Improved Deterministic $(\Delta+1)$-Coloring in Low-Space MPC

Artur Czumaj
University of Warwick
Coventry, United Kingdom
A.Czumaj@warwick.ac.uk

Peter Davies
IST Austria
Klosterneuburg, Austria
Peter.Davies@ist.ac.at

Merav Parter
Weizmann Institute of Science
Rehovot, Israel
Merav.Parter@weizmann.ac.il

ABSTRACT

We present a *deterministic* $O(\log\log\log n)$-round *low-space* Massively Parallel Computation (MPC) algorithm for the classical problem of $(\Delta+1)$-coloring on n-vertex graphs. In this model, every machine has sublinear local space of size n^ϕ for any arbitrary constant $\phi \in (0,1)$. Our algorithm works under the relaxed setting where each machine is allowed to perform exponential local computations, while respecting the n^ϕ space and bandwidth limitations.

Our key technical contribution is a novel derandomization of the ingenious $(\Delta+1)$-coloring local algorithm by Chang-Li-Pettie (STOC 2018, SIAM J. Comput. 2020). The Chang-Li-Pettie algorithm runs in $T_{local} = poly(\log\log n)$ rounds, which sets the state-of-the-art randomized round complexity for the problem in the local model. Our derandomization employs a combination of tools, notably pseudorandom generators (PRG) and bounded-independence hash functions.

The achieved round complexity of $O(\log\log\log n)$ rounds matches the bound of $\log(T_{local})$, which currently serves an upper bound barrier for all known *randomized* algorithms for locally-checkable problems in this model. Furthermore, no deterministic sublogarithmic low-space MPC algorithms for the $(\Delta+1)$-coloring problem have been known before.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**; • **Mathematics of computing** → **Graph algorithms**; • **Theory of computation** → **Pseudorandomness and derandomization**.

KEYWORDS

Massively Parallel Computation; Coloring; Derandomization

ACM Reference Format:
Artur Czumaj, Peter Davies, and Merav Parter. 2021. Improved Deterministic $(\Delta+1)$-Coloring in Low-Space MPC. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC '21), July 26–30, 2021, Virtual Event, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3465084.3467937

Due to space constraints, some proofs and results are deferred to the full version of this paper, available on the authors' web pages, e.g., at https://www.dcs.warwick.ac.uk/~czumaj/Publications.html.

PODC '21, July 26–30, 2021, Virtual Event, Italy

ACM ISBN 978-1-4503-8548-0/21/07…$15.00
https://doi.org/10.1145/3465084.3467937

ACKNOWLEDGMENTS

This work is partially supported by a Weizmann-UK Making Connections Grant, the Centre for Discrete Mathematics and its Applications (DIMAP), IBM Faculty Award, EPSRC award EP/V01305X/1, European Research Council (ERC) Grant No. 949083, the Minerva foundation with funding from the Federal German Ministry for Education and Research No. 713238, and the European Union's Horizon 2020 programme under the Marie Skłodowska-Curie grant agreement No 754411.

1 INTRODUCTION

In this paper, we study the deterministic complexity of the $(\Delta+1)$ (list) coloring problem in the low-space MPC setting. The *Massively Parallel Computation (MPC)* model, introduced by Karloff, Suri and Vassilvitskii [27], is a nowadays standard theoretical model for parallel algorithms. This model shares many similarities to earlier models of parallel computation (e.g., PRAM), and is also closely related to various distributed models, such as the LOCAL and the CONGESTED CLIQUE models. We focus on the *low-space* MPC regime in which machines have space n^ϕ for some constant $\phi \in (0,1)$, where n is the number of nodes in the graph. This model has attracted a lot of attention recently [3, 5, 11, 13, 14, 17–19, 21, 23, 24, 28], especially in the context of *local* graph problems. Recent works have provided many *randomized* algorithms with sublogarithmic round complexities, for fundamental graph problems such as maximal matching, maximal independent set and $(\Delta+1)$ coloring. However, much less is known on the corresponding deterministic complexity of these problems. In particular, to this date no *sublogarithmic deterministic* algorithm is known, in the low-space MPC model, for any of the canonical symmetry breaking problems.

We study deterministic low-space MPC algorithms for the $(\Delta+1)$ (list) coloring problem, which is arguably among the most fundamental graph problems in parallel and distributed computing with numerous implications. In this problem, we are given an input graph $G=(V,E)$ with maximum degree Δ, for which every vertex has a palette (list) of $\Delta+1$ colors. The goal is to compute a legal vertex coloring, namely, where no two neighbors have the same color, in which each node is assigned a color from its own palette. A sequence of recent exciting breakthrough results have led to a dramatic improvement in the randomized and the deterministic complexity of the problem, in the classical distributed models, as we highlight next.

$(\Delta+1)$ *Coloring in the LOCAL Model:* The LOCAL model has been introduced by Linial [30] with the purpose of developing symmetry breaking methodologies in decentralized networks. In this model, each node in the communication graph is occupied by a processor. The processors communicate in synchronous message passing

rounds where per round each processor can send one message to each of its neighbors in the network. Since its introduction, the model has focused on four canonical problems and their variants: maximal independent sets, $(\Delta+1)$ coloring, and their *edge* analogs, namely, maximal matching and edge coloring. As this model abstracts away congestion issues, it provides the most convenient platform for studying the locality aspects of symmetry breaking.

The study of the $(\Delta+1)$ coloring problem in this model has quite a long history with several important milestones. We first focus on randomized algorithms, and then address the deterministic aspects of the problem. Logarithmic solutions for $(\Delta+1)$ coloring are known since the 80's, e.g., by the classical Luby-MIS algorithm [33]. Barenboim et al. [4] presented the shattering technique, which in the context of coloring, reduces the problem, within $O(\log\Delta)$ randomized rounds, into independent subproblems of $\text{poly}\log n$ size, which can be then solved deterministically. Harris, Schneider and Su [26] presented a new graph decomposition technique that provided the first sublogarithmic solution for the problem. Finally, in a subsequent remarkable breakthrough result, Chang, Li and Pettie (CLP) [10, 12] presented an $O(Det_d(\text{poly}\log n))$-round solution for the problem, where $Det_d(n')$ is the deterministic complexity of the $(\deg+1)$-list coloring problem on an n'-vertex graph. In the latter problem, every vertex v has a palette of only $\deg(v)+1$ colors. Their upper bound *morally* matches the lower bound of $\Omega(Det(\text{poly}\log n))$ rounds shown by Chang, Kopelowitz and Pettie [9] with the only distinction being that $Det(n')$ is the deterministic complexity of the $(\Delta+1)$-list coloring problem. Combining the CLP algorithm with the recent deterministic network decomposition result of Rozhoň and Ghaffari [40], yields an $\text{poly}(\log\log n)$-round algorithm for the $(\Delta+1)$ list coloring problem, which sets the state-of-the-art bound for the problem. The randomized complexity for the related $(\deg+1)$ coloring is $O(\log\Delta)+\text{poly}(\log\log n)$ by [4, 40].

Obtaining *deterministic* coloring solutions of polylogarithmic-time has been one of the most major open problems in the area. This was resolved recently by the groundbreaking network decomposition result of Rozhoň and Ghaffari [40]. Even more recently, Ghaffari and Kuhn [20] improved the time bounds into $O(\log^2\Delta\log n)$ rounds, by using the more direct approach of rounding fractional color assignments. Due to the shattering-based structure of the CLP solution, any deterministic algorithm for the problem immediately improves also the (randomized) CLP bound.

$(\Delta+1)$ *Coloring in the CONGESTED CLIQUE Model:* In the CONGESTED CLIQUE model, introduced by Lotker, Pavlov and Patt-Shamir [31, 32], the network is represented as a fully connected graph, where each node is occupied by a machine which stores the node' edges. The machines communicate in an all-to-all fashion, where in each round, every pair of machines can exchange $O(\log n)$ bits of information. The local memory and computation power are assumed to be *unlimited*. As we will see, this model is considerably more relaxed than the low-space MPC model that we consider in this paper.

There has been a sequence of recent results concerning the randomized complexity of the $(\Delta+1)$ coloring in this model. Parter [38] presented an $O(\log\log\Delta)$-round algorithm for the problem that is based on combining the CLP algorithm with a recursive degree reduction. By employing a palette sparsification technique, Parter and Su [39] improved the complexity into $O(\log^*\Delta)$ rounds. Finally, the randomized complexity of the problem has been settled into $O(1)$ rounds, by Chang et al. [11]. Their algorithm also supports the list variant of the problem, by employing a new randomized partitioning of both the nodes and their *colors*. Recently, Czumaj, Davies and Parter [14] provided a simplified $O(1)$-round deterministic algorithm for the problem. In contrast to prior works, their algorithm is not based on the CLP algorithm. Prior deterministic (logarithmic) bounds were also given by Parter [39] and Bamberger, Kuhn and Maus [3].

$(\Delta+1)$ *Coloring in the Low-Space MPC Model:* In the MPC model, there are M machines and each of them has S words of space. Initially, each machine receives its share of the input. In our case, the input is a collection V of nodes and E of edges and each machine receives approximately $\frac{n+m}{M}$ of them (divided arbitrarily), where $|V|=n$ and $|E|=m$. The computation proceeds in synchronous *rounds* in which each machine processes its local data and performs an arbitrary local computation on its data without communicating with other machines. At the end of each round, machines exchange messages. Each message is sent only to a single machine specified by the machine that is sending the message. All messages sent and received by each machine in each round have to fit into the machine's local space. Hence, their total length is bounded by S. This, in particular, implies that the total communication of the MPC model is bounded by $M\cdot S$ in each round. The messages are processed by recipients in the next round. At the end of the computation, machines collectively output the solution. The data output by each machine has to fit in its local space of S words.

We focus on the *low-space* regime where $S=n^\phi$ for any given constant $\phi\in(0,1)$. A major challenge underlying this setting is that the local space of each machine might be too small to store all the edges incident to a single node. This poses a considerable obstacle for simulating LOCAL algorithms compared to the linear space regime. To overcome this barrier, both randomized and deterministic algorithms in this model are based on graph sparsification techniques.

Chang et al. [11] presented the first randomized algorithm for $(\Delta+1)$ coloring in this model, which as described before[1], employs a random node and palette partitioning which breaks the problem into independent coloring instances. A sparsified variant of the CLP algorithm is then applied on each of the instances, in parallel. This approach when combined with the network decomposition result of [40] provides an $O(\log\log\log n)$ round algorithm, which is currently the state-of-the-art bound for the problem.

The deterministic complexity of the $(\Delta+1)$ coloring in low-space MPC has been studied independently by Bamberger, Kuhn and Maus [3] and by Czumaj, Davies and Parter [14]: [3] presented an $O(\log^2\Delta+\log n)$ round solution for the $(\deg+1)$ list coloring problem; [14] presented an $O(\log\Delta+\log\log n)$-round algorithm for the $(\Delta+1)$ list coloring problem. No sublogarithmic bounds are currently known. To the best of our knowledge, the only sublogarithmic deterministic solutions in this model are given for the ruling set problem[2] by Kothapalli, Pai and Pemmaraju [28].

[1] Their MPC algorithm is similar to their CONGESTED CLIQUE algorithm.
[2] In the β ruling set problem, it is required to compute an independent set S such that every vertex as a β-hop neighbor in S.

On the connection between low-space MPC and LOCAL models. Many of the existing algorithms for local problems in the low-space MPC model are based on LOCAL algorithms for the corresponding problems, e.g., [5, 11, 17, 37]. Specifically, using the graph exponentiation technique, T-round LOCAL algorithms can be simulated within $O(\log T)$ MPC rounds, provided that the T-balls of each node fits the space of the machine. Since in many cases the balls are too large, this technique is combined with other round compression approaches, such as graph sparsification, that are aimed at simulating many LOCAL rounds using few MPC rounds. The upper bound limit of all current approaches is $O(\log T_{\text{LOCAL}})$ MPC rounds, where T_{local} is the LOCAL complexity of the problem.

In a recent inspiring paper, Ghaffari, Kuhn and Uitto [21] established a connection between these two models in the reverse direction (see also a revised framework in [15]). They presented a general technique that allows one to lift *lower bound* results in the LOCAL model into lower bounds in the low-space MPC model, conditioned on the connectivity conjecture. Using this approach they provided conditional lower bounds of $\Omega(\log(T_{\text{LOCAL}}))$ MPC rounds given an $\Omega(T_{\text{LOCAL}})$-round LOCAL lower bound for the corresponding problem. While the original framework from [21] holds only for randomized algorithms, the revised framework in [15] applies also to deterministic algorithms. One caveat of these results is that they hold only for the class of *component-stable* MPC algorithms. Roughly speaking, in this class of algorithms the output of a node depends only on its connected component. We note that the deterministic algorithms presented in this paper are *not* component stable. Our algorithm matches the logarithm of the randomized LOCAL complexity of the $(\Delta+1)$ list coloring problem, which is currently an upper bound limit even for randomized algorithms, for most of the canonical local graph problems.

1.1 Our Results

Our key result is an $O(\log\log\log n)$-time deterministic algorithm for the $(\Delta+1)$ (list) coloring problem in the low-space MPC model. Our algorithm employs exponential (in n^{ϕ}) local computation at each machine, while respecting its (sublinear) space requirement.

Theorem 1. *There exists a deterministic algorithm that, for every n-vertex graph $G=(V,E)$ with maximum degree Δ, computes a $(\Delta+1)$ (list) coloring for G using $O(\log\log\log n)$ rounds, in the low-space MPC model with global space $\widetilde{O}(|E|+n^{1+\phi})$. The algorithm employs exponential (in n^{ϕ}) local computation at each machine while respecting the space and bandwidth limitations.*

Alternatively, we can also state the result as a non-explicit, non-uniform, polynomial-computation deterministic low-space MPC algorithm, if one is allowed to hardcode n^{ϕ} bits of information to each machine (which do not depend on the input graph G). Our result improves over the state-of-the-art deterministic $O(\log\Delta+\log\log n)$-round algorithm for this problem by [14] which works in standard low-space MPC model (i.e., with polynomial local computation). This also matches the *randomized* complexity of the problem as given by Chang et al. [11].

Low-Space MPC with exponential local computation. As noted in previous works, e.g., Andoni et al. [1], the main focus of the low-space MPC model is on the *information-theoretic* question of understanding the round complexity within sublinear space restrictions *(i.e., even with unbounded computation per machine)*. This point of view might provide an explanation for the inconsistency and ambiguity concerning the explicit restrictions on local computation in the low-space MPC model. Many of the prior work explicitly *allow* for an *unlimited local computation*, e.g., [1, 2, 5, 6, 22]. Other works only recommend having a polynomial time computation [21, 23], and some explicitly restrict the local computation to be polynomial [11, 19]. In this work we take the distributed perspective on the MPC model by adopting the standard assumption in which local computation comes for *free*, as assumed in all the classical distributed models, LOCAL, CONGEST and CONGESTED CLIQUE. The main motivation for such an assumption is that it decouples *communication* from *computation*. Our results may indicate that allowing exponential local computation might provide an advantage in the context of distributed and parallel derandomization.

1.2 Key Techniques

Our approach is based on a derandomization of the CLP algorithm using *pseudorandom generator* [41]. As a starting point, we assume that the maximum degree Δ is in the range $\Delta \in [\text{poly}\log n, n^{\phi/c}]$ for a sufficiently large constant c. The upper bound degree assumption is made possible by employing first a recursive graph partitioning, inspired by [14], that uses bounded independence hash functions to break the problem into several independent instances with lower degree of at most $n^{\phi/c}$. This allows us to allocate a machine M_v for every node v in the graph, and store on that machine the $O(1)$-radius ball of v in G. Since most of the CLP procedures are based inspecting the $O(1)$-radius balls, that would be very useful. To handle small (polylogarithmic) degrees, we employ a derandomization of the state-of-the-art $(deg+1)$-coloring algorithm of Barenboim et al. [4]. Assuming that $\Delta = \Omega(\text{poly}\log n)$ provides us a more convenient start point for the CLP derandomization, since in this degree regime, all the local randomized CLP procedures succeed with high probability, of $1-1/n^{c'}$, for any desired constant c'.

The common derandomization approach in all-to-all communication models is based on a combination of obtaining a small search space (i.e., using short random seed) and the method of conditional expectations [8, 34]. The main obstacle in derandomizing the CLP algorithm is that it applies local[3] randomized procedures that seem to require almost full independence. It is thus unclear how derandomize them using the standard bounded independence tools of e.g., hash functions. For example, one of the key procedures for coloring dense regions in the graph (denoted as *almost-cliques*) is based on a *randomized permutation* of the clique' nodes. It is unclear how simulate such a permutation using a small seed and in polynomial time computation. We therefore sacrifice the latter requirement, by allowing exponential local computation.

A *pseudorandom generator* [36, 41] is a function that gets a short random seed and expands it to into a long one which is *indistinguishable* from a random seed of the same length for a given class of algorithms. Informally, a PRG function $\mathcal{G}:\{0,1\}^a \to \{0,1\}^b$,

[3] By local we mean that these procedures are part of the local computation of the nodes.

where $a \ll b$, is said to ϵ-fool a given class of randomized algorithms C that uses b random coins as part of their input, if the following holds for every algorithm $C \in C$: the success probability of C under b pseudorandom coins $\mathcal{G}(X)$, where X is a vector of a random coins, is within $\pm\epsilon$ of the success probability of C when using b truly random coins. Explicit PRG constructions with small seed length have been provided for a collection of Boolean formulas [25], branching program with bounded widths [7, 35], and small depth circuits [16, 36]. Unfortunately none of these computational settings fits the local randomized computation of the CLP procedures that we wish to derandomize. A useful property, however, of the CLP procedures is that they run (locally) in polynomial time in the maximum degree of the graph.

Our derandomization is based on a brute-force construction of PRG functions that can ϵ-fool the family of all polynomial time computation using a seed length of $O(\log n)$ bits, for $\epsilon = 1/n^c$. The drawback of these PRGs is that they are non-explicit (though can be found by an expensive brute-force computation), and require space which is exponential in the seed length to specify. This, in particular, implies that even if we relax the local computation constraint, in order to fit the space limitations of the low-space MPC model, we must introduce an additive *sublinear* error of $1/n^\alpha$ for some small constant α that depends on the low-space exponent ϕ. In other words, one *can* simulate the CLP procedures using a PRG which fits in machines' local space, but this PRG requires (i) local computation which is exponential in the local space bound, and (ii) a weakened success guarantee to $1 - 1/n^\alpha$ for a small constant $\alpha \in (0, 1)$. We next explain how to handle this larger probability of errors.

Handing sublinear errors. The increase in the error using small seeds creates complications in several CLP procedures, for the following reason. The CLP procedures are highly sensitive to the *order* in which the nodes get colored. In particular for certain classes of nodes, the analysis is based on showing the each coloring step did not color *too many* neighbors of a given node, while at the same time, colored a sufficiently many neighbors of that node. In other words, a given node (or a cluster of nodes) is happy at the end of a given randomized procedure if its coloring status satisfies a given (in many cases delicate and non-monotone) invariant that also depends on the coloring status of its neighbors.

It is non-trivial to derandomize such procedures when suffering from a sublinear error. To see this, assume that the machines can compute a PRG that ϵ-fools the CLP local procedures with $\epsilon = 1/\sqrt{n}$ with a random seed of $\ell = o(\log n)$ bits. Using standard voting on the 2^ℓ possible seeds, the machines can compute the seed Z^* which maximizes the number of happy nodes. Due to the error of ϵ, this implies that all but $\sqrt{n}$ of the nodes are happy. This appears to be quite a large amount of progress. Indeed, at first glance it may seem that one can complete the computation with only one more recursive step over the remaining $\sqrt{n}$ unhappy nodes. The key complication of this approach is that it might now be impossible to make the remaining $\sqrt{n}$ nodes happy under the current color selection to their happy neighbors, since this may have destroyed some necessary properties for the coloring algorithm. Furthermore, if one now starts canceling the colors already assigned to happy neighbors, it might create long cancellation chains, ending up with uncoloring all the nodes.

We overcome this impasse using several different approaches, depending on the precise properties of the CLP procedure and of the node class on which it is applied. For example, for one derandomization procedure (Section 4.2), we combine PRGs with bounded independence hash functions. Informally, the latter are used to partition nodes into groups, to which we apply our PRG in turn, in such a way that error from the PRG can only cause damage within each group, and any of the remaining groups still have the necessary properties to make all nodes happy. In another procedure (Section 4.4), where we apply the PRG to *clusters* of nodes, we extend the happiness property of a cluster S to also include conditions on neighboring clusters as well as well as S itself. By carefully choosing these conditions, we will then see that we *can* safely uncolor clusters that do not satisfy their self-related conditions, without violating the necessary conditions of their neighbors. In this way we avoid causing chains of cancellations.

In Sec. 2.3, we provide the formal PRG definitions, and describe the general (partial) derandomization in more details.

2 ALGORITHM DESCRIPTIONS

2.1 Terminology and a Quick Exposition of the CLP Algorithm

In the description below, we focus on the main randomized part (a.k.a the *pre-shattering* part) of the CLP algorithm [12], that runs in $O(\log^* \Delta)$ rounds. We start by providing useful definitions, originally introduced by Harris, Schneider and Su [26].

Definition 2. *For an $\epsilon \in (0, 1)$, an edge $e = (u, v)$ is called an ϵ-*friend *if $|N(u) \cap N(v)| \geq (1-\epsilon)\Delta$. The endpoints of an ϵ-friend edge are called ϵ-*friends*. A node v is denoted as ϵ-*dense *if v has at least $(1-\epsilon)\Delta$ many ϵ-friends, otherwise it is ϵ-*sparse*. An ϵ-*almost clique *is a connected component of the subgraph induced by the ϵ-dense nodes and their incident ϵ-friend edges.*

The next lemma (Lemma 3.1 of [12]) summarizes the key properties of the almost-cliques. Throughout, assume that $\epsilon < 1/5$ and let $V_\epsilon^d, V_\epsilon^s$ be the subsets of ϵ-dense (ϵ-sparse) nodes.

Lemma 3. *For every ϵ-almost clique C and every $v \in C$ it holds:*

- $|(N(v) \cap V_\epsilon^d) \setminus C| \leq \epsilon\Delta$ *(i.e., small external degree w.r.t ϵ-dense nodes).*
- $|C \setminus (N(v) \cup \{v\})| < 3\epsilon\Delta$ *(small antidegree).*
- $|C| \leq (1 + 3\epsilon)\Delta$ *(small size).*
- $\mathrm{dist}_G(u, v) \leq 2$ *for each $u, v \in C$.*

The CLP algorithm starts by applying a $O(1)$-round randomized procedure that colors a subset of the nodes in a way that generates for the remaining uncolored nodes a *slack* in their number colors. Formally the slack of a node is measured by the difference between the number of colors available in its palette and its uncolored degree. Our focus will be coloring on the uncolored nodes, denoted by V^*. The procedure is based on computing a hierarchy of ϵ-almost cliques for a sequence of increasing ϵ values $\epsilon_1 < \ldots < \epsilon_\ell$. The hierarchy partitions V^* into $\ell = O(\log \log \Delta)$ *layers* as follows. Layer 1 is defined by $V_1 = V^* \cap V_{\epsilon_1}^d$ and $V_i = V^* \cap (V_{\epsilon_i}^d \setminus V_{\epsilon_{i-1}}^d)$ for every $i \in \{2, \ldots, \ell\}$. Letting $V_{sp} = V^* \cap V_\ell^s$, we have that $(V_1, \ldots, V_\ell, V_{sp})$ is a partition of V^*. The nodes of V_i are denoted as

layer-i nodes, these nodes are both ϵ_i-dense and also ϵ_{i-1}-sparse. The nodes of V_{sp} are denoted as *sparse* nodes.

Blocks: The layer-i nodes V_i are further partitioned into *blocks*, which refer to a set of layer-i nodes in a given almost-clique. Letting $(C_1, \ldots, C_k)$ be the list of ϵ_i-almost cliques, define the block $B_j = C_j \cap V_i$. The block-list $(B_1, \ldots, B_k)$ is a partition of V_i. A block $B_j \subseteq V_i$ is called a *layer-i block*. The blocks are classified into three types based on their size: *small, medium and large*. A layer-i block B is *large-eligible* if $|B| \geq \Delta/\log(1/\epsilon_i)$. The division into the three types depends on the relations between the blocks which can be captured by a rooted tree $\mathcal{T}$. For $i < i'$, a layer-i block B is a descendant of a layer-i' block B' if both are subsets of the same $\epsilon_{i'}$-almost clique. The root of the tree $\mathcal{T}$ is the set V_{sp} of the sparse nodes. The set of *large* blocks is a maximal set of large-eligible and independent blocks (i.e., which are not ancestors or descendants of each other) which prioritizes by size and breaking ties by layer. *Medium* blocks are large-eligible blocks which are not large, and the remaining blocks are *small*. Let V_i^S, V_i^M and V_i^L be the set of layer-i nodes in a layer-i small (medium and large, resp.) blocks. For each $X \in \{S, M, L\}$, let $V_{2+}^X = \bigcup_{i=2}^{\ell} V_i^X$. The nodes $V^* \setminus V_{sp}$ are colored in six stages according to the order

$$(V_{2+}^S, V_1^S, V_{2+}^M, V_1^M, V_{2+}^L, V_1^L) .$$

This ordering ensures that when a given node is considered to be colored, it has sufficiently many remaining colors in its palette. At the end of these six stages, there will be a small subset $U \subset V^* \setminus V_{sp}$ of uncolored nodes. The sets $V_{sp} \cup U$ will be colored later on efficiently within $O(\log^* \Delta)$ rounds. The main benefit of defining the six classes is in providing a sufficient amount of slack when considering a given node for coloring.

Lemma 4. *[Lemma 3.3 of [12]] For each layer $i \in [1, \ell]$, the following are true:*

- *$\forall v \in V_i^S$ with $|N(v) \cap V^*| \geq \Delta/3$, we have $|N(v) \cap (V_{2+}^M \cup V_1^M \cup V_{2+}^L \cup V_1^L \cup V_{sp})| \geq \Delta/4$.*
- *For each $v \in V_i^M$, we have $|N(v) \cap (V_{2+}^L \cup V_L^1 \cup V_{sp})| \geq \Delta/(2\log(1/\epsilon_i))$.*

Since the nodes in small and medium blocks have many neighbors in the other sets, when coloring these nodes we enjoy their excess in the number of colors (restricted to their neighbors in the given class). In what follows, we provide a derandomization scheme for each of the randomized procedures applied in the CLP algorithm. We adhere, in general, to the same notation used by the CLP algorithms in [12].

2.2 High-Level Description of our Algorithm

Throughout, a degree bound Δ' is said to be *medium* if $\Delta' = O(n^\beta)$ for some constant β sufficiently smaller than ϕ. In addition, Δ' is *low* if it is polylogarithmic. Low-degree graphs can be handled by derandomizing the (deg +1) coloring algorithm of Barenboim et al. [4]. The main algorithm for $\Delta \geq \log^c n$, for some constant c, has two main steps.

Step 1: In the first step, we reduce the problem to graphs with maximum degree n^β for any desired constant $\beta \in (0, 1)$. This step takes $O(1)$ number of rounds, using bounded independent hash functions. Our deterministic graph partitioning has the same properties as the randomized partitioning of Chang et al. [11].

Step 2: The second step of the algorithm assumes that $\Delta = O(n^\beta)$. This allows one to store a constant-radius ball of a node on a given machine. Similarly to the CLP algorithm, the derandomization has three main parts: (i) initial coloring (which generates the initial excess in colors) (ii) dense coloring (e.g., coloring nodes with almost-clique neighborhoods) and (iii) coloring bidding which colors nodes with excess colors. Parts (i) and (ii) are derandomized within $O(1)$ number of rounds, and part (iii) is derandomized in $O(\log^* \Delta)$ rounds. In our algorithm, we apply only a partial implementation of procedure (ii), as our goal is to reduce the uncolored degree to a polylogarithmic bound. The coloring of the dense nodes is completed within $O(\log\log\log n)$ rounds by derandomizing the (deg +1) list coloring algorithm by Barenboim et al. [4].

Road-map. In Sec. 2.3 and 2.4 we present our derandomization tools of PRG and bounded-independence hash functions. Note that Sec. 2.3 introduces notations that will be used throughout our algorithms. In Sec. 4.1, we first provide a deterministic (deg +1) coloring algorithm for graphs with polylogarithmic degrees. We therefore assume from now on that $\Delta \geq \log^c n$ for a sufficiently large constant c. In Sec. 3 we describe the first step of our coloring algorithm, where we apply a recursive partitioning which results in medium degree coloring instances. The derandomization of CLP of Step 2 spans over Sec. 4.2,4.3 and 4.4. In Sec. 4.2 we provide a derandomization of the OneShotColoring procedure for generating the *initial* color excess for every node as a function its neighborhood sparsity. Then we turn to consider the coloring of the dense vertices $V \setminus V_{sp}$. Recall that these nodes are partitioned into the classes $(V_{2+}^S, V_1^S, V_{2+}^M, V_1^M, V_{2+}^L, V_1^L)$. In Sec. 4.3, we provide a derandomization of the CLP procedures for coloring the dense nodes in small and medium blocks $V_{2+}^S, V_1^S, V_{2+}^M, V_1^M$. Sec. 4.4 considers the remaining dense nodes in the large blocks V_{2+}^L, V_1^L. Our derandomization reduces the uncolored degrees of the nodes in V_{2+}^L as a function of their sparsity. In addition, it reduces the degrees of the uncolored nodes in V_1^L to polylogarithmic. The coloring of the remaining V_1^L can be then completed in $O(\log\log\log n)$ rounds. Sec. 4.5 handles the remaining uncolored vertices in layer ≥ 2, as well as the sparse vertices V_{sp}.

2.3 Pseudorandom Generators and Derandomization

We will now formally define pseudorandom generators (PRGs). A PRG is a function that gets a short random seed and expands it to a long one which is indistinguishable from a random seed of the same length for a given class of algorithms. We will use the following definitions from [41]: in the latter, U_k denotes the uniform distribution on $\{0,1\}^k$.

Definition 5 (**Computational Indistinguishability**, Def. 7.1 in [41]). *Random variables X and Y taking values in $\{0,1\}^m$ are (t, ϵ)* indistinguishable *if for every nonuniform algorithm T running in time at most t, we have $|\Pr[T(X) = 1] - \Pr[T(Y) = 1]| \leq \epsilon$.*

Definition 6 (**PRG**, Definition 7.3 in [41]). *A deterministic function $\mathcal{G} : \{0,1\}^d \to \{0,1\}^m$ is an (t, ϵ) pseudorandom generator (PRG) if: (1) $d \leq m$ and (2) $\mathcal{G}(U_d)$ and U_m are (t, ϵ) indistinguishable.*

A simple counting argument (given in, e.g., [41]) shows that there must *exist* PRGs with short seeds:

Proposition 7 (Proposition 7.8 in [41]). *For all $m \in \mathbb{N}$ and $\epsilon > 0$, there exists a (non-explicit) (m, ϵ) PRG $\mathcal{G} : \{0,1\}^d \to \{0,1\}^m$ with seed length $d = O(\log m + \log 1/\epsilon)$.*

The next lemma follows by a brute-force PRG construction from [15].

Lemma 8. *For all $m \in \mathbb{N}$ and $\epsilon > 0$, there exists an algorithm for computing the (m, ϵ) PRG of Proposition 7 with seed length $d = O(\log m + \log 1/\epsilon)$, in time $\exp(\text{poly}(m/\epsilon))$ and space $\text{poly}(m/\epsilon)$.*

Derandomization with PRG. A randomized LOCAL algorithm $\mathcal{A}$ is said to be *nice* if the local computation, per round, performed at each node is polynomial in Δ. All the randomized LOCAL procedures that we derandomize with the PRG framework in this paper, will indeed be nice. We will also have the property that $\text{poly}(\Delta)$ bits fit the local space of each machine. To illustrate the technique, assume that $\mathcal{A}$ is a *two* round randomized algorithm such that after applying $\mathcal{A}$, each node satisfies a given desired property that depends only on its 1-radius ball, with high probability of $1 - 1/n^c$. It is convenient to view this two-round LOCAL algorithm in a way that decouples the randomness from the computation. Specifically, we assume that each node a priori generates its own pool of random coins, and we simulate $\mathcal{A}$ in the LOCAL model by letting each node v first collects its two-hop ball $v \cup N_{G^2}(v)$, as well as, the initial states and the private coins of each of its 2-hop neighbors. Then, each node v locally applies an algorithm $\mathcal{A}_v$ on this information. In this view, every algorithm $\mathcal{A}$ consists of n sub-algorithms $\{\mathcal{A}_v, v \in V\}$ where each $\mathcal{A}_v$ is a randomized $\text{poly}(\Delta)$-time algorithm, the randomized decisions made by each node u are consistent with all the algorithms run by its 2-hop neighbors. We then say that a node v is *happy* if a certain property holds for $\{v\}$. Our goal is to show that using the PRG framework, there is a low-space MPC deterministic algorithm that derandomizes $\mathcal{A}$ in a way that makes at least $1 - 1/n^\alpha$ fraction of the nodes happy, for some constant $\alpha \in (0, 1)$.

The first preprocessing step for the derandomization computes $O(\log \Delta)$-bit identifiers for the nodes such that the identifiers are distinct in each 2-radius ball. This can be done in $O(\log^* n)$ deterministic MPC rounds [29, 30].

Claim 9. *Given that all the nodes have $O(\log \Delta)$-bit identifiers (unique in each 2-radius ball), there exists a low-space MPC deterministic algorithm that causes a collection of at least $(1 - 1/n^\alpha)|V|$ nodes to be happy, for some constant α sufficiently smaller than ϕ. The round complexity of the algorithm is $O(1)$, and it requires .*

The derandomization is based on two parts. First, we show that there is a weaker variant of algorithm $\mathcal{A}$ that uses only a shared random seed of $c \cdot \phi \cdot \log n$ bits for a sufficiently small constant $c \in (0, 1]$. This weaker variant suffers an additive error of $\pm 2/n^\alpha$ (for some constant α sufficiently smaller than ϕ), compared to the fully-randomized algorithm. Then, we derandomize this weaker variant in $O(1)$ MPC rounds in such a way that at least $(1-1/n^\alpha)|V|$ nodes are happy.

We start with the first step, which is the part that exploits the PRG machinery. Let $t = \text{poly}(\Delta)$ be an upper bound on the local time complexity of all $\{\mathcal{A}_v, v \in V\}$ algorithms, and let $N = \Delta^c$ be an upper bound on the largest node identifier. The local randomized algorithm $\mathcal{A}_v$ applied locally at each node v can be represented as a *deterministic* algorithm that gets as input a vector of $N \cdot t$ random coins interpreted as follows: the i^{th} chunk of t coins specifies the random coins for a node with ID i for every $i \in \{1, \ldots, N\}$. Since the algorithm $\mathcal{A}_v$ runs in time t, it is sufficient to specify at most t random bits for each node in $\{v\} \cup N_{G^2}(v)$. The weaker randomized algorithm, denoted by $\mathcal{A}'_v$, will be given a collection of $N \cdot t$ *pseudorandom* coins obtained by applying a PRG function $\mathcal{G}^*$ on a shared random seed of only $c\phi \log n$ random coins, for a sufficiently small constant $c \in (0, 1]$.

Specifically, by Prop. 7, there is an $(N \cdot t, \epsilon)$ pseudorandom generator $\mathcal{G}^* : \{0,1\}^d \to \{0,1\}^{Nt}$ with a seed length $d = O(\log(Nt) + \log 1/\epsilon)$. This $\mathcal{G}^*$ function ϵ-fools the collection of all t-time randomized algorithms up to an additive error of ϵ. In particular, it fools the collection of all $\{\mathcal{A}_v \mid v \in V\}$ algorithms. We choose the constant in the seed length to be small enough so that the machines will be able to locally compute $\mathcal{G}^*$ within their space limitations. Since the PRG computation consumes $2^d \text{poly}(\Delta)$ space, we can support an additive error of $\epsilon = 1/(2n^\alpha)$ for some small constant α. This provides a seed of length $d = c\phi \log n$ for a small constant $c \in (0, 1]$. Let $Z \in \{0,1\}^d$ be a random seed of length d. We then have that when each node v simulates $\mathcal{A}_v$ using $\mathcal{G}^*(Z)$ as the source of Nt pseudorandom coins, the node v is happy with probability of $1 - 1/n^c - 1/(2n^\alpha) \leq 1 - 1/n^\alpha$. It is important to note that since all nodes use the shared random seed Z, and since the pseudorandom coins assigned by each local algorithm $\mathcal{A}'_v$ to node $u \in \{v\} \cup N(v)$ are determined by the $O(\log \Delta)$-bit identifier of u, the output of u is consistent by all the algorithms $\mathcal{A}_w$ for every $w \in \{u\} \cup N_{G^2}(u)$.

It remains to show that these weaker algorithms $\{\mathcal{A}'_v\}$ can be derandomized within $O(1)$ rounds. This is done by allocating a machine M_v for every node v that stores also the 2-radius ball of v in G. Every machine M_v simulates algorithm $\mathcal{A}'_v$ under each $Z \in \{0,1\}^d$. Specifically, for each $Z \in \{0,1\}^d$, it simulates $\mathcal{A}_v$ using $\mathcal{G}^*(Z)$ as the input of random coins. This allows each machine M_v to determine if v is happy under each possible seed Z. As there are $o(n^\phi)$ seeds, this fits the local space. Using standard sorting procedures, in constant rounds the machines can compute the seed Z^* that maximizes the number of happy nodes. Finally, all machines simulate $\mathcal{A}'_v$ using the seed Z^*, which defines the output of the algorithm. Since the expected number of happy nodes with a random seed $Z \in \{0,1\}$ is at least $(1 - 1/n^\alpha)|V|$, the number of happy nodes under the best seed Z^* is at least $(1 - 1/n^\alpha)|V|$ as well.

In Section 4.4, the procedure is slightly more complicated: there, the definition of a happy *cluster* S will contain both *self-invariants* (properties about S) and *neighbor-invariants* (properties about neighboring clusters to S). We use the PRG to show that the number of happy clusters is at least a $(1-1/n^\alpha)$-proportion of the total number of clusters, as above. However, unhappy clusters are then uncolored. By the choice of the happiness properties, we show that clusters that were happy still satisfy their self-invariants, which will be sufficient to allow unhappy uncolored neighbors the chance to become happy even when running the coloring algorithm on the remaining unhappy nodes. The analysis will show that within $O(1/\alpha)$

iterations, all clusters become happy. These arguments involve introducing some extra slack to the bounds of CLP: specifically, our bounds (e.g., on the desired uncolored degrees) at the end of this derandomization will be larger by a factor of $(1/\alpha) = O(1)$ than those obtained by the randomized CLP procedures.

Throughout the paper we use the parameter α to determine the exponent of the additive error of the pseudorandom algorithms. That is, we will only be considering LOCAL randomized algorithms that succeeds at each node with high probability, and using the space limitation of the machines, we will a get a pseudorandom LOCAL algorithm that succeeds with probability of $1 - 1/n^{\alpha}$ for some sufficiently small α. Consequently, the derandomization will cause at least a $(1 - 1/n^{\alpha})$ fraction of the nodes to be happy.

Useful Observations for the CLP Algorithm. The above mentioned general derandomization scheme fits well into our setting of derandomizing the CLP algorithm. Specifically, we observe the following useful property for the CLP algorithm. This allows us to work, throughout, with $O(\log \Delta)$-bit identifiers, which is crucial for the derandomization procedure. We show:

Observation 10. *All the randomized procedures of the CLP algorithm are nice and run in $O(1)$ rounds*[4]*. In addition, these algorithms can be simulated in an analogous manner in the following setting: all nodes are given $O(\log \Delta)$-bit identifiers that are unique within each $O(1)$-radius ball (for any desired constant), and all nodes of the same identifier are given the same set of* $\mathrm{poly}(\Delta)$ *random coins to simulate their random decisions. Provided the Δ is at least polylogarithmic, the procedures satisfy some desired properties at each node* with high probability.

2.4 Bounded-Independence Hash Functions

Some of the local randomized procedures considered in this paper requires a more light-weight derandomization scheme which based on bounded-independence hash functions. Like PRGs, these functions can approximate the effect of random choices using only a small seed (which, additionally, can be computed efficiently in polynomial time, rather than in exponential time as in the PRG setting). The major benefit of bounded-independence hash functions over PRGs is that they do not incur *error*, as the PRGs we use do. That is, they provide *exactly* the same bounds as true randomness for analysis that only requires independence between a bounded number of random choices. The families of hash functions we require are specified as follows:

Definition 11. *For $N, L, k \in \mathbb{N}$ such that $k \leq N$, a family of functions $\mathcal{H} = \{h : [N] \to [L]\}$ is* k-wise independent *if for all distinct $x_1, \ldots, x_k \in [N]$, the random variables $h(x_1), \ldots, h(x_k)$ are independent and uniformly distributed in $[L]$ when h is chosen uniformly at random from $\mathcal{H}$.*

We will use the following well-known lemma (cf. [41, Corollary 3.34]).

Lemma 12. *For every a, b, k, there is a family of k-wise independent hash functions $\mathcal{H} = \{h : \{0,1\}^a \to \{0,1\}^b\}$ such that choosing a random function from $\mathcal{H}$ takes $k \cdot \max\{a, b\}$ random bits, and evaluating a function from $\mathcal{H}$ takes time $poly(a, b, k)$.*

[4]The sparse coloring procedure runs in $\ell = O(\log^* \Delta)$ rounds but it is based on ℓ applications of a $O(1)$-round procedure, which we will derandomize with PRGs.

2.5 The Method of Conditional Expectations

To agree on seeds specifying a particular hash function, we make use of a distributed implementation of the classical *method of conditional expectations*, as employed in [13, 14]. The properties of this method can be stated as follows:

Assume that, over the choice of a random hash function $h \in \mathcal{H}$, the expectation of some objective function (which is a sum of functions q_x calculable by individual machines) is at least some value Q. That is,

$$\mathbf{E}_{h \in \mathcal{H}}\big[q(h) := \sum_{\text{machines } x} q_x(h)\big] \geq Q \ .$$

Then, if $|\mathcal{H}| = poly(n)$, i.e., hash functions from $\mathcal{H}$ can be specified using $O(\log n)$ bits, there is an $O(1)$-round deterministic low-space MPC algorithm allowing all machines to agree on some specific $h^* \in \mathcal{H}$ with $q(h^*) \geq Q$.

The method with which we fix seeds for our PRGs can also be thought of as a special case of this implementation of the method of conditional expectations; the difference is that the seeds for our PRGs are short enough that the entire seed space fits in a machine's memory and can be searched at once, whereas our hash function seeds are a constant-factor longer and the seed space must be searched in $O(1)$ iterations.

3 REDUCING TO MEDIUM-DEGREE INSTANCES

In this section, we provide a deterministic (recursive) degree reduction procedure, and show the following:

Lemma 13. *Assume that for every n, there is an $O(\log\log\log n)$-round low-space MPC algorithm $\mathcal{A}$ for computing $(\Delta+1)$ list coloring in a graph G provided that (i) $\Delta = n^{O(\zeta)}$ for a constant ζ sufficiently smaller than ϕ, and (ii) each vertex has a palette of size $\min\{\deg_G(v), \Delta - \Delta^{3/5}\}$. Then there is an $O(\log\log\log n)$-round $(\Delta+1)$ list coloring algorithm for* any *n-vertex graph G.*

We do so by using an extension of a method introduced in [14], which deterministically reduces an instance of coloring with high degree to a collection of instances of lower degree, of which many can be solved in parallel. We run the following algorithm up to a recursion depth of $\frac{\log_n \Delta}{\zeta} - 23$ (note that we have control of ζ, and set it so that this is an integer) - upon reaching that recursion depth we apply algorithm $\mathcal{A}$ in place of the recursive call to LowSpaceColorReduce. Throughout we use families of $O(1)$-wise independent hash functions. See Sec. 2.4 for definitions.

Algorithm 1 LowSpaceColorReduce(G)

$G_1, \ldots, G_{n^\zeta} \leftarrow$ LowSpacePartition(G).
For each $i = 1, \ldots, n^\zeta - 1$, perform LowSpaceColorReduce(G_i) in parallel.
Update color palettes of G_{n^ζ}, perform LowSpaceColorReduce(G_{n^ζ}).

This algorithm employs a partitioning procedure to divide the input instance into *bins*, which are then solved recursively:

Algorithm 2 LowSpacePartition(G)

Let hash function $h_1 : [poly(n)] \to [n^{\zeta}]$ map each node $v \notin V_0$ to a bin $h_1(v) \in [n^{\zeta}]$.
Let hash function $h_2 : [poly(n)] \to [n^{\zeta} - 1]$ map colors γ to a bin $h_2(\gamma) \in [n^{\zeta} - 1]$.
Let $G_1, \ldots, G_{n^{\zeta}}$ be the graphs induced by bins $1, \ldots, n^{\zeta}$ respectively.
Restrict palettes of nodes in $G_1, \ldots, G_{n^{\zeta}-1}$ to colors assigned by h_2 to corresponding bins.
Return $G_1, \ldots, G_{n^{\zeta}}$.

(If the ranges of the hash functions are not powers of 2, we can instead map to a sufficiently large (but $poly(n)$) power of 2, and then map intervals of this range to $[n^{\zeta}]$ or $[n^{\zeta} - 1]$ as equally as possible. As in [13, 14], we incur some error in the distributions of random hash function outputs, but can easily ensure that it is negligibly small, e.g. n^{-3}.) The analysis of this procedure is similar to [14], and deferred to the full version.

4 IMPROVED COLORING VIA DERANDOMIZATION OF CLP

In this section, we show a deterministic coloring algorithm that runs in $O(\log\log\log n)$ rounds, by derandomizing the $(\Delta + 1)$-list coloring algorithm of [12]. Throughout, we assume that $\Delta \in [\log^c n, n^{\phi/c}]$ for a sufficiently large constant c. Due to Lemma 13, this can be assumed, *almost* without loss of generality. Specifically, the guarantee of Lemma 13 is that each v has a palette of $\min\{\deg_G(v), \Delta - \Delta^{3/5}\}$ colors, instead of $\Delta + 1$ colors. This was obtained also for the randomized procedures of [11] and [38]. As observed in [38], the CLP algorithm works exactly the same for that setting, up to minor modifications in the constants of the lemma statements. For simplicity of that section, we assume the standard $(\Delta + 1)$ list coloring setting, and then in the full version explain the minor adaptations to handle the palettes of Lemma 13.

Smaller IDs. Our approach is based on derandomizing t-round local algorithms for some constant t. The first preliminary step for this derandomization is to assign the nodes unique identifiers of $O(\log\Delta)$ bits, that are unique in each t-radius ball. To do that, the algorithm applies the well-known algorithm of Linial [30] to compute Δ^{2t} coloring in G. This can be done in $O(\log^* n)$ rounds. This part is important for the PRG based simulations as explained in Sec. 2.3. We next iterate over the key CLP procedures and derandomize them using PRGs.

The key challenging part is in derandomizing the dense nodes in the large blocks of Sec. 4.4. These nodes have excess of colors due to neighbors in different classes, and thus the randomized coloring procedure is highly sensitive to the order in which the nodes get colored.

4.1 Low-Deg Coloring

We start by providing a $O(\log\log\log n)$-round algorithm for computing $(deg + 1)$-coloring in graphs with maximum degree at most $\log^c n$ for any constant c. This allows us later on to focus on graphs with maximum degree $\Delta \geq \log^c n$ for which we provide a derandomization of the CLP algorithm.

Lemma 14. *For any n-node graph G with maximum degree $d \leq \log^c n$, there exists an $O(\log\log\log n)$-deterministic algorithm for computing* $(\deg+1)$ *list coloring.*

Proof. The algorithm derandomizes the state-of-the-art local algorithm for the $(\deg+1)$ list coloring problem of [4] that runs in $T = O(\log d) + \text{poly}(\log\log n)$ rounds (when combined with the network decomposition result of [40]). First, the MPC algorithm allocates a machine M_v for every node v which collects the $3T$-radius ball of v in G. Since $d^T = n^{o(1)}$ this fits the local space of each machine. Via the standard graph exponentiation technique, these balls can be collected in $\log T$ rounds.

Since the T-round randomized algorithm of [4] and [40] employs a polynomial(in d) time computation at each node, we can again use the PRG machinery as follows. In what follows we provide an $O(\log\log\log n)$-round MPC procedure to color $(1 - 1/n^{\alpha})$ of the nodes. Repeating this procedure on the remaining uncolored graph guarantees that within $O(1/\alpha)$ repetitions, all nodes are colored.

The algorithm of [4] has two steps. The first is a randomized pre-shattering procedure that in $O(\log d)$ rounds guarantees that the every going component has size at most $\text{poly}(d)\log n$, w.h.p. This holds even if the random decisions of nodes at distance c for some constant c are adversarially correlated. That is, it sufficient to have independence in each $O(1)$-radius ball. This implies that we can continue using our $O(\log\Delta)$-bit identifiers that are unique in each c-radius ball for the purposes of derandomization via PRG.

The second step completes the coloring by applying the $\text{poly}(\log N)$ deterministic algorithm for coloring for $N = \text{poly}(\log n)$. Since we have already collected the T-radius ball around each node v onto a machine M_v, each machine M_v can now simulate this second step immediately.

It remains to focus on the derandomization of the pre-shattering step. Recall that each machine M_v stores the $2T$-radius ball of v, collected in $O(\log T)$ rounds. The machine M_v computes the PRG function $\mathcal{G}$ that ϵ-fools every $\text{poly}(d)$ time computation for $\epsilon = 1/(2n^{\alpha})$ for some small constant $\alpha \in (0, 1)$. Specifically, the parameter α is chosen using Lemma 8 in a way that guarantees that the computation of the function $\mathcal{G}$ can be done in space of n^{ϕ}. This yields a seed of length $d < \log n/\phi$. Simulating the T-round randomized algorithm with a random d-length seed guarantees that each node remains uncolored with probability of at most $1/n^{\alpha}$. The machines can then compute a seed Z^* that matches this expected value, and all machines simulate the T-round randomized algorithm using Z^*. □

4.2 Initial Slack Generation

The CLP algorithm starts by applying a $O(1)$-round randomized procedure, called OneShotColoring, which translates the sparsity in nodes' neighborhoods into a slack (excess) in the number of colors. In this procedure, each uncolored node participates with probability p, and each participating node v randomly selects a color $c(v)$ from its palette $\Phi(v)$. This color is selected for v only if none of its neighbors picked that color.

Lemma 15. *[Lemma 2.5 from [12]] There is a $O(1)$-round randomized LOCAL algorithm that colors a subset of the nodes V, such that the following are true for every node v with $deg(v) \geq \frac{5}{6}\Delta$:*

- *(P1) With probability* $1 - 1/n^c$*, the number of uncolored neighbors of* v *is at least* $\Delta/2$.
- *(P2) With probability* $1 - 1/n^c$*,* v *has at least* $\Omega(\epsilon^2\Delta)$ *excess colors, where* ϵ *is the highest value such that* v *is* ϵ*-sparse for* $\epsilon \geq 1/\Delta^{1/10}$.

We derandomize this procedure and sketch our method here.

We will partition nodes into $\Theta(1)$ groups using bounded-independence hash functions, in such a way that we ensure that the local sparsity property is maintained *within each group*. We will then apply the PRG to directly derandomize Lemma 15 within each group. The reason for the partitioning is that the error of the PRG will cause some nodes to fail to meet the criteria. However, since we have multiple groups, and any one group can provide a node with sufficient slack, we will be able to show that the PRG will succeed in at least one group for each node. For this reason our PRG derandomization here is simpler than its applications to other parts of the CLP algorithm: we need only ensure that, when coloring each group, the number of nodes which have not yet received sufficient slack decreases by an n^α-factor. After coloring $1/\alpha = O(1)$ groups, we ensure that all nodes have sufficient slack, reaching Theorem 16.

Theorem 16. *[Derandomization of Lemma 2.5 from [12]] There is a* $O(1)$*-round deterministic low-space MPC algorithm that colors a subset of the nodes* V*, such that the following are true for every node* v *with* $deg(v) \geq (5/6)\Delta$*:*

- *(P1) The number of uncolored neighbors of* v *is at least* $\Delta/2$.
- *(P2)* v *has at least* $\Omega(\epsilon^2\Delta)$ *excess colors, where* ϵ *is the highest value such that* v *is* ϵ*-sparse for* $\epsilon \geq 1/\Delta^{1/10}$.

4.3 Dense Coloring with Slack

This section handles the collection of the dense nodes in the small and medium size clusters, namely, a set $S \in \{V_1^S, V_1^M, V_{2+}^S, V_{2+}^M\}$. Each of these nodes has sufficiently many neighbors not in the current node set S, which provides a slack in the number of available colors (regardless of how we color S).

The set S is a collection of clusters $S_1, \ldots, S_g$ of weak diameter 2. It is assumed that the edges within S are oriented from the sparser to the denser endpoint, breaking ties by comparing IDs. For $v \in \bigcup_j S_j$, let $N_{out}(v)$ be the outgoing[5] neighbors of v in S. The dense coloring CLP procedures for these sets are based on applying procedure DenseColoringStep (version 1) which works as follows. For each cluster S_j, a leader node $u \in S_j$ collects its cluster nodes and their current palettes, and colors the nodes in S_j sequentially according to some random permutation. For each node v (in that ordering), the leader picks a free color uniformly at random from the list of available colors of v. This color is set as permanent, if it does not conflict with the colors of $N_{out}(v)$. While it is not so clear how to derandomize this procedure efficiently (i.e., in polynomial time) using the common derandomization techniques (such as bounded-independence hash functions), we show in this section that using PRGs a derandomization is possible, but at the cost of exponential local computation.

We start by describing the dynamics of the coloring procedure for dense nodes under the *randomized* procedure DenseColoringStep of [12], and then explain how to derandomize it within $O(1)$ number of rounds. Every dense node is associated with two parameters which governs its coloring probability in the DenseColoringStep procedure when applied simultaneously by a given set $S = S_1 \cup \ldots \cup S_g$:

- A parameter Z_v which provides a lower bound on the number of excess colors of v w.r.t the nodes in S. I.e., the palette size of v minus $|N_{out}(v) \cap S|$ is at least Z_v.
- A parameter D_v which provides an upper bound on the external degree of v, i.e., $|N_{out}(v) \setminus S_j| \leq D_v$ where S_j is the cluster of v.

The ratio between these bounds, denoted by $\delta_v = D_v/Z_v$, determines the probability that a node $v \in S_j$ remains uncolored after a single application of the DenseColoringStep procedure. Two clusters S_i and S_j are *neighbors* if there exist $u \in S_i$ and $v \in S_j$ such that $(u, v) \in E(G)$. For any positive integer $r \geq 1$, let $N^r(S_j)$ be the r-hop neighboring clusters of S_j in S. For $r = 1$, we may omit the index and simply write $N(S_j)$ to denote the immediate cluster neighbors of S_j in S.

Theorem 17. *[Derandomization of Lemma 4.2 of [12]] Let* $S \in \{V_{2+}^S, V_{2+}^M\}$*. Suppose that each layer-*i *node* $v \in S$ *has at least* $\frac{\Delta}{2\log(1/\epsilon_i)}$ *excess colors w.r.t* S*. Then, there exists a* $O(1)$*-round deterministic algorithm that colors a subset of* S *meeting the following condition. For each node* $v \in V^*$*, and for each* $i \in [2, \ell]$*, the number of uncolored* i*-layer neighbors of* v *in* S *is at most* $\epsilon_i^5\Delta$.

To handle the layer-1 nodes in small and medium clusters, we derandomize Lemma 4.3 of [12]. The randomized procedure of that lemma colors *all* the nodes in these clusters with high probability. We next show that we can color all these nodes deterministically.

Theorem 18. *[Derandomization of Lemma 4.3 of [12]] Let* $S \in \{V_1^S, V_1^M\}$*. Suppose that each node* $v \in S$ *has at least* $\Delta/(2\log(1/\epsilon_i))$ *excess colors w.r.t* S*. There exists a* $O(1)$*-round deterministic algorithm that colors all the nodes of* S.

4.4 Dense Coloring without Slack

In this section we consider the more challenging subset of dense nodes, in the large blocks V_1^L and V_{2+}^L, that have no slack in their colors due to neighbors in other subsets. The randomized CLP procedure colors these nodes by employing a modified variant of the dense coloring procedure, denoted as DenseColoringStep (version 2).

Let $S \in \{V_1^L, V_{2+}^L\}$. For every node $v \in S$, let $N^*(v)$ be the neighbors of v in S of layer number smaller than or equal to the layer number of v. Let the clusters of $S = S_1 \cup \ldots \cup S_g$ be ordered based on nondecreasing order their layer number. Each cluster, and each node in a cluster have an ID that is consistent with this ordering. We use the term *antidegree* of $v \in S_j$ to the number of *uncolored* nodes in $S_j \setminus (N(v) \cup \{v\})$. The term *external degree* of $v \in S_j$ refers to the number of uncolored nodes in $N^*(v) \setminus S_j$. The rate by which a dense cluster $S_j \in S$ gets colored is determined by the following parameters:

- A parameter D_j that provides an upper bound on the uncolored external degree and antidegree[6] of each node $v \in S_j$.

[5] An edge $e = \{u, u'\}$ is oriented as (u, u') if u is at layer i, u' at layer i' and $i > i'$, or if $i = i'$ and $ID(u) > ID(v)$.

[6] In contrast to Sec. 4.3, here the bounds depend on $N^*(v)$ rather than on $N_{out}(v)$.

That is, $|N^*(v) \setminus S_j| \leq D_j$ and $|S_j \setminus (N^*(v) \cap \{v\})| \leq D_j$, respectively.
- A lower bound L_j on the size of the uncolored part of S_j.
- An upper bound U_j on the size of the uncolored part of S_j.
- A shrinking rate $\delta_j \geq D_j \log(|U_j|/D_j)/|L_j|$ that determines the speed at which the cluster S_j shrinks (due to the coloring of its nodes).

Version 2 of the DenseColoringStep procedure is again run by each cluster leader and works as follows. First, the leader of cluster S_j picks a $1-\delta_j$ fraction of the nodes in S_j uniformly at random, and computes a permutation on the elected nodes. It then iterates over these nodes according to the permutation order, picking a free color uniformly at random for each such node. These colors are fixed as the permanent colors, only if there are no collisions with their outgoing external neighbors. The dense coloring CLP procedure is based on having multiple applications of this DenseColoringStep procedure. The key property that underlies the correctness of the DenseColoringStep procedure is summarized in the next lemma.

Lemma 19. *[Lemma 6.2 of [12]] Consider an execution of the* DenseColoringStep *procedure (version 2). Let T be any subset of S and let $\delta = \max_{j:S_j \cap T \neq \emptyset} \delta_j$. For any number t, the probability that the number of uncolored nodes in T is at least t is at most $\binom{|T|}{t} \cdot (O(\delta))^t$.*

The invariants. We will have $O(1)$ iterations of applying the DenseColoringStep procedure. At the beginning of each iteration i, each cluster is required to satisfy certain desired properties concerning the uncolored external and antidegrees of its nodes, and the size of the uncolored cluster. Specifically, at the beginning of iteration i, each cluster S_j is required to satisfy that the antidegree of S_j is at most $D_j^{(i)}$, and that the number of uncolored nodes in S_j is in the range $[L_j^{(i)}, U_j^{(i)}]$. Both in the procedures of coloring V_1^L and V_{2+}^L, the invariants regarding the bounds $L_j^{(i)}, U_j^{(i)}, D_j^{(i)}$ hold with high probability (since $\Delta \geq \log^c n$). Specifically, for $\Delta > \log^c n$, the CLP analysis shows that after the k^{th} application of Procedure DenseColoringStep all blocks satisfy the $(k+1)^{th}$ invariant with probability of $1 - 1/n^c$. Due to the sublinear space limitation of our machines, we will satisfy these properties with probability of $1 - 1/n^\alpha$ using a seed of length $o(\log n)$, for some constant α. To handle this error, we provide a more careful derandomization, which is based on the following terminology.

We classify the invariants of the CLP algorithm into two types: *self-invariant* and *neighbor-invariant*. Roughly speaking, the self-invariant property of a cluster S_j depends only on the coloring status of the nodes of the cluster (e.g., all nodes in S_j should have an antidegree at most x). In contrast, the neighbor-invariant of S depends only on the coloring status of the neighboring clusters $N(S_j)$ of S_j (e.g., all nodes in S_j should have an external degree at most x), where $N(S_j)$ are all the clusters that have at least one neighbor of S_j in the current collection of clusters considered. The algorithm defines $K = O(1)$ invariants where the k^{th} phase of the algorithm assumes that all nodes satisfy the k^{th} invariant. The k^{th} invariant provides lower bound value $L_j^{(k)}$ and an upper bound $L_j^{(k)}$ on the current uncolored size of the cluster S_j. In addition, it provides an upper bound on the (i) external uncolored degree of a node in a cluster, (ii) uncolored antidegree of a node in a cluster, and possibly also on (iii) number of layer-i uncolored neighbors of each node in a cluster.

Definition 20. *A cluster S_j γ-satisfies the k^{th} invariant if it satisfies the neighbor-invariant up to a factor of γ (e.g., the number of uncolored neighbors in different clusters is increased by a factor of γ than the required invariant bounds).*

In our derandomization scheme, we assume that there exists a randomized algorithm with the following properties. As shown in the main paper such an algorithm exists for coloring V_1^L and V_{2+}^L.

Lemma 21. *Let $\mathcal{S}' \subseteq \mathcal{S}$ be a subset of clusters such that all clusters in $\mathcal{S} \setminus \mathcal{S}'$ r-satisfies the $(k+1)^{th}$ invariant, and in addition, each cluster $S_j \in \mathcal{S}'$ satisfies:*

- *(P1) the* $(k+1)^{th}$ neighbor-invariant, *up to a multiplicative factor of r, w.r.t its neighbors in $\mathcal{S} \setminus \mathcal{S}'$ (if such exist).*
- *(P2) the* k^{th} neighbor-invariant *w.r.t its neighbors in $\mathcal{S}'$, and*
- *(P3) the* k^{th} self-invariant.

By applying procedure DenseColoringStep only *to the nodes in $\mathcal{S}'$, w.h.p. it holds that:*

- *(P4) every $S_j \in \mathcal{S}'$ $(r+1)$-satisfies the $(k+1)^{th}$ invariant; and*
- *(P5) every neighboring cluster $S_{j'} \in \bigcup_{S_j \in \mathcal{S}'} N(S_j)$ (where $S_{j'}$ is possibly in $\mathcal{S} \setminus \mathcal{S}'$) $(r+1)$-satisfies the $(k+1)^{th}$ (neighbor) invariant.*

Lemma 22. *Let $\mathcal{S}' \subseteq \mathcal{S}$ be a collection of clusters that satisfy the properties of Lemma 21. Then, given the randomized LOCAL algorithm of Lemma 21, one can provide a $O(1)$-round deterministic MPC procedure that extends the coloring (of nodes in $\mathcal{S}'$), resulting in a subset $\mathcal{S}'' \subseteq \mathcal{S}'$ such that (i) $|\mathcal{S}''| \geq (1 - 1/n^\alpha)|\mathcal{S}'|$ and in addition (ii) the clusters of $\mathcal{S}''$ satisfy (P4,P5).*

The deterministic coloring procedures of the subsets V_1^L and V_{2+}^L follow by the following theorem (upon specifying the precise invariants for each of these subsets).

Theorem 23. *Given the randomized algorithm of Lemma 21 and assuming that the given collection of clusters $\mathcal{S}$ satisfies the k^{th} invariant, there exists a $O(1)$-round deterministic algorithm that extends the current coloring such that all $\mathcal{S}$ clusters $(\lceil 1/\alpha \rceil)$-satisfy the $(k+1)^{th}$ invariant.*

4.5 Sparse Coloring

Let U be the remaining uncolored dense nodes, and recall that V_{sp} is the collection of sparse nodes. The sets U and V_{sp} are colored in [12] by applying an $O(\log^* \Delta)$-round randomized algorithm that w.h.p. colors all nodes in V_{sp} and U. The coloring of these two subsets is obtained by derandomizing the corresponding CLP procedure while maintaining the same order of the number of rounds.

Theorem 24 (Derandomization of Lemma 2.1 of [12]). *Consider a directed acyclic graph G', where each node v is associated with a parameter $p_v \leq |\Phi(v)| - \deg(v)$. Let d^* be the maximum out-degree of the graph, let $p^* = \min_{v \in V} p_v$. Suppose that $d^*, p^* \geq \log^c n$, and that there is a number $C = \Omega(1)$ such that every node v satisfies $\sum_{u \in N_{out}(v)} 1/p_u \leq 1/C$. Then, there is a deterministic algorithm for coloring G' within $O(\log^*(p^*) - \log^* C + 1)$ rounds.*

REFERENCES

[1] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel Algorithms for Geometric Graph Problems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*. 574–583.

[2] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. 2019. Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC)*. 461–470.

[3] Philipp Bamberger, Fabian Kuhn, and Yannic Maus. 2020. Efficient Deterministic Distributed Coloring with Small Bandwidth. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC)*. 243–252.

[4] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2016. The Locality of Distributed Symmetry Breaking. 63, 3 (2016), 20:1–20:45.

[5] Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M. Karp, and Jara Uitto. 2019. Massively Parallel Computation of Matching and MIS in Sparse Graphs. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC)*. 481–490. A preliminary version of a merge of CoRR abs/1807.06701 and CoRR abs/1807.05374.

[6] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Łącki, and Vahab S. Mirrokni. 2019. Near-Optimal Massively Parallel Graph Connectivity. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*. 1615–1636.

[7] Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. 2014. Pseudorandom generators for regular branching programs. *SIAM J. Comput.* 43, 3 (2014), 973–986.

[8] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. 2017. Derandomizing Local Distributed Algorithms under Bandwidth Restrictions. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC)*. 11:1–11:16.

[9] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. 2019. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. 48, 1 (2019), 122–143.

[10] Yi-Jun Chang, Wenzheng Li, and Seth Pettie. 2018. An Optimal Distributed $(\Delta+1)$-Coloring Algorithm?. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*. 445–456.

[11] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. 2019. The Complexity of $(\Delta+1)$ Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC)*. 471–480.

[12] Yi-Jun Chang, Wenzheng Li, and Seth Pettie. 2020. Distributed $(\Delta+1)$-Coloring via Ultrafast Graph Shattering. 49, 3 (2020), 497–539.

[13] Artur Czumaj, Peter Davies, and Merav Parter. 2020. Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space. In *Proceedings of the 32nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 175–185.

[14] Artur Czumaj, Peter Davies, and Merav Parter. 2020. Simple, Deterministic, Constant-Round Coloring in the Congested Clique. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC)*. 309–318.

[15] Artur Czumaj, Peter Davies, and Merav Parter. 2021. Component Stability in Low-Space Massively Parallel Computation. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*.

[16] Anindya De, Omid Etesami, Luca Trevisan, and Madhur Tulsiani. 2010. Improved Pseudorandom Generators for Depth 2 Circuits. In *Proceedings of the 13th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and of the 10th the International Conference on Randomization and Computation (RANDOM)*. 504–517.

[17] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. 2018. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*. 129–138.

[18] Mohsen Ghaffari, Christoph Grunau, and Ce Jin. 2020. Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond. In *Proceedings of the 34th International Symposium on Distributed Computing (DISC)*, Vol. 179. 34:1–34:18.

[19] Mohsen Ghaffari, Ce Jin, and Daan Nilis. 2020. A Massively Parallel Algorithm for Minimum Weight Vertex Cover. In *Proceedings of the 32nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 259–268.

[20] Mohsen Ghaffari and Fabian Kuhn. 2020. Deterministic Distributed Vertex Coloring: Simpler, Faster, and without Network Decomposition. *CoRR* abs/2011.04511 (2020).

[21] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. 2019. Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*. 1650–1663.

[22] Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. 2020. Faster Algorithms for Edge Connectivity via Random 2-Out Contractions. In *Proceedings of the 2020 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1260–1279.

[23] Mohsen Ghaffari and Jara Uitto. 2019. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1636–1653.

[24] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, Searching, and Simulation in the MapReduce Framework. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC)*. 374–383.

[25] Parikshit Gopalan, Raghu Meka, Omer Reingold, Luca Trevisan, and Salil P. Vadhan. 2012. Better Pseudorandom Generators from Milder Pseudorandom Restrictions. In *Proceedings of the 53rd IEEE Symposium on Foundations of Computer Science (FOCS)*. 120–129.

[26] David G. Harris, Johannes Schneider, and Hsin-Hao Su. 2016. Distributed $(\Delta+1)$-Coloring in Sublogarithmic Rounds. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*. 465–478.

[27] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A Model of Computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 938–948.

[28] Kishore Kothapalli, Shreyas Pai, and Sriram V. Pemmaraju. 2020. Sample-And-Gather: Fast Ruling Set Algorithms in the Low-Memory MPC Model. In *Proceedings of the 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. 28:1–28:18.

[29] Fabian Kuhn. 2009. Weak Graph Colorings: Distributed Algorithms and Applications. In *Proceedings of the 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 138–144.

[30] Nathan Linial. 1992. Locality in Distributed Graph Algorithms. 21, 1 (Feb. 1992), 193–201.

[31] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. 2005. Minimum-Weight Spanning Tree Construction in $O(\log\log n)$ Communication Rounds. 35, 1 (2005), 120–131.

[32] Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. 2003. MST Construction in $O(\log\log n)$ Communication Rounds. In *Proceedings of the 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 94–100.

[33] Michael Luby. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. 15, 4 (1986), 1036–1053.

[34] Michael Luby. 1993. Removing Randomness in Parallel Computation without a Processor Penalty. *Journal of Computer and System Sciences* 47, 2 (1993), 250–286.

[35] Raghu Meka, Omer Reingold, and Avishay Tal. 2019. Pseudorandom generators for width-3 branching programs. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*. 626–637.

[36] Noam Nisan and Avi Wigderson. 1988. Hardness vs. Randomness (Extended Abstract). In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (FOCS)*. 2–11.

[37] Krzysztof Onak. 2018. Round Compression for Parallel Graph Algorithms in Strongly Sublinear Space. *CoRR abs/1807.08745* (2018). arXiv:1807.08745

[38] Merav Parter. 2018. $(\Delta+1)$ Coloring in the Congested Clique Model. In *Proceedings of the 45th Annual International Colloquium on Automata, Languages and Programming (ICALP)*. 160:1–160:14.

[39] Merav Parter and Hsin-Hao Su. 2018. Randomized $(\Delta+1)$-Coloring in $O(\log^* \Delta)$ Congested Clique Rounds. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*. 39:1–39:18.

[40] Václav Rozhoň and Mohsen Ghaffari. 2020. Polylogarithmic-time Deterministic Network Decomposition and Distributed Derandomization. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*. 350–363.

[41] Salil P. Vadhan. 2012. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science* 7, 1-3 (2012), 1–336.