Improved Deterministic Connectivity in Massively Parallel Computation

Manuela Fischer
ETH Zürich, Switzerland

Jeff Giliberti
ETH Zürich, Switzerland

Christoph Grunau
ETH Zürich, Switzerland

Abstract

A long line of research about connectivity in the Massively Parallel Computation model has culminated in the seminal works of Andoni et al. [FOCS'18] and Behnezhad et al. [FOCS'19]. They provide a randomized algorithm for low-space MPC with conjectured to be optimal round complexity $O(\log D + \log\log_{\frac{m}{n}} n)$ and $O(m)$ space, for graphs on n vertices with m edges and diameter D. Surprisingly, a recent result of Coy and Czumaj [STOC'22] shows how to achieve the same deterministically. Unfortunately, however, their algorithm suffers from large local computation time.

We present a deterministic connectivity algorithm that matches all the parameters of the randomized algorithm and, in addition, significantly reduces the local computation time to nearly linear.

Our derandomization method is based on reducing the amount of randomness needed to allow for a simpler efficient search. While similar randomness reduction approaches have been used before, our result is not only strikingly simpler, but it is the first to have efficient local computation. This is why we believe it to serve as a starting point for the systematic development of computation-efficient derandomization approaches in low-memory MPC.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms; Theory of computation → MapReduce algorithms

Keywords and phrases Massively Parallel Computation, MPC, MapReduce, Deterministic Algorithms, Connectivity, Hitting Set, Maximum Matching, Derandomization

Digital Object Identifier 10.4230/LIPIcs.DISC.2022.22

Funding *Christoph Grunau*: Supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 853109).

1 Introduction

Due to the ever-increasing amount of data available, memory has grown to become a major bottleneck, which makes many traditional graph algorithms inefficient or even inapplicable. To overcome this obstacle, inspired by the MapReduce paradigm [17], several computation frameworks for large-scale graph processing across multiple machines have been proposed. The Massively Parallel Computation (MPC) model is a clean, theoretical abstraction of these frameworks and thus serves as a basis for the systematic study of memory-restricted distributed algorithms. Introduced by Karloff et al. [27] and Feldman et al. [19] in 2010, it was later refined in a sequence of works and has become tremendously popular over the past decade.

© Manuela Fischer, Jeff Giliberti, and Christoph Grunau;
licensed under Creative Commons License CC-BY 4.0
36th International Symposium on Distributed Computing (DISC 2022).
Editor: Christian Scheideler; Article No. 22; pp. 22:1–22:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

MPC Model

In the MPC model, the distributed network consists of $\mathbf{M}$ machines, having local memory $\mathbf{S}$ 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 $\mathbf{S}$ 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 of an algorithm are its *global memory* usage – i.e., the number of machines times the memory per machine required – as well as the *total computation* performed by machines to run the algorithm, i.e., the (asymptotic) sum of the local computation performed by each machine.

We 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 $\mathbf{S} = O(n^{\delta})$ each, for some constant $0 < \delta < 1$, so that the global space is $\mathbf{S}_{Global} = \Omega(n + m)$.

Graph Algorithms and Connectivity

In this model, fundamental graph 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, vertex cover, coloring, and many more (see, e.g., [6, 20, 12, 15, 7, 23, 16]).

One particularly important (and arguably the most central) graph problem that has received increasing attention over the past few years is the one of connectivity. This is not only a problem of independent interest, but it serves as a subroutine for many algorithms.

▶ **Definition 1** (Connectivity Problem). *Let $G = (V, E)$ be an undirected graph. The goal is to compute a function $cc\colon V \to \mathbb{N}$ 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, 29, 7, 33, 10, 7] on this problem culminated in a randomized algorithm by Behnezhad et al. [7] that finds all connected components of a graph with diameter D in $O(\log D + \log\log_{\frac{m}{n}} n)$ rounds.

In a very recent breakthrough, Coy and Czumaj [12] obtained the same round complexity with a deterministic algorithm. Their derandomization approach, however, comes at a cost of heavy local computation, which makes it impractical for large-scale applications.

Deterministic Algorithms and Derandomization

While the problem of connectivity is of independent interest, it is instructive to view the above results in a broader context of deterministic algorithms and derandomization.

Notably, 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 (low-memory) MPC model restricted to deterministic algorithms [5, 16, 13, 15, 12]. They demonstrate that several graph problems can be solved deterministically with (asymptotic) complexity bounds that are comparable to those of the randomized algorithms. The main ingredients of these results are derandomization methods specifically tailored to the low-memory MPC model: they are designed to cope with the limited memory per machine while exploiting the power of *local computation* and *all-to-all communication* in this setting.

This quest for efficient derandomization techniques has become one of the main problems of the area. Unfortunately, current derandomization frameworks suffer from long local running time (e.g., large polynomial or even exponential in n^δ). In fact, as noted in [16], allowing heavy local computation might provide an advantage in the context of distributed and parallel derandomization. However, especially in performance-oriented scenarios, local computation may quickly become a critical parameter. It thus emerges as a natural direction to study deterministic algorithms whose total computation matches that of their randomized counterparts.

1.1 Our Contribution

We address this issue by presenting the first computation-efficient deterministic algorithm for the problem of graph connectivity in the strongly sublinear memory regime of MPC.

▶ **Theorem 2** (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.*

The total computation of our algorithm significantly improves over the poly(n)-bound of Coy and Czumaj [12], with no loss in the round complexity. In fact, our algorithm matches even the state-of-the-art randomized algorithm [7] in all parameters up to a polylogarithmic factor in the local running time.

While the connectivity algorithm is of independent interest, our result provides a number of other qualitative advantages. For instance, our analysis relies only on pairwise independence as opposed to the almost $O(\log n)$-wise independence of [12]. Moreover, to the best of our knowledge, our result is the first that uses the framework of limited independence for derandomization 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 the more systematic development of computation-efficient derandomization.

1.2 Randomized Connectivity Algorithms in a Nutshell

We present the intuition of the randomized connectivity algorithms by Andoni et al. [3] and Behnezhad et al. [7]. For a broader overview of connectivity algorithms, see Section 1.4.

Vertex Contraction

The main idea behind connectivity algorithms working in $\tilde{O}(\log D)$ rounds is to repeatedly perform *vertex contractions* [3]. Contracting (often also called *relabeling*) 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 first appoint a random subset of the vertices as *leaders* (by letting each vertex independently with probability $\frac{1}{2}$ become a leader), 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 [31] – 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. [7])

To further improve the round complexity, Behnezhad et al. [7] 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 budgets' 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. [7])

To allow each vertex to start with a poly $\log n$ budget, a randomized constant-round algorithm (see [7, 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 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 thus allows to contract $\Omega(n)$ vertices as desired.

1.3 Deterministic Connectivity: Comparison with the State-of-the-Art

We next present the main ideas behind the recent deterministic connectivity algorithm of Coy and Czumaj [12].

Coy and Czumaj [12] identify and extract the only two sources of randomization from the algorithms of [3, 7], namely matching and hitting set. On the one hand, as outlined in Section 1.2, 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 variant of set cover, which we refer to as hitting set with all sets of the same size (see Definition 9 for a precise definition).

As these are the only steps involving randomness (as outlined in Section 1.2), the (efficient) derandomization of these two constant-round key algorithmic primitives immediately leads to an (efficient) deterministic connectivity algorithm. In fact, their derandomization together with the $O(\log D + \log\log n)$ randomized algorithm due to Behnezhad et al. [7] results in the state-of-the-art deterministic connectivity algorithm in low-memory MPC [12].

Interestingly, because of the conditional lower bound framework (conditioned on the widely believed 1-vs-2-cycles conjecture for low-space MPC algorithms) due to Ghaffari et al. [22] and its extension to the deterministic setting due to Czumaj et al. [14], the two underlying problems of matching and hitting set do not admit any *component-stable*[1] constant-round deterministic algorithm. Hence, the authors in [12] incorporate in their work derandomization techniques that are highly non-component-stable.

While their adopted derandomization framework is well-established, its efficient implementation for obtaining a deterministic connectivity algorithm on an MPC with low local space and optimal global space requires to overcome several challenges. Although the algorithm from [12] achieves optimal space guarantees, the computation is suboptimal for both derandomization steps. We refine these to obtain a more efficient deterministic connectivity algorithm, as explained next.

Maximum Matching

In [12], 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, with each evaluation taking poly $\log n$ time, the resulting total computation is $\tilde{O}(n^3)$. We reduce the *seed length*, i.e., the total 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 [12] 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 implementation of the 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 [9, 21, 39] solve the hitting set problem deterministically in the context of graph spanners in CONGEST and CONGESTED-CLIQUE using similar derandomization techniques. 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 matching and hitting set algorithms is significantly simplified as we can perform a simple brute force search instead of using the method of conditional expectation.

[1] The notion of component-stability intuitively refers to the property that the choices of any vertex over the course of the algorithm are affected only by vertices in its same connected component.

1.4 Further Related Work

The connectivity problem in low-memory MPC was studied by Andoni et al. [3] who presented an $O(\log D \cdot \log\log_{\frac{m}{n}} n)$ randomized algorithm, which improves upon the classic $O(\log n)$ bound derived from earlier works in the PRAM model. Concurrently, for graphs with large spectral gap λ, i.e., $\Omega(1/\text{poly}\log(n))$, the bound was improved in [4] developing a randomized $O(\log\log n + \log(1/\lambda))$ algorithm. Then, a near-optimal parallel randomized algorithm that in $O(\log D + \log\log_{\frac{m}{n}} n)$ rounds determines all connected components was developed by Behnezhad et al. [7]. Subsequently, Liu et al. [33] extended the same result to the arbitrary CRCW PRAM model, which is less computationally powerful than MPC, achieving such result with good probability[2]. Moreover, by developing a method that converts randomized PRAM algorithms to highly randomness-efficient MPC algorithms, Charikar et al. [10] achieved a super-polynomial saving in the randomness used in [7], showing that $(\log n)^{O(\log D + \log\log_{m/n} n)}$ random bits suffice (with good probability), provided that the global space is $\Omega((n+m)\cdot n^{\delta})$. The current deterministic state-of-the-art algorithm for connectivity is due to Coy and Czumaj [12] who obtained a deterministic $O(\log D + \log\log_{\frac{m}{n}} n)$ algorithm with asymptotically optimal space.

Finally, let us note that the connectivity problem has been studied in other regimes as well. Lattanzi et al. [30] gave a constant-round MPC connectivity algorithm in the superlinear regime, i.e., each machine has local space $\Omega(n^{1+\delta})$. By well-known connections between linear memory MPC and the CONGESTED-CLIQUE model, [26] yields a $O(1)$-rounds randomized connectivity MPC algorithm with optimal global space. Then, Nowicki [38] showed that the same problem can be solved deterministically in $O(1)$ MPC rounds with the same memory guarantees.

On the hardness side, one of the most outstanding problems for low-space MPCcomplexity is the problem of distinguishing whether an input graph is an n-vertex cycle or consists of two $\frac{n}{2}$-vertex cycles (see, e.g., [41, 37] for more information). Based on the conjectured $\Omega(\log n)$ low-memory MPC round-complexity lower bound for the 1-vs-2-cycles problem, Behnezhad et al. [7] show an $\Omega(\log D)$ lower bound for computing connected components in general graphs with diameter $D \geq \log^{1+\Omega(1)} n$. Coy and Czumaj in [12] extend the same conditional lower bound to the entire spectrum of D proving that no connectivity algorithm can achieve $o(\log D)$ MPC round complexity.

2 Preliminaries

2.1 Primitives in Low-Space MPC

There are a number of 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 stricly sublinear space per machine and linear global space. We will use the following lemma to refer to them:

▶ **Lemma 3** ([25, 24]). *For any positive constant δ, sorting, filtering, prefix sum, predecessor, duplicate removal, and colored summation task* [3] *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.*

[2] with success probability at least $1 - 1/\text{poly}((m\log n)/n)$

[3] Given a sequence of n pairs of numbers $\langle color_i, x_i\rangle, i \in [n]$, with $C = \{color_i \mid i \in [n]\}$, compute $S_c = \sum_{i:color_i=c} x_i$ for all $c \in C$. Note that this problem can be easily solved by a constant sequence of map, shuffle, and reduce steps with $\langle color_i, x_i\rangle$ as key-value pairs.

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 blocks of 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.

2.2 Derandomization Framework

In this section, we give an overview of the common derandomization techniques used in all-to-all communication models [9, 34] with a focus on deterministic algorithms in the strongly sublinear memory regime of MPC. A systematic introduction to the framework of limited independence can be found for example in [40, 36, 2, 35, 8, 42].

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 4** (k-wise independence). *Let $N, k, \ell \in \mathbb{N}$ with $k \leq N$. A family of hash functions $\mathcal{H} = \{h : [N] \to \{0,1\}^\ell\}$ is k-wise independent if for all $I \subseteq \{1, \dots, n\}$ with $|I| \leq k$, the random variables $X_i := h(i)$ 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. Random variables sampled from a pairwise independent family of hash functions are called pairwise independent random variables.*

The following is a well-known result about the existence and construction of such hash families:

▶ **Lemma 5** ([1, 11, 18]). *For every $N, \ell, k \in \mathbb{N}$, there is a family of k-wise independent hash functions $\mathcal{H} = \{h : [N] \to \{0,1\}^\ell\}$ 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 $\mathrm{poly}(\ell, \log N)$ time.*

If there is a randomized algorithm, over the choice of a random hash function, that gives good results in expectation, one can derandomize it by finding the right choice of (random) bits. To achieve that, if the seed length is small, one can brute force it without incurring an overhead in the global space.

In previous works this was usually not possible due to a seed length depending on n of $\Omega(\log n)$ bits, which results in hash families of size larger than the space $\mathbf{S}$ of a single machine. Instead, they used the method of conditional expectation or probabilities. 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 (or probability). This can be done with global coordination. We refer the interested reader for more details of the method of conditional expectation to [12, Section 2.5, Appendix A].

2.3 Reducing The Seed Length via Coloring

The following technique plays a central role for reducing the seed length of randomized processes solving *local* graph problems. As showed in [5, 16, 15], if the outcome of a vertex depends only on the random choices of its neighbors, then k-wise independence among random variables of *adjacent* vertices is sufficient. Whenever this is the case, we can find a mapping from vertex IDs to shorter names (colors) such that adjacent vertices are assigned different names. Linial gave a 1-round distributed coloring algorithm with $O(\Delta^2 \log(n))$ colors [32]. We here adapt a more explicit 1-round distributed coloring algorithm with $O(\Delta^2 \log^2_\Delta(n))$ colors by Kuhn [28] to the MPC model, which leads to the following lemma:

▶ **Lemma 6.** *Let $G = (V, E)$ be a graph of maximum degree $\Delta \leq n^\delta$. There exists a deterministic algorithm which computes an $O(\Delta^2 \log^2_\Delta n)$ coloring of G in $O(1)$ MPC rounds using $O(n^\delta)$ local space, $O(n \cdot \mathrm{poly}(\Delta))$ global space, and $\tilde{O}(n \cdot \mathrm{poly}(\Delta))$ 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 $10\Delta \log_\Delta(n) \leq p \leq 20\Delta \log_\Delta(n)$. It is well known that such a prime always exist. Moreover, let $d = \lceil \log_\Delta(n) \rceil$. 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 $p^2 = O(\Delta^2 \log^2_\Delta n)$ such tuples.

Let $C_i = \{(x, f_i(x)) \colon x \in \mathbb{F}_p\}$. 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 3 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 $\mathrm{poly}(\log n, \Delta)$. Computing the color $c(i)$ boils down to $O(\Delta \cdot p^2) = \mathrm{poly}(\log n, \Delta)$ polynomial evaluations. Hence, the total computation time is $\tilde{O}(n \cdot \mathrm{poly}(\Delta))$, as desired. ◀

3 Constant Approximation of Maximum Matching

The first algorithmic step for the derandomization of the connectivity algorithm from [7] consists of solving approximate maximum matching in graphs of maximum degree two. Coy and Czumaj proved the following theorem:

▶ **Theorem 7** (Theorem 4.2 of [12]). *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 $\mathbf{S}_{Global} = O(n)$.*

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

▶ **Theorem 8.** *There exists an algorithm with the same properties as those in Theorem 7 using $\tilde{O}(n)$ total computation.*

We start by reviewing the main idea used in the algorithm proving Theorem 7.

Randomized Algorithm. The algorithm of Theorem 7 is based on derandomizing the following simple random process. Let $\{X_e \colon e \in E\}$ 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 of the random variable. Hence, they can be specified by a seed of length $2 \log n + O(1)$ by Lemma 5. As explained in [12], this allows to use 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 variables are independent. This motivates the following approach. First, we assign to each edge e a color $c(e)$ from the set $\{1, 2, \ldots, C\}$ for $C = O(\log^2 n)$ by applying Lemma 6 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. Now we are ready to present our deterministic MPC algorithm that proves Theorem 8. 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 5. 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', where $h(i)$ denotes the length-2 bit sequence assigned to i by the corresponding integer in $\{0, \ldots, 3\}$.

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 $\mathcal{M}(h^*) \geq m/8$. We efficiently compute $|\mathcal{M}(h)|$ for every $h \in \mathcal{H}$ and choose one *good* hash function that yields a matching of size at least $m/8$.

First, we efficiently compute the coloring c using Lemma 6. Next, we compute the approximate maximum matching in G by derandomizing the sampling approach analyzed above. Since the size of our family of pairwise independent hash functions 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 $|\mathcal{H}| \cdot \text{poly}(\log n) \cdot O(n) = \tilde{O}(n)$.

4 Computation-Efficient Derandomization of Hitting Set

In this section, we give a deterministic MPC algorithm for the following hitting set variant defined in [12]:

▶ **Definition 9** (*Hitting Set for Leader Election*). *Let $S_1, \ldots, S_n$ 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 [12] gave an algorithm with the same parameters as those of the random sampling approach in [7], except that they need large $\operatorname{poly}(n)$ computation.

▶ **Theorem 10** (Theorem 5.6 of [12]). *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 $\mathbf{S}_{Global} = O(nb)$, and total computation $\operatorname{poly}(n)$.*

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 11.** *There exists an algorithm with the same properties as those in Theorem 10 with two differences. The total computation reduces to $O(n \cdot \operatorname{poly}(b))$ and the global space increases to $O(n \cdot \operatorname{poly}(b))$.*

We will show in Section 5 that the algorithm from Theorem 11 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 $b \geq \log^{10}(n)$ 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 [12] use so-called k-wise ε-approximately independent random variables for $k = 15\log_b(n)$ and $\varepsilon = n^{-6}$. In particular, the starting point of their algorithm is the following theorem.

▶ **Theorem 12** ([12, Theorem 5.2]). *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 $X_1, X_2, \ldots, X_n$ 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 hold with probability at least $1 - 9n^{-3}$:*

(1) $\sum_{j \in S_i} X_j > 0$ *for every* $1 \leq i \leq n$, *and*

(2) $\sum_{i=1}^{n} X_i \leq 2nb^{-\frac{1}{5}}$.

Next, we explain our randomized approach, which bears some similarities with that of 12, and proceed to the reduction of 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, this implies that a set is bad, i.e., no element is sampled from it, with probability at most $\frac{1}{b^{4/5}}$. This directly follows from the following lemma:

▶ **Lemma 13.** *Let $X_1, \ldots, X_n$ be pairwise independent random variables taking values in $[0, 1]$. Let $X = X_1 + \ldots + X_n$ and $\mu = \mathbb{E}[X]$. Then $\mathbb{V}\mathrm{ar}[X] = \sum_{i=1}^{n} \mathbb{V}\mathrm{ar}[X_i] \leq \mu$ and*

$$\Pr\left[|X - \mu| \geq \mu\right] \leq \frac{\sum_{i=1}^{n} \mathbb{V}\mathrm{ar}[X_i]}{\mu^2} \leq \frac{1}{\mu}.$$

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 simple 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 $\mathrm{poly}(b)$ 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 a lot of 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 $\Delta_{conflict}$ 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 $C = O(\Delta_{conflict}^2 \log^2(n)) = O(b^6 \log^2 n)$ 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.

We are now ready to present our randomized process that produces a hitting set with the desired properties. Let $\{X_c \colon c \in [C]\}$ be a family of pairwise independent random variables with $X_c = 1$ with probability $p = b^{-1/5}$ and $X_c = 0$ otherwise. For simplicity, we assume that $1/p$ is a power of 2, i.e., there exists $\ell \in \mathbb{N}$ with $2^\ell = b^{1/5}$. According to Lemma 5, we can generate these random 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 analysis and discussion above, $\mathcal{L}$ is a hitting set of expected size $O(nb^{-1/5})$.

MPC Algorithm

It remains to discuss the MPC implementation, which will prove Theorem 11. 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 $O(n\text{poly}(b))$ 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 us to compute for each element in 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 $S = \{e_1, e_2, \ldots, e_b\}$ creates $\binom{b}{2}$ entries $\{\{e_i, e_j\} \colon i \neq j \in [b]\}$. This can easily be done with 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 3. Hence, we can efficiently generate all these edges in parallel. Afterwards, we use the duplicate removal procedure of Lemma 3 to remove duplicate edges.

As $G_{conflict}$ has maximum degree b^3, we can use Lemma 6 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 $\mathcal{H} = \{h \colon [C] \mapsto \{0,1\}^\ell\}$ 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 poly$(\ell, \log C) = \text{poly}(\log b)$ time. Lemma 5 guarantees the existence of such a family.

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 $\{0, \ldots, \ell - 1\}$. 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 3, 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 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 hitting set $\mathcal{L}_{h^*}$ of smallest size. As remarked above, $\mathcal{L}_{h^*}$ has size $O(nb^{-1/5})$, which finishes the proof.

5 Connectivity Algorithm

In this section, we discuss the necessary changes to the randomized connectivity algorithm of Behnezhad et al. [7] and its analysis in order to prove the main result of this paper.

The deterministic approximate matching from Section 3 is used to replace steps 5 and 6 of Algorithm 2 of [7]. The same modification was already done by [12] 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. All the steps of the modified deterministic algorithm can be implemented by invoking the primitives of Lemma 3 $O(1)$ times, which in particular ensures that the algorithm can be implemented with total computation $\tilde{O}(m)$. Hence, we can from now on assume that $m \geq n \log^C n$, for a given constant C. It remains to prove that Algorithm 1 of [7] 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 14** ([12, Lemma 6.3]). *Let S_i denote the set of saturated vertices at level i after Step 2 of the* RelabelIntraLevel *routine in [7], 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 [7] remains correct with the same asymptotic local and global space complexity.

We extend the above lemma to make it work with the deterministic hitting set from Section 4 by proving the following slight modification of it. The main technical challenge will be to ensure that our deterministic hitting set algorithm, which adds a polynomial factor (in b) increase in the memory and computation required, can still be run in parallel with linear global space and total computation.

▶ **Lemma 15.** *Let $c \geq 3$ be the smallest integer such that both the global space and the total computation required by the algorithm from Theorem 11 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 14 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 $n\beta_i^c$ 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 $\left(\frac{m}{n}\right)^{1/2}$ assigned to each vertex with $\left(\frac{m}{n}\right)^{1/2c}$ in Algorithm 1 of [7]. Then, the connectivity algorithm of [7] 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 [7] do not affect its correctness nor its bounds on local and global memory. As in [12], we need to prove the following three key properties:

(1) for any vertex v, the value of $\ell(v)$ never exceeds $O(\log \log_{m/n} n)$ (cf. [7, Lemma 15]),
(2) the global space used is $O(\mathbf{S}_{Global})$ (cf. [7, Lemma 17]),
(3) the sum of the squares of the budgets does not exceed $O(\mathbf{S}_{Global})$ (cf. [7, Lemma 21]).

(1) Recall that the budget of each vertex is increased as $\beta_{i+1} := \beta_i \cdot (\min\{\beta_i, n^{\varepsilon}\})^{\frac{\gamma}{4c}}$ and that $\beta_0 = \left(\frac{m}{n}\right)^{1/2c}$. 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.

(2) 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 [7], it is shown that the total sum of the budget increases over the course of 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 [12], we first show that $\beta_{i+1}^c \cdot n_{i+1} \leq \beta_i^c \cdot n_i$. We have that 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:

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

It then follows that

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

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 11. This yields

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

(3) Follows by the same line of reasoning as in property (2).

By the choice of c, repeating the same calculations as in property (b) 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 3 implies that all the other steps of the algorithm can be implemented with total computation $\tilde{O}(m)$. ◀

We are now ready to prove our main result.

Proof of Theorem 2. We apply Lemma 15 using our Hitting Set for Leader Election algorithm from Theorem 11 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 [12] combined with Lemma 15 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 [12, 7]. Finally, as noted in [12], our extension of Lemma 15 in [7] 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. ◀

References

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. doi:10.1016/0196-6774(86)90019-2.

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. doi:10.1109/FOCS.2018.00070.

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, pages 461–470, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331596.

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, pages 243–252, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3382734.3404504.

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

7 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. doi:10.1109/FOCS.2019.00095.

8 J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. doi:10.1016/0022-0000(79)90044-8.

9 Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *Distributed Computing*, 33(3):349–366, June 2020. doi:10.1007/s00446-020-00376-1.

10 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, pages 431–433, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467951.

11 Benny Chor and Oded Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5(1):96–106, 1989. doi:10.1016/0885-064X(89)90015-0.

12 Sam Coy and Artur Czumaj. Deterministic massively parallel connectivity. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 162–175, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520055.

13 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, pages 309–318, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3382734.3405751.

14 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, pages 481–491, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467903.

15 Artur Czumaj, Peter Davies, and Merav Parter. Graph sparsification for derandomizing massively parallel computation with low space. *ACM Trans. Algorithms*, 17(2), May 2021. doi:10.1145/3451992.

16 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, pages 469–479, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467937.

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

18 Guy Even, Oded Goldreich, Michael Luby, Noam Nisan, and Boban Veličković. Efficient approximation of product distributions. *Random Structures & Algorithms*, 13(1):1–16, 1998. doi:10.1002/(SICI)1098-2418(199808)13:1<1::AID-RSA1>3.0.CO;2-W.

19 Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Cliff Stein, and Zoya Svitkina. On distributing symmetric streaming computations. *ACM Trans. Algorithms*, 6(4), September 2010. doi:10.1145/1824777.1824786.

20 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. doi:10.4230/LIPIcs.DISC.2020.34.

21 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 für Informatik. doi:10.4230/LIPIcs.DISC.2018.29.

22 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. doi:10.1109/FOCS.2019.00097.

23 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. doi:10.1137/1.9781611975482.99.

24 Michael T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999. doi:10.1137/S0097539795294141.

25 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. doi:10.1007/978-3-642-25591-5_39.

26 Tomasz Jurdziński and Krzysztof Nowicki. MST in O(1) Rounds of Congested Clique. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2620–2632, 2018. doi:10.1137/1.9781611975031.167.

27 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. doi:10.1137/1.9781611973075.76.

28 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, pages 138–144, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1583991.1584032.

29 Jakub Lacki, Vahab S. Mirrokni, and Michal Wlodarczyk. Connected components at scale via local contractions. *CoRR*, abs/1807.10727, 2018. arXiv:1807.10727.

30 Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: A method for solving graph problems in mapreduce. In *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 85–94, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1989493.1989505.

31 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, pages 295–296, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1835698.1835772.

32 Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

33 Sixue Cliff Liu, Robert E. Tarjan, and Peilin Zhong. *Connected Components on a PRAM in Log Diameter Time*, pages 359–369. Association for Computing Machinery, New York, NY, USA, 2020. doi:10.1145/3350755.3400249.

34 Michael Luby. Removing randomness in parallel computation without a processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993. doi:10.1016/0022-0000(93)90033-S.

35 Michael Luby and Avi Wigderson. Pairwise independence and derandomization. *Foundations and Trends in Theoretical Computer Science*, 1(4):237–301, 2006. doi:10.1561/0400000009.

36 Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

37 Danupon Nanongkai and Michele Scquizzato. Equivalence classes and conditional hardness in massively parallel computations. *Distributed Computing*, 35(2):165–183, 2022. doi:10.1007/s00446-021-00418-2.

38 Krzysztof Nowicki. A deterministic algorithm for the mst problem in constant rounds of congested clique. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1154–1165, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451136.

39 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 für Informatik. doi:10.4230/LIPIcs.DISC.2018.40.

40 Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988. doi:10.1016/0022-0000(88)90003-7.

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

42 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. doi:10.1109/SFCS.1979.26.