defined as follows.

Definition 1.2 (Maximal Independent Set) Find a set $\mathcal{I} \subseteq V$ such that no two vertices in \mathcal{I} are adjacent and every vertex $u \in V \setminus \mathcal{I}$ has a neighbor in \mathcal{I} .

Definition 1.3 (Maximal Matching) Find a set $\mathcal{M} \subseteq E$ such that no two edges in \mathcal{M} are adjacent and every edge $e \in E \setminus \mathcal{M}$ has a neighbor in \mathcal{M} .

Randomized MIS and MM One of the recent highlights in low-memory MPC has been a series of works on the MIS and MM problems in *bounded arboricity graphs*. Brandt et al. [16] were the first to break the linear memory barrier proving an upper bound of $O(\log^3 \log n)$ for trees. Behnezhad et al. [11] extended it to graphs of arbitrary arboricity giving a general $O(\log^2 \log n)$ -time algorithm that reduces the maximum degree of the graph to $\text{poly}(\max(\lambda, \log n))$. The subsequent algorithm of Ghaffari et al. [35] gives an improved $O(\log \log n)$ -time degree reduction. This result combined with the fastest randomized MIS and MM algorithms for general graphs [40] running in $O(\sqrt{\log \Delta} \log \log \Delta + \log \log n)$ time, where Δ denotes the maximum degree, yields a $O(\log \log n)$ -time algorithm for graphs of polylogarithmic arboricity.

Deterministic MIS and MM The fastest deterministic MIS and MM algorithms for general graphs run in $O(\log \Delta + \log \log n)$ by a result of Czumaj et al. [25]. When restricting our attention to deterministic algorithms in terms of the arboricity, only a $O(\log^2 \log n)$ -time bound for MIS and MM on trees (i.e., $\lambda = 1$) is known due to Latypov and Uitto [50].

Arboricity Coloring The problem of coloring a graph is a corner-stone *local* problem with numerous applications that has been extensively studied. The goal of a K -coloring algorithm is the following.

Definition 1.4 (K-Coloring) Find a function $\varphi: V \rightarrow \{1, \dots, K\}$ such that for every edge $\{u, v\} \in E$ vertices u and v are assigned distinct colors, i.e., $\varphi(u) \neq \varphi(v)$.

There has been a long succession of randomized and deterministic results based on recursive graph partitioning procedures in the models of CONGESTED-CLIQUE and linear-memory MPC [5, 19, 43, 66, 67]. The randomized complexity of $(\Delta + 1)$ -Coloring and its $(\Delta + 1)$ -list-coloring generalization was settled to $O(1)$ time in a breakthrough by Chang et al. [19]. Subsequently, Czumaj et al. [23] proved the same result deterministically.

Much of the research on this topic has focused on coloring with $\Delta + 1$ colors. However, for sparse graphs, the maximum degree Δ may be much larger than the arboricity λ , and thus an arboricity-dependent coloring would be more satisfactory. In this regard, Barenboim and Khazanov [10] observed that any problem in λ -arboricity graphs can be solved in $O(\lambda)$ deterministic time (since there are at most λn edges) and presented the first non-trivial deterministic upper bound of λ^ε , for any constant $\varepsilon > 0$. On the randomized side, Ghaffari and Sayyadi [39] show that a $O(\lambda)$ coloring can be computed in a constant number of rounds, settling the randomized complexity of the problem. Note that any graph admits a 2λ -coloring, and this bound is tight.

1.1.2 Deterministic Algorithms and Derandomization

Apart from being of independent interest, it is instructive to view the above results in a broader context of deterministic algorithms and derandomization. Derandomization of local algorithms has attracted much attention in the parallel setting (see [18] for an overview). However, for almost a decade, (almost) all the research in the domain of Massively Parallel Computation has focused on the study of randomized algorithms. Only recently, a sequence of works has aimed at exploring the power of the MPC model in the context of deterministic algorithms [5, 22, 23, 25, 26]. They demonstrate that several graph problems can be solved deterministically with (asymptotic) complexity bounds that compare favorably with those of known randomized algorithms.

These results follow the clever derandomization scheme of [18], which enhances classic derandomization methods [58, 61] with the power of (potentially unbounded) local computation and global communication, in spite of the limited bandwidth. In a nutshell, a randomized process that works under limited independence is derandomized by computing the conditional expectation. This technique deterministically returns a result that is at least *as good as* the expectation of the randomized process, even when concentration bounds fail to hold with high probability. On the other hand, the *shattering effect* no longer appears under limited independence, as opposed to the full-independent setting. The lack of the shattering phenomena poses major challenges in obtaining deterministic algorithms with round complexities that match those of randomized algorithms, most of which crucially rely on shattering to have n -dependencies in the running time replaced by Δ . In addition, the sublinear regime of MPC requires derandomization methods that are specifically designed to cope with the limited memory per machine, for instance via sparsification [25] or partitioning [26].

This quest for efficient derandomization techniques is now one of the main problems in the area. Unfortunately, current derandomization frameworks suffer from long local running time (e.g., large polynomial or even exponential in n^δ) or large global memory (e.g., superlinear in the input size).

In fact, as noted in [26], allowing heavy local computation might provide an advantage in the context of distributed and parallel derandomization. However, especially in performance-oriented scenarios, these parameters may quickly become critical. It thus emerges as a natural direction to study deterministic algorithms whose global memory and total computation are as close as possible (or even match) those of their randomized counterparts.

1.2 Our Contribution

We introduce several improved deterministic algorithms for the four fundamental graph problems presented above in the MPC setting. Our approach builds upon the best-known randomized algorithms and follows the two-step derandomization method outlined in Section 1.1.2. We design randomized experiments for these problems and analyze their local probability of failure under pairwise or constant-wise independence. Then, we find *good random seeds* by invoking the distributed method of conditional expectation or simply by brute-forcing the search space. While the implementation of this process in a constant number of rounds is relatively well-understood, current derandomization techniques are often inferior in terms of global memory and total computation bounds. Our algorithms demonstrate in a number of cases not only how randomized round complexities can be achieved, but also how memory and computational complexities can benefit from careful derandomization approaches.

Connectivity We present the first computation-efficient deterministic algorithm for the problem of graph connectivity in the strongly sublinear memory regime of MPC. The total computation of $\tilde{O}(m)$ significantly improves over the $\text{poly}(n)$ -bound¹ of Coy and Czumaj [22], with no loss in the round complexity. In fact, our algorithm matches even the state-of-the-art randomized algorithm [13] in all parameters up to a polylogarithmic factor in the local running time. While the connectivity algorithm is of independent interest, our result provides several other qualitative advantages. For instance, our analysis relies only on pairwise independence as opposed to the almost $O(\log n)$ -wise independence of [22]. Moreover, to the best of our knowledge, this is the first derandomization result that uses the framework of limited independence without incurring a significant loss in one of the parameters (e.g., in the total computation time), and hence may be of practical interest. Furthermore, due to their simplicity, our analyses may serve as a friendly introduction to deterministic algorithms via the framework of bounded independence and, hopefully, as a stepping stone to a more systematic development of computation-efficient derandomization methods.

¹An explicit lower bound on the total computation is $\Omega(n^5)$, which outweighs the communication cost of $\tilde{O}(m)$.

MIS and MM We give a deterministic algorithm that matches the randomized arboricity-dependent algorithm [35] for both problems in low-memory MPC. In addition, we obtain the first deterministic strongly sublinear algorithms that use linear global memory for general graphs and as a function of the arboricity. Our arboricity-dependent result provides the first $O(\log \log n)$ -time algorithm on any graph family other than the family of graphs with bounded degree. Even for *unoriented trees* the best previous deterministic result has running time of $O(\log^2 \log n)$.

We begin by derandomizing a key degree-reduction method (parameterized by the arboricity) of Barenboim et al. [9], which shows that MIS and MM are reducible to instances with maximum degree $\text{poly}(\lambda, \log n)$. We first show that a tighter analysis for the sublinear regime of MPC of Barenboim et al.’s degree-reduction [9] gives the following: (i) the maximum degree can be reduced deterministically up to $\text{poly}(\lambda)$, even when $\lambda = o(\log n)$, (ii) each high-degree node can select a polynomially (in Δ) smaller set of low-degree neighbors, (iii) pairwise independence provides the same guarantees in expectation. We then incorporate this derandomization in the pipeline framework of Ghaffari et al. [35] to achieve a deterministic degree reduction running in $O(\log \log n)$ time. An (almost) immediate consequence of this is a round complexity $O(\log \lambda + \log \log n)$ by invoking the algorithm of Czumaj et al. [25] on the remaining graph.

When restricting our attention to the near-linear global memory regime, we derive three new results. We give the first $O(\log n)$ -round deterministic algorithm based on a slight modification of the derandomization of Czumaj et al.’s algorithm [25]. The crux in obtaining this is the definition of a pessimistic estimator that uses only knowledge of the one-hop neighborhood of each node to find a good random seed. Then, for the family of graphs with $\lambda = o(2^{\frac{\log n}{\log \log n}})$, we prove a $O(\log \lambda \log \log n)$ -round complexity bound using our degree-reduction routine followed by a graph exponentiation process. Further, we improve this bound to $O(\log \log n)$ rounds for graphs of *bounded arboricity* (see Section 4.3.2 for a precise statement) by adapting the arboricity coloring algorithms due to Barenboim and Elkin [6].

Arboricity Coloring We devise a strikingly simple constant-round deterministic distributed algorithm for the problem of computing a coloring with $O(\lambda)$ colors in the linear memory regime of MPC. This settles the round complexity of the problem in the relaxed setting where the global memory allowed is on the order of $n \cdot \text{poly}(\lambda)$. Its simplicity lies in the fact that after *one single step* of our deterministic graph partitioning procedure, all induced subgraphs are of linear size and can thus be collected and solved locally, whereas the state-of-the-art randomized algorithm [39] is based on a more sophisticated speed up technique via opportunistic information gathering.

For the setting with linear global memory, we derive a $O(\lambda)$ -coloring algorithm running in $O(\log \lambda)$ time by applying the constant-round algorithm for $(\Delta + 1)$ -coloring and $(\Delta + 1)$ -list coloring due to Czumaj et al. [23]. Both results improve exponentially upon the $\lambda^{\Omega(1)}$ -round algorithm of [10].

Our main technical contribution is a derandomization scheme that overcomes the inapplicability of the classic method of conditional expectation. In fact, our derandomization is based on a global quality measure (the arboricity) that cannot be decomposed into functions computable by single machines. To circumvent this, we show that we can find a subset of candidate partitions whose parts can be tested efficiently, yet limiting the local space to be linear and allowing only a $\text{poly}(\lambda)$ overhead factor in the total memory.

1.3 Structure of the Thesis

In Chapter 2, we start with overviewing basic techniques that are frequently used in the subsequent chapters, with the emphasis on arboricity-based results and derandomization methods in all-to-all communication models. We then proceed (Chapter 3) to the connectivity problem. We begin by reviewing the state-of-the-art randomized and deterministic connectivity algorithms and continue describing our algorithm based on the paper [33]. In Chapter 4, we turn our attention to MIS and MM. We first introduce the derandomization and pipelining of the arboricity-based degree reduction. Next, we present its applications together with algorithms in the near-linear and superlinear global memory regimes in Section 4.3 and Section 4.4, resp. Chapter 5 is devoted to arboricity-dependent coloring. After explaining how to use $(\Delta + 1)$ -list coloring for computing arboricity-based colorings efficiently, we describe our fast deterministic arboricity-coloring algorithm. Finally, in Chapter 6, we summarize our findings and discuss potential improvements.

Tools and Techniques

This chapter overviews the techniques that make up the *algorithmic toolbox* for the design of distributed graph algorithms. Our focus is on introducing the tools that are employed by the algorithms of the subsequent chapters. After some necessary notation and definitions, we explain the decomposition techniques adopted in the LOCAL model in Section 2.2 and those applied to all-to-all communication models in Sections 2.3 and 2.4. For a broader overview, we refer the reader to [8, 32].

2.1 Notation and Preliminaries

Let us start by introducing the notations used throughout this manuscript. For an integer $k \geq 1$, we will frequently denote $[k]$ as the set $\{1, 2, \dots, k\}$. Let $G = (V, E)$ be the undirected n -node m -edge graph given in input to our algorithms. Let the maximum degree of G be denoted by Δ and the arboricity of G be denoted by λ , which we introduce along with its properties later. Denote $\deg_G(u)$ as the degree of vertex u in graph G and $N_G(u)$ as the set of neighbors of u in G (when G is clear from the context, we may omit it). The distance from u to v , $\text{dist}_G(u, v)$, is the length of the shortest path from u to v . For a vertex subset $U \subseteq V$ or edge subset $U \subseteq E$, we use $\deg_U(u)$ and $N_U(u)$ to refer to the subgraph of G induced by U denoted by $G[U]$. Similarly, for a vertex subset $V' \subseteq V$, let $N_G(V')$ be the union of the neighbors of each $v \in V'$, let $\text{dist}_G(u, V')$ be the minimum distance from u to any vertex in V' , and let $\text{dist}_{V'}(u, v) = \text{dist}_{G[V']}(u, v)$ be the minimum distance from u to v .

Further, we denote G^k , for integer $k \geq 2$, as the graph obtained from connecting each pair of vertices at distance at most k in G . Then, the k -hop neighborhood of a vertex v in G is the subgraph containing all vertices u satisfying $\text{dist}_G(u, v) \leq k$, together with every edge that lies on some path starting from v with length at most k . The k -hop neighborhood of a vertex set $V' \subseteq V$ is the union of the k -hop neighborhoods of $v \in V'$.

Graph Arboricity

The notion of arboricity characterizes the sparsity of a graph. It is equal to the minimum number of forests into which the edge set of the graph can be partitioned. Nash-Williams [63, 64] showed that the arboricity is equal to the density of the graph, that is, the maximum value of $\lceil m_S / (n_S - 1) \rceil$, where m_S, n_S are the number of edges and vertices in any subgraph S . In the following lemma, we state some basic properties of this parameter. These play a central role in the design of λ -dependent distributed graph algorithms.

Lemma 2.1 (Arboricity Properties [8, 9]) *Let G be a graph of m edges, n nodes, and arboricity λ . Then*

1. $m < \lambda n$.
2. The arboricity of any subgraph $G' \subseteq G$ is at most λ .
3. The number of nodes with degree at least $t \geq \lambda + 1$ is less than $\lambda n / (t - \lambda)$.
4. The number of edges whose endpoints both have degree at least $t \geq \lambda + 1$ is less than $\lambda m / (t - \lambda)$.

Remark 2.2 *While our algorithms may seem to require knowing λ , by employing the standard technique of [46] to run the algorithm with doubly-exponentially increasing estimates for λ in parallel, combined with global communication, we can find an estimate for λ that produces (asymptotically) the same result incurring only a $\tilde{O}(1)$ factor overhead in the global space and total computation.*

2.2 Local Decomposition Algorithms

Symmetry-breaking algorithms often rely on primitives that decompose the input graph into smaller subgraphs that can be processed separately, in parallel or sequentially, faster, and more easily than the original graph. We will make use of the following coloring algorithm and of a certain graph-theoretic structure introduced by Nash-Williams [63, 64] as forests-decomposition and known as H -partition in the setting of distributed computing [6].

Fast LOCAL Coloring

A proper coloring $\varphi: V \mapsto C$, for $C \in \mathbb{N}$, is a mapping from vertices to colors such that no two incident neighbors share the same color, i.e., $\varphi(v) \neq \varphi(u)$ for each edge $(u, v) \in E$. In his seminal work, Linial [54] showed that a valid $O(\Delta^2)$ coloring can be computed deterministically within $\log^* n + O(1)$ time. Specifically, his algorithm, given a proper C -coloring, produces a new legal coloring with a color palette of size $O(\Delta \log^2 C)$. The proof relies on a purely combinatorial structure [29], which can be derandomized using an algebraic construction based on polynomials [8, 29, 47].

Lemma 2.3 *Let $G = (V, E)$ be a graph of maximum degree Δ and assume we are given proper C -coloring of G . There is a deterministic algorithm that computes a $O(\Delta^2 \log^2 C)$ coloring of G in $O(1)$ LOCAL rounds. Moreover, if $C = O(\Delta^3)$, then the number of colors can be reduced to $O(\Delta^2)$.*

As an example of application, a legal C -coloring φ can be used for computing an MIS within C rounds. For $i \in [C]$, each vertex v with $\varphi(v) = i$ without a neighbor in the MIS joins the MIS and inform its neighbors about this. Since vertices colored by color i form an independent set the correctness follows.

A coloring can be useful also when the decision of each node may depend on nodes within distance k . In fact, such dependencies can be modeled using the power- k graph G^k . A valid coloring of G^k assigns colors to nodes such that nodes at distance at most k have distinct colors and thus their decisions are decoupled. The execution of a (coloring) algorithm on G^k incurs a multiplicative factor k overhead in the runtime in LOCAL, while in all-to-all communication models this will be analyzed on a case-by-case basis.

H-Partition

The arboricity properties seen in Section 2.1 give a surprisingly simple distributed algorithm that decomposes the vertex set of a graph G into parts such that each part has degree linear in the arboricity. This decomposition is known as H -partition and often serves as a basic building block for arboricity-dependent algorithms [6, 7, 8, 11]. It is defined as follows.

Definition 2.4 (H-Partition) *An H -partition \mathcal{H} of a λ -arboricity graph G parameterized by its degree d , for any $d > 2\lambda$, is a partition of the vertices of G into layers H_1, \dots, H_L such that every vertex $v \in H_i$ has at most d neighbors in layers with indexes equal to or higher than i , i.e., $\bigcup_{j=i}^L L_j$.*

Each layer can be computed easily in a constant number of rounds by applying the classic greedy peeling algorithm as follows: Repeatedly, vertices of remaining degree at most d are peeled off from the graph and form a new layer. Since the number of vertices of degree larger than d is at most $\frac{2\lambda}{d} \cdot n$, the size L of \mathcal{H} , i.e., the overall number of layers, is at most $\lceil \log_{d/2\lambda} n \rceil = O(\log_{\frac{d}{\lambda}} n)$.

Another of its features is to give a d forest decomposition or, equivalently, an acyclic orientation of the edges such that every vertex has outdegree at most d . An edge (u, v) is oriented toward the vertex with a higher index layer in the given H -partition or, if they are in the same layer, toward the vertex with a greater ID. By construction, the orientation is acyclic and each vertex has at most d outneighbors. Now, let each node v assign a distinct label from the set $\{1, \dots, d\}$ to each of its outgoing edges. The set of edges labeled by label i forms a forest. For a more elaborate proof, we refer to [8].

2.3 Local Simulation via Global Communication

Primitives in Low-Space MPC

There are many well-known MPC primitives that will be used as black-box tools. These have been studied in the MapReduce framework and can be implemented in the MPC model with strictly sublinear space per machine and linear global space. We will use the following lemma to refer to them:

Lemma 2.5 ([41, 42]) *For any positive constant δ , sorting, filtering, prefix sum, predecessor, duplicate removal, and colored summation task on a sequence of n tuples can be performed deterministically in MapReduce (and therefore in the MPC model) in a constant number of rounds using $\mathbf{S} = n^\delta$ space per machine, $O(n)$ global space, and $\tilde{O}(n)$ total computation.*

Finally, observe that these basic primitives allow us to perform all of the basic computations on graphs deterministically that we will need in a constant number of MPC rounds. This includes the tasks of computing the degree of every vertex, ensuring neighborhoods of all vertices are stored on contiguous machines, sums of values among a vertex neighborhood, and collecting the 2-hop neighborhoods provided that they fit in the memory of a single machine. Moreover, it is a common assumption that nodes are assigned to machines in a balanced way, i.e., the underlying system takes care of load balancing.

Graph Exponentiation

One central technique that often leads to an exponential speed-up of LOCAL algorithms is *graph exponentiation* due to Lenzen and Wattenhofer [52]. This approach leverages the power of global communication and its main idea is to let each node send its r -hop neighborhood to each of the nodes in it so that in one round every node learns the topology of the ball of radius $2r$ around itself. By doing so, each node can simulate $2r$ rounds of any LOCAL algorithm by computing the choices performed by the nodes in its $2r$ -hop neighborhood. In general, a t -round LOCAL algorithm can be simulated in $O(\frac{t}{r})$ rounds using the knowledge of the $\Omega(r)$ -hop neighborhood and allowing a constant-round global coordination step every $\Theta(r)$ rounds. However, the main challenge is to ensure that the amount of information in the r -hop neighborhood of every vertex is small enough to respect the communication and memory constraints. We next discuss an application of graph exponentiation in low-memory MPC.

Lemma 2.6 *Let G be a graph with maximum degree $\Delta \leq O(n^\alpha)$ and let k be the largest integer such that $\Delta^{2^k} \leq n^\alpha$. Let \mathcal{A} be a LOCAL algorithm that reduces the problem size, i.e., the number of active nodes or relevant edges, by a factor $\Omega(\Delta)$ within t rounds. Then T LOCAL rounds of \mathcal{A} can be simulated in $O(t \cdot k + \frac{T}{2^k})$ rounds on a low-memory MPC using local space $O(n^\alpha)$ and $O(n + m)$ global space.*

Proof The algorithm consists of two stages. The first stage of the algorithm applies k graph exponentiation steps interleaved by a $O(t)$ -round simulation phase, which reduces the space required by the graph exponentiation process. For $i = 0, 1, \dots, k - 1$, the algorithm performs the following steps:

1. Each node simulates $s \cdot t \cdot 2^i$ rounds of algorithm \mathcal{A} , for constant $s > 0$.
2. Each vertex learns its 2^{i+1} -hop neighborhood via graph exponentiation.

In the second stage, each node simulates the remaining $O(T - t \cdot k)$ LOCAL rounds of algorithm \mathcal{A} in $O(\frac{T}{2^k})$ MPC rounds, interleaved by a constant-round global coordination step to update the neighborhood of each node.

We prove by induction that at the end of iteration i , each node knows its 2^{i+1} -hop neighborhood. The base case for $i = 0$ clearly holds. Assume that each node knows its 2^i -hop neighbors by the end of iteration $i - 1 \geq 0$. By step number two above, each node v receives the 2^i -hop neighborhoods of the vertices at distance at most 2^i from v . From this fact, it follows that every node learns the topology of its 2^{i+1} -hop neighborhood.

Next, we discuss the correctness of the algorithm. Each node v simulates locally on its machine 2^i rounds of the LOCAL algorithm \mathcal{A} . After this simulation, v learns its status after 2^i rounds of \mathcal{A} and communicates it to each of its 2^i -hop neighbors. Thus, v can simulate another 2^i rounds of algorithm \mathcal{A} , and repeat this process for $s \cdot t$ times in each iteration. This requires $O(s \cdot t) = O(t)$ MPC rounds per iteration, and $O(kt)$ time overall.

Let us now analyze the space usage of the algorithm. The local memory constraint is respected as each node uses $O(\Delta^{2^k}) = O(n^\delta)$ memory to store its k -hop neighborhood. To see that also the global memory constraint is not violated, we show by induction that after step number one of the above algorithm the global space usage is $O(n/\Delta^{2^{i+1}})$. For the base case $i = 0$, $s \cdot t$ executions of \mathcal{A} reduce the number of vertices (or edges) to $O(n/\Delta^s) \leq O(n/\Delta^2)$, for $s \geq 2$. Now, by induction hypothesis, assume that iteration $i > 0$ starts with $O(n/\Delta^{2^i})$ vertices (or edges). After the simulation of $s \cdot t \cdot 2^i$ rounds of \mathcal{A} , the number of nodes (or edges) decreases by a factor $\Delta^{s \cdot 2^i}$, since every t rounds of \mathcal{A} reduce the problem size by a factor $\Omega(\Delta)$. Thus, the global space usage after this step becomes $O(n/\Delta^{2^i + s \cdot 2^i}) = O(n/\Delta^{2^{i+1}})$, as desired. Then, by observing that $i + 1$ -th step of graph exponentiation may duplicate a vertex or an edge at most Δ^{i+1} times, we deduce that the total memory occupied is $O(n/\Delta^{2^{i+1}} \cdot \Delta^{2^{i+1}}) = O(n)$ per iteration. Assuming that inactive nodes (or edges) require $O(n + m)$ memory, the claim follows. \square

2.4 Derandomization in All-to-All Models

Derandomization techniques in all-to-all communication models emerged as a powerful tool to obtain simple deterministic algorithms of complexity comparable to that of their randomized counterparts. Our approach follows the derandomization framework of [18, 22, 23, 25, 26], which traces back to [58]. For a more systematic treatment of derandomization approaches see for example [2, 17, 59, 60, 69, 71].

The first step is to obtain a *randomized process* that produces good results in expectation based on a small search space (i.e., short random seed) by using random variables with some limited independence. We will use a k -wise independent family of hash functions, which is defined as follows:

Definition 2.7 (k -wise independence) *Let $N, k, \ell \in \mathbb{N}$ with $k \leq N$. A family of hash functions $\mathcal{H} = \{h : [N] \rightarrow \{0, 1\}^\ell\}$ is k -wise independent if for all $I \subseteq \{1, \dots, n\}$ with $|I| \leq k$, the random variables $X_i := h(i)^1$ with $i \in I$ are independent and uniformly distributed in $\{0, 1\}^\ell$, when h is chosen uniformly at random from \mathcal{H} . If $k = 2$ then \mathcal{H} is called pairwise independent.*

The following is a well-known result about its construction:

Lemma 2.8 ([1, 21, 30]) *For every $N, k, \ell \in \mathbb{N}$, there is a family of k -wise independent hash functions $\mathcal{H} = \{h : [N] \rightarrow \{0, 1\}^\ell\}$ such that choosing a uniformly random function h from \mathcal{H} takes at most $k(\ell + \log N) + O(1)$ random bits, and evaluating a function from \mathcal{H} takes time $\text{poly}(\ell, \log N)$ time.*

If there is a randomized algorithm, over the choice of a random hash function h from \mathcal{H} , that gives good results in expectation, then one can obtain a result as good as the expectation by finding the right choice of (random) bits. To achieve that, if the seed length that defines the family of hash functions \mathcal{H} is small, then one can simply brute force all possible sequences of bits to find one good seed. Instead, when brute-forcing is undesirable or not possible, one relies on a distributed implementation of the method of conditional expectation (or probability). There, one divides the seed into several parts and fixes one part at a time in a way that does not decrease the conditional expectation. This is done by having nodes agree globally on each part in a voting-like manner. Next, we explain both approaches in more detail.

We define an objective function (which is a sum of functions calculable by individual machines) that is at least (or at most) some value Q , namely

$$\mathbb{E}_{h \in \mathcal{H}} \left[q(h) \stackrel{\text{def}}{=} \sum_{\text{machines } x} q_x(h) \right] \geq Q.$$

¹ $h(\cdot)$ denotes the length- ℓ bit sequence by the corresponding integer in $\{0, \dots, 2^\ell - 1\}$.

Then, by the probabilistic method, this implies the existence of a hash function $h^* \in \mathcal{H}$ for which $q(h^*)$ is at least (or at most) Q . If the family of hash functions fits into the memory of a single machine, then each machine x can locally compute $q(h)$ (which is a function of the vertices machine x is assigned) for each h . Then, we sum the values $q(h)$ computed by every machine for each h and find one good hash function h^* in $O(1)$ MPC rounds. When the family of hash functions cannot be collected onto a single machine but its size is $\text{poly}(n)$, i.e., each function can be specified using $O(\log n)$ bits, we can still find one good hash function in $O(1)$ MPC rounds as follows.

Let $\tau = O(\log n)$ denote the length of the random seed and $\chi = \log \mathbf{S} = \alpha \log n$. We proceed in $\lceil \frac{\tau}{\chi} \rceil$ iterations. We search for a good function h^* by having machines agree on a χ -bit chunk at a time. Let $\mathcal{H}^{(i)}$ be the subset of hash functions from \mathcal{H} considered at iteration i . Clearly, $\mathcal{H}^{(0)} = \mathcal{H}$ and $\mathcal{H}^{(i)} \supseteq \mathcal{H}^{(i+1)}$. We will ensure that $\mathbb{E}_{h \in \mathcal{H}^{(i)}} [q(h)] \geq \mathbb{E}_{h \in \mathcal{H}^{(i-1)}} [q(h)]$ until $\mathcal{H}^{(i)}$ consists of a single element defining h^* . In phase i , for $1 \leq i \leq \lceil \frac{\tau}{\chi} \rceil$, we determine $\mathcal{H}^{(i)}$ by fixing the bits at positions $1 + (i-1) \lceil \frac{\tau}{\chi} \rceil, \dots, i \lceil \frac{\tau}{\chi} \rceil$. We assume that each machine knows the prefix \mathbf{b} chosen in previous iterations. Next, each machine considers the extension of \mathbf{b} by all possible seeds of length χ . Let $\mathcal{H}_1, \dots, \mathcal{H}_{2^\chi}$ be a partition of $\mathcal{H}^{(i-1)}$ such that \mathcal{H}_j contains all hash functions $h \in \mathcal{H}^{(i-1)}$ whose bit sequence consists of \mathbf{b} as prefix followed by the bit representation b_j of j .

For every \mathcal{H}_j , each machine computes locally $\mathbb{E}_{h \in \mathcal{H}_j} [q(h)]$ for all $1 \leq j \leq 2^\chi$. This computation can be performed *locally* by taking each hash function $h \in \mathcal{H}_j$ and computing $q_x(h)$, that is the target function for the nodes (or edges) that machine x is responsible for. Once we know $q_x(h)$ for every $h \in \mathcal{H}_j$, then $\mathbb{E}_{h \in \mathcal{H}_j} [q_x(h)]$ is simply the average of all these numbers. An alternative way to compute $\mathbb{E}_{h \in \mathcal{H}_j} [q_x(h)]$ is by letting each machine compute the expected value of $q_x(h)$ for each of its vertices (or edges) conditioned on $\mathbf{b} \circ b_j$ (note that each node u can usually compute this probability as it often depends only on its neighbors and by knowing their IDs, u can simulate their choices). Using Lemma 2.5, we compute $\sum_{\text{machines } x} \mathbb{E}_{h \in \mathcal{H}_j} [q_x(h)]$ by summing up the individual expectations. Note that by linearity of expectations, it holds that $\sum_{\text{machines } x} \mathbb{E}_{h \in \mathcal{H}_j} [q_x(h)] = \mathbb{E}_{h \in \mathcal{H}_j} [q(h)]$. Then, we are ensured that there is an index k such that $\mathbb{E}_{h \in \mathcal{H}_k} [q(h)] \geq Q$, i.e., picking a u.a.r. function h from \mathcal{H}_k gives an objective that is at least (or at most) our objective Q . Thus, we select $\mathcal{H}^{(i)} = \mathcal{H}_k$ and extend the bit prefix $\mathbf{b} \leftarrow \mathbf{b} \circ b_k$ with the bit representation b_k of k . In the next iteration, we repeat the same process until when one good hash function h^* is found. Since $\lceil \frac{\tau}{\chi} \rceil = O(1/\alpha) = O(1)$, this distributed implementation of the method of conditional expectation runs in $O(1)$ MPC rounds.

Concentration Inequalities To show that a randomized process requires only k -wise independence, we will use the following results. For $k = 2$, a bound that is usually satisfactory is easily obtained from Chebyshev’s inequality, as follows.

Theorem 2.9 (Chebyshev) *Let X_1, \dots, X_n be random variables taking values in $[0, 1]$. Let $X = X_1 + \dots + X_n$ denote their sum and $\mu \leq \mathbb{E}[X]$. Then*

$$\Pr[|X - \mathbb{E}[X]| \geq \mu] \leq \frac{\text{Var}[X]}{\mu^2}.$$

Corollary 2.10 *If X_1, \dots, X_n are pairwise independent, then $\text{Var}[X] = \sum_{i=1}^n \text{Var}[X_i] \leq \mathbb{E}[X]$ and*

$$\Pr[|X - \mathbb{E}[X]| \geq \mu] \leq \frac{\sum_{i=1}^n \text{Var}[X_i]}{\mathbb{E}[X]^2} \leq \frac{1}{\mathbb{E}[X]} \leq \frac{1}{\mu}.$$

When the expectation of X is small, the following more general tail inequality due to Bellare and Rompel [14] is often applied: The bound is in terms of n , k and the expectation.

Lemma 2.11 (Lemma 2.3 of [14]) *Let $k \geq 4$ be an even integer. Let X_1, \dots, X_n be random variables taking values in $[0, 1]$. Let $X = X_1 + \dots + X_n$ denote their sum and let $\mu \leq \mathbb{E}[X]$ satisfying $\mu \geq k$. Then, for any $\varepsilon > 0$, we have*

$$\Pr[|X - \mathbb{E}[X]| \geq \varepsilon \cdot \mathbb{E}[X]] \leq 8 \left(\frac{2k}{\varepsilon^2 \mu} \right)^{k/2}.$$

Reducing The Seed Length via Coloring The following technique plays a central role in reducing the seed length of randomized processes solving *local* graph problems. As shown in [5, 25, 26], if the outcome of a vertex depends only on the random choices of its neighbors at distance at most t , then k -wise independence among random variables of vertices within pairwise distance t is sufficient. Whenever this is the case, we can find a mapping from vertex IDs to shorter names such that “adjacent” vertices are assigned different names. This can be achieved via a coloring of the power- t graph G^t using the coloring algorithm introduced earlier in Lemma 2.3. We here adapt a more explicit 1-round distributed coloring algorithm with $O(\Delta^2 \log^2(n))$ colors by Kuhn [48] to the MPC model, which leads to the following lemma:

Lemma 2.12 *Let $G = (V, E)$ be a graph of maximum degree $\Delta \leq n^\delta$. There exists a deterministic algorithm which computes an $O(\Delta^2 \log^2 n)$ coloring of G in $O(1)$ MPC rounds using $O(n^\delta)$ local space, $O(n + m)$ global space, and $\tilde{O}(n \cdot \text{poly}(\Delta))$ total computation.*

Proof We start by recalling the high-level idea and then we give an efficient MPC implementation. We assume that each vertex in G is given a unique ID between 1 and n . Let p be a prime with $10\Delta \log(n) \leq p \leq 20\Delta \log(n)$. It is well known that such a prime always exists. Moreover, let $d = \lceil \log(n) \rceil$. There exists $p^{d+1} \geq n$ distinct polynomials of degree at most d over \mathbb{F}_p . We denote by f_i the i -th such polynomial. Each color corresponds to a tuple over \mathbb{F}_p . Note that there are $p^2 = O(\Delta^2 \log^2 n)$ such tuples.

Let $C_i = \{(x, f_i(x)) : x \in \mathbb{F}_p\}$. Using $\Delta d < p$ together with the fact that a non-zero polynomial of degree d can have at most d zeros implies that each vertex can choose a color $c(i) \in C_i$ such that $c(i) \notin C_j$ for every neighbor j . Now, assigning each vertex i the color $c(i)$ results in a valid coloring. It remains to discuss the MPC implementation. By using the basic primitives of Lemma 2.5 and the assumption that $\Delta \leq n^\delta$, we can assume that the machine responsible to compute the coloring of the i -th vertex also stores the IDs of all the neighbors of i . Note that a given polynomial can be evaluated in time $\text{poly}(\log n, \Delta)$. Computing the color $c(i)$ boils down to $O(\Delta \cdot p^2) = \text{poly}(\log n, \Delta)$ polynomial evaluations. Hence, the total computation time is $\tilde{O}(n \cdot \text{poly}(\Delta))$, as desired. \square

2.4.1 Example of Pairwise Independent Analysis

In this section, we illustrate a pairwise independent analysis of the randomized $O(\log n)$ -round algorithm for maximal matching by Israeli and Itai [44]. We find it a useful introductory exercise to bounded dependency analyses that will be used throughout the paper.

The Algorithm Given a graph $G(V, E)$ with maximum degree Δ , the algorithm performs the following four steps for $O(\log n)$ iterations.

1. Each $v \in V$ chooses randomly an adjacent edge and directs it outward.
2. Each vertex such that $\deg^{\text{in}}(v) \geq 1$ selects an incoming edge, where $\deg^{\text{in}}(v)$ refers to the in-degree of vertex v . Let E_S be the set of selected edges, where the orientations are now ignored.
3. Each vertex chooses randomly an incident edge of E_S . An edge $e \in E_S$ belongs to the matching \mathcal{M} if it was chosen by both its endpoints.
4. Remove from G all the edges of \mathcal{M} , with all their incident edges to get G' —the input graph for the next phase.

Theorem 2.13 *The above algorithm from [44] computes a maximal matching of an input graph G in $O(\log n)$ LOCAL rounds with high probability using only pairwise independence.*

Our goal is to show that in each iteration every vertex decreases its degree linearly with constant probability. Let us consider a fixed iteration of the algorithm. Define the *proposal level* $p(v)$ of a vertex $v \in V$ as the expected number of incoming edges after the first step of the algorithm, namely

$$p(v) = \sum_{u \in N(v)} \frac{1}{\deg(u)}.$$

We say that vertex v is good if $p(v) \geq \varepsilon$, for $\varepsilon = 1/4$. As we shall see, being good for a vertex means to be matched with constant probability. For analysis purposes, we assume wlog² that $p(v)$ is at most $1/2$.

Lemma 2.14 *If v is good, then v has an incoming edge after step one with probability at least $\varepsilon/2$.*

Proof By pairwise independence, we have

$$\begin{aligned} \Pr[\deg^{in}(v) \geq 1] &\geq \sum_{u \in N(v)} \Pr[u \text{ chooses } v] - \sum_{u, u' \in N(v)} \Pr[u, u' \text{ choose } v] \\ &\geq \sum_{u \in N(v)} \frac{1}{\deg(u)} - \sum_{u, u' \in N(v)} \frac{1}{\deg(u) \cdot \deg(u')} \\ &\geq \sum_{u \in N(v)} \frac{1}{\deg(u)} \left(1 - \sum_{u' \in N(v)} \frac{1}{\deg(u')} \right) \geq \frac{\varepsilon}{2}. \quad \square \end{aligned}$$

Lemma 2.15 *If $\deg^{in}(v) \geq 1$, then v is matched with probability at least $1/2$.*

Proof A vertex with $\deg^{in}(v) \geq 1$ is part of the matching if the edge that vertex v chooses in step three is chosen by the other endpoint as well. Since $\deg^{in}(u) \leq 2$ for every $u \in V$ and random choices are pairwise independent, the other endpoint u selects such edge with probability at least $1/2$. \square

We would like to say that a constant fraction of vertices is good, but this is unfortunately not true. Therefore, instead of looking just at a vertex v , we look at the direct neighbors of v . We will show that if the neighborhood of v contains *many* good nodes, then the degree of v drops by a constant factor with constant probability. On the other hand, when there are few good nodes in v 's neighborhood, chances are that v chooses a bad neighbor that will be matched with v with constant probability.

Lemma 2.16 *If v has at least $\frac{\deg(v)}{2}$ good neighbors then its degree shrinks by a constant factor with probability at least $1/100$.*

²This can be achieved by restricting the neighborhood of v under consideration. In fact, every vertex with a neighbor of degree at most 3 has a probability of at least $1/3$ of being matched, thus we can assume that every neighbor has degree at least 4.

Proof Every good vertex has $\deg^{in}(v) \geq 1$ with probability at least $\varepsilon/2$ by Lemma 2.14. Since step one and two of the algorithm use different randomness, we deduce that every good vertex is matched with probability at least $\varepsilon/4$ by Lemma 2.15. Let X denote the number of unmatched vertices in v 's neighborhood. Using linearity of expectations, the expected value of X is at most $(1 - \frac{\varepsilon}{8}) \deg(v)$. By an application of Markov's inequality, X is at most $(1 - \frac{\varepsilon}{8}) \frac{100 \deg(v)}{99} < \frac{98}{100} \deg(v)$ with probability at least $1/100$. \square

Lemma 2.17 *If v has less than $\frac{\deg(v)}{2}$ good neighbors, then its degree shrinks by a constant factor with probability at least $1/4$.*

Proof Let us analyze the probability of the event A that v chooses a vertex $u \in N(v)$ such that $\deg^{in}(u) = 1$. If event A happens, then v is matched with probability at least $3/4$. By pairwise independence and by restricting $N(v)$ to bad vertices, we have

$$\begin{aligned}
 \Pr[A] &\geq \sum_{u \in N(v)} \Pr[v \text{ chooses } u] - \sum_{v, u' \in N(u)} \Pr[v, u' \text{ choose } u] \\
 &\geq \sum_{u \in N(v)} \frac{1}{\deg(v)} - \sum_{v, u' \in N(u)} \frac{1}{\deg(v) \deg(u')} \\
 &= \sum_{u \in N(v)} \frac{1}{\deg(v)} \left(1 - \sum_{u' \in N(u)} \frac{1}{\deg(u')} \right) \quad \left(\sum_{u' \in N(u)} \frac{1}{\deg(u')} < \varepsilon = \frac{1}{4} \right) \\
 &\geq \sum_{u \in N(v)} \frac{3}{4 \deg(v)} \\
 &\geq \frac{3}{8}.
 \end{aligned}$$

By the reasoning above, vertex v is matched with probability at least $\frac{3}{8} \cdot \frac{3}{4} = \frac{9}{32}$. By a simple application of Markov's inequality on the expected degree of v , the degree of v decreases by a constant factor with probability at least $1/4$. \square

We are now ready to prove Theorem 2.13. By the above lemmas, one iteration of the algorithm decreases the degree of each vertex by a constant factor with probability at least $1/100$. After $O(\log \Delta + \log n) = O(\log n)$ rounds, by Markov's inequality, every vertex has degree less than one with high probability. Thus, every vertex is matched or all its neighbors are matched.

Deterministic Work-Efficient Massively Parallel Connectivity

One of the most fundamental problems in the model of Massively Parallel Computation with strongly sublinear memory is that of connectivity, that is, determining whether any two vertices are in the same connected component. A long line of research about connectivity culminated in a randomized algorithm for low-space MPC that finds all connected components of a graph with diameter D in near-optimal round complexity $O(\log D + \log \log_{\frac{m}{n}} n)$ and optimal $O(m)$ space. In Section 3.1, we briefly overview this algorithm that is due to Behnezhad et al. [13] and builds on that of Andoni et al. [3]. Surprisingly, a recent result of Coy and Czumaj [22] shows how to achieve the same deterministically. Unfortunately, however, their algorithm suffers from large local computation time, as we review in Section 3.1. In this chapter, we describe a deterministic connectivity algorithm that matches all the parameters of the randomized algorithm mentioned above and, in addition, significantly reduces the local computation time to nearly linear. Our result is based on reducing the amount of randomness needed to solve two algorithmic primitives: Constant Approximation of Maximum Matching (Section 3.3.1) and Hitting Set (Section 3.3.2). We will see in Section 3.4 how these two ingredients can be incorporated into the algorithm of [13] to obtain the first deterministic, derandomization-based algorithm that has efficient local computation. Moreover, another virtue of our algorithm is its simplicity, which is why we hope it serves as a starting point for the systematic development of computation-efficient derandomization approaches in low-memory MPC. Our main result is the following:

Theorem 3.1 (Deterministic Connectivity) *There is a strongly sublinear MPC algorithm that given a graph with diameter D , identifies its connected components in $O(\log D + \log \log_{\frac{m}{n}} n)$ rounds deterministically using $O(n + m)$ global space and $\tilde{O}(m)$ total computation.*

3.1 Randomized Connectivity Algorithms in a Nutshell

We start by giving some intuition behind the (randomized) connectivity algorithms by Andoni et al. [3] and Behnezhad et al. [13].

Vertex Contraction The main idea behind connectivity algorithms working in $\tilde{O}(\log D)$ rounds is to repeatedly perform *vertex contractions* [3]. Contracting a vertex u to an adjacent vertex v means deleting the edge $\{u, v\}$ and connecting v to all the vertices adjacent to u . The simplest way to implement this contraction-based approach is to appoint a random subset of the vertices as *leaders* (by letting each vertex become a leader with probability $\frac{1}{2}$), and then to contract non-leader vertices to one of their leader neighbors (if any). This approach requires $O(\log n)$ rounds with high probability.

Vertex Contraction with Levels and Budgets (Andoni et al. [3]) A crucial observation to speed up the vertex contractions—going back to the graph exponentiation approach by Lenzen and Wattenhofer [52]—is to let each vertex expand its neighborhood to neighbors of neighbors by adding new edges (without changing the connectivity). In fact, if every vertex reaches degree $\Omega(d)$ by expanding its neighborhood in $O(\log D)$ rounds, we can mark vertices to be a leader with probability $\approx \frac{\log n}{d}$. As a result, each non-leader vertex has a leader in its neighborhood and the number of remaining vertices is $\tilde{O}(\frac{n}{d})$.

In their algorithm, Andoni et al. [3] assign a level to every vertex which has not been contracted yet. Vertices at level i have a budget of b_i for expanding their neighborhood, i.e., each vertex at level i can add at most b_i neighbors. The initial budget b_0 is set to $\min(n^{\delta/2}, \sqrt{\frac{m}{n}})$ to maintain global space $O(m)$. At iteration i , every vertex either increases its degree to b_i or finds its connected component. As explained above, we thus can mark leader vertices with probability $\frac{\log n}{b_i}$ and perform contractions to reduce the problem size to $\tilde{O}(n/b_i)$. Hence, the budgets of remaining vertices can be updated to $b_{i+1} = b_i^{1+c}$, for a small constant c , while using the same global space. Overall, after $O(\log \log_{\frac{m}{n}} n)$ iterations, there will be a unique vertex left in each connected component.

Random Leader Contraction (Behnezhad et al. [13]) To further improve the round complexity, Behnezhad et al. [13] design an algorithm that applies vertex contractions and increases the budgets of vertices in an asynchronous manner, e.g., at a given time two active vertices can have different budgets. In each round, their algorithm (informally) ensures that each vertex either learns its 2-hop neighborhood or increases its budget. We here focus on the routine that defines the budget increase, as this is the only step involving randomness.

Consider the subgraph induced by vertices with budget level i . The crucial observation is that if a vertex has $\Omega(b_i)$ many neighbors of the same level, then contracting all of them allows us to recuperate $\Omega(b_i^2)$ budget. If each vertex is elected as a leader with probability $\approx \frac{\log n}{b_i}$, and non-leader vertices contracted to an arbitrary neighboring leader, then leaders can increase their level without exceeding the total memory.

Increasing Initial Budget using Matching (Behnezhad et al. [13]) To allow each vertex to start with a poly $\log n$ budget, a randomized constant-round algorithm (see [13, Algorithm 3]) reduces the number of vertices of G by a constant factor. By running it for $O(\log \log n)$ MPC rounds, the problem size decreases from n to $n/\text{poly } \log n$. Intuitively, this algorithm works by contracting a constant fraction of the vertices to their lowest-ID neighbors as follows. Each vertex proposes to be contracted to its neighbor with the smallest ID. A deterministic conflict resolving phase results in a graph of size $\Omega(n)$ consisting of *vertex-disjoint paths*. Contracting along the edges of a constant-approximate *maximum matching* in this graph with maximum degree 2 results in the removal of $\Omega(n)$ vertices as desired.

3.2 Comparison with the State-of-the-Art

We next present the main ideas behind the recent deterministic connectivity algorithm of Coy and Czumaj [22]. In [22], the authors identify and extract the only two sources of randomization from the algorithms of [3, 13], namely, random sampling for matching and hitting set. On the one hand, as outlined in Section 3.1, a constant approximation of matching in graphs with maximum degree 2 can be used for the initial budget increase. On the other hand, the random leader contraction can be formulated as a hitting set instance with all sets of the same size (see Definition 3.4 for a precise definition).

As these are the only steps involving randomness, the constant-round derandomization of them immediately leads to a deterministic connectivity algorithm. In fact, their derandomization together with the $O(\log D + \log \log n)$ randomized algorithm due to Behnezhad et al. [13] results in the state-of-the-art deterministic connectivity algorithm in low-memory MPC [22].

While their adopted derandomization framework is well-established, its efficient implementation for obtaining a deterministic connectivity algorithm in low-memory MPC requires overcoming several challenges. Although their algorithm achieves optimal space guarantees, the computation is suboptimal for both derandomization steps. We refine these two to obtain a computation-efficient deterministic connectivity algorithm, as explained next.

Maximum Matching: The problem of approximating maximum matching in graphs of maximum degree at most two is solved by searching the space of a randomized process based on pairwise independent hash functions, which are specified by $(2 \log n + O(1))$ random bits. As each of the $O(n^2)$ hash functions is evaluated $O(n)$ times using poly $\log n$ computation, the total computation is $\tilde{O}(n^3)$. We reduce the *seed length*, i.e., the number of random bits needed, to $O(\log \log n)$ and, as a result, obtain $\tilde{O}(n)$ total computation.

Hitting Set: For a hitting set instance with n elements and a collection of n subsets of size b , the algorithm from [22] finds a hitting set of size $O(nb^{-1/5})$ by derandomizing a simple random sampling approach based on a $O(\log_b(n))$ -wise $1/\text{poly}(n)$ -approximately independent family of hash functions of size $\text{poly}(n)$. The (distributed) method of conditional expectation for this process takes global space $O(nb)$ and $\text{poly}(n)$ total computation.

We provide a low-memory MPC algorithm that solves the same hitting set instance using only pairwise independent random choices with $n \cdot \text{poly}(b)$ global space and $n \cdot \text{poly}(b)$ total computation. Thus, the dependency on n improves polynomially when $b \ll n$. It turns out that using this hitting set algorithm as a subroutine in our connectivity algorithm allows us to obtain an algorithm with total computation $\tilde{O}(m)$. We also note that several other works [18, 36, 68] solve the hitting set problem deterministically in the context of graph spanners in CONGEST and CONGESTED-CLIQUE. However, these are not straightforward to implement in the low-memory MPC model.

Finally, it is worth observing that because of the shorter seeds, the MPC implementation of both primitives is significantly simplified as we can perform a brute-force search instead of using the method of conditional expectation.

3.3 Derandomization of Algorithmic Primitives

3.3.1 Approximation of Maximum Matching

The first algorithmic step for the derandomization of the connectivity algorithm of [13] consists in solving approximate maximum matching in graphs of maximum degree two. Coy and Czumaj proved the following theorem:

Theorem 3.2 (Theorem 4.2 of [22]) *Let $G = (V, E)$ be an undirected simple graph with maximum degree $\Delta \leq 2$. One can deterministically find a matching \mathcal{M} of G of size at least $m/8 = \Omega(m)$ in $O(1)$ MPC rounds with local space $\mathbf{S} = O(n^\delta)$, and global space $\mathbf{S}_{\text{Global}} = O(n)$.*

By extending their algorithm with the seed reduction technique mentioned earlier, we prove the following result.

Theorem 3.3 *There exists an algorithm with the same properties as those in Theorem 3.2 that uses $\tilde{O}(n)$ total computation.*

We start by reviewing the main idea used in the algorithm of Theorem 3.2.

Randomized Algorithm The algorithm of Theorem 3.2 is based on derandomizing the following simple random process. Let $\{X_e : e \in E\}$ be a family of pairwise independent random variables with $X_e = 1$ with probability $p = 1/4$ and $X_e = 0$ otherwise. Now, let \mathcal{M} be the matching that includes each edge e with $X_e = 1$ and $X_{e'} = 0$ for every neighboring edge e' . The expected size of this matching is:

$$\begin{aligned} \mathbb{E}[|\mathcal{M}|] &= \sum_{e \in E} \Pr[e \in \mathcal{M}] \geq \sum_{e \in E} \Pr[X_e = 1] - \sum_{\substack{e' \in E \setminus \{e\}: \\ e' \cap e \neq \emptyset}} \Pr[X_e = 1 \cap X_{e'} = 1] \\ &\geq m \cdot (p - 2p^2) \geq \frac{m}{8}, \end{aligned}$$

where the second inequality follows from pairwise independence. Hence, random variables can be specified by a seed of length $2 \log n + O(1)$ by Lemma 2.8. As explained in [22], this allows using the method of conditional expectation to deterministically find a matching of size at least $m/8$ in $O(1)$ MPC rounds.

Reducing the Seed Length We next show how one can further reduce the seed length to $O(\log \log n)$. The main observation is that the above analysis holds as long as for any two neighboring edges the two corresponding random variables are pairwise independent. This motivates the following approach. First, we assign to each edge e a color $c(e)$ from the set $\{1, 2, \dots, C\}$ for $C = O(\log^2 n)$ by applying Lemma 2.12, such that two neighboring edges get assigned a different color. Let $\{X_c : c \in [C]\}$ be a family of pairwise independent random variables with $X_c = 1$ with probability $p = 1/4$ and $X_c = 0$ otherwise. We now include each edge e in \mathcal{M} if $X_{c(e)} = 1$ and $X_{c(e')} = 0$ for every neighboring edge e' . The same calculations as above shows that $\mathbb{E}[\mathcal{M}] \geq \frac{m}{8}$.

MPC Algorithm We are now ready to present our deterministic MPC algorithm that proves Theorem 3.3. In the following, we say that something can be efficiently computed if there exists a deterministic MPC algorithm running in $O(1)$ rounds with local space $\mathbf{S} = O(n^\delta)$, global space $\mathbf{S}_{Global} = O(n)$ and using $\tilde{O}(n)$ total computation.

Let $\mathcal{H} = \{h : [C] \mapsto \{0, 1\}^2\}$ be a family of 2-wise independent hash functions of size at most $2^{2 \cdot \log C + O(1)} = \text{poly}(\log n)$ obtained using Lemma 2.8. Observe that each hash function $h \in \mathcal{H}$ defines a matching $\mathcal{M}(h)$ that includes each edge e with $h(c(e)) = 0$ and $h(c(e')) \neq 0$ for every neighboring edge e' .

The analysis of the randomized algorithm above implies that choosing a hash function h uniformly at random from \mathcal{H} results in a matching of expected size at least $m/8$. In particular, this guarantees the existence of a hash function h^* with $\mathcal{M}(h^*) \geq m/8$. Next, we explain how one can efficiently compute $|\mathcal{M}(h)|$ for every $h \in \mathcal{H}$ and choose a hash function that yields a matching of size at least $m/8$.

First, we efficiently compute the coloring c using Lemma 2.12. Then, since the size of \mathcal{H} is poly $\log n$, we can store one number per hash function on every machine. Each machine M_j , which is responsible for some edges $\mathcal{E}_j \subseteq [E]$, can compute locally the number of edges $\mathcal{M}^j(h) \subseteq E_j$ in the matching generated by $h \in \mathcal{H}$ within a single round. Then, we efficiently aggregate these numbers across all machines to compute the size of the matching $\mathcal{M}(h) = \sum_j \mathcal{M}^j(h)$ for every hash function h . The best $h^* \in \mathcal{H}$ for which $\mathcal{M}(h^*) \geq \frac{m}{8}$, breaking ties arbitrarily, yields our approximate maximum matching. Finally, let us note that the global memory occupied by the hash functions across all machines \mathbf{M} is $\mathbf{M} \cdot |\mathcal{H}| \ll \mathbf{M} \cdot O(n^\delta) = O(n)$ and the overall computation performed to evaluate each hash function for every edge is $|\mathcal{H}| \cdot \text{poly}(\log n) \cdot O(n) = \tilde{O}(n)$.

3.3.2 Hitting Set

In this section, we give a deterministic MPC algorithm for the following hitting set variant defined in [22]:

Definition 3.4 (Hitting Set for Leader Election) *Let S_1, \dots, S_n be subsets of $[n]$ with $i \in S_i$ and $|S_i| = b$, for each $i \in [n]$. The goal is to find a (small) hitting set $\mathcal{L} \subseteq [n]$, that is, a set for which $S_i \cap \mathcal{L} \neq \emptyset$ holds for all $i \in [n]$.*

Coy and Czumaj [22] gave an algorithm with the same parameters as those of the random sampling approach in [13], except that they need large poly(n) computation.

Theorem 3.5 (Theorem 5.6 of [22]) *Let b and n be integers with $\log^{10}(n) \leq b \leq n$. One can deterministically find a subset $\mathcal{L} \subseteq [n]$ that solves the Hitting Set for Leader Election problem with $|\mathcal{L}| \leq O(n(\min\{b, \mathbf{S}\})^{-1/5})$ within a constant number of MPC rounds using local space $\mathbf{S} = O(n^\delta)$, global space $\mathbf{S}_{\text{Global}} = O(nb)$, and total computation poly(n).*

We extend the randomized approach their algorithm relies on by using the method of alterations and reducing the amount of randomness needed to prove the following result:

Theorem 3.6 *There exists an algorithm with the same properties as those in Theorem 3.5 with two differences. The total computation reduces to $O(n \cdot \text{poly}(b))$ and the global space increases to $O(n \cdot \text{poly}(b))$.*

We will show in Section 3.4 that the algorithm from Theorem 3.6 together with minor changes to the parameters of the connectivity algorithm results in a deterministic connectivity MPC algorithm with near-linear total computation.

Review of Hitting Set Algorithm of Coy and Czumaj

Consider adding each element to \mathcal{L} with probability $p = b^{-1/5}$. Assuming full independence, the assumption that $b \geq \log^{10}(n)$ together with a simple Chernoff Bound implies that \mathcal{L} is a hitting set with high probability. The high probability bound still holds with $O(\log_b n)$ -wise independence, but fails to hold with $o(\log_b n)$ -wise independence. As n k -wise independent random variables require a seed length of $\Omega(k \log n)$, using $O(\log_b n)$ -wise independence would not result in a seed length of $O(\log n)$, which is necessary for an $O(1)$ MPC round derandomization based on the method of conditional expectation. To shorten the seed length, the authors of [22] use so-called k -wise ε -approximately independent random variables for $k = 15 \log_b(n)$ and $\varepsilon = n^{-6}$. In particular, their starting point is the following theorem.

Theorem 3.7 (Theorem 5.2 of [22]) *Let $\log^{10}(n) \leq b \leq n$, k be even with $k = 15 \log_b(n) \geq 4$, $\varepsilon = n^{-6}$, and $p = b^{-1/5}$. Then, if X_1, X_2, \dots, X_n are k -wise ε -approximately independent random variables with $X_i = 1$ with probability $b^{-1/5}$ and $X_i = 0$ otherwise. Then each of the following $n + 1$ events holds with probability at least $1 - 9n^{-3}$:*

- $\sum_{j \in S_i} X_j > 0$ for every $1 \leq i \leq n$, and
- $\sum_{i=1}^n X_i \leq 2nb^{-\frac{1}{5}}$.

Next, we explain our randomized approach that bears some similarities with that of Theorem 3.7 but relies only on pairwise independence and computes a *partial* independent set, which can be fixed to become a full hitting set deterministically. Then, we discuss how to reduce its seed length and its deterministic implementation on an MPC with strongly sublinear memory.

Pairwise Analysis As a first step, we show that a minor modification to their randomized hitting set algorithm results in a hitting set of expected size at most $2nb^{-1/5}$, assuming only pairwise independence. As before, each element joins \mathcal{L} with probability $p = b^{-1/5}$. In expectation, $b \cdot p = b^{4/5}$ elements are sampled from each set. Using only pairwise independence and Chebyshev's inequality (Corollary 2.10), this implies that a set is bad, i.e., no element is sampled from it, with probability at most $\frac{1}{b^{4/5}}$. Hence, by adding for each unhit set an arbitrary element to \mathcal{L} , at most $n/b^{4/5}$ additional elements are added to \mathcal{L} in expectation, resulting in a hitting set of expected size at most $n(b^{-1/5} + b^{-4/5})$.

Reducing The Seed Length From the pairwise analysis above, we directly get a seed length of $O(\log n)$. Next, we show how to reduce the seed length to $O(\log b)$, which allows for a simpler brute-force search. We again employ a coloring idea, which is based on the simple observation that we only require pairwise independence between elements contained in the same set. Hence, the goal is to color the elements with $\text{poly}(b)$ colors such that all elements in a given set S_i are colored with a different color.

In general, this may not be possible as there might exist elements which are contained in many sets. Fortunately, a simple calculation shows that there exist at most n/b elements which are contained in more than b^2 different sets. Hence, by directly adding these elements to \mathcal{L} , we can assume “for free” that each element is contained in at most b^2 sets, which we will do from now on.

We can then obtain a coloring with the desired properties by finding a proper coloring in the graph G_{conflict} , defined as follows. The vertex set consists of one vertex for each of the n elements. Moreover, two elements are connected by an edge if there exists a set which contains both elements. Note that the maximum degree Δ_{conflict} of G_{conflict} is upper bounded by b^3 . This follows from our assumption that each element is contained in at most b^2 sets. Therefore, we can efficiently color G_{conflict} with $C = O(\Delta_{\text{conflict}}^2 \log^2(n)) = O(b^6 \log^2 n)$ colors. For each $i \in [n]$, let $c(i)$ denote the color assigned to the i -th element. Note that it directly follows from the definition of G_{conflict} that all elements in a given set are assigned a different color.

Our randomized process that produces a hitting set with the desired properties works as follows. Let $\{X_c : c \in [C]\}$ be a family of pairwise independent random variables with $X_c = 1$ with probability $p = b^{-1/5}$ and $X_c = 0$ otherwise. We assume that $1/p$ is a power of 2, i.e., there is $\ell \in \mathbb{N}$ with $2^\ell = b^{1/5}$. By Lemma 2.8, we can generate these variables with a seed of length $2(\ell + \log C) + O(1) = O(\log b)$. Now, we add each element i with $X_{c(i)} = 1$ to \mathcal{L} . Then, for each set S_i with $\sum_{j \in S_i} X_{c(j)} = 0$, we add the element $i \in S_i$ to \mathcal{L} . By the above reasoning, \mathcal{L} is a hitting set of expected size $O(nb^{-1/5})$.

MPC Algorithm It remains to discuss the MPC implementation to prove Theorem 3.6. In the following, we say that something can be efficiently computed if there exists a deterministic MPC algorithm running in $O(1)$ rounds with local space $= O(n^\delta)$, global space $O(n \text{poly}(b))$, and using $O(n \cdot \text{poly}(b))$ total computation. In the preprocessing step, we add all elements which are contained in at least b^2 sets to the hitting set and remove all sets which contain at least one such element from consideration. The preprocessing step requires computing for each element how many sets it is contained in. This can be done efficiently by using the colored summation primitive.

Next, we explain how to efficiently construct the graph G_{conflict} . We generate the edges of G_{conflict} in two steps. First, each set $S = \{e_1, e_2, \dots, e_b\}$ creates $\binom{b}{2}$ entries $\{\{e_i, e_j\} : i \neq j \in [b]\}$. This can easily be done with $\text{poly}(b)$ global space per set and $\min(\mathbf{S}, \text{poly}(b))$ local space in $O(1)$ rounds by using the primitives of Lemma 2.5. Hence, we can efficiently generate all these edges in parallel. Afterwards, we use the duplicate removal procedure of Lemma 2.5 to remove duplicate edges.

As G_{conflict} has maximum degree b^3 , we can use Lemma 2.12 to efficiently compute a coloring of G_{conflict} with $C = O(b^6 \log^2 n) = \text{poly}(b)$ colors. As before, we denote with $c(i)$ the color assigned to the i -th element. For $\ell := \log_2(b^{1/5})$, let $\mathcal{H} = \{h : [C] \mapsto \{0, 1\}^\ell\}$ be a family of 2-wise independent hash functions of size at most $2^{2(\ell + \log C) + O(1)} = \text{poly}(b)$ such that evaluating a function from \mathcal{H} takes time $\text{poly}(\ell, \log C) = \text{poly}(\log b)$ time, by Lemma 2.8.

For each function $h \in \mathcal{H}$, we define a hitting set \mathcal{L}_h as follows. First, each element i with $h(c(i)) = 0$ is contained in \mathcal{L}_h , where $h(c(i))$ denotes the length- ℓ bit sequence for $c(i)$ by the corresponding integer in $\{0, \dots, \ell - 1\}$. Moreover, if for a given set S_i no element contained in it was added in the first step, then we add element i to \mathcal{L}_h . The discussion above implies that there exists at least one hash function $h \in \mathcal{H}$ with $|\mathcal{L}_h| = O(nb^{-1/5})$. Using Lemma 2.5, it is easy to see that for a single hash function $h \in \mathcal{H}$, we can efficiently compute \mathcal{L}_h and its size. As \mathcal{H} only contains $\text{poly}(b)$ hash functions, this implies that we can efficiently compute \mathcal{L}_h for every $h \in \mathcal{H}$. After we have done this, we can output the smallest hitting set \mathcal{L}_{h^*} . As remarked above, \mathcal{L}_{h^*} has size $O(nb^{-1/5})$, which finishes the proof.

3.4 Connectivity Algorithm

In this section, we discuss the necessary changes to the randomized connectivity algorithm of Behnezhad et al. [13] and its analysis, in order to prove the main result of this section. The deterministic approximate matching from Section 3.3.1 is used to replace steps 5 and 6 of Algorithm 2 of [13]. The same modification was already done by [22] and they showed that the total number of vertices drop by a constant factor, assuming that no isolated vertex exists. Hence, by applying this modified algorithm $O(\log \log \frac{m}{n} n)$ times, one can in $O(\log \log \frac{m}{n} n)$ rounds ensure that $m \geq n \log^C n$, for a given constant C . Moreover, all the steps of the modified deterministic algorithm can be implemented by invoking the primitives of Lemma 2.5, which in particular ensures that the algorithm can be implemented with total computation $\tilde{O}(m)$. Thus, it remains to prove that Algorithm 1 of [13] can be implemented deterministically with the same asymptotic complexity and using $\tilde{O}(m)$ total computation, assuming $m \geq n \log^C(n)$ for a sufficiently large constant C . To this end, Coy and Czumaj proved the following lemma.

Lemma 3.8 (Lemma 6.3 of [22]) *Let S_i denote the set of saturated vertices at level i after Step 2 of the RELABELINTRALEVEL routine in [13], let L_i denote the set of selected leaders at level i after Step 3 of the same execution of RELABELINTRALEVEL, let β_i denote the budget of vertices at level i , let $b(v)$ denote the budget of vertex v , and let γ, ε be arbitrary constants such that $0 < \gamma, \varepsilon < 1$. If we make the following modifications to RELABELINTRALEVEL:*

- *set $\beta_{i+1} := \beta_i \cdot (\min\{\beta_i, n^\varepsilon\})^{\gamma/4}$,*
- *replace Step 3 of RELABELINTRALEVEL with any MPC algorithm that in $O(1)$ rounds selects $O\left(\frac{|S_i|}{(\min\{\beta_i, n^\varepsilon\})^\gamma}\right)$ leaders for each level i with high probability or deterministically, and*
- *replace the budget update rule in Step 4 of RELABELINTRALEVEL with*

$$b(v) := b(v) \cdot (\min\{b(v), n^\varepsilon\})^{\gamma/4},$$

then the connectivity algorithm of [13] remains correct with the same asymptotic local and global space complexity.

We prove the same result as the above lemma using the deterministic hitting set algorithm from Section 3.3.2. The main technical challenge will be to ensure that our procedure, which adds a polynomial factor (in b) increase in the memory and computation required, can be run in parallel using linear global space and total computation.

Lemma 3.9 *Let $c \geq 3$ be the smallest integer such that both the global space and the total computation required by the algorithm from Theorem 3.6 are bounded by $n \cdot b^c$, and let $\varepsilon = \delta/c$ so that $n^{c \cdot \varepsilon} \leq n^\delta$. The same result as that of Lemma 3.8 can be achieved with the following modifications to RELABELINTRALEVEL:*

- *set $\beta_{i+1} := \beta_i \cdot (\min\{\beta_i, n^\varepsilon\})^{\frac{\gamma}{4c}}$,*
- *replace Step 3 of RELABELINTRALEVEL with any MPC algorithm that in $O(1)$ rounds selects $O\left(\frac{|S_i|}{(\min\{\beta_i, n^\varepsilon\})^\gamma}\right)$ leaders for each level i with high probability or deterministically using at most $n\beta_i^\varepsilon$ global space and total computation, and*
- *replace the budget update rule in Step 4 of RELABELINTRALEVEL with*

$$b(v) := b(v) \cdot (\min\{b(v), n^\varepsilon\})^{\frac{\gamma}{4c}},$$

and by replacing the initial budget $(\frac{m}{n})^{1/2}$ assigned to each vertex with $(\frac{m}{n})^{1/2c}$ in Algorithm 1 of [13]. Then, the connectivity algorithm of [13] remains correct with the same asymptotic local and global space complexity. Moreover, the resulting total computation is $O(m)$.

Proof We need to show that all claims and lemmas involving the modified steps of Algorithm 1 of [13] do not affect its correctness nor its bounds on local and global memory. As in [22], we prove the following three key properties:

- i) for any vertex v , $\ell(v)$ never exceeds $O(\log \log_{m/n} n)$ (cf. [13, Lemma 15]),
 - ii) the global space used is $O(\mathbf{S}_{Global})$ (cf. [13, Lemma 17]),
 - iii) the sum of the squares of the budgets is $O(\mathbf{S}_{Global})$ (cf. [13, Lemma 21]).
- i) Recall that the budget of each vertex is increased as $\beta_{i+1} := \beta_i \cdot (\min\{\beta_i, n^\epsilon\})^{\frac{\gamma}{4c}}$ and that $\beta_0 = (\frac{m}{n})^{1/2c}$. Since the budget of any vertex cannot exceed n , we have that there are at most $O(\log \log_{m/n} n)$ levels as required.
- ii) Let n_i denote the number of vertices which *ever* reach level i over the course of the algorithm. In the proof of Lemma 17 of [13], it is shown that the total sum of the budget increases throughout the algorithm is $O(m)$, namely

$$\sum_{i=1}^L \beta_i n_i = O(m).$$

We extend this claim and prove that the total sum of the global space used by all hitting set instances over all iterations of the algorithm is bounded by $O(m)$, that is

$$\sum_{i=1}^L \beta_i^c \cdot n_i = O(m).$$

Analogously to [22], we first show that $\beta_{i+1}^c \cdot n_{i+1} \leq \beta_i^c \cdot n_i$. Specifically, the number of vertices at level i removed from the graph (i.e., not marked as a leader) per vertex marked as leader is at least:

$$\frac{|S_i \setminus L_i|}{|L_i|} = \frac{|S_i| - |L_i|}{|L_i|} = \Omega((\min\{\beta_i, n^\epsilon\})^\gamma) \gg (\min\{\beta_i, n^\epsilon\})^{\gamma/2}.$$

It then follows that

$$\begin{aligned} \beta_{i+1}^c \cdot n_{i+1} &= \left(\beta_i \cdot (\min\{\beta_i, n^\epsilon\})^{\frac{\gamma}{4c}} \right)^c n_{i+1} \\ &< \left(\beta_i^c \cdot (\min\{\beta_i, n^\epsilon\})^{\frac{\gamma}{4}} \right) \left(n_i (\min\{\beta_i, n^\epsilon\})^{-\gamma/2} \right) \\ &\leq \beta_i^c \cdot n_i. \end{aligned}$$

Using the fact that the maximum possible level for a vertex is $L = O(\log \log n)$, we obtain

$$\sum_{i=1}^L \beta_i^c \cdot n_i \leq L \cdot (\beta_0^c \cdot n_0) \leq O(\log \log n) \cdot \left(\frac{m}{n} \right)^{\frac{1}{2}} \cdot n,$$

where the last inequality comes from the fact that $\beta_0 = \left(\frac{m}{n}\right)^{\frac{1}{2c}}$. Note that we can assume that $m \geq n \log^{20c}(n)$ and therefore each vertex has an initial budget of $\beta_0 = (m/n)^{1/2c} \geq \log^{10}(n) \gg O(\log \log n)$, as required by Theorem 3.6. This yields

$$O(\log \log n) \cdot \left(\frac{m}{n}\right)^{\frac{1}{2}} \cdot n \ll \left(\frac{m}{n}\right)^{\frac{1}{2}} \cdot \left(\frac{m}{n}\right)^{\frac{1}{2}} \cdot n = O(m).$$

iii) Follows by the same line of reasoning as in property (ii).

By the choice of c , repeating the same calculations as in property (ii) proves that the total computation required by running our deterministic hitting set algorithm over all instances in each iteration of the algorithm does not exceed $O(m)$. Moreover, Lemma 2.5 implies that all the other steps of the algorithm can be implemented with total computation $\tilde{O}(m)$. \square

We are now ready to prove Theorem 3.1. We apply Lemma 3.9 using our Hitting Set for Leader Election algorithm from Theorem 3.6 setting $\gamma = \frac{1}{5}$ (Note that $m \geq n \log^C(n)$ for a sufficiently large constant C implies $\beta_0 \geq \log^{10} n$). Then, it follows directly from Lemma 6.4 of [22] combined with Lemma 3.9 that copies of our hitting set algorithms can be run in parallel, for each possible level and in a constant number of rounds, within optimal global space and $\tilde{O}(m)$ total computation. Thus, we proved that all relevant aspects of the proof of correctness have been adjusted in comparison to [13, 22]. Finally, as noted in [22], our extension of Lemma 15 in [13] proves that the number of iterations remains asymptotically the same and that the deterministic algorithms replacing the $O(1)$ -round random sampling approach take asymptotically the same number of rounds. Thus, we conclude that the round complexity is not affected.

Deterministic MPC Algorithms for MIS and MM

We study the deterministic complexity of MIS and MM in terms of the arboricity in Massively Parallel Computation with strongly sublinear memory. Our $O(\log \log n)$ -round algorithm that reduces the maximum degree to $\text{poly}(\lambda)$ is presented in two parts. In Section 4.1, we derandomize a key degree-reduction primitive by Barenboim et al. [9] which is plugged into the pipeline framework of Ghaffari et al. [35] (Section 4.2) to obtain our result. In the near-linear global memory regime (Section 4.3), we give the first $O(\log n)$ -round deterministic algorithm for the problems of MIS and MM. We further improve this bound to $O(\log^2 \log n)$ for $\text{poly}(\log n)$ -arboricity graphs and $O(\log \log n)$ for graphs with arboricity at most $\log^{O(1)} \log n$. Finally, in the super-linear global memory regime (Section 4.4), we get a round complexity of $O(\log \lambda + \log \log n)$ by combining our degree reduction with a result of Czumaj et al. [25].

4.1 Arboricity-Based Degree Reduction

In this section, we present a fast degree-reduction algorithm based on the derandomization of the LOCAL arboricity-based degree reduction due to Barenboim et al. [9]. In a nutshell, this algorithm works in constant-round iterations. It appoints a set of high-degree nodes with many low-degree neighbors and let low-degree neighbors with certain properties mark themselves with a probability that depends on the current maximum degree. Marked nodes join the MIS if a marked node does not have any marked neighbor. Similarly, in the MM case, a low-degree node sends a matching proposal to a random high-degree neighbor, which accepts one arbitrarily. The pairwise analysis and its subsequent derandomization shows that in each iteration the number of high-degree nodes shrinks deterministically

by a factor Δ and that only a constant-hop neighborhood around (good) high-degree nodes is involved.

4.1.1 LOCAL Algorithmic Primitive

As the first step, we describe our LOCAL algorithm **ARB-ALG** that adapts the arboricity-based degree reduction from [9]. We will prove the following:

Lemma 4.1 *Let $\Delta = O(n^\delta)$. Suppose the input graph G has arboricity at most λ and maximum degree at most $\Delta \geq \text{poly}(\lambda)$. Let $H_G := \{v : \deg_G(v) \geq \sqrt{\Delta}\}$ denote the set of high-degree vertices in G . There is a LOCAL algorithm (**ARB-ALG**) that in expectation computes an independent set \mathcal{I} (or a matching \mathcal{M}) of G such that $|H_{G'}| \leq |H_G|/\Delta$, where G' is the subgraph of G where vertices in \mathcal{I} and their neighbors (or matched vertices in \mathcal{M}) are removed. The algorithm has $r = O(1)$ round complexity and the following properties.*

1. *Let H^+ denote the r -hop neighborhood of H_G . The algorithm involves only (a subset of) nodes at distance at most r from H_G . Then, vertices outside H^+ will stay in G' , and their degrees in G' are unchanged.*
2. *The state description and (random) seed for each vertex in each round have length at most $\text{poly} \log(n)$ bits. The overall information exchanged with neighbors, the space usage for computing the new state of vertex v , and the messages sent from v consist of at most $\deg_G(v) \cdot \text{poly} \log(n)$ bits.*

The algorithm has two phases: a preparation phase in which a subset of low-degree nodes is selected, and an MIS and MM phase in which low-degree nodes selected in the previous phase form a partial solution. We next present each of these two phases and discuss their derandomization in Section 4.1.2.

Preparation Phase Let G be the graph under consideration and let $H = \{v \in V \mid \deg_G(v) \geq \sqrt{\Delta}\}$ be the set of high-degree vertices. The algorithm partitions H into three subsets $\mathcal{J}, H_{\text{Bad}}, H_{\text{Good}}$ and identifies a subset $\mathcal{S} \subset V$ consisting of low-degree nodes that are incident to H . Then, it will select a subset $\mathcal{S}_{\text{Good}} \subseteq \mathcal{S}$ of good low-degree nodes.

Let $\mathcal{J} = \{v \in H \mid \deg_H(v) \geq \deg_G(v)/2\}$ be the set of (high-degree) vertices that have a majority of their neighbors in H . Roughly speaking, vertices in \mathcal{J} have too few low-degree neighbors and are thus removed from consideration. It follows that any node $v \in H' \stackrel{\text{def}}{=} H \setminus \mathcal{J}$ has $\deg_{V \setminus H}(v) \geq \deg_G(v)/2 \geq \sqrt{\Delta}/2$ since at most $\deg_G(v)/2$ of its neighbors can be in H . The set H' is further split in H_{Bad} and H_{Good} .

Let each node $v \in H'$ discard all but $2 \cdot \Delta^{0.25}$ edges such that $\deg_{V \setminus H}(v) = 2 \cdot \Delta^{0.25}$. Accordingly, let \tilde{E} be the set of remaining edges crossing the cut $(H, V \setminus H)$ and define $\mathcal{S} = \{u \mid v \in H' \text{ and } \{u, v\} \in \tilde{E}\}$ to be the low-degree

neighborhood of H' with respect to \tilde{E} . We now define the subsets $\mathcal{S}_{\text{Good}}$ and $H_{\text{Bad}}, H_{\text{Good}}$ meeting the following conditions, where $\beta = \Delta^{0.025}$.

Good Low-Degree: Good \mathcal{S} -nodes have less than β neighbors in H' and less than β^2 neighbors in \mathcal{S} , i.e., $\mathcal{S}_{\text{Good}} = \{u \in \mathcal{S} \mid \deg_{\tilde{E}}(u) < \beta, \deg_{\mathcal{S}}(u) < \beta^2\}$. Nodes $\mathcal{S} \setminus \mathcal{S}_{\text{Good}}$ can be ignored for the rest of the algorithm.

Good High-Degree: Good H -nodes have at least $\Delta^{0.25}$ good neighbors in \mathcal{S} , i.e., $H_{\text{Good}} = \{v \in H' \mid \deg_{(\mathcal{S}_{\text{Good}}, \tilde{E})}(v) \geq \Delta^{0.25}\}$, where $\deg_{(\mathcal{S}_{\text{Good}}, \tilde{E})}(v)$ denotes the degree of v in the respective subgraph. Then, $H_{\text{Bad}} = H' \setminus H_{\text{Good}}$.

Further, let each vertex $v \in H_{\text{Good}}$ select exactly $\Delta^{0.25}$ nodes among its $\mathcal{S}_{\text{Good}}$ neighbors arbitrarily and discard the other edges incident to v . Then, since the maximum degree in the graph induced by $\mathcal{S}_{\text{Good}}$ is less than β^2 , the neighborhood of v contains a subset $T(v) \subset (\mathcal{S}_{\text{Good}} \cap N(v))$ of at least $\Delta^{0.25}/\beta^4$ nodes, each pair of which is at distance at least 3 with respect to $\mathcal{S}_{\text{Good}}$. We greedily select $T(v)$ by letting every $\mathcal{S}_{\text{Good}}$ -node send its neighborhood in \mathcal{S} to each of the at most β nodes in H_{Good} it is incident to. Using this information, v can compute $T(v)$ locally. Note that the total information exchanged in this step is only $\sum_{v \in H'} \Delta^{0.25} \cdot \beta^2 \leq |H'| \cdot \sqrt{\Delta} = O(m)$. This step concludes our preparation phase.

We obtain that the number of *bad* nodes that are in either \mathcal{J} or H_{Bad} is upper bounded by $\frac{5\lambda|H|}{\beta-\lambda}$, which follows immediately from [9, Theorem 7.2]. Thus, in the next phase, we focus on removing all but a β fraction of H_{Good} nodes. In particular, each node $u \in \bigcup_{v \in H_{\text{Good}}} T(v) \subseteq \mathcal{S}_{\text{Good}}$ will take active part in the MIS (MM) algorithm to remove neighboring H_{Good} nodes.

MIS and MM Phase We analyze a simple random sampling approach that uses limited independence to build an independent set \mathcal{I} (matching \mathcal{M}) on the vertex set $\mathcal{S}_{\text{Good}}$. We prove that as a result of this randomized process every H_{Good} -node is removed from the graph, i.e., neighboring a vertex in the independent set or matched, with probability at least $1 - 1/\Delta^{\Omega(1)}$. This implies that at most a $\Delta^{\Omega(1)}$ fraction of H_{Good} -nodes survive in expectation.

Independent Set: We sample each node $u \in \mathcal{S}_{\text{Good}}$ with probability $p = 1/\beta^3$ using pairwise independent random variables. Then, u joins the MIS \mathcal{I} if none of its (at most β^2) neighbors in $\mathcal{S}_{\text{Good}}$ is sampled. Consider now an arbitrary node v in H_{Good} . Let $\{X_u\}_{u \in T(v)}$ be the random variables denoting the event that u joins \mathcal{I} and define $X = \sum_{u \in T(v)} X_u$ to be their sum. We have

$$\Pr[X_u = 1] \geq \Pr[u \text{ sampled}] - \sum_{u' \in N_{\mathcal{S}}(u)} \Pr[u, u' \text{ sampled}] \geq p - \beta^2 p^2, \quad (4.1)$$

where the second inequality follows from pairwise independence. It follows that the expected number of neighbors of v in the MIS is at least

$$\mathbb{E}[X] = \sum_{u \in T(v)} \Pr[X_u = 1] \geq \frac{\Delta^{0.25}}{\beta^4} \cdot (p - \beta^2 p^2) \stackrel{\text{def}}{=} \mu.$$

Next, we apply Chebyshev's inequality (Theorem 2.9) to prove the following upper bound on the probability that v has no neighbors in \mathcal{I} :

$$\Pr[X = 0] \leq \Pr[|X - \mathbb{E}[X]| \geq \mathbb{E}[X]] \leq \frac{\text{Var}[X]}{\mu^2} \leq \frac{5}{\beta}.$$

For any two vertices in $u, u' \in T(v)$ we have that $\mathbb{E}[X_u X_{u'}] \leq p^2$ by pairwise independence and disjointness of the 1-hop neighborhoods of u and u' . This together with Eq. 4.1 gives the following bounds on $\text{Var}[X_u]$ and $\text{Cov}[X_u, X_{u'}]$.

$$\begin{aligned} \text{Var}[X_u] &= \mathbb{E}[X_u^2] - \mathbb{E}[X_u]^2 \leq p - p^2(1 - \beta^2 p)^2 \leq p \quad \{p \leq \frac{1}{2\beta^2}\}, \\ \text{Cov}[X_u, X_{u'}] &= \mathbb{E}[X_u X_{u'}] - \mathbb{E}[X_u]\mathbb{E}[X_{u'}] \leq p^2 - p^2(1 - \beta^2 p)^2 \\ &\leq 2\beta^2 p^3. \end{aligned}$$

Then, let us simplify the subsequent calculations by observing that

$$\mu^2 = \left(\frac{\Delta^{0.25}}{\beta^4}\right)^2 \cdot (p - \beta^2 p^2)^2 \geq \frac{1}{2} \left(\frac{\Delta^{0.25}}{\beta^4}\right)^2 \cdot p^2 \quad \{p \leq \frac{1}{4\beta^2}\}.$$

Thus, by applying Theorem 2.9 and plugging in the above bounds, we get

$$\begin{aligned} \frac{\text{Var}[X]}{\mu^2} &\leq \frac{\sum_{w \in T(v)} \text{Var}[X_w] + \sum_{w, w' \in T(v)} \text{Cov}[X_w, X_{w'}]}{\mu^2} \\ &\leq \frac{2\beta^4}{\Delta^{0.25}} \cdot \frac{1}{p} + 4\beta^2 p = \frac{2\beta^7}{\Delta^{0.25}} + \frac{4}{\beta} \leq \frac{5}{\beta}, \end{aligned}$$

where the last equality follows from our choice of $p = 1/\beta^3$ and $\beta = \Delta^{0.025}$. We conclude that the expected number of H_{Good} -nodes that do not have a neighbor in the MIS, and therefore survive to this phase, is at most $5|H_{\text{Good}}|/\beta$. Combined with the upper bound on \mathcal{J} and H_{Bad} from [9], the expected number of remaining high-degree nodes is at most

$$|\mathcal{J}| + |H_{\text{Bad}}| + \frac{5}{\beta} \cdot |H_{\text{Good}}| < \frac{5\lambda|H|}{\beta - \lambda} + \frac{5|H_{\text{Good}}|}{\beta} < \frac{10\lambda|H|}{\beta - \lambda}.$$

Since $\beta = \Delta^{0.025} \geq \text{poly}(\lambda)$, the number of high-degree nodes is reduced by a factor $\Delta^{\Omega(1)}$. Thus, after $O(1)$ repetitions of this two-phase procedure, the number of remaining high-degree nodes is at most $|H|/\Delta$ in expectation.

Matching: Similarly, our goal is to find a matching \mathcal{M} that matches all but $H_{\text{Good}}/\Delta^{\Omega(1)}$ good high-degree nodes in expectation. We run the following randomized process. Each $u \in \mathcal{S}_{\text{Good}}$ chooses an edge $\{u, v\}$ with $u \in T(v)$ uniformly at random and proposes to v that $\{u, v\}$ is included in the matching. Any $v \in H_{\text{Good}}$ receiving a proposal accepts one arbitrarily.

A node $v \in H_{\text{Good}}$ has at least $|T(v)|$ neighbors $u \in \mathcal{S}_{\text{Good}}$ with $\deg_{\mathcal{S}_{\text{Good}}}(u) < \beta$. Thus, the number of proposals that v receives in expectation is at least $|T(v)|/\beta = \frac{\Delta^{0.25}}{\beta^5}$. Using Chebyshev's inequality for pairwise independent random variables (Corollary 2.10), we directly obtain that v is unmatched with probability at most $\frac{5}{\beta}$. Note that the calculations are simplified as the matching proposal of each $\mathcal{S}_{\text{Good}}$ -node does not depend on that of its neighbors. As in the case of MIS, the overall number of high-degree nodes is reduced by a factor Δ by repeating this procedure $O(1)$ times.

Algorithm Properties: Let us now turn our attention to the two properties of Lemma 4.1.

1. Property 1 is proved by observing that low-degree nodes in \mathcal{I} (resp. \mathcal{M}) are at distance exactly one from H_{Good} , meaning that nodes at distance at least three are not removed from the graph and thus the degrees of nodes at distance at least four from H_{Good} remain unchanged. Moreover, since the algorithm requires $O(1)$ rounds and it is repeated a constant number of times, the overall running time r is $O(1)$.
2. Property 2 follows from the fact that the size of the messages exchanged in the two phases is at most $\deg(v) \cdot \text{poly} \log(n)$ for all vertices.

4.1.2 Derandomization and MPC Implementation

We present separately the derandomization of a single run and multiple runs of the algorithm ARB-ALG from Lemma 4.1 in low-memory MPC.

Single Run Derandomization Our goal is to turn ARB-ALG into a fully-scalable deterministic algorithm with the same guarantees using $O(\log \Delta + \log \log n)$ random bits per run. The central observation is that the randomized steps of vertex sampling and matching proposal need 2-wise independence only among nodes in $T(v)$ and their neighbors in $\mathcal{S}_{\text{Good}}$, for every $v \in H_{\text{Good}}$. We legally color a *conflict* graph on nodes in $\mathcal{S}_{\text{Good}}$ in $O(1)$ rounds, where every two dependent nodes are incident to each other. Specifically, we use $O((\Delta')^2 \log^2 n)$ colors, for some Δ' , and assign distinct colors to nodes at distance¹ at most four. The conflict subgraph of G is constructed as follows.

¹Here, the distance between two vertices in $\mathcal{S}_{\text{Good}}$ is with reference to the subgraph on the nodes $\mathcal{S}_{\text{Good}} \cup H_{\text{Good}}$ including edges with both endpoints in $\mathcal{S}_{\text{Good}}$ and the union of the relevant edges for each high-degree node, that is, $\bigcup_{v \in H_{\text{Good}}} \bigcup_{u \in T(v)} \{u, v\}$.

Every $v \in H_{\text{Good}}$ adds an edge between each pair of vertices in $T(v) \cup N_{\mathcal{S}}(T(v))$, i.e., the inclusive neighborhood of $T(v)$ in \mathcal{S} becomes pairwise connected. The number of edges that each node v adds is at most

$$|T(v) \cup N_{\mathcal{S}}(T(v))|^2 \leq \left(\frac{\Delta^{0.25}}{\beta^4} + \frac{\Delta^{0.25}}{\beta^4} \cdot (\beta^2 - 1) \right)^2 \leq \Delta^{0.5} \leq \deg(v),$$

which is within our memory constraints and fits into the memory of a single machine. Then, the resulting degree of each node $u \in \mathcal{S}_{\text{Good}}$ is at most

$$\deg_{\mathcal{S}}(u) + \sum_{v \in H_{\text{Good}} \cap N(u)} \left(|T(v) \cup N_{\mathcal{S}}(T(v))| + \sum_{w \in T(v)} \deg_{\mathcal{S}}(w) \right) \leq \beta + \beta^3 \cdot \frac{\Delta^{0.25}}{\beta^4} \leq \Delta^{0.25}.$$

Thus, every $\mathcal{S}_{\text{Good}}$ -node has degree at most $\Delta' = \Delta^{0.25}$ in the augmented graph and can compute an $O(\Delta^{0.5} \log^2 n)$ coloring of $\mathcal{S}_{\text{Good}}$ -nodes in a constant number of rounds using Lemma 2.12. This concludes our coloring stage.

Next, we use the assigned colors as vertex IDs for the derandomization step. We map the set of $C \stackrel{\text{def}}{=} O(\Delta^{0.5} \log^2 n)$ colors to $\{0, 1\}^k$, for some integer $k > 0$, using a 2-wise independent hash family. Without affecting asymptotic results, we assume that $1/p$ is a power of 2, i.e., there exists $\ell \in \mathbb{N}$ with $2^\ell = \beta$.

Independent Set: Let $\mathcal{H} = \{h: [C] \mapsto \{0, 1\}^{3\ell}\}$ be a family of 2-wise independent hash functions of size at most $2^{O(\log \Delta + \log \log n)} = O(\text{poly}(n^\delta)) = O(n^\alpha)$, for a suitable choice of δ , by Lemma 2.8. Every function $h \in \mathcal{H}$ induces an independent set \mathcal{I}_h defined as follows. If a vertex $v \in \mathcal{S}_{\text{Good}}$ with $h(c(v)) = 0$ has no neighbors $w \in N(v) \cap \mathcal{S}_{\text{Good}}$ with $h(c(w)) = 0$, then we add v to \mathcal{I}_h . Then, each machine locally computes the number of H_{Good} -nodes that have no neighbors in \mathcal{I}_h for every hash function h . By the above analysis, there is one hash function $h^* \in \mathcal{H}$ with $|H_{\text{Good}} \setminus N(\mathcal{I}_{h^*})| \leq 5|H_{\text{Good}}|/\beta$. We aggregate these numbers across machines for every hash function and select one such h^* . Then, we add \mathcal{I}_{h^*} to \mathcal{I} and remove nodes in $\mathcal{I}_{h^*} \cup N(\mathcal{I}_{h^*})$ from the graph.

Matching: The procedure is analogous to the one above for MIS. Let $\mathcal{H} = \{h: [C] \mapsto \{0, 1\}^\ell\}$ be a family of pairwise independent hash functions. Each $h \in \mathcal{H}$ defines a matching \mathcal{M}_h as follows. Each $v \in H_{\text{Good}}$ selects an arbitrary neighboring vertex $u \in T(v)$ with $h(c(u)) = v$. Note that u can map in an arbitrary but consistent manner values in $[\beta - 1]$ to its neighbors' IDs. Then, each machine locally computes the number of H_{Good} -nodes that remains unmatched. We aggregate these numbers and find one hash function $h^* \in \mathcal{H}$ with $|H_{\text{Good}} \setminus \mathcal{M}_{h^*}| \leq 5|H_{\text{Good}}|/\beta$. We add the edges in \mathcal{M}_{h^*} to the matching \mathcal{M} and remove matched nodes in \mathcal{M}_{h^*} along with their edges.

Algorithm Properties: While Property 1 follows directly from the same argument as in the proof of Lemma 4.1, the size of the messages exchanged in the coloring step is $\deg(v) \cdot \text{poly log}(n)$ for all vertices except for a subset of vertices in $\mathcal{S}_{\text{Good}}$ for which the degree in the conflict graph of the coloring phase is $O(\Delta^{0.25})$, which may be larger than their degree in G . Therefore, the information exchanged at each node is at most $\max\{\Delta^{0.25}, \deg_G(v) \cdot \text{poly log}(n)\}$ bits. The number of edges in the conflict graph is $O(m)$ since every high-degree node v was responsible for the creation of $O(\deg(v))$ many edges. With this information, each vertex can then compute its color by exchanging messages of length at most $\text{poly log}(n)$ along every edge. Therefore, the total space needed is $O(n + m)$. Then, the state of each vertex is determined by the bits specifying h^* , i.e., the chosen hash function, of size $O(\log n)$. Lastly, communicating whether a vertex is in \mathcal{I} (resp. \mathcal{M}) requires $O(1)$ bits.

The following lemma summarizes our discussion in this paragraph.

Lemma 4.2 *The LOCAL algorithm ARB-ALG can be simulated in $O(1)$ rounds on an MPC with low-memory with the same guarantees as that of Lemma 4.1 such that the number of high-degree nodes in G decreases by a factor Δ deterministically.*

Multiple Run Derandomization Let G_0 be the graph under consideration. We assume that every node has collected into one machine its dr -hop neighborhood in G_0 and that we are given a proper coloring $\psi: V \mapsto [C]$ of G_0^8 in input, for which it holds that $d \cdot \log C = O(\delta \log n)$. Recall that in G_0^8 nodes at distance at most 8 in G_0 are adjacent to each other. Observe that the coloring ψ remains valid throughout the cd simulations of ARB-ALG since edges can be only removed. We will discuss how to obtain the coloring ψ later.

With these assumptions, our goal is now to show that cd executions of ARB-ALG can be simulated in $O(c)$ MPC rounds for some positive constant c . We group the cd runs of ARB-ALG into c stages consisting of d runs. We first show that each stage can be implemented deterministically in a constant number of rounds in the MPC model, using $O(n^\alpha)$ space per machine and $\tilde{O}(n + m)$ total space. Then, we discuss how to execute c stages sequentially, interleaved by a constant-round coordination step between any two consecutive stages. For simplicity, we first consider the MIS case and later extend it to MM.

Implementing one stage: For each execution we employ a 2-wise independent family \mathcal{H} , which maps $[C]$ to $\{0, 1\}^{3\ell}$ with $\ell = O(\log \Delta)$. Each hash function from \mathcal{H} requires $O(\log C + \log \Delta)$ bits to be specified. To simulate d executions at once, we consider a sequence of hash functions h_1, \dots, h_d that can be specified using $d \cdot O(\log C + \log \Delta) = O(\delta \log n)$ by choosing $d = O(\log_\Delta n)$. Hence, all possible sequences h_1, \dots, h_d of d hash functions from \mathcal{H} can be stored and evaluated on any single machine.

Therefore, for every high-degree node v , we distribute to the machine designated to v the entire family \mathcal{H} of pairwise independent hash functions. Let G_i denote the graph obtained after the i -th simulation, $0 \leq i \leq d$; we will construct G_1, \dots, G_d iteratively. The randomized algorithm at simulation i takes in input h_i chosen i.u.r. from \mathcal{H} and runs ARB-ALG on G_{i-1} . The new graph G_i is obtained from G_{i-1} by removal of $\mathcal{I}_{h_i} \cup N(\mathcal{I}_{h_i})$.

To derandomize these d simulations, each node considers all possible sequences of hash functions h_1, \dots, h_d from \mathcal{H} . The number of such sequences is $|\mathcal{H}|^d$. We go through all sequences of hash functions $\mathbf{h} = \langle h_1, \dots, h_d \rangle$ from \mathcal{H} and find independent sets $\mathcal{I}_{h_1}, \dots, \mathcal{I}_{h_d}$ for each sequence. Let $\mathcal{I}^{\mathbf{h}}$ be the union of the independent sets found in the d executions for the sequence of hash functions \mathbf{h} . Let $G^{\mathbf{h}}$ be the graph G_d obtained at the end of the simulations using sequence \mathbf{h} . At least one sequence \mathbf{h} ensures that the resulting graph $G^{\mathbf{h}}$ has at most n_0/Δ_0^d high-degree nodes, where n_0 is the number of high-degree nodes in G_0 . Our algorithm will select the sequence \mathbf{h}^* that minimizes the number of high-degree nodes in the resulting graph $G^{\mathbf{h}^*}$, and will output $G^{\mathbf{h}^*}$ and $\mathcal{I}^{\mathbf{h}^*}$ to the next stage.

Further, we prove by induction that every high-degree node $v \in G_0$ can determine whether it is still of high degree after d executions of ARB-ALG for a fixed sequence \mathbf{h} of hash functions, solely based on the (dr) -hop neighborhood of v in G_0 . For the base case of $d = 1$, each high-degree node v simulates ARB-ALG and finds its set of relevant low-degree neighbors $T(v)$ for the current simulation. Then, node v must check whether $h_1(u) = 0$ and $h_1(u') \neq 0$ for any $u \in T(v)$ and $u' \in N_{S_{\text{good}}}(u)$ in G_0 , in which case $v \in N_{G_0}(\mathcal{I}_{h_1})$. Similarly, every low-degree node u adjacent to a high-degree node can detect whether it is part of (or incident to) the independent set \mathcal{I}_{h_1} . Specifically, node u can simulate ARB-ALG and learn this information using the knowledge of its r -hop neighborhood.

For the j -th execution, by induction, every node knows the topology of its $(rd - r(j - 1))$ -th hop neighborhood in G_{j-1} . Recall that every execution of ARB-ALG takes r LOCAL rounds. Since $r(d - (j - 1)) \geq r$, for $j \leq d$, every node v can simulate the j -th execution of ARB-ALG in G_{j-1} and determine $\hat{N}_{G_{j-1}}(v) \cap (\mathcal{I}_{h_j} \cup N_{G_{j-1}}(\mathcal{I}_{h_j}))$. Hence, v can also compute its degree in G_j .

Therefore, for every node v , we run on the machine storing v the algorithm above: for every sequence \mathbf{h} of hash functions from \mathcal{H} , check whether $v \in \mathcal{I}^{\mathbf{h}} \cup N(\mathcal{I}^{\mathbf{h}})$ and compute its degree in $G^{\mathbf{h}}$. We can aggregate this information efficiently and find a sequence of hash functions \mathbf{h}^* that minimizes the number of high-degree nodes in $G^{\mathbf{h}^*}$. Such sequence of hash functions is used to define the independent set $\mathcal{I}^{\mathbf{h}^*}$. As a result, the obtained graph $G^{\mathbf{h}^*}$ has at most n_0/Δ_0^d high-degree nodes as desired.

Coordination step: It remains to prove that the above simulations can be repeated c times, resulting in the executions of $c \cdot d$ runs. To achieve that, it is enough to maintain updated the dr -hop neighborhoods at the beginning of each stage. Suppose that a node u knows its dr -hop neighborhood at the beginning of a stage in graph G_0 . Notice that G_d is obtained from G_0 by removing $\mathcal{I}_{h^*} \cup N(\mathcal{I}_{h^*})$. Therefore, to update the dr -hop neighborhood of u in G_d , it suffices that u knows all nodes in its dr -hop neighborhood in G_0 that are in $\mathcal{I}_{h^*} \cup N(\mathcal{I}_{h^*})$. This can be easily achieved in a single MPC round by letting every node $v \in \mathcal{I}_{h^*} \cup N(\mathcal{I}_{h^*})$ send whether it is part of the current solution to all nodes in its dr -hop neighborhood in G_0 .

Extension to Matching: The same approach can be easily extended to maximal matching by running the matching version of the algorithm ARB-ALG. Therefore, we obtain that $c \cdot d$ executions of ARB-ALG can be simulated deterministically using $O(n^\alpha)$ space per machine and $\tilde{O}(n + m)$ total space; provided a coloring ψ as defined above and that each node knows its dr -hop neighborhood. Accordingly, we get a matching \mathcal{M}_{h^*} , induced by a hash function sequence \mathbf{h}^* from \mathcal{H} , that reduces high-degree nodes to n_0 / Δ_0^{cd} .

Let us summarize the above discussion in the following Lemma.

Lemma 4.3 *Let G denote the input graph, Δ denote its maximum degree, and ψ be a given proper vertex-coloring of G^δ with C colors such that $d \cdot \log C = O(\delta \log n)$. Assume that every node in G knows its rd -hop neighborhood in G for some positive integer $d = O(\log_\Delta n)$. Then, there exists a strongly sublinear MPC algorithm that simulates cd executions of the LOCAL algorithm ARB-ALG in $O(c)$ rounds with the same guarantees as that of Lemma 4.1 such that the number of high-degree nodes in G decreases by a factor Δ^{cd} deterministically.*

4.2 Pipelining of Arboricity-Based Degree Reduction

In this section, we present a low-memory MPC algorithm that reduces the maximum degree of G to $\text{poly}(\lambda)$ in $O(\log \log n)$ rounds for any input graph G with maximum degree $\Delta = O(n^\delta)$ and arboricity λ . We refer to this algorithm as ARB-ALG PIPELINED and is summarized in the following theorem.

Theorem 4.4 *There is a strongly sublinear MPC algorithm (ARB-ALG PIPELINED) that given a graph with arboricity λ and maximum degree $\Delta = O(n^\delta)$ deterministically computes an independent set \mathcal{I} and a matching \mathcal{M} such that every node in the resulting subgraph G' has degree at most $\text{poly}(\lambda)$, where G' denotes the subgraph of G where vertices in \mathcal{I} and their neighbors (or matched vertices in \mathcal{M}) are removed. The algorithm takes $O(\log \log n)$ MPC rounds and uses $O(n^\alpha)$ local space, and $\tilde{O}(m + n)$ total memory.*

Our approach builds upon the work of [35] that introduces a key pipelining technique. Their pipeline consists of multiple concurrent simulations of the randomized LOCAL arboricity-based degree reduction of [9] combined with graph exponentiation. In the previous sections, we showed that such LOCAL algorithm can be derandomized and multiple executions of it can be efficiently simulated on a low-memory MPC. We now adapt their pipeline framework to the deterministic version of (roughly) the same algorithm. In the following by **ARB-ALG** we refer to the deterministic version of such algorithm obtained by Lemmas 4.2 and 4.3 .

Partial LOCAL simulation and MPC derandomization As introduced in [35], the first ingredient for pipelining multiple executions of a LOCAL algorithm, which involves only some parts of the graph, is the LOCAL simulation with incomplete information. As the current phase progresses and more vertices become irrelevant in the current phase, we can start running the next phase on the vertices that are no longer participating in the previous phase. When running phase ℓ , vertices that have not finished their computation in previous phases $1, 2, \dots, \ell - 1$ are called *pending*. As the state of pending vertices at the start of phase ℓ is still unknown, the messages sent from these vertices are temporarily marked as pending. If a vertex receives a pending message, also its state becomes pending and so on. Thus, when running phase ℓ , we are only able to simulate the behavior of non-pending vertices. As earlier phases terminate for some vertex, the state of this vertex in phase ℓ becomes known. This allows us to remove the pending marks and resume the computation of phase ℓ . The authors in [35] give the following simple observation:

Observation 4.5 ([35]) *Define a LOCAL algorithm A' that is the same as A except for the following additional rule: the messages sent from a pending vertex are marked as pending, and a vertex receiving a pending message becomes pending. If vertex v has no pending vertices in its R -hop neighborhood, then v 's state after R rounds of A' is non-pending, and is the same as its state in A after R rounds.*

Using this observation, we adapt **ARB-ALG** to account for the possibility of having pending vertices throughout its executions. We remark that **ARB-ALG** in MPC is equivalent to its randomized execution in LOCAL where every (non-pending) node makes its random decision according to the hash function that MPC machines agreed upon. Thus, we can combine observation 4.5 with Lemmas 4.2 and 4.3 to obtain the following statement.

Corollary 4.6 *Let n_0 be the number of non-pending vertices whose $(r \cdot cd)$ -hop neighborhood does not contain any pending vertex in G . The simulation of cd executions of the LOCAL algorithm **ARB-ALG** can be done in $O(c)$ rounds on an MPC with low-memory under the assumptions of Lemmas 4.2 and 4.3 such that n_0 is reduced by at least a factor Δ^{cd} deterministically.*

Moreover, we need the following lemma, which is equivalent to Lemma 5 of [35], to reduce the space usage of MPC simulations. When simulating R LOCAL rounds of ARB-ALG if we somehow know a subset U of vertices that will be eliminated by ARB-ALG then we only need to collect v 's R -hop neighborhood in the induced subgraph $G[V \setminus U]$ along with the messages sent from U when they are alive.

Lemma 4.7 *Suppose the algorithm ARB-ALG runs on an input graph $G = (V, E)$, and $U_0 \subseteq V$ is the set of vertices that are eliminated during the execution of ARB-ALG. Let $U \subseteq U_0$, and $v \in V \setminus U$. Then, we can simulate the behavior of v for the derandomization of d executions of ARB-ALG, provided that we have information of:*

1. *The dr -hop neighborhood of v in the induced subgraph $G[V \setminus U]$.*
2. *A coloring ψ as defined in Lemma 4.3.*
3. *The complete communication history along every edge $(u, w) \in E$ where $u \in U$ and w belongs to the neighborhood defined in Item 1, until u gets eliminated by ARB-ALG.*

Proof Using the complete information history, every vertex v deduces the state of each node in its dr -hop neighborhood. By doing that, vertex v can check whether there are any pending vertices in its dr -hop neighborhood. Similarly, if node v was marked pending at some earlier iteration, by using the communication history of nodes that are not present anymore at the current iteration, v can maintain its dr -hop neighborhood updated and resume the execution of the current phase. Provided that there are no pending marks in the dr -hop neighborhood, node v can simulate ARB-ALG by Corollary 4.6. \square

Pipelined Algorithm Let $\Delta_\ell := \Delta^{1/2^\ell}$, where Δ is the maximum degree of the input graph. The unpipelined algorithm sequentially runs $L = O(\log \log \Delta)$ phases, where phase ℓ , for $\ell \in [L]$ runs ARB-ALG for $O(\log_{\Delta_\ell} n)$ times with degree parameter $\Delta_{\ell-1}$, and reduces the maximum degree from $\Delta_{\ell-1}$ to Δ_ℓ .

The pipelined MPC algorithm runs in $O(\log \log n)$ iterations, each taking $O(1)$ MPC rounds. In each iteration, there are multiple phases concurrently being simulated by the MPC algorithm. The start of phase ℓ and that of $\ell + 1$ are interleaved by a short lag $t = O(1)$. This means that the simulation of phase ℓ starts at iteration $j = (\ell - 1)t + 1$, by running INITIALIZE(ℓ). It is convenient to denote the current iteration j by $j = (\ell - 1)t + i$ with $i \geq 1$, i.e., j is the i -th iteration since the start of phase ℓ . In each iteration j , the algorithm performs subroutines SIMULATE(ℓ, j), UPDATE, and EXPAND(ℓ, j), concurrently for all active phases ℓ (i.e, phases that have already started). The pseudocode for this procedure provided below is precisely the same as [35, Algorithm 1].

Algorithm 1 Structure of the pipelined MPC algorithm

Number of phases $L = O(\log \log \Delta)$.
 Lag parameter $t = O(1)$.
for iteration $j \leftarrow 1, 2, \dots, O(\log \log n)$ **do**
 Let L_{active} be the number of phases that have already started;
 if $j = (\ell - 1)t + 1$ for some $\ell \in [L]$ **then**
 INITIALIZE(ℓ) (*Phase ℓ starts in this iteration*)
 SIMULATE(ℓ, j) for all $\ell \in [L_{\text{active}}]$ concurrently
 UPDATE
 EXPAND(ℓ, j) for all $\ell \in [L_{\text{active}}]$ concurrently

In the following, we include the algorithm explanation and the proof steps of [35] for completeness with the a few changes for their derandomization. In iteration j , concurrently for each active phase ℓ , SIMULATE(ℓ, j) performs (part of) the LOCAL simulation and its derandomization, and removes the matched vertices (or the independent set and its neighbors) from the graph. After removing the vertices, in UPDATE we compute the degrees of vertices in the remaining graph, and update some other information. Then, EXPAND(ℓ, j) performs one graph exponentiation step and doubles the radius of the neighborhoods stored in memory. We set an upper bound $\Theta(\log_{\Delta_\ell} n)$ on the number of graph exponentiation steps performed in phase ℓ . Note that when expanding the neighborhood we may encounter pending vertices. Even though their degrees may not be upper bounded by $\Delta_{\ell-1}$, we include them in the graph exponentiation process as well.

Let $G(j)$ denote the subgraph induced by the remaining vertices at iteration j . Let $H_\ell(j)$ denote the set of vertices v with $\deg_{G(j)}(v) > \Delta_\ell$, and let $H_\ell^+(j)$ be the r -hop neighborhood of $H_\ell(j)$ in graph $G(j)$. From Lemma 4.1 (Property (1)), we observe that vertices outside $H_\ell^+(j)$ are not affected by phase $1, 2, \dots, \ell$ in the following iterations $j + 1, j + 2, \dots$. At the end of iteration j , we maintain the following invariants for all active phases ℓ .

Property 4.8 Let $d_i := 2^i r$ and $s := \lceil 10/\delta \rceil$. At the end of iteration j , we have:

1. **(Number of finished LOCAL executions)** For every $v \in H_\ell^+(j - 1)$, if there is no $u \in H_{\ell-1}^+(j - 1)$ satisfying $\text{dist}_{H_\ell^+(j-1)}(u, v) \leq sd_i \cdot r$, then we have computed v 's state after sd_i executions of ARB-ALG in phase ℓ .
2. **(Radius of collected neighborhoods)** For every $v \in H_\ell^+(j)$, we have collected the following information into one machine:
 - a) The d_i -hop neighborhood of v in graph $H_\ell^+(j)$.
 - b) The communication history during phase ℓ along every edge (u, w) , where vertex $u \notin H_\ell^+(j)$ was eliminated during phase ℓ , and w belongs to the neighborhood defined in (a).

Next, we describe how to implement $\text{INITIALIZE}(\ell)$, $\text{SIMULATE}(\ell, j)$, UPDATE , $\text{EXPAND}(\ell, j)$ in iteration $j = (\ell - 1)t + i$ using $O(1)$ MPC rounds and maintain property 4.8.

Lemma 4.9 *Assume property 4.8 holds at the end of iteration $j - 1$, for $j = (\ell - 1)t + i$. We can implement $\text{SIMULATE}(\ell, j)$, UPDATE , $\text{EXPAND}(\ell, j)$ (concurrently for all phases $\ell \in [L_{\text{active}}]$) using $O(1)$ MPC rounds so that at the end of iteration j , property 4.8 holds for all phases $\ell \in [L_{\text{active}}]$.*

Proof Note that $\text{SIMULATE}(\ell, j)$ and $\text{EXPAND}(\ell, j)$ are independent across all phases ℓ and can be executed concurrently.

- $\text{SIMULATE}(\ell, j)$. To maintain property 4.8 Item 1, it suffices to simulate ARB-ALG for $s \cdot 2^i$ times. We apply Lemma 4.7, with U being the set of vertices $u \notin H_\ell^+(j - 1)$ which were eliminated during phase ℓ . By property 4.8 Item 2, we already have the (d_{i-1}) -hop neighborhood of every vertex in $H_\ell^+(j - 1)$ (as well as the communication history required by Lemma 4.7) stored into one machine.

Then, we compute a coloring ψ of non-pending vertices in $(H_\ell^+(j - 1))^8$ where vertices at distance at most 8 are assigned different colors, by applying Lemma 2.12. Since $d_{i-1} = r2^{i-1} \geq 16 \cdot 2^i$, for $r \geq 32$, the number of colors C used by ψ is at most $\text{poly}(\Delta) \cdot \log^{(i)} n$ for $i < \log^* n$ and at most $\text{poly}(\Delta)$ otherwise. In particular this implies that $2^i \cdot \log C = O(\delta \log n)$, which satisfies the conditions of Corollary 4.6.

Thus, we can simulate $s \cdot 2^i$ executions of ARB-ALG for all $v \in H_\ell^+(j - 1)$ in $O(sd_i/d_{i-1}) = O(1)$ MPC rounds. When we simulate the behavior of v in phase ℓ , we also record its communication history. Since the degree in phase ℓ is at most $\Delta_{\ell-1}$, messages have size $O(\Delta_{\ell-1}^2 \cdot \text{poly} \log n)$.

- UPDATE . After vertices are removed from $G(j - 1)$ by $\text{SIMULATE}(\ell, j)$, let $G(j)$ be the induced subgraph of the remaining vertices, and compute the vertex degrees in graph $G(j)$. Then, for every ℓ , delete from $H_\ell(j - 1)$ the vertices whose degree dropped to $\leq \Delta_\ell$ and the ones that got removed, and obtain $H_\ell(j)$. Then similarly obtain $H_\ell^+(j)$. We also delete these vertices from the stored neighborhoods.
- $\text{EXPAND}(\ell, j)$. To maintain property 4.8 Item 2, we perform one graph exponentiation step in $O(1)$ MPC rounds as long as $\Delta_\ell^{d_i} = O(n^\alpha)$, i.e., until the d_i -hop neighborhood fits in the memory of a single machine. This condition implies that we need to stop the neighborhood expansion when $d_i = r2^i = O(\alpha \log_{\Delta_\ell} n)$. However, executing ARB-ALG for $\Omega(\log_{\Delta_\ell} n)$ times reduces the number of high-degree vertices by a factor $\Delta_\ell^{\Omega(\log_{\Delta_\ell} n)}$. Therefore, as soon as $i = O(\log \log_{\Delta_\ell} n)$ for phase ℓ , non-pending high-degree vertices will be removed after $O(1)$ iterations.

To perform one step of graph exponentiation, every $u \in H_\ell^+(j)$ requests the neighborhood of every other v in the stored d_{i-1} -hop neighborhood of u . The new radius becomes $2d_{i-1} = d_i$. Then it is easy to collect the required communication history of vertices in this neighborhood. \square

Lemma 4.10 *At iteration $j = (\ell - 1)t + 1$, we can implement INITIALIZE(ℓ) using $O(1)$ MPC rounds so that at the end of this iteration property 4.8 holds for phase ℓ .*

Proof We apply Corollary 4.6 in phase ℓ to run ARB-ALG on non-pending vertices for $2s$ times. This can be done in $O(s) = O(1)$ MPC rounds with $\tilde{O}(n + m)$ global memory. Then, we collect the $2r$ -hop neighborhood of vertices in $H_\ell^+(j)$ (and their communication history) in $O(1)$ MPC rounds. \square

It remains to show that our algorithm runs in $O(\log \log n)$ and satisfies the local memory and global memory constraints. The proof almost immediately follows from [35]; we discuss the changes in their proofs that have to be made in order to simulate the derandomized version of the LOCAL algorithm.

Theorem 4.11 *ARB-ALG PIPELINED runs in $O(\log \log n)$ rounds, requires $O(n^\alpha)$ for storing the d_i -hop neighborhood of each node, and uses $\tilde{O}(n + m)$ global memory.*

Proof Runtime: After $t \cdot L$ rounds, all $L = O(\log \log \Delta)$ phases have started their simulation. Recall that in phase ℓ after $\log_{\Delta_\ell} n$ executions of ARB-ALG, there are no remaining vertices with degree higher than Δ_ℓ . After another $O(\log \log n)$ rounds, every phase ℓ has at least been running for $i > \log \log_{\Delta_\ell} n$ iterations. Since $d_i = \Omega(\log_{\Delta_\ell} n)$, each phase completes in $O(1)$ rounds if there are non-pending vertices. By a simple induction, all phases terminate sequentially within $O(\log \log n)$ rounds, proving the claimed round complexity. The maximum degree of the remaining graph $G(j + L - 1)$ is at most $\Delta_L = \text{poly}(\lambda)$, as required.

Memory: We give only a proof sketch and refer to [35] for more details. For every phase ℓ at iteration j and for every node $v \in H_\ell^+(j)$, consider the $(sd_i + 2)r$ -hop neighborhood. Partition vertices in the subgraph $H_\ell^+(j)$ in ℓ parts defined as $P_\ell^0(j) \cup \dots \cup P_\ell^{\ell-1}(j)$ such that $P_\ell^q(j)$ contains vertices that have highest degree $(\Delta_{\ell-q}, \Delta_{\ell-q-1}]$ in their $(sd_i + 2)r$ neighborhood.

- Vertices in $P_\ell^0(j)$ are called *non-pending* and each of them can simulate sd_i executions. Thus, at the end of iteration j , we have $P_\ell^0(j) \leq \frac{n}{\Delta_{\ell-1}^{sd_i}} \cdot \Delta_{\ell-1}^{r+1}$, where the first term follows from sd_i executions of ARB-ALG according to property 4.8, and the second term by the definition of $H_\ell^+(j)$.
- By the definition of $P_\ell^q(j)$, $q > 0$, vertices in this class have a node in the subgraph $H_{\ell-q}(j - 1)$ at distance at most $(sd_i + 2)r$, and are called *pending*. This yields $P_\ell^q(j) \leq H_{\ell-q}(j - 1) \cdot \Delta_{\ell-q-1}^{(sd_i+3)r}$, since every vertex in $H_{\ell-q}(j - 1)$ may slow down vertices at distance at most $(sd_i + 2)r$.

Using the above upper bounds and the same argument as in [35], by induction, we get that $H_\ell^+(j) \leq \frac{2n}{\Delta_{\ell-1}^{(s-1)d_i}}$, where i is uniquely determined by j for each phase ℓ . Summing up over all phases, we observe that the term $\Delta_{\ell-q-1}^{(s-1)d_i} = \Delta_{\ell-1}^{((s-1)d_i 2^{q^{t-1}})/2^q}$ at the denominator dominates the sum for a sufficiently large t , making the overall summation dominated by a geometric series.

Then, the local per-node-memory for a node in $P_\ell^q(j)$ is at most $\Delta_{\ell-q-1}^{d_i+1} \cdot \Delta_{\ell-1}^2$. poly $\log n$, which account for both the graph exponentiation process and the communication history.

- **Local memory:** We show it for each vertex based on the part $P_\ell^q(j)$ it is in for $0 \leq q \leq \ell$. We consider two cases. When $q = 0$, we have

$$P_\ell^0(j) \geq 1 \Rightarrow \frac{n}{\Delta_{\ell-1}^{(s-1)d_i}} \geq 1 \Rightarrow n^\delta \geq \Delta_{\ell-1}^{\delta(s-1)d_i} \geq \Delta_{\ell-1}^{d_i+3}.$$

For $q \in [1, \ell]$, it holds that

$$\frac{2n}{\Delta_{\ell-q-1}^{(s-1)d_i}} \geq H_{\ell-q}(j-1) \geq 1 \Rightarrow (2n)^\delta \geq \Delta_{\ell-q-1}^{\delta(s-1)d_i} \geq \Delta_{\ell-q-1}^{d_i+3}.$$

- **Global memory:** We consider each phase $\ell = O(\log \log \Delta)$ separately, incurring only a $\tilde{O}(1)$ multiplicative factor overhead. The bound follows from the same calculations as in [35] by showing that the summation of the local space used by each node in phase ℓ , that is, $P_\ell^0(j) \cdot \Delta_{\ell-1}^{d_i+3} + \sum_{q=1}^{\ell-1} P_\ell^q(j) \cdot \Delta_{\ell-q-1}^{d_i+3}$, is dominated by a geometric series. \square

4.3 Near-Linear Global Space Algorithms

When the maximum degree of the graph under consideration is poly(λ) we switch to another algorithm depending on the desired global space bound. This section concerns near-linear total space, i.e., $\tilde{O}(n+m)$, and is devoted to the following theorem.

Theorem 4.12 *There is a fully-scalable MPC algorithm that given a graph with arboricity λ computes a maximal independent set and a maximal matching deterministically with round complexity on the order of*

$$\min \left\{ \log n, \log \lambda \cdot \log \log n \right\},$$

and $O(\lambda^{1+\varepsilon} + \log \log n)$ for maximal matching and $O(\lambda^{2+\varepsilon} + \log \log n)$ for maximal independent set, for arbitrary constant $\varepsilon > 0$. All the algorithms use $O(n^\alpha)$ local space and $\tilde{O}(m+n)$ global space.

4.3.1 High-Arboricity Case

When the input graph G has arboricity $\lambda = n^{\Omega(1)}$, any $O(\log \lambda)$ -round algorithm can afford to spend $O(\log n)$ rounds for computing a solution. Therefore, classic $O(\log n)$ -round algorithms such as Luby's, which rely only on pairwise independence, are well-suited to obtain $O(\log \lambda)$ -round deterministic algorithms for this class of graphs. The main challenge in converting these to fully-scalable MPC algorithms is that a machine may not be able to store the whole neighborhood of a node in its memory.

To overcome such limitation, Czumaj et al. [25] devise a constant-round sparsification procedure that reduces the maximum degree to $O(n^\delta)$, for arbitrary constant $\delta > 0$, while preserving a (partial) solution that decreases the problem size by a constant factor. Their end-result is the conversion of each step of Luby's MIS and MM *randomized* algorithms into $O(1)$ *deterministic* algorithms to obtain $O(\log n)$ -round fully scalable *deterministic* algorithms.

However, their derandomization need to have two-hop neighborhoods stored on a single machine in order to find a good matching or independent set, thus requiring $O(n^{1+2\delta})$ total space. Our approach is to define a *pessimistic estimator* that produces asymptotically the same results and can be computed using only knowledge of the one-hop neighborhood, i.e., $O(\min\{n^\delta, \deg(v)\})$ local space. We use a lower sampling probability so that nodes neighboring a sampled node can optimistically assume that will be removed. Since sampled nodes have a constant probability of being part of the solution, we can then bound the effect of sampled nodes that are not part of \mathcal{I} (or \mathcal{M}).

Theorem 4.13 *There is a low-memory MPC algorithm that computes MIS and MM deterministically with round complexity $O(\log n)$ and $O(m + n)$ global space.*

Maximal Matching

Our starting point is the maximal matching sparsification procedure of [25, Section 3], which is summarized in the following lemma.

Lemma 4.14 ([25]) *Given in input a graph $G = (V, E)$, there is a strongly sub-linear MPC constant-round sparsification procedure that returns a subset of nodes $B \subseteq V$ and a subset of edges $E' \subseteq E$ with the following properties:*

1. *For every $v \in V$ it holds that $\deg_{E'}(v) = O(n^\delta)$, for any $\delta, 0 < \delta < 1$.*
2. *Every node $v \in B$ either satisfies $\sum_{\{u,v\} \in E'} \frac{1}{\deg_{E'}(\{u,v\})} \geq \frac{1}{27}$, or is incident to an edge $\{u, v\} \in E'$ whose degree in E' is 0.*
3. *For the subset B of V it holds that $\sum_{v \in B} \deg(v) \geq \frac{\delta|E|}{8}$.*

Moreover, the algorithm is deterministic and uses $O(n + m)$ total space.

Lemma 4.14 returns a subset E' of edges and a subset B of vertices that our algorithm uses to construct a matching $\mathcal{M} \subseteq E'$ that is incident to $\Omega(\delta|E|)$ edges while utilizing $O(n + m)$ total space.

We define for each $v \in B$ a subset of *relevant* edges $S(v) \subseteq E'(v)$ such that $\sum_{e \in S(v)} \frac{1}{\deg_{E'}(\{u,v\})} \in [\frac{1}{27}, 1]$, by Property 2. Let \mathcal{H} be a family of pairwise independent hash functions and let $h \in \mathcal{H}$ map each edge $e = \{u, v\}$ in E' to a value $z_e \in [n^3]$. We say that e is *sampled* and joins the set of sampled edges S_h iff $z_e < \frac{n^3}{3 \deg_{E'}(e)}$. Let $S_h(v) \stackrel{\text{def}}{=} S(v) \cap S_h$ denote the set of edges in $S(v)$ that are sampled. Then, an edge $e = \{u, v\} \in S_h(v)$ is a *matched from u 's side*, for a hash function h , iff $\{e\} = \{e' \in S_h \mid u \in e'\}$, i.e., e is the only sampled edge in u 's neighborhood. Further, a node v is a *matching candidate* under h if the following two conditions hold: (i) $|S_h(v)| = 1$ and (ii) $e = \{u, v\} \in S_h(v)$ is matched from u 's side.

Lemma 4.15 *For an arbitrary hash function h , we can find a matching \mathcal{M}_h such that every matching candidate vertex under h is part of \mathcal{M}_h .*

Proof Consider the subgraph induced by edges in S_h . Every node $v \in B$ with exactly one sampled edge e^* in its neighborhood marks edge e^* . Using this information, every node v can deduce whether it is a matching candidate. Then, \mathcal{M}_h consists of marked edges incident to matching candidates. \square

Our goal is to define a pessimistic estimator $P(h)$ that lower bounds the number of edges removed, i.e., $\mathcal{M}_h \cup N(\mathcal{M}_h)$ and that its expected value is $\Omega(\delta|E|)$ over a random choice of $h \in \mathcal{H}$. Moreover, its computation should require only knowledge of the 1-hop neighborhood.

Each node performs the following operations to compute $P(h)$. Let each node $v \in B$ add $\deg(v)$ to $P(h)$ iff $|S_h(v)| = 1$, i.e., v respects the first condition to become a matching candidate. Next, for each vertex $v \in B$ and for each of its sampled edges $e = \{u, v\} \in S_h(v)$, node u subtracts $\deg(v)$ to $P(h)$ iff edge e is not matched from u 's side, i.e., there is $e' = \{u, w\} \in S_h$ with $w \neq v$ and thus v does not respect the second condition. Observe that only matching candidates positively contribute to $P(h)$. Concretely, we define

$$P(h) = \sum_{v \in B: |S_h(v)|=1} \deg(v) - \sum_{v \in B} \deg(v) \cdot \left(\sum_{\{u,v\} \in S_h(v)} \mathbb{1}_{\{\exists e' \in S_h \sim_e e' \ni u\}} \right).$$

Lemma 4.16 *For any node $v \in B$ the probability that v is a matching candidate, for a random hash function $h \in \mathcal{H}$, is at least $\frac{1}{250}$.*

Proof If v has an adjacent edge e with degree 0, then $S(v) = \{e\}$ and e will join \mathcal{M}_h with probability 1. Otherwise, for any edge $e = \{u, v\} \in E'$ and any

$h \in \mathcal{H}$ it holds that the probability of the event A_e that $e \in S_h$ is

$$\frac{1}{3 \deg_{E'}(e)} - \frac{1}{n^3} \leq \Pr[A_e] \leq \frac{1}{3 \deg_{E'}(e)}.$$

Let B_e be the event $\{\exists e' \in E' \sim_e A_{e'}\}$ that e is not matched from u 's side. Conditioned on A_e and using pairwise independence of z_e and $z_{e'}$, we obtain

$$\Pr[B_e | A_e] \leq \sum_{e' \in E' \sim_e} \Pr[A_{e'}] \leq \deg_{E'}(e) \cdot \frac{1}{3 \deg_{E'}(e)} = \frac{1}{3}.$$

Then, by applying the inclusion-exclusion principle, we get

$$\begin{aligned} \Pr[|S_h(v)| = 1] &\geq \sum_{e \in S(v)} \Pr[A_e] - \sum_{e, e' \in S(v), e \neq e'} \Pr[A_e, A_{e'}] \\ &\geq \sum_{e \in S(v)} \Pr[A_e] - \sum_{e \in S(v)} \sum_{e' \in S(v)} \Pr[A_e] \cdot \Pr[A_{e'}] \\ &\geq \sum_{e \in S(v)} \Pr[A_e] \left(1 - \sum_{e' \in S(v)} \Pr[A_{e'}]\right). \end{aligned}$$

where the second inequality follows from pairwise independence. Observe that $\Pr[A_e \wedge B_e] = \Pr[A_e] \Pr[B_e | A_e] \leq \frac{1}{3} \Pr[A_e]$ by the calculations above. Hence, the probability that v is a matching candidate is at least

$$\begin{aligned} \Pr[|S_h(v)| = 1] - \sum_{e \in S(v)} \Pr[A_e \wedge B_e] &= \sum_{e \in S(v)} \Pr[A_e] \left(\frac{2}{3} - \sum_{e' \in S(v)} \Pr[A_{e'}]\right) \\ &\geq \sum_{e \in S(v)} \Pr[A_e] \left(\frac{2}{3} - \frac{1}{3} \sum_{e' \in S(v)} \frac{1}{\deg_{E'}(e')}\right) \geq \frac{1}{3} \cdot \sum_{e \in S(v)} \Pr[A_e] \\ &\geq \frac{1}{3} \cdot \sum_{e \in S(v)} \left(\frac{1}{3 \deg_{E'}(e)} - \frac{1}{n^3}\right) \geq \frac{1}{9} \cdot \sum_{e \in S(v)} \frac{1}{\deg_{E'}(e)} - \frac{1}{3n^2} \geq \frac{1}{243} - \frac{1}{3n^2}, \end{aligned}$$

where the last inequality follows from Property 2. The claim now follows from $\frac{1}{243} - \frac{1}{3n^2} \geq \frac{1}{250}$ for n large enough. \square

Lemma 4.16 and Property 3 give the following lower bound on $\mathbb{E}[P(h)]$.

$$\begin{aligned} \mathbb{E}[P(h)] &= \sum_{v \in B} \deg(v) \cdot \Pr[|S_h(v)| = 1] - \sum_{v \in B} \deg(v) \cdot \left(\sum_{e \in S(v)} \Pr[A_e \wedge B_e]\right) \\ &= \sum_{v \in B} \deg(v) \cdot \left(\Pr[|S_h(v)| = 1] - \sum_{e \in S(v)} \Pr[A_e \wedge B_e]\right) \\ &\geq \frac{1}{250} \sum_{v \in B} \deg(v) \geq \frac{\delta|E|}{2000}. \end{aligned}$$

Summarized, a maximal matching can be found deterministically in the MPC model with strongly sublinear space per machine and $O(n + m)$ total space. Our algorithm applies the sparsification procedure (Lemma 4.14) followed by the derandomization process described above. By the method of conditional expectation, using objective function $P(h)$, we select a hash function h^* such that $P(h^*) \geq \frac{\delta}{2000} |E|$. Then, by Lemma 4.15, the matching \mathcal{M}_{h^*} matches all matching candidates and decreases the number of edges by $\frac{\delta}{4000} |E|$. Repeating it $O(\log n)$ times ensures that the returned matching is maximal.

Maximal Independent Set

In this section, we modify the approach of Section 4.3.1 to find a maximal independent set with analogous properties. Our starting point is again the MIS sparsification of [25, Section 4], which has the following properties.

Lemma 4.17 ([25]) *Given in input a graph $G = (V, E)$, there is a strongly sub-linear MPC constant-round sparsification procedure that returns subsets of nodes $B, Q \subseteq V$ and a subset of edges $E' \subseteq E$ such that:*

1. For every $v \in B \cup Q$ it holds that $\deg_{E'}(v) = O(n^\delta)$ and that $\deg_Q(v) = O(n^\delta)$, for any $\delta, 0 < \delta < 1$.
2. Every node $v \in B$ either satisfies $\sum_{u \in N_{E'}(v)} \frac{1}{\deg_Q(u)} \geq \frac{\delta}{10}$, or it has a neighbor $u \in Q$ whose degree in Q is 0.
3. For the subset B of V it holds that $\sum_{v \in B} \deg(v) \geq \frac{\delta |E|}{8}$.

Moreover, the algorithm is deterministic and uses $O(n + m)$ total space.

Lemma 4.17 returns a subset E' of edges and subsets B, Q of vertices that are used to find an independent set $\mathcal{I} \subseteq Q$ such that $\mathcal{I} \cup N(\mathcal{I})$ is incident to a linear number of edges. Since by Property 1 every node has degree $O(\min\{n^\delta, \deg(v)\})$, $O(n^\delta)$ local space and $O(n + m)$ total space suffice.

For each $v \in B$, denote by $S(v) \subseteq N_{E'}(v)$, with $\sum_{u \in S(v)} \frac{1}{\deg_Q(u)} \in [\delta/10, 1]$, a subset of *relevant* nodes. Let \mathcal{H} be a family of pairwise independent hash functions and let $h \in \mathcal{H}$ map each node v in Q to a value $z_v \in [n^3]$. We say that v is *sampled* and joins the set of sampled nodes S_h iff $z_v < \frac{n^3}{3 \deg_Q(v)}$. Let

$S_h(u) \stackrel{\text{def}}{=} S(u) \cap S_h$ denote the set of nodes in $S(u)$ that are sampled for each $u \in B$. Then, a node $v \in Q$ is a *independent*, for a random hash function h , iff $S_h \cap N_Q(v) = \emptyset$. Further, a node $v \in B$ is *ruled* under h if the following two conditions hold: (i) $|S_h(v)| = 1$ and (ii) $u \in S_h(v)$ is independent.

Lemma 4.18 *For an arbitrary hash function h , we can find an independent set \mathcal{I}_h such that every ruled vertex under h is in $\mathcal{I}_h \cup N(\mathcal{I}_h)$.*

Proof Build the independent set \mathcal{I}_h by adding every *independent* node under h to it. By definition, \mathcal{I}_h is an independent set. Since each *ruled* vertex has an *independent* neighbor the lemma follows. \square

Our goal is to define a pessimistic estimator denoted by $P(h)$ that lower bounds the number of edges removed, i.e., $E(\mathcal{I}_h \cup N(\mathcal{I}_h))$, and its expected value is $\Omega(\delta^2|E|)$ over a random choice of $h \in \mathcal{H}$. The computation of $P(h)$ should require only knowledge of the 1-hop neighborhood.

Let us think of $P(h)$ as a counter and let each node $v \in B$ add $\deg(v)$ to $P(h)$ iff $|S_h(v)| = 1$, i.e., v respects the first condition for becoming ruled. Accordingly, for each vertex $v \in B$ and for each of its sampled neighbors $u \in S_h(v)$, node u subtracts $\deg(v)$ to $P(h)$ iff u is not independent, that is, v does not respect the second condition for becoming ruled. Observe that only ruled vertices positively contribute to $P(h)$. Concretely, we define

$$P(h) = \sum_{v \in B: |S_h(v)|=1} \deg(v) - \sum_{v \in B} \deg(v) \cdot \left(\sum_{u \in S_h(v)} \mathbb{1}\{\exists u' \in N_Q(u) u' \in S_h\} \right).$$

Lemma 4.19 *For any node $v \in B$ the probability that v is ruled, for a random hash function $h \in \mathcal{H}$, is at least $\frac{\delta}{100}$.*

Proof If v has an adjacent neighbor u with degree 0 in Q , then $S(v) = \{u\}$ and u will join \mathcal{I}_h with probability 1. Otherwise, for any node $u \in Q$ and any $h \in \mathcal{H}$ it holds that the probability of the event A_u that $u \in S_h$ is

$$\frac{1}{3 \deg_Q(u)} - \frac{1}{n^3} \leq \Pr[A_u] \leq \frac{1}{3 \deg_Q(u)}.$$

Let B_u denote the event $\{\exists u' \in N_Q(u) u' \in S_h\}$ that u is not independent. Conditioned on A_u and by independence of $z_u, z_{u'}$, the probability of B_u is

$$\Pr[B_u | A_u] \leq \sum_{u' \in N_Q(u)} \Pr[A_{u'}] \leq \deg_Q(u) \cdot \frac{1}{3 \deg_Q(u)} = \frac{1}{3}.$$

Then, by applying the inclusion-exclusion principle and observing that $\Pr[A_u, A_{u'}] = \Pr[A_u] \cdot \Pr[A_{u'}]$ by pairwise independence, we get

$$\begin{aligned} \Pr[|S_h(v)| \geq 1] &\geq \sum_{u \in S(v)} \Pr[A_u] - \sum_{\substack{u, u' \in S(v), \\ u \neq u'}} \Pr[A_u] \cdot \Pr[A_{u'}] \\ &\geq \sum_{u \in S(v)} \Pr[A_u] \left(1 - \sum_{u' \in S(v)} \Pr[A_{u'}] \right). \end{aligned}$$

Observe that $\Pr[A_u \wedge B_u] = \Pr[A_u] \cdot \Pr[B_u \mid u \in S_h] \leq \frac{1}{3} \Pr[A_u]$ by the calculations above. Hence, the probability that v is ruled is at least

$$\begin{aligned} & \Pr[|S_h(v)| = 1] - \sum_{u \in S(v)} \Pr[A_u \wedge B_u] \geq \sum_{u \in S(v)} \Pr[A_u] \left(\frac{2}{3} - \sum_{u' \in S(v)} \Pr[A_{u'}] \right) \\ &= \sum_{u \in S(v)} \Pr[A_u] \left(\frac{2}{3} - \frac{1}{3} \sum_{u' \in S(v)} \frac{1}{\deg_Q(u')} \right) \geq \frac{1}{3} \cdot \sum_{u \in S(v)} \Pr[A_u] \\ &\geq \frac{1}{3} \cdot \sum_{u \in S(v)} \left(\frac{1}{3 \deg_Q(u)} - \frac{1}{n^3} \right) \geq \frac{1}{9} \cdot \sum_{u \in S(v)} \frac{1}{\deg_Q(u)} - \frac{1}{3n^2} \geq \frac{\delta}{90} - \frac{1}{3n^2}, \end{aligned}$$

where the last inequality follows from Property 2. The claim now follows from $\frac{\delta}{90} - \frac{1}{3n^2} \geq \frac{\delta}{100}$ for n large enough. \square

Lemma 4.19 and Property 3 yield the following lower bound on $\mathbb{E}[P(h)]$.

$$\begin{aligned} \mathbb{E}[P(h)] &= \sum_{v \in B} \deg(v) \cdot \Pr[|S_h(v)| = 1] - \sum_{v \in B} \deg(v) \cdot \left(\sum_{u \in S(v)} \Pr[A_u \wedge B_u] \right) \\ &= \sum_{v \in B} \deg(v) \cdot \left(\Pr[|S_h(v)| = 1] - \sum_{e \in S(v)} \Pr[A_u \wedge B_u] \right) \\ &\geq \frac{\delta}{100} \sum_{v \in B} \deg(v) \geq \frac{\delta^2 |E|}{800}. \end{aligned}$$

We are now ready to complete the proof of Theorem 4.13. Our algorithm applies the sparsification procedure of Lemma 4.17 followed by the derandomization process described above. By the method of conditional expectation, using objective function $P(h)$, we select a hash function h^* such that $P(h^*) \geq \frac{\delta^2}{800} |E|$. By applying Lemma 4.18, we then find an independent set \mathcal{I}_{h^*} incident to all ruled vertices and to at least $\frac{\delta}{1600} |E|$ edges from the graph. Thus, after $O(\log n)$ iterations the returned independent set is maximal.

4.3.2 Low-Arboricity Case

By the algorithm from Section 4.3.1 of round complexity $O(\log n)$, we can assume that $\lambda = O(n^{\delta/4})$ for an arbitrary constant $\delta \in (0, 1)$. Using this assumption, we show that one can obtain a constant number of subgraphs of maximum degree $O(n^\delta)$, which can be processed one at a time using $O(n^\alpha)$ local space. Then, for each subgraph, we run `ARB-ALG PIPELINED` to reduce the maximum degree to $\text{poly}(\lambda)$ and, following that, a general MIS or MM algorithm depending on the desired runtime. First, we construct a H -partition of degree $d = n^{\delta/3}$ with $O(\log_{d/\lambda} n) = O(1)$ layers. Then, we process one layer at a time depending on the problem we want to solve as follows. The pseudocode summarizing the two procedures is given below.

Maximal Independent Set: We compute an MIS of the subgraph induced by vertices in H_i for $i = 1, \dots, L$ sequentially. This is possible since each vertex has at most d neighbors in the same layer. Each time, before proceeding to the next layer, we remove vertices in higher layers that have a neighbor in the solution of the layer that we just computed.

Maximal Matching: We compute a matching between vertices in H_i and H_1, \dots, H_{i-1} for $i = 2, \dots, L$ such that vertices with degree larger than n^δ , are matched. Once the matching for the i -th layer is computed, before proceeding to the next layer, we remove edges incident to the current matching. After this step, every unmatched node has degree $O(n^\delta)$ in the resulting graph, and we can thus compute an MM on the remaining vertex set.

Algorithm 2 MPC MIS Arboricity-Based Algorithm

$H_1, \dots, H_L \stackrel{\text{def}}{=} H\text{-partition}(G, n^{\delta/3}).$
for $i \leftarrow 1, 2, \dots, L$ **do**
 ARB-ALG PIPELINED(H_i, \mathcal{I}).
 Run MIS algorithm on H_i .
 $G \leftarrow G \setminus (\mathcal{I} \cup N(\mathcal{I})).$

Algorithm 3 MPC MM Arboricity-Based Algorithm

$H_1, \dots, H_L \stackrel{\text{def}}{=} H\text{-partition}(G, n^{\delta/3}).$
for $i \leftarrow 2, \dots, L$ **do**
 MATCH HIGH-DEG($H_i, \bigcup_{j=1}^{i-1} H_j, \mathcal{M}$).
 $G \leftarrow G \setminus \mathcal{M}.$
 ARB-ALG PIPELINED(G, \mathcal{M}).
 Run MM algorithm on $G \setminus \mathcal{M}.$

Matching High-Degree Nodes The procedure MATCH HIGH-DEG reduces the maximum degree of G to d^3 . The algorithm is based on the random degree reduction technique of [11]. The algorithm consists of two phases: a *marking* phase in which a random set of edges incident to nodes in layer i is marked, and a *selection phase* in which a valid subset of edges is selected as part of the matching. The pseudocode of this procedure follows.

Algorithm 4 MATCH HIGH-DEG($H_i, (H_1, \dots, H_{i-1}), \mathcal{M}$)

Every $u \in \bigcup_{j=1}^{i-1} H_j$ randomly marks one $e = \{u, v\}$ with $v \in H_i \cap N(u)$.
 Let P be the set of marked edges.
if $v \in H_i$ and $\deg_P(v) \geq 1$ **then**
 v adds one incident marked edge to \mathcal{M} arbitrarily.

Lemma 4.20 *Every vertex in H_i with degree at least d^3 gets removed or all but at most d^3 of its incident edges are removed.*

Proof Let v be a node in layer i with degree at least d^3 . Since v has at most d neighbors in H_i , there are at least $d^3 - d \gg 2d^2$ in lower layers. Let X_u be a family of k -wise independent random variables with $X_u = 1$

when node u marks the edge $\{u, v\}$ and $X_u = 0$ otherwise, for each $u \in U(v) \stackrel{\text{def}}{=} N(v) \cap \bigcup_{j=1}^{i-1} H_j$. Define $X = \sum_{u \in U(v)} X_u$ and observe that by linearity of expectation $\mathbb{E}[X] \geq 2d$, since X_u takes value 1 with probability at least $1/d$ and $|U(v)| \geq 2d^2$. By applying Lemma 2.11, we obtain

$$\Pr[X = 0] \leq 8 \left(\frac{1}{d}\right)^{k/2} \leq n^{-c}. \quad (k = 7c/\delta)$$

We are now ready to present the MPC derandomization of this algorithm. Let $\mathcal{H} = \{h: V \mapsto [d]\}$ be a family of k -wise independent hash functions specified by a seed of length at most $k \cdot (\log n + \log d) + O(1) = O(\log n)$ by Lemma 2.8. Each function $h \in \mathcal{H}$ defines a matching $\mathcal{M}(h)$ that includes each node $v \in H_i$ with at least one marked edge, i.e., if $h(u) = v$ for $u \in U(v)$.

To implement our derandomization scheme, each node $v \in H_i$ of degree at least d^3 distributes its edges across machines as follows. We group the edges of v incident to $U(v)$ in blocks of size exactly $d^2 = O(n^\delta)$, and remove from consideration the other at most $d^2 - 1$ edges. Each block is allocated on a single machine, which will ensure that such block is marked. A block is *marked* for a hash function h if it has at least one marked edge.

The analysis of the above randomized algorithm together with a union bound over the set of all blocks implies that choosing a hash function h uniformly at random from \mathcal{H} makes all blocks *marked* with probability at least n^{-c+2} . In particular, this guarantees the existence of a hash function h^* that matches every high-degree node $v \in H_i$ for $c \geq 3$. Formally, each machine x , which is responsible for some blocks of edges $B_j(v)$ incident to node v , can compute locally the number of edges $\{u, v\} \in B_j(v)$ such that $h(u) = v$ for each hash function $h \in \mathcal{H}$ and determine whether $B_j(v)$ is marked. By the distributed implementation of the method of conditional expectation from Section 2.4 with objective function $q_x(h) = \mathbb{1}\{B_j(v) \text{ is marked for every } v\}$, we can find a function h^* which makes all blocks marked, in a constant number of rounds. Then, we compute the matching $\mathcal{M}(h^*)$ induced by h^* by letting every high-degree node choose one incident marked edge arbitrarily. \square

Memory and Running Time Trade-Off

A runtime bound of $O(\log \lambda + \log \log n)$ as Theorem 4.23, explained in the next section, while using near-linear global memory would be very desirable. In the following, we take a step in this direction and show how one can improve the total space to near-linear while trading off the running time. Specifically, we prove the $O(\log \lambda \log \log n)$ bound of Theorem 4.12. Whenever $\lambda = o(2^{\frac{\log n}{\log \log n}})$, this algorithm achieves sublogarithmic runtime. As we shall see, the main idea behind it is the application of graph exponentiation in the framework of Lemma 2.6.

For $i = 0, 1, \dots, \log \log_{\Delta} n^{\alpha} + O(1)$, the algorithm performs the following:

1. Run the derandomized version of Luby's algorithm for $2^i \cdot \Theta(\log \Delta)$ LOCAL rounds.
2. Each vertex performs one graph exponentiation step until $\Delta^{2^i} = O(n^{\alpha})$.

Lemma 4.21 *The above algorithm computes an MIS and an MM deterministically of an input graph G with maximum degree $\Delta = \text{poly}(\lambda)$ in $O(\log \lambda \log \log n)$ round using $O(n^{\alpha})$ local space and $\tilde{O}(m + n)$ global space.*

Proof We apply the MPC simulation result of Lemma 2.6 with maximum graph exponentiation radius $k = \Theta(\log \log_{\Delta} n^{\alpha})$, as algorithm \mathcal{A} the MPC derandomization of Luby's algorithms as explained in Theorem 4.13 (cf. [25, Section 5]), and the number of LOCAL rounds to simulate $T = \Theta(\log n)$. In particular, algorithm \mathcal{A} satisfies the condition that a $\Omega(\Delta)$ progress in the problem size is achieved in $t = O(\log \Delta) = O(\log \lambda)$ LOCAL rounds and the choice of $T = \Theta(\log n)$ ensures that the returned set is maximal. This proves that the round complexity is $O(\log \lambda \cdot \log \log_{\lambda} n)$, as claimed. \square

Applications of Arboricity Coloring

In this section, we describe a $O(\log \log n)$ -round algorithm for the family of graphs with $\lambda = \log^{O(1)} \log n$. This improves the previous runtime bound of Lemma 4.21 by a factor $\log \lambda$ and completes the proof of Theorem 4.12.

Lemma 4.22 *For a graph G with arboricity λ , there is a deterministic low-memory MPC algorithm that computes an MIS and an MM of G in time $O(\lambda^{2+\varepsilon} + \log \log n)$ and $O(\lambda^{1+\varepsilon} + \log \log n)$, respectively, for any constant $\varepsilon > 0$, using linear total space. The same algorithm runs in time $O(\log \log n)$ whenever λ is on the order of $\log^{1/2-\delta} \log n$ and $\log^{1-\delta} \log n$, respectively, for some constant $\delta > 0$.*

Proof Recall that by earlier results, we can assume that G has maximum degree $\Delta \leq \text{poly}(\lambda) = O(n^{\alpha})$. First, we compute a degree $d = \lambda^{1+\varepsilon}$ H -partition in $O(\log \log n)$ rounds, as follows. This approach is similar to that of [15, Lemma 4.2]. Let k be the largest integer such that $\Delta^{2^k} \leq n^{\alpha}$. In rounds $i = 0, \dots, k + O(1)$, the algorithm performs the following steps:

1. Nodes simulate $s \cdot 2^i$ rounds of the H -partition algorithm for $s = O(1)$.
2. Nodes in layers $(s \cdot (2^i - 1), s \cdot (2^{i+1} - 1)]$ remove themselves from G .
3. If $i < k$, then each node performs one step of graph exponentiation.

The correctness follows from the H -partition algorithm. For the runtime, we apply Lemma 2.6 with maximum graph exponentiation radius k , as algorithm \mathcal{A} the H -partition algorithm, and the number of LOCAL rounds to simulate $T = \Theta(\log_{\lambda} n)$. In particular, algorithm \mathcal{A} satisfies the condition that a $\Omega(\Delta)$

progress in the problem size is achieved in $t = O(1)$ LOCAL rounds by the properties of the H -partition. This gives a round complexity of $O(\log \log_\lambda n)$.

Obtained the H -partition, the MIS result follows from computing a $O(\lambda^{2+\epsilon})$ coloring in $O(\log^* n)$ rounds applying the algorithm `ARB-LINIAL'S COLORING` by Barenboim and Elkin [6]. Then, an MIS is found by adding the vertices of each color that have no neighbor in the independent set so far computed.

For the MM case, the algorithm by Panconesi and Rizzi [65] takes in input the d forest-decomposition produced by the H -partition and returns a maximal matching in $O(d)$ rounds. A sketch of their algorithm follows. First, compute in parallel a 3-coloring of each forest in $O(\log^* n)$ rounds. In a sequential fashion, for each forest $i \in [d]$ and color $j \in [3]$, node v colored by j sends a matching proposal over its outgoing edge labeled by i (if any). Each vertex that has received proposals accepts one arbitrarily. Finally, matched vertices inform their neighbors that they became matched and delete themselves from the graph. Since the set of vertices with color j in forest i forms an independent set, there are no conflicts. Moreover, the matching is maximal as each edge is either added to the matching or discarded.

For the special case of $\lambda = O(\log^{1-\delta} \log n)$ for MM, with constant $\delta \in (0, 1)$, the $O(\log \log n)$ bound is achieved by setting $\epsilon = \delta/(1 - \delta)$. A similar calculation applies to the case $\lambda = O(\log^{1/2-\delta} \log n)$ for MIS. \square

4.4 Superlinear Global Space Algorithm

In the setting where $O(n^{1+\epsilon})$ global memory is allowed, one obtains faster MIS and MM deterministic algorithms running in $O(\log \lambda + \log \log n)$ deterministic time by invoking the algorithm due to Czumaj, Davis, and Parter [25].

Theorem 4.23 *There is a fully-scalable MPC algorithm that given a graph with arboricity λ computes a maximal independent set and a maximal matching deterministically with round complexity $O(\log \lambda + \log \log n)$ using $O(n^\alpha)$ local space and $\tilde{O}(n^{1+\epsilon} + m)$ global space, for an arbitrary positive constant ϵ .*

Proof Whenever $\Delta = \text{poly}(\lambda)$, we invoke the deterministic MIS and MM algorithm of [25] that runs in $O(\log \Delta + \log \log n)$ rounds on an input graph with maximum degree Δ and uses $O(n^{1+\epsilon} + m)$ global space. Otherwise, we first reduce the maximum degree to $\text{poly}(\lambda)$ as follows. We apply the preprocessing step explained in Section 4.3.2 that decreases the maximum degree to $O(n^\delta)$, for an arbitrarily small positive constant δ . Then, the algorithm of Theorem 4.4 can be applied to find a partial solution that reduces the maximum degree to $\text{poly}(\lambda)$ in $O(\log \log n)$ deterministic time. \square

Deterministic Arb-Coloring via All-to-All Communication

In this chapter, we study the complexity of deterministic arboricity-dependent coloring in the context of all-to-all communication models, that is, when the underlying communication network is a complete graph. We consider two well-studied models of distributed and parallel computation: CONGESTED-CLIQUE and linear-memory MPC, where each machine has $O(n)$ memory. In these models, Czumaj, Davis, and Parter [23] give a deterministic algorithm that solves $(\Delta + 1)$ coloring, and its list-coloring variant, in a constant number of rounds. Henceforth, we will refer to this algorithm as CDP's algorithm. Interestingly, the algorithm of CDP implies improved upper bounds for arboricity-dependent coloring as well, as we prove in Section 5.1. Specifically, it provides a $O(\lambda)$ coloring of λ -arboricity graphs in $O(\log \lambda)$ rounds and a coloring with $O(\lambda^{1+\epsilon})$ colors in just $O(1)$ time.

In earlier work, Barenboim and Khazanov [10] gave the first non-trivial upper bound of $O(\lambda^\epsilon)$, for any constant $\epsilon > 0$, to $O(\lambda)$ -color a graph. Turning our attention to the randomized setting, Ghaffari and Sayyadi [39] show that a $O(\lambda)$ coloring can be computed in a constant number of rounds in CONGESTED-CLIQUE, with high probability. This settles the randomized complexity of the problem. Inspired by their work and motivated by our quest for deterministic algorithms, we take a step forward in addressing the following natural question about arboricity colorings.

“For what values of λ an $O(\lambda)$ coloring can be computed in a constant number of rounds deterministically in all-to-all models with $O(n)$ local memory?”

We show that a $O(\lambda)$ coloring can be obtained deterministically in $O(1)$ rounds when the available local space per machine is $O(n)$ and the total memory is on the order of $n \cdot \text{poly}(\lambda)$. Whenever $\lambda = O(n^\delta)$, for a suitably small constant δ , our algorithm uses $O(n^2)$ total space, and can thus be implemented in the CONGESTED-CLIQUE model as well.

To achieve our result, we observe that a key step of the constant-round randomized algorithm of [39] is a binning procedure that partitions the vertices of an λ -arboricity graph in k subsets, each of which induces a subgraph of arboricity at most $O(\lambda/k)$, for any $\lambda = \Omega(\log n)$. The main feature of this procedure is that the product of its parameters, i.e., the number of bins and the arboricity of each part, is linear, that is, $O(\lambda/k) \cdot k = O(\lambda)$. In this way, a $O(\lambda/k)$ coloring of each part can then be computed separately and independently. We combine this partitioning step with the powerful derandomization framework of CDP [23]. This is done by employing several technical ingredients including, most notably, limited independence, pessimistic estimators, the method of brute-forcing (as opposed to the method of conditional expectation), and randomness reduction via coloring. As we shall see, the method of brute forcing all possible random seeds incurs a $\text{poly}(\lambda)$ overhead factor in the global space, which appears to be inherent to the current approach. However, our algorithm is surprisingly simple. After *one single step* of our graph partitioning algorithm, each of the k parts induces a subgraph of linear size and arboricity $O(\lambda/k)$, and thus it can be solved offline. Moreover, bad-behaving nodes, which were excluded by the partitioning algorithm due to their high degree, will induce a subgraph of size $O(n)$ that can be colored locally using a fresh color palette of size 2λ .

5.1 Connection with $(\Delta + 1)$ List Coloring

Corollary 5.1 *Given a graph G with arboricity λ and a parameter d satisfying $2\lambda < d \leq \lambda^{1+\varepsilon}$, for an arbitrary $\varepsilon > 0$, a legal $O(d)$ vertex coloring of G can be computed deterministically in $O(\log_{d/\lambda} \lambda)$ rounds of CONGESTED-CLIQUE and linear-memory MPC with optimal global memory.*

Proof The algorithm consists of three stages. The first stage computes the first $L = O(\log_{d/\lambda} \lambda)$ layers of the H -partition of G with degree d , and removes nodes in these L layers from the graph. The number of remaining nodes whose index layer is higher than L is at most $(\frac{\lambda}{d})^L n$ by the definition of the H -partition. Since the arboricity of the graph cannot increase, at most $\lambda \cdot (\frac{\lambda}{d})^L n = O(n)$ edges remain. Thus, the remaining graph can be gathered in a single machine and colored with $d + 1$ colors locally.

The second stage of the algorithm colors vertices in layers $L, \dots, 2$ in a sequential fashion using d additional colors. Consider an arbitrary layer $\ell \in [2, L]$. Each node has a palette of $2d + 1$ colors and at most d neighbors already colored. Let each node discard colors blocked by its neighbors and observe that layer ℓ induces an instance of $(\Delta + 1)$ -list coloring solvable in $O(1)$ rounds by CDP's algorithm. The global memory required to store the color palettes is $n_\ell \cdot O(d)$, where n_ℓ is the number of nodes in layer ℓ . Since $n_\ell \leq \frac{\lambda n}{d}$ for $\ell \geq 2$, the algorithm requires $\frac{\lambda n}{d} \cdot O(d) = O(m)$ total space.

In the third stage of the algorithm, we color nodes in the first layer using the constant-round $(\Delta + 1)$ -coloring algorithm of CDP and a fresh palette of $d + 1$ colors, so that the global memory remains linear. The final coloring is a legal vertex coloring with $3d + 2 = O(d)$ colors, as desired. \square

5.2 Constant-Round $O(\lambda)$ Coloring

In this section, we prove the following theorem, which is our main result.

Theorem 5.2 *Given a graph G with arboricity λ , a legal $O(\lambda)$ vertex coloring of G can be computed deterministically in $O(1)$ rounds of CONGESTED-CLIQUE and linear-memory MPC using $n \cdot \text{poly}(\lambda)$ global memory.*

Algorithm Outline Our algorithm consists of four steps.

- A first preparation step decomposes the graph into a constant number of subgraphs ensuring that each has maximum degree at most $\text{poly}(\lambda)$.
- The second step computes a $\text{poly}(\lambda)$ coloring of each subgraph by applying CDP's algorithm. This step is crucial to reduce the randomness required and will be discussed last for ease of exposition.
- In the third and main step, we apply a careful derandomization of our graph partitioning algorithm. Concretely, vertices are randomly partitioned in many subgraphs B_1, \dots, B_ℓ of maximum *out-degree* $O(\frac{\lambda}{\ell})$ and size $O(\frac{n}{\ell})$. We show that such a partition exists by using the concept of pessimistic estimators. Then, a good partition is found by defining the global quality of a partition as a function of quantities computable by individual machines. However, doing so requires two steps. First, we select among all possible partitions those whose parts have *all* linear size. Further, for each partition, we locally check how many *bad nodes* there are in each of its parts and sum these numbers up to find a good partition.
- Finally, once a good partition is found, each of its ℓ subgraphs can be $O(\frac{\lambda}{\ell})$ -colored locally while the subgraph induced by bad nodes is colored with 2λ colors (locally as well).

Preprocessing Step: The algorithm computes the first $L = O(1)$ layers of the H -partition of G with degree $\Delta \stackrel{\text{def}}{=} \lambda^{1+\delta}$, for $0 < \delta \leq \frac{2\epsilon}{1-2\epsilon}$. The number of edges induced by unlayered nodes is at most $O(n)$ by the arboricity properties. Thus, we can gather and color unlayered vertices with 2λ colors onto a single machine. In the rest of the algorithm, we discuss how to color the subgraph H_j induced by the j -th layer, for $j \in [L]$, with a fresh palette of $O(\lambda)$ colors. Since $L = O(1)$, the bound on the overall number of colors follows.

The Graph Partitioning Algorithm: The graph partitioning is parameterized by $\ell = \lambda^{0.6}$. The partition $V = B_1 \cup \dots \cup B_\ell$ is obtained placing each node v in a part selected according to a k -wise independent random choice. Denote $\deg_B^{\text{out}}(v)$ as the outdegree of vertex v in the subgraph induced by the vertices $B \subseteq V$. We require that the output of the partitioning algorithm satisfies the following properties. For the sake of the analysis, let us imagine that we are given an orientation of the edges of G with outdegree 2λ .

i) Size of Each Part: Bin i in $[\ell]$ is *good* if $|B_i| = O(\frac{n}{\ell})$. Observe that $\mathbb{E}[|B_i|] = \frac{n}{\ell} = \Omega(n^{0.4})$. Consequently, the probability that $|B_i|$ is at most $2 \cdot \mathbb{E}[|B_i|]$ is upper bounded by n^{-3} , for $k \geq 16$, by applying Lemma 2.11 with X_1, \dots, X_n as indicator random variables for the events that the i -th vertex is placed in B_i . Moreover, by the union bound over ℓ bins, with probability at least $1 - n^{-2}$, all bins contain fewer than $\frac{2n}{\ell}$ nodes.

ii) Outdegree of Each Vertex: A node v in bin i is *good* if $\deg_{B_i}^{\text{out}}(v) = O(\frac{\lambda}{\ell})$. It is required that all $v \in V$ but at most n/λ^3 nodes are good in expectation. We apply Lemma 2.11 with $X_1, \dots, X_{\deg^{\text{out}}(v)}$ as indicator random variables for the events that each outneighbor of v is placed in the same bin as v . These variables are $(k-1)$ -wise independent and each has expectation ℓ^{-1} . Accordingly, the expected outdegree $\mathbb{E}[X]$ of v is at most $\frac{2\lambda}{\ell} = \Omega(\lambda^{0.4})$. Therefore, its outdegree is larger than $\frac{4\lambda}{\ell}$ with probability less than λ^{-3} , for $k \geq 17$, and we can conclude the required claim.

Finding a Good Partition: Let \mathcal{H} be a k -wise independent family of hash functions $h: [n] \mapsto [\ell]$. Each hash function h induces a partition of the graph into ℓ parts $B_1(h), \dots, B_\ell(h)$. We analyze the likelihood that a randomly chosen hash function h induces any bad bin and many bad nodes by defining the following pessimistic estimator $P(h)$:

$$P(h) = |\{\text{bad nodes under } h\}| + |\{\text{bad bins under } h\}| \cdot n.$$

Combining the bounds from the previous paragraph gives:

$$\mathbb{E}[P(h)] = \mathbb{E}[|\{\text{bad nodes}\}|] + \mathbb{E}[|\{\text{bad bins}\}|] \cdot n \leq \frac{n}{\lambda^3} + \frac{1}{n} \leq \frac{2n}{\lambda^3}.$$

Therefore, by the probabilistic method, there is at least one *good* hash function h^* that gives no bad bins and at most $\frac{2n}{\lambda^3}$ bad nodes. However, $P(h)$ cannot be easily computed without the orientation used for the analysis. Because of that, we relax the above conditions and identify first a subset $\mathcal{H}' \subseteq \mathcal{H}$ of candidate hash functions such that $h^* \in \mathcal{H}'$ and each $h \in \mathcal{H}'$ gives a graph partitioning in which each subgraph has size $O(n)$. We then find a good hash function by analyzing each of its subgraphs separately and concurrently.

i) Select Candidate Functions: Observe that *every* good hash function is in \mathcal{H}' . Indeed, the bound on the size of each part together with the bound on the outdegree of each vertex gives

$$|E(G[B_i(h)])| = \sum_{v \in B_i(h)} \deg_{B_i(h)}(v) \leq \frac{2n}{\ell} \cdot \frac{4\lambda}{\ell} + \frac{2n}{\lambda^3} \cdot \Delta = O(n).$$

To deterministically find \mathcal{H}' , we assign each hash function $h \in \mathcal{H}$ to a machine x_h . Let each node compute its degree under h and send it to x_h . The machine x_h is then able to determine whether each subgraph induced by h has size $O(n)$ by summing up the degrees of vertices in each particular bin. This process requires a total space of $O(\lambda n |\mathcal{H}|)$.

ii) Find a Good Partitioning: For every $h \in \mathcal{H}'$ and $i \in [\ell]$, we compute a $\frac{10\lambda}{\ell}$ -coloring of the subgraph $G[B_i(h)]$ and sum up the number of nodes that remain uncolored after this coloring step across all bins for each h . We prove that there is a function $h^* \in \mathcal{H}'$ that leaves only $O(n/\lambda)$ nodes uncolored, which can be colored locally with a new palette of size 2λ . Since $|G[B_i(h)]| = O(n)$, this computation can be made offline for each subgraph on a single machine, using $O(\lambda n |\mathcal{H}|)$ total space.

Consider a subgraph $G[B_i(h)]$ that satisfies the two conditions above. This subgraph may have both good and bad nodes. We run the H -partition algorithm with degree $d = \frac{10\lambda}{\ell} - 1$ until when there is no vertex of degree at most d left. Once this process finishes, we obtain a decomposition \mathcal{H} of $B_i(h)$ that can be $(d+1)$ -colored. Then, nodes in $B'_i(h) \stackrel{\text{def}}{=} B_i(h) \setminus \mathcal{H}$ induce an uncolored subgraph of minimum degree at least $d+1$. This implies that $|E(G[B'_i(h)])| \geq \frac{5\lambda}{\ell} \cdot |B'_i(h)|$. On the other hand, good nodes in $B'_i(h)$ contribute with at most $\frac{4\lambda}{\ell} \cdot |B'_i(h)|$ edges and bad nodes with fewer than $\frac{2n}{\lambda^3} \cdot \Delta$ edges. This yields

$$\frac{5\lambda}{\ell} \cdot |B'_i(h)| \leq \frac{4\lambda}{\ell} \cdot |B'_i(h)| + \frac{2n}{\lambda^3} \cdot \Delta \iff |B'_i(h)| \leq \frac{2\ell\Delta n}{\lambda^4} \leq \frac{n}{\lambda^2},$$

which proves the existence of such good $h^* \in \mathcal{H}'$. We then find a partition whose bins can be $\frac{10\lambda}{\ell}$ -colored (with different palettes) leaving at most $\ell \cdot \frac{n}{\lambda^2}$ nodes uncolored across all parts. Hence, uncolored vertices induce at most $\lambda \cdot \frac{\ell n}{\lambda^2} = O(n)$ edges in the original graph.

Reducing the Total Space: The total space requirement of $O(\lambda n |\mathcal{H}|)$ can be made on the order of $n \cdot \text{poly}(\lambda)$ by reducing the size of \mathcal{H} to be polynomial in the arboricity. To this end, we first note that wlog one can assume that $\lambda \leq n^{\frac{1}{2}-\varepsilon}$, for an arbitrary constant ε , $0 < \varepsilon < \frac{1}{2}$, as otherwise $\text{poly}(n) = \text{poly}(\lambda)$. Second, we observe that k -wise independence is required only among the random variables associated with vertices that share a neighbor.

An assignment of nodes to IDs such that no two nodes with a shared neighbor are assigned the same ID can be equivalently stated as a coloring of the subgraph H_j^2 . In the subgraph H_j^2 , each vertex has degree at most $\Delta^2 = \lambda^{2+2\delta} = O(n)$, by our choice of δ and the assumption that $\lambda \leq n^{\frac{1}{2}-\epsilon}$. Applying CDP's algorithm on H_j^2 provides a $\text{poly}(\lambda)$ coloring in $O(1)$ time, as desired.

We can conclude that our algorithm runs in a constant number of rounds, properly color vertices with $L \cdot (\ell \cdot \frac{10\lambda}{\ell} + 2\lambda) = O(\lambda)$ colors, and requires $O(n \cdot \text{poly}(\lambda))$ global space.

Conclusion

This work continues a recent long line of work on derandomization in the sublinear and linear memory regimes of Massively Parallel Computation. We develop improved deterministic algorithms for several (variants of) fundamental graph problems: *connectivity*, *maximal independent set*, *maximal matching* and *vertex coloring*. Our results are obtained by applying the nowadays classic derandomization framework based on bounded independence together with constant-wise analyses of randomized algorithms, specifically tailored to each problem. Some of the most intriguing improvements and related open questions on these problems are the following:

Connectivity: For graphs with diameter $o(\log \log n)$, it is open whether the $O(\log \log n)$ term in the running time can be improved, even for randomized algorithms.

MIS and MM: In the sublinear regime of MPC with near-linear global memory, we obtained three different runtime bounds, each of which is asymptotically superior for different values of λ . It would be interesting to explore whether there is a deterministic $O(\log \lambda + \log \log n)$ -round algorithm with nearly-optimal global memory. Alternatively, whether the round complexity $O(\log \log n)$ of MIS for bounded arboricity graphs can be extended to graph with arboricity $O(\log^{1-\delta} \log n)$, for constant $\delta > 0$, as in the case of matching. The latter direction may be seen as finding a $O(\lambda^{1+\epsilon})$ coloring in graphs of maximum degree $\text{poly}(\lambda)$, as opposed to the current $O(\lambda^{2+\epsilon})$ -coloring, with constant $\epsilon > 0$.

Coloring: We settled the complexity of deterministic $O(\lambda)$ -coloring λ -arboricity graphs in linear-memory MPC model with global memory $n \cdot \text{poly}(\lambda)$. Improving this bound on the global memory seems to require different ideas. Another interesting direction would be the study of the randomized (and deterministic) round complexity of arboricity-coloring in low-memory MPC.

Bibliography

- [1] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.
- [2] Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2016.
- [3] Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel graph connectivity in log diameter rounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 674–685, 2018.
- [4] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC '19*, page 461–470, New York, NY, USA, 2019. Association for Computing Machinery.
- [5] Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Efficient deterministic distributed coloring with small bandwidth. In *Proceedings of the 39th Symposium on Principles of Distributed Computing, PODC '20*, page 243–252, New York, NY, USA, 2020. Association for Computing Machinery.
- [6] Leonid Barenboim and Michael Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, 22(5):363–379, 2010.
- [7] Leonid Barenboim and Michael Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *J. ACM*, 58(5), 2011.

- [8] Leonid Barenboim and Michael Elkin. Distributed graph coloring: Fundamentals and recent developments. *Synthesis Lectures on Distributed Computing Theory*, 4(1):1–171, 2013.
- [9] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *J. ACM*, 63(3), 6 2016.
- [10] Leonid Barenboim and Victor Khazanov. Distributed symmetry-breaking algorithms for congested cliques. In Fedor V. Fomin and Vladimir V. Podolskii, editors, *Computer Science – Theory and Applications*, pages 41–52, Cham, 2018. Springer International Publishing.
- [11] 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.
- [12] Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Brief announcement: Semi-mapreduce meets congested clique. *CoRR*, abs/1802.10297, 2018.
- [13] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab Mirrokni. Near-optimal massively parallel graph connectivity. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1615–1636, 2019.
- [14] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 276–287, 1994.
- [15] Sebastian Brandt, Manuela Fischer, and Jara Uitto. Matching and MIS for uniformly sparse graphs in the low-memory MPC model. *CoRR*, abs/1807.05374, 2018.
- [16] Sebastian Brandt, Manuela Fischer, and Jara Uitto. Breaking the linear-memory barrier in MPC: Fast MIS on trees with strongly sublinear memory. In Keren Censor-Hillel and Michele Flammini, editors, *Structural Information and Communication Complexity*, pages 124–138, Cham, 2019. Springer International Publishing.
- [17] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

-
- [18] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *Distributed Computing*, 33(3):349–366, Jun 2020.
- [19] 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.
- [20] Moses Charikar, Weiyun Ma, and Li-Yang Tan. Brief announcement: A randomness-efficient massively parallel algorithm for connectivity. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 431–433, New York, NY, USA, 2021. Association for Computing Machinery.
- [21] Benny Chor and Oded Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5(1):96–106, 1989.
- [22] Sam Coy and Artur Czumaj. Deterministic massively parallel connectivity. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 162–175, New York, NY, USA, 2022. Association for Computing Machinery.
- [23] Artur Czumaj, Peter Davies, and Merav Parter. Simple, deterministic, constant-round coloring in the congested clique. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, page 309–318, New York, NY, USA, 2020. Association for Computing Machinery.
- [24] Artur Czumaj, Peter Davies, and Merav Parter. Component stability in low-space massively parallel computation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 481–491, New York, NY, USA, 2021. Association for Computing Machinery.
- [25] Artur Czumaj, Peter Davies, and Merav Parter. Graph sparsification for derandomizing massively parallel computation with low space. *ACM Trans. Algorithms*, 17(2), 5 2021.
- [26] Artur Czumaj, Peter Davies, and Merav Parter. Improved deterministic $(\delta+1)$ coloring in low-space mpc. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 469–479, New York, NY, USA, 2021. Association for Computing Machinery.

- [27] Artur Czumaj, Jakub Łacki, Aleksander Madry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, page 471–484, New York, NY, USA, 2018. Association for Computing Machinery.
- [28] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 1 2008.
- [29] Paul Erdős, Peter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel J. Math*, 51(1-2):79–89, 1985.
- [30] Guy Even, Oded Goldreich, Michael Luby, Noam Nisan, and Boban Veličković. Efficient approximation of product distributions. *Random Structures & Algorithms*, 13(1):1–16, 1998.
- [31] Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Cliff Stein, and Zoya Svitkina. On distributing symmetric streaming computations. *ACM Trans. Algorithms*, 6(4), 9 2010.
- [32] Manuela Fischer. *Local Algorithms for Classic Graph Problems*. Doctoral thesis, ETH Zurich, Zurich, 2021.
- [33] Manuela Fischer, Jeff Giliberti, and Christoph Grunau. Improved Deterministic Connectivity in Massively Parallel Computation, 2022.
- [34] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the 2016 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 270–277, 2016.
- [35] Mohsen Ghaffari, Christoph Grunau, and Ce Jin. Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
- [36] Mohsen Ghaffari and Fabian Kuhn. Derandomizing Distributed Algorithms with Small Messages: Spanners and Dominating Set. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing (DISC 2018)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

-
- [37] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1650–1663, 2019.
- [38] Mohsen Ghaffari and Christiana Lymouri. Simple and Near-Optimal Distributed Coloring for Sparse Graphs. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing (DISC 2017)*, volume 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [39] Mohsen Ghaffari and Ali Sayyadi. Distributed Arboricity-Dependent Graph Coloring via All-to-All Communication. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 142:1–142:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [40] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1636–1653, 2019.
- [41] Michael T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.
- [42] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In Takao Asano, Shin-ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *Algorithms and Computation*, pages 374–383, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [43] James W. Hegeman and Sriram V. Pemmaraju. Lessons from the congested clique applied to mapreduce. *Theoretical Computer Science*, 608:268–281, 2015. Structural Information and Communication Complexity.
- [44] Amos Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986.
- [45] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010.

- [46] Amos Korman, Jean-Sébastien Sereni, and Laurent Viennot. Toward more localized local algorithms: removing assumptions concerning global knowledge. *Distributed Computing*, 26(5):289–308, 2013.
- [47] Fabian Kuhn. Weak graph colorings: Distributed algorithms and applications. In *Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures, SPAA '09*, page 138–144, New York, NY, USA, 2009. Association for Computing Machinery.
- [48] Fabian Kuhn. Weak graph colorings: Distributed algorithms and applications. In *Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures, SPAA '09*, page 138–144, New York, NY, USA, 2009. Association for Computing Machinery.
- [49] Jakub Lacki, Vahab S. Mirrokni, and Michal Włodarczyk. Connected components at scale via local contractions. *CoRR*, abs/1807.10727, 2018.
- [50] Rustam Latypov and Jara Uitto. Deterministic 3-coloring of trees in the sublinear MPC model. *CoRR*, abs/2105.13980, 2021.
- [51] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing, PODC '13*, page 42–50, New York, NY, USA, 2013. Association for Computing Machinery.
- [52] Christoph Lenzen and Roger Wattenhofer. Brief announcement: Exponential speed-up of local algorithms using non-local communication. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC '10*, page 295–296, New York, NY, USA, 2010. Association for Computing Machinery.
- [53] Nathan Linial. Distributive graph algorithms global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 331–335, 1987.
- [54] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.
- [55] Sixue Cliff Liu, Robert E. Tarjan, and Peilin Zhong. *Connected Components on a PRAM in Log Diameter Time*, page 359–369. Association for Computing Machinery, New York, NY, USA, 2020.
- [56] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $o(\log \log n)$ communication rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.

-
- [57] M Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, STOC '85*, page 1–10, New York, NY, USA, 1985. Association for Computing Machinery.
- [58] Michael Luby. Removing randomness in parallel computation without a processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.
- [59] Michael Luby and Avi Wigderson. Pairwise independence and de-randomization. *Foundations and Trends in Theoretical Computer Science*, 1(4):237–301, 2006.
- [60] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.
- [61] Rajeev Motwani, Joseph (Seffi) Naor, and Moni Naor. The probabilistic method yields deterministic parallel algorithms. *Journal of Computer and System Sciences*, 49(3):478–516, 1994. 30th IEEE Conference on Foundations of Computer Science.
- [62] Danupon Nanongkai and Michele Scquizzato. Equivalence classes and conditional hardness in massively parallel computations. *Distributed Computing*, 35(2):165–183, 2022.
- [63] C. St.J. A. Nash-Williams. Edge-Disjoint Spanning Trees of Finite Graphs. *Journal of the London Mathematical Society*, s1-36(1):445–450, 01 1961.
- [64] C. St.J. A. Nash-Williams. Decomposition of Finite Graphs Into Forests. *Journal of the London Mathematical Society*, s1-39(1):12–12, 01 1964.
- [65] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed computing*, 14(2):97–100, 2001.
- [66] Merav Parter. (Delta+1) Coloring in the Congested Clique Model. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 160:1–160:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [67] Merav Parter and Hsin-Hao Su. Randomized (Delta+1)-Coloring in $O(\log^* \Delta)$ Congested Clique Rounds. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing (DISC 2018)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

- [68] 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)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [69] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.
- [70] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits (on lower bounds for modern parallel computation). *J. ACM*, 65(6), nov 2018.
- [71] Mark N. Wegman and J. Lawrence Carter. New classes and applications of hash functions. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 175–182, 1979.
- [72] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10*, page 10, USA, 2010. USENIX Association.