Ghaffari, Mohsen; Kuhn, Fabian; Uitto, Jara

Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds

Published in:
Proceedings - 2019 IEEE 60th Annual Symposium on Foundations of Computer Science, FOCS 2019

DOI:
10.1109/FOCS.2019.00097

Published: 01/11/2019

Document Version
Peer reviewed version

Please cite the original version:
Ghaffari, M., Kuhn, F., & Uitto, J. (2019). Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. In *Proceedings - 2019 IEEE 60th Annual Symposium on Foundations of Computer Science, FOCS 2019* (pp. 1650-1663). [8948686] IEEE. https://doi.org/10.1109/FOCS.2019.00097

Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds

Mohsen Ghaffari
ETH Zurich
ghaffari@inf.ethz.ch

Fabian Kuhn
University of Freiburg
kuhn@cs.uni-freiburg.de

Jara Uitto
ETH Zurich & U. of Freiburg
jara.uitto@inf.ethz.ch

Abstract

We present the first conditional hardness results for massively parallel algorithms for some central graph problems including (approximating) maximum matching, vertex cover, maximal independent set, and coloring. In some cases, these hardness results match or get close to the state of the art algorithms. Our hardness results are conditioned on a widely believed conjecture in massively parallel computation about the complexity of the connectivity problem. We also note that it is known that an unconditional variant of such hardness results might be somewhat out of reach for now, as it would lead to considerably improved circuit complexity lower bounds and would concretely imply that $\mathsf{NC}_1 \subsetneq \mathsf{P}$. We obtain our conditional hardness result via a general method that lifts unconditional lower bounds from the well-studied LOCAL model of distributed computing to the massively parallel computation setting.

1 Introduction and Related Work

We present conditional hardness results for massively parallel algorithms for some central graph problems. Our main technical contribution is a general method that lifts unconditional lower bounds from the more classic LOCAL model of distributed computing to the massively parallel computation setting. We start with a review of the massively parallel computation setting and the state of the art for graph problems, and we then present our results.

1.1 Massively Parallel Computation: Model and State of the Art

Massively parallel computation and the emergence of several popular practical frameworks for it — such as MapReduce [DG04], Hadoop [Whi12], Spark [ZCF+10], and Dryad [IBY+07] — has been one of the impactful developments of the past 10 to 15 years in computer science. There is also a growing interest from theoretical computer science to build the theoretical foundations of computation in these settings. This growth is fueled by the understanding that (such coarse-grained notions of) parallelism will be an essential necessity in the future of computation, as the data sets are growing faster than the capacity of single processors. Karloff et al. [KSV10] introduced a theoretical model, by now referred to as the *Massively Parallel Computation* (MPC) model, as a simple abstraction that captures essential aspects of parallelism in these settings. This model and algorithmic developments in it have been receiving increasingly more attention over the past few years [KSV10, GSZ11, LMSV11, BKS13, ANOY14, BKS14, HP15, AG15, RVW16, IMS17, CLM+18, Ass17, ABB+19, GGK+18, HLL18, BFU18, ASW18, BEG+18, ASS+18, Ona18, GKMS18, GU19, BHH19, ACK19].

The MPC model: We have a number of machines, each with S bits of memory, that can communicate with each other in synchronous rounds. Per round, each machine can send $O(S)$ bits to the other machines in total, it can receive $O(S)$ bits from other machines, and it can also perform some local computation, ideally of time at most $\text{poly}(S)$. We assume that the number of machines M is at least at large as N/S, where N is the size of the input, but ideally not much larger. For graph problems, the input size is $N = O(m + n)$, where m denotes the number of edges and n denotes the number of vertices. We sometimes refer to S as local memory of the system while $M \cdot S$ is the global memory of the system. The common interpretation is that, when trying to apply such algorithms to a setting, the parameter S will be dictated to us by what is physically available, and we then should achieve the best round complexity for that local memory. Hence, the main objective is to obtain MPC algorithms that have a small *round complexity* for as small as possible local *memory*. Ideally, the algorithm should be using global memory that is almost equal to the input size; though we note that minimizing the global memory is sometimes treated as secondary target.

State of the Art for Graph Problems: Several graph problems have been studied in the MPC model. We briefly review the state of the art for some of the central graph problems.

One of most central problems is connectivity, i.e., identifying the connected components of a graph. Karloff et al. [KSV10] showed that this problem can be solved in $O(\log n)$ rounds using machines with local memory $S = n^\alpha$ for any constant $\alpha \in (0, 1)$, using Boruvka-style algorithms. Whether this round complexity can be improved remains unknown but there is strong evidence to suggest that this bound is the best possible: Beame et al. [BKS13] showed that a natural class of algorithms cannot run in $o(\log n)$ rounds. The problem is even open for the seemingly trivial case where the algorithm should distinguish whether the input is one n-node cycle or two $n/2$-node cycles. The problem has come to be believed to be inherently hard by now, and is conjectured to require $\Omega(\log n)$ rounds. Proving the $\Omega(\log n)$ lower bound might be hard, however. In fact, as argued by Roughgarden et al. [RVW16], proving any super-constant round complexity MPC lower bound for

any problem in P and for $S = n^\alpha$ is presumably a difficult task, as it would imply strong circuit complexity lower bounds and particularly show that $NC_1 \subsetneq \mathsf{P}$. That is, it would show that there is a problem in P that does not admit a logspace-uniform circuit family of fan-in 2 with $\text{poly}(n)$ gates and $O(\log n)$ depth. Recently, Andoni et al. [ASS+18] showed that if each component of the input has diameter at most D, then the round complexity can be improved to $O(\log D \cdot \log\log_{MS/n} n)$. For much larger local memory, there are faster algorithms. Lattanzi et al. [LMSV11] gave an $O(1)$-round algorithm for the strongly superlinear memory regime, where $S = n^{1+\epsilon}$ for a constant $\epsilon > 0$. An $O(1)$-round algorithm for near-linear local memory of $S = O(n \log^3 n)$ is implicit in the sketching work of Ahn, Guha, and McGregor for the streaming setting [AGM12a, AGM12b], and an $O(1)$-round algorithm for linear local memory $S = O(n)$ is implied by the results of Jurdinski and Nowicki [JN18].

Another central graph problem in the recent developments has been maximum matching and its approximation, as well as other closely related problems such as vertex cover and maximal independent set (MIS). The filtering method of Lattanzi et al. [LMSV11] gives an $O(1)$ round algorithm for maximal matching, and thus a 2-approximation of maximum matching, for strongly superlinear-local memory $S = n^{1+\Omega(1)}$. However, the task is significantly harder for lower memory regimes. A recent breakthrough of Czumaj et al. [CLM+18] gave an $O((\log\log n)^2)$-round algorithm for $(1+\varepsilon)$-approximation of maximum matching when $S = \tilde{O}(n)$, for any constant $\varepsilon > 0$. The round complexity was improved later to $O(\log\log n)$ by Assadi et al. [ABB+19] and Ghaffari et al. [GGK+18]. Similar approaches give $O(\log\log n)$-round algorithms for $(2+\varepsilon)$-approximation of minimum vertex cover [GGK+18] and for computing a maximal independent set, both for $S = \tilde{O}(n)$. Behnezhad et al. [BHH19] also gave an $O(\log\log n)$-round algorithm for maximal matching and 2-approximation of vertex cover. When the local memory is strongly sublinear $S = n^\alpha$ for a constant $\alpha \in (0,1)$ — which is the ideal target memory regime as the size of the graphs is constantly growing — the best known results are considerably slower: (A) An $\tilde{O}(\sqrt{\log \Delta})$ round algorithm due to Ghaffari and Uitto [GU19] for $(1+\varepsilon)$-approximate maximum matching, maximal independent set, maximal matching, and 2-approximation of vertex cover and an algorithm of Onak [Ona18] with the same round complexity for $(1+\varepsilon)$-approximate maximum matching. Here Δ denotes the maximum degree. (B) An $\tilde{O}(\sqrt{\log \lambda}) + O(\log^3 \log n)$-round algorithms by Behnezhad et al. [BBD+19] for graphs with arboricity at most λ for $(1+\varepsilon)$-approximation of maximum matching, maximal independent set, maximal matching, and 2-approximation of vertex cover.

Another basic graph problem that was studied recently is vertex coloring: Harvey et al. [HLL18] gave an $O(1)$-round algorithm for $(1+o(1))\Delta$-coloring for strongly superlinear local memory $S = n^{1+\varepsilon}$, Assadi et al. [ACK19] gave an $O(1)$-round algorithm for $(\Delta+1)$-coloring for near-linear local memory $S = O(n \log^3 n)$, and Chang et al. [CFG+19] gave a $(\Delta+1)$-coloring for the strongly sublinear memory regime of $S = n^\alpha$ for any constant $\alpha \in (0,1)$ that runs in $O(\sqrt{\log\log n})$ rounds.

To the best of our knowledge, there are no hardness or conditional hardness (and particularly super-constant) results known for any of the above problems in the MPC setting, including matching approximation, vertex cover approximation, maximal independent set, and coloring. The only exception is the hardness for the connectivity problem discussed above: an $\Omega(\log n)$-round lower bound is conjectured and widely believed to hold in the strongly sublinear local memory regime, even for distinguishing one cycle from two cycles.

1.2 Our Contribution

We present conditional hardness results for several of the above graph problems — matching and vertex cover approximation, vertex coloring, maximal independent set, etc. — for the strongly sublinear memory regime where each machine has memory $S = n^\alpha$ for any constant $\alpha \in (0,1)$. Our hardness results apply to any component-stable MPC algorithm, which means algorithms where the outputs in one connected component are independent of other components. See Section 2 for a formal

definition. To the best of our knowledge, all known algorithms in the literature are component-stable or can easily be made component-stable with no asymptotic increase in the round complexity. Our conditional hardness results are based on the popular conjecture mentioned above that distinguishing one cycle from two cycles needs $\Omega(\log n)$ rounds, in the strongly sublinear memory regime. The hardness can be based also on a seemingly even more robust assumption that $\Omega(\log D)$ rounds are needed for D-diameter s-t connectivity where the algorithm should say YES when s and t are endpoints of a path of length at most D and it should say NO when s and t are disconnected.

Perhaps of a more conceptual significance, our hardness results are obtained by lifting unconditional lower bounds from the classic LOCAL model of distributed computing to the MPC model. The LOCAL model was introduced by Linial [Lin87] and it has been studied extensively since the 1980s. By now, it hosts a range of algorithmic and lower bound techniques, developed specifically to cope with the issue of *locality* in decentralized computation or to exhibit the limitations that it imposes. The issue of locality (roughly speaking, having to determine each part of the output without a view of the global topology) appears to be at the heart of the technical difficulty in massively parallel computation, especially with strongly sublinear memory. Hence, we are hopeful that the connection exhibited in this paper will strengthen the ties between the two models and lead to further developments, including conditional hardness results for a wider range of problems in MPC. Previously, connections between these two models where made in the direction of importing algorithms from the LOCAL model to the MPC model [CLM+18, GGK+18, BBD+19, GU19, Ona18, ACK19, CFG+19], sometimes implicitly, and often with considerable speed-ups. In this paper, we are exhibiting a connection in the reverse (and conceptually harder) direction, by exporting hardness results from the LOCAL model to the MPC model. This can also be viewed as the possibility of transferring component-stable MPC algorithms to the LOCAL model (with a certain slow down, and conditioned on the hardness of connectivity). To state our results, we first review the LOCAL Model.

Distributed LOCAL Model [Lin87]: The communication network is abstracted as an n-node graph $G = (V, E)$, with one processor on each node. Initially, each processor knows only its own neighbors. Processors communicate in synchronous message passing rounds where per round each processor can send one message to each of its neighbors. The processors want to solve a graph problem about their network G — e.g., compute coloring or a matching of G — and at the end, each processor/node should know its own part of the output, e.g., its color or its matching mate, if it is matched. The measure of complexity is the round complexity of graph problems, i.e., the number of rounds until which all nodes output and terminate.

Our Hardness Results — Matching, Vertex Cover, MIS: Our main result is an $\Omega(\log \log n)$ conditional hardness for any constant approximation of maximum matching, which also applies to any constant approximation of vertex cover or for computing a maximal independent set.

Theorem 1.1. *Unless there is an $o(\log n)$ round* MPC *algorithm for connectivity with local memory $S = n^\alpha$ for a constant $\alpha \in (0, 1)$ and any $\text{poly}(n)$ global memory, there is no component-stable* MPC *algorithm with local memory $S = n^\alpha$ and $\text{poly}(n)$ global memory that computes a constant approximation of maximum matching, a constant approximation of vertex cover, or a maximal independent set in $o(\log \log n)$ rounds. For maximal matching, the lower bound holds even on trees.*

This result is obtained via lifting an $\Omega(\sqrt{\log n / \log \log n})$ round lower bound of Kuhn, Moscibroda, and Wattenhofer [KMW16] in the LOCAL model for the same problems to the MPC model. We note that for trees, this hardness comes close to the state of the art algorithm for maximal matching, which runs in $O((\log \log n)^3)$ round algorithm [BBD+19]. For general graphs, obtaining a $\text{poly}(\log \log n)$-round strongly sublinear memory MPC algorithm, or even one that is faster than $\tilde{O}(\sqrt{\log n})$ [GU19, Ona18], remains an intriguing open question.

Our Hardness Results — Coloring: By lifting a celebrated lower bound of Linial [Lin87], we show that even the seemingly trivial problem of constant coloring nodes of a cycle needs at least $\Omega(\log(\log^* n))$ rounds — conditioned on the hardness of connectivity. This $\Omega(\log(\log^* n))$ bound is tight. Perhaps as more substantial hardness, we also show that it is probably difficult to improve on the $O(\sqrt{\log\log n})$ round $(\Delta+1)$-vertex coloring algorithm of Chang et al. [CFG+19], by showing that any $o(\sqrt{\log\log n})$-round randomized MPC algorithm would imply a faster than $2^{O(\sqrt{\log n})}$-round LOCAL algorithm for the same problem, hence improving a well-known result of Panconesi and Srinivasan from 1992 [PS92]; the latter remains one of the central and long-standing open problems of the LOCAL model (see, e.g., [BE13, GKM17]).

Theorem 1.2. *Suppose that there is no $o(\log n)$ round MPC algorithm for connectivity with local memory $S = n^\alpha$ for a constant $\alpha \in (0,1)$ and any $\mathrm{poly}(n)$ global memory. Then, if there is a component-stable MPC algorithm with local memory $S = n^\alpha$ and $\mathrm{poly}(n)$ global memory that solves $(\Delta+1)$-coloring in $o(\sqrt{\log\log n})$ rounds, then there is a deterministic LOCAL algorithm that solves $(\Delta+1)$-coloring in $2^{o(\sqrt{\log n})}$ rounds.*

Our Hardness Results — Lovász Local Lemma and Sinkless Orientation: Our lifting method can be applied to essentially any LOCAL-model hardness result. By applying it to a recent lower bound of Brandt et al. [BFH+16], we can prove that any MPC algorithm for a constructive version of the Lovász Local Lemma — even when the bad events of the LLL satisfy much stronger local union bound requirements — needs at least $\Omega(\log\log\log n)$ rounds. This hardness holds even for a very special instance of the LLL, where we should orient the edges of a d-regular graph where $d \geq 4$ such that each node has at least one outgoing edge.

Theorem 1.3. *Suppose that there is no $o(\log n)$ round MPC algorithm for connectivity with local memory $S = n^\alpha$ for a constant $\alpha \in (0,1)$ and any $\mathrm{poly}(n)$ global memory. Then, there is no component-stable MPC algorithm with local memory $S = n^\alpha$ and $\mathrm{poly}(n)$ global memory that in $o(\log\log\log n)$ rounds solves the Lovász Local Lemma problem, or even solves the special instance known as sinkless orientation where we should orient the edges of a d-regular graph, where $d \geq 4$, such that each node has at least one outgoing edge.*

From LOCAL to MPC Hardness — Our Main Technical Contribution: All the discussed conditional hardness results are based on one main technical result, which shows that a LOCAL model lower bound for a graph problem $\mathcal{P}$ together with a fast sublinear local memory component-stable MPC algorithm can be used to build a fast sublinear local memory MPC algorithm for connectivity.

Theorem 1.4. *Let $\mathcal{P}$ be a graph problem that has a $D = T(n)$-round lower bound in the randomized LOCAL model with shared randomness. Suppose that $T(n) \leq \log^\gamma n$ for a constant $\gamma \in (0,1)$ and that that there is an $o(\log D)$-round component-stable MPC algorithm $\mathcal{A}_\mathcal{P}$ for $\mathcal{P}$, with strongly sublinear local memory. Then, there exist strongly sublinear local memory MPC algorithms that solve D-diameter s-t connectivity in $o(\log n)$ rounds and that can distinguish one n-node cycle from two $n/2$-node cycles in time $o(\log n)$.*

In Section 5, we show that the relevant existing randomized LOCAL lower bounds also hold if the nodes have access to an arbitrary amount of shared randomness. Theorems 1.1 to 1.3 then directly follow from Theorem 1.4 and from these shared randomness LOCAL lower bounds (Theorem 5.1 and Corollaries 5.2, 5.4 and 5.5).

1.3 The General Structure of Our Hardness Proofs

All our conditional hardness results are obtained using two ingredients: (A) The best known unconditional lower bound for randomized LOCAL algorithms for the same problem, and (B) The assumed

hardness about the connectivity problem. In a very rough and informal sense, the connection goes as follows: Suppose that there is a problem $\mathcal{P}$ and it has a lower bound of D rounds in the LOCAL model. Then, we show that if there is a component-stable MPC algorithm with strongly sublinear memory that solves this problem in $o(\log D)$ rounds, then this algorithm must somehow be able to "*see some faraway information*" very fast. More concretely, for a certain network graph (from the family of graphs for which the LOCAL lower bound holds), a node v in this graph is able "detect" some information that is related to the parts of the graph that are further than D hops from v, in $o(\log D)$ rounds of the MPC model. We call such an MPC algorithm *farsighted*. Then, we use this farsightedness property to build another MPC algorithm that solves the connectivity problem in certain D-diameter graphs in $o(\log D)$ rounds. The latter is in contradiction with the assumed hardness in (B), which completes the line of argument and shows that there cannot be any component-stable MPC algorithm for problem $\mathcal{P}$ that has strongly sublinear local memory and round complexity $o(\log D)$.

Some comments are in order. First, to turn the above vague intuition into a formal connection, we need to build a number of formal concepts related to *locality*. This, for instance, includes a notion of *sensitivity* to distant information, which allows us to formalize the above line of thought about the MPC algorithm being "farsighted". We are hopeful that the framework developed here may find applications besides the results presented in this paper. As described above, the overall argument is composed of two steps: (I) showing that MPC algorithms that are more than exponentially faster compared to the LOCAL bound must be farsighted, and (II) farsighted component-stable MPC algorithms allow us to build a fast connectivity algorithm that is in contradiction with the assumed hardness. We explain these two parts in Section 3 and Section 4, respectively.

Second, regarding the hardness of connectivity, we discuss two separate conditional hardness assumptions in Section 4. One of these, which is about the connectivity problem in low-diameter instances (which we call D-diameter s-t connectivity) appears to be considerably more stable in the sense that if the hardness of this connectivity problem goes down slightly, our conditional hardness results also get weakened to the same extent. The other is the more standard one cycle vs. two cycles $\Omega(\log n)$ lower bound, which we show to imply the assumed hardness of the former.

Third, for the LOCAL-model lower bounds to be applicable to our framework, it is important that they apply to randomized LOCAL algorithms that also have access to shared randomness. Although common lower bounds from the LOCAL model only assume private randomness, in Section 5, we explain how to extend a number of them to also incorporate shared randomness, without any asymptotic loss in their round complexity.

2 Formal Definitions

Graph-Theoretic Notations and Definitions. Given a graph $G = (V, E)$ and a subset $S \subseteq V$ of its nodes, we use $G[S]$ to denote the subgraph of G induced by the nodes in S. Further, for a node $v \in V$ and an integer $r \geq 0$, we use $N_r(v) := \{u \in v : d_G(u, v) \leq r\}$ to denote the set of nodes in the r-hop neighborhood of v. Further, the r-hop neighborhood of a node $v \in V$ is defined as the graph $G[N_r(v)]$ induced by the set of nodes within distance at most r from v.

When arguing about distributed algorithms and their relation to MPC algorithms, we often have to consider graphs from the viewpoint of a specific node. We thus define the notion of a *centered graph* as a graph $G = (V, E)$ together with a distinguished *center* $v \in V$. A centered graph G with center v has radius $\text{rad}(G)$ if the maximum distance of between v and any other node in G is $\text{rad}(G)$. Two centered graphs G and G' with centers v and v' are called r-hop-isomorphic for an integer $r \geq 0$ if there is an isomorphism φ between the r-hop neighborhoods $G[N_r(v)]$ and $G'[N_r(v')]$ that maps v to v', i.e., $\varphi(v) = v'$. We note that by definition, G is r-hop isomorphic to itself for any $r \geq 0$.

Distributed Graph Problems. We consider graph problems, where when applied to a graph $G = (V, E)$ each node $v \in V$ has to output some label from a given alphabet Σ (e.g., a color in the vertex coloring problems or an indicator 1 or 0 when computing an independent set). For simplicity, we do not explicitly treat output labels for edges, however all our arguments directly extend to allowing output labels for edges. Alternatively, outputs for edges can also be delegated to the incident nodes (e.g., in a matching, each node can output whether it is matched and if yes, which neighbor its matching partner is).

LOCAL model algorithms. Let $\mathcal{G}$ be some family of graphs. We consider randomized LOCAL algorithms that are run on a graph $G = (V, E)$ from $\mathcal{G}$ and where the nodes in V have access to shared randomness. Each node is given an upper bound n on the number of nodes of G as input. Each node is further given a unique ID from a domain $\mathcal{N} := \{1, \ldots, N\}$, where $N \geq n$. For some integer $r \geq 0$, let $\mathcal{T}_{\mathcal{G},r}$ be the set of possible labeled r-hop neighborhoods of n-node graphs in $\mathcal{G}$, where each node is given a unique label from $\mathcal{N}$. For a running time function $t(n)$, a $t(n)$-round randomized LOCAL algorithm is formally defined as a function

$$f_{\mathsf{LOCAL}} : \mathcal{S}_L \times \mathcal{T}_{\mathcal{G},t(n)} \to \Sigma,$$

where $\mathcal{S}_L$ is a shared string of random bits and Σ is the set of possible output labels of the distributed problem solved by the algorithm. To determine its output, a node $v \in G$, collects its labeled $t(n)$-hop neighborhood and it applies the function f_{LOCAL} to it and the shared randomness $\mathcal{S}_L$. An algorithm is said to be correct with high probability (w.h.p.) if for any n-node graph G in $\mathcal{G}$ and any possible ID assignment to the nodes of G, it outputs a correct solution with probability at least $1-1/n$ (where the probability is taken over the shared randomness $\mathcal{S}_L$).

MPC algorithms: Let $n \geq 1$ and $M \geq 1$ be integers and let $\mathcal{G}$ be some family of graphs. We consider randomized MPC algorithms that are run on an n-node graph $G = (V, E)$ from $\mathcal{G}$. The system is composed of M machines, all of whom have access to shared randomness $\mathcal{S}_P$. Each node of G is given a unique ID from a domain $\{1, \ldots, N\}$, where $N = n^{O(1)}$. Each edge of G is given as input to one of the M machines. In the end, the algorithm needs to assign one output in Σ to each node and each machine needs to know the outputs of all nodes for which it initially holds edges. The algorithm is said to be correct w.h.p. if it outputs a correct solution with probability at least $1 - 1/n$ (where the probability is taken over the shared randomness $\mathcal{S}_P$).

For our conditional lower bounds, we consider a natural class of MPC algorithms, which we call **component-stable**, where the outputs of nodes in different connected components are independent. Formally, assume that for a graph G, $\mathcal{D}_G$ denotes the initial distribution of the edges of G among the M machines and the assignment of unique IDs to the nodes of G. For a subgraph H of G let $\mathcal{D}_H$ be defined as $\mathcal{D}_G$ restricted to the nodes and edges of H. Let H_v be the connected component of node v. An MPC algorithm $\mathcal{A}$ is called component-stable if for each node $v \in V$, the output of v depends (deterministically) on the node v itself, the initial distribution and ID assignment $\mathcal{D}_{H_v}$ of the connected component H_v of v, and on the shared randomness $\mathcal{S}_M$.

3 Distributed Lower Bounds Imply Farsighted MPC Algorithms

We will now present the key part of transferring an existing LOCAL model lower bound for a given graph problem to an argument that says that, for any MPC algorithm that solves the problem, the output of some nodes has to depend on far-away nodes. We will see in Section 4.2 how this property of an MPC algorithm can be used to prove a conditional time lower bound in the MPC model. We

first formally define what it means for an MPC algorithm to depend on far-away information in the graph.

Definition 3.1 ((r,ε)-Sensitive MPC Algorithm)**.** *For an integer $r \geq 0$ and some $\varepsilon > 0$, an* MPC *algorithm $\mathcal{A}$ for some graph problems is called* (r,ε)-sensitive *w.r.t. two r-hop-isomorphic centered graphs $G = (V, E)$ and $G' = (V', E')$ with centers $v \in V$ and $v' \in V'$ if the following is true.*

We consider two dependent runs of $\mathcal{A}$, one on G and one on G'. In both runs, each node in the isomorphic r-hop neighborhoods $G[N_r(v)]$ and $G'[N_r(v')]$ are assigned the same uniformly chosen random IDs from the ID space of algorithm $\mathcal{A}$. All the remaining nodes, in G and G', are assigned independently and uniformly chosen random IDs from the ID space of algorithm $\mathcal{A}$. Each edge $\{u, w\}$ of G or G' is labeled by the set of IDs $\{\mathrm{ID}(u), \mathrm{ID}(w)\}$. For each edge label, we independently choose a machine uniformly at random and we give all edges with that label to that machine. That is, in particular, the corresponding edges of $G[N_r(v)]$ and $G'[N_r(v')]$ are given to the same machines in both runs (note that for a sufficiently large ID space, with very high probability, these are the only edges where the machines are not chosen independently).

Then, if the MPC *algorithm $\mathcal{A}$ is run on G and G' with the same shared randomness, the probability that the two centers v and v' output different values is at least ε (the probability is taken over the choice of random IDs, the random distribution of the edges, and the shared randomness).*

We note that in the above definition, it is fine if G and G' are isomorphic beyond distance r from the centers and in particular if G and G' are the same graph. We next show that for every graph problem $\mathcal{P}$ for which there is a $T(n)$-round LOCAL lower bound, there exists a $(T(k), 1/k^{O(1)})$-sensitive pair of centered graphs G and G' of size $k \leq n^\delta$ for a suitably small constant $\delta > 0$.

Lemma 3.1. *Let $\mathcal{P}$ be a graph problem for which there is no $T(n)$-round* LOCAL *algorithm that solves $\mathcal{P}$ on n-node graphs from a given family $\mathcal{G}$ with probability at most $1 - 1/n$. Assume that this holds for any sufficiently large n, even if the* LOCAL *algorithm has access to shared randomness and even if all nodes initially know n. Then, for every* MPC *algorithm $\mathcal{A}_{\mathsf{MPC}}$ that solves $\mathcal{P}$ with probability at least $2/3$, for every constant $\delta > 0$ and some $t(n)$ for which $\log t(n) = \Theta(\log T(n))$, there are two $t(n)$-hop-isomorphic centered graphs G and G' of size less than n^δ such that $\mathcal{A}_{\mathsf{MPC}}$ is $(t(n), 1/n^\delta)$-sensitive w.r.t. G and G'.*

Proof. We first note that when running $\mathcal{A}_{\mathsf{MPC}}$ on a graph G of size at most n^δ, we can boost the probability that $\mathcal{A}_{\mathsf{MPC}}$ succeeds to $1 - 1/n$ by adding $O(\log n)$ additional independent copies of G. These additional components can be produced by using the public randomness of the MPC algorithm.

Let $\delta > 0$ be an arbitrarily chosen constant. For convenience, we define $k := \lfloor n^{\kappa\delta} \rfloor$ for a sufficiently small constant $\kappa > 0$ and we define $\varepsilon := n^{-\delta}$. Let us assume that there are not two centered graphs G and G' of size at most k such that the given MPC algorithm $\mathcal{A}_{\mathsf{MPC}}$ is $(T(k), \varepsilon)$-sensitive w.r.t. G and G' (holds also if $G = G'$). We use this fact to construct a $T(k)$-round randomized LOCAL algorithm $\mathcal{A}_{\mathsf{LOCAL}}$ to solve $\mathcal{P}$ on graphs of size k with probability at least $1 - 1/k$, which is a contradiction to the assumption of the lemma that no such algorithm exists. Note that because $T(k)$ can grow at most linearly with k (because every graph problem on graphs of size k can be solved in time at most k in the LOCAL model). The choice of k then implies that $\log T(k) = \Theta(\log T(n))$ and we can thus choose $t(n) = T(k)$. The contradiction thus implies the claim of the lemma.

We start the discussion by describing how to construct a LOCAL algorithm $\mathcal{A}_{\mathsf{LOCAL}}$ from the given MPC algorithm $\mathcal{A}_{\mathsf{MPC}}$. We then show that $\mathcal{A}_{\mathsf{LOCAL}}$ solves $\mathcal{P}$ with the right success probability.

Construction of the LOCAL algorithm. For convenience, we will use $R := T(k)$, such that we can construct an R-round LOCAL algorithm. In order to prove the existence of such an algorithm $\mathcal{A}_{\mathsf{LOCAL}}$, we have to prove that there is a function f_{LOCAL} that takes an arbitrary labeled R-hop

neighborhood and an arbitrary bit string $\mathcal{S}$ and that maps this to an output label. The label is the output of a node v in $\mathcal{A}_{\mathsf{LOCAL}}$ if v's labeled R-hop neighborhood is as given and if the shared randomness of $\mathcal{A}_{\mathsf{LOCAL}}$ is equal to $\mathcal{S}$. For every labeled k-node graph G, the function should produce a valid solution for $\mathcal{P}$ on G for at least a $(1-1/k)$-fraction of all possible bit strings $\mathcal{S}$. We next show how to construct such a function f_{LOCAL}.

Assume that we are given a sufficiently long bit string $\mathcal{S}$. For simplicity, we will assume that $\mathcal{S}$ is given as two strings $\mathcal{S}_0$ and $\mathcal{S}_1$ ($\mathcal{S}_0$ could for example consist of the entries of $\mathcal{S}$ at even positions and $\mathcal{S}_1$ could consist of the entries of $\mathcal{S}$ at odd positions). Assume further that we are given a labeled R-hop neighborhood, which is given as a centered graph H of radius at most R, where each node of H is labeled with a unique ID. Let v_H be the center node of the centered graph H. Further, let $\mathcal{H}$ be the family of all k-node centered graphs that are R-hop-isomorphic to H. That is, $\mathcal{H}$ is the set of centered graphs, where the R-hop neighborhood of the center is equal to H (without the labels). We use the short notation $\mathrm{ID}(H)$ for the given assignment of IDs to H and we call the triple $(\mathrm{ID}(H), \mathcal{S}_0, \mathcal{S}_1)$ the input of the LOCAL algorithm.

Let $\mathcal{H} = H_1, \dots, H_q$ be an enumeration of all graphs in $\mathcal{H}$. W.l.o.g., assume that the possible set of output labels of a node are $\mathcal{L} := \{1, \dots, \ell\}$ for some $\ell \geq 2$. For each graph H_a, we compute a probability distribution over the ℓ possible output labels by evaluating the following random process. We first assign a uniformly random ID to each node $u \in H_a$ that is not in the R-hop neighborhood of the center node v of H_a. That is, we assign a uniformly random ID to each node of H_a that is not in H and thus does not have an ID assigned yet. Now, we deterministically compute an output label for the center node v as follows. We use the shared random bit string $\mathcal{S}_0$ to determine the assignment of the edges of H_a to the M machines of the MPC model. This assignment is done in way such that we use disjoint parts of $\mathcal{S}_0$ to determine the machine of each possible edge, where each possible edge is identified by the two IDs of its two nodes. This guarantees that if we choose $\mathcal{S}_0$ as a uniformly random bit string, the edges are assigned to machines independently. We further make sure that the assignment is done in a way such that if $\mathcal{S}_0$ is chosen uniformly at random, the assignment of edges to machines is also done uniformly at random. Given the assignment of IDs and the distribution of the edges of H_a among the M machines, we then run the MPC algorithm $\mathcal{A}_{\mathsf{MPC}}$ with the second shared random bit string $\mathcal{S}_1$. Note that after fixing the input $I = (\mathrm{ID}(H), \mathcal{S}_0, \mathcal{S}_1)$, and the random IDs of the nodes that are not in H, $\mathcal{A}_{\mathsf{MPC}}$ is a deterministic algorithm. For each output label $x \in \{1, \dots \ell\}$, we define $p_{a,x}(I)$ as the probability that node v outputs label x. We set

$$f_{\mathsf{LOCAL}}(I) = f_{\mathsf{LOCAL}}(\mathrm{ID}(H), \mathcal{S}_0, \mathcal{S}_1) := \arg\max_{x \in \{1,\dots,\ell\}} \sum_{a=1}^{|\mathcal{H}|} p_{a,x}(I),$$

that is, algorithm $\mathcal{A}_{\mathsf{LOCAL}}$ outputs the label that has the largest probability when using a random graph from $\mathcal{H}$ for the given input I.

Correctness of the LOCAL algorithm. Let $\mathcal{I}$ be the set of possible inputs $(\mathrm{ID}(H), \mathcal{S}_0, \mathcal{S}_1)$. For an input $I \in \mathcal{I}$ and a pair of graphs $H_a, H_b \in \mathcal{H}$, we define $P_{a,b}(I) := \sum_{x=1}^{\ell} p_{a,x}(I) \cdot p_{b,x}(I)$, which is the probability that when running the above process on graphs H_a and H_b with input I, the centers of H_a and H_b output the same value. Similarly, for each $H_a \in \mathcal{H}$, we define $P_a(I) := \sum_{x=1}^{\ell} p_{a,x}^2(I)$, which is is the probability that two independent runs on H_a result in the same output. Because we assumed that there are no two centered graphs G and G' of size at most k such that the given MPC algorithm $\mathcal{A}_{\mathsf{MPC}}$ is $(T(k), \varepsilon)$-sensitive w.r.t. G and G', we know that for every $H_a, H_b \in \mathcal{H}$, $P_{a,b}(I) \geq 1-\varepsilon$ and $P_a(I) \geq 1-\varepsilon$ if I is chosen uniformly at random. We thus have $\sum_{I \in \mathcal{I}} P_{a,b}(I) \geq (1-\varepsilon)|\mathcal{I}|$ and $\sum_{I \in \mathcal{I}} P_a(I) \geq (1-\varepsilon)|\mathcal{I}|$. We call an input $I \in \mathcal{I}$ *good for the pair* (H_a, H_b) if $P_{a,b}(I) \geq 1-\varepsilon^{1/3}$ and we call I *good for* H_a if $P_a(I) \geq 1-\varepsilon^{1/3}$. Let $\eta > 0$ be the fraction of inputs that are not good for a

pair (H_a, H_b). We have $\varepsilon|\mathcal{I}| \geq \sum_{I\in\mathcal{I}}(1 - P_{a,b}(I)) \geq \eta \cdot \varepsilon^{1/3} \cdot |\mathcal{I}|$ and thus $\eta \leq \varepsilon^{2/3}$. Hence, for every pair (H_a, H_b), at least a $(1-\varepsilon^{2/3})$-fraction of all inputs $I \in \mathcal{I}$ is good for (H_a, H_b). Similarly, we get that for every $H_a \in \mathcal{H}$, at least a $(1-\varepsilon^{2/3})$-fraction of all inputs $I \in \mathcal{I}$ is good for H_a.

Assume that I is good for some pair $(H_a, H_b) \in \mathcal{H}^2$. Then if ε is chosen such that $\varepsilon^{1/3} < 1/2$, there exists an output label $x \in \{1, \ldots, \ell\}$ for which $p_{a,x}(I) \geq 1 - \varepsilon^{1/3}$ and $p_{b,x}(I) \geq 1 - \varepsilon^{1/3}$. To see this, let x be the output label that maximizes $p_{a,x}(I)$, let y be the output label that maximizes $p_{b,y}(I)$, and assume that $p_{a,x} = 1-\alpha$ and $p_{b,y}(I) = 1-\beta$. W.l.o.g., assume that $\alpha \geq \beta$. The value of $P_{a,b}(I)$ is a convex combination of all the $p_{a,z}(I)$-values and it is thus maximized if $p_{b,z}(I)$ is non-zero only if $p_{a,z}(I) = p_{a,x}(I) = 1-\alpha$. We thus get that $P_{a,b}(I) \leq 1-\alpha$ and thus $\alpha \leq \varepsilon^{1/3}$ because we know that $P_{a,b}(I) \geq 1-\varepsilon^{1/3}$. Because we assumed that $\varepsilon^{1/3} < 1/2$, we get that $\alpha < 1/2$ (and thus also $\beta < 1/2$). In this case, $P_{a,b}(I)$ can only be larger than $1/2$ if $x = y$, which proves that there is an x for which $p_{a,x}(I) \geq 1-\varepsilon^{1/3}$ and $p_{b,x}(I) \geq 1-\varepsilon^{1/3}$. With the same argument, we can also conclude that if I is good for H_a, there is an output label x for which $p_{a,x}(I) \geq 1-\varepsilon^{1/3}$. In the following, we say that x is a dominant output label for input I and graph H_a if $p_{a,x}(I) \geq 1-\varepsilon^{1/3}$.

We further define when an input $I \in \mathcal{I}$ is good for the graph H, the common central R-neighborhood in all graphs $H_a \in \mathcal{H}$. We say that $I \in \mathcal{I}$ is *good for* H if it is good for at least a $(1-\varepsilon^{1/3})$-fraction of all the pairs (H_a, H_b) (where we assume that $a < b$). Let η be the fraction of inputs that is not good for H. Further, let $\mathcal{P}$ be the set of possible pairs (H_a, H_b) with $a < b$. We have at least $\eta|\mathcal{I}|\varepsilon^{1/3}|\mathcal{P}|$ combinations of $I \in \mathcal{I}$ and $(H_a, H_b) \in \mathcal{P}$ such that I is not good for (H_a, H_b). Because for every pair, a $(1-\varepsilon^{2/3})$-fraction of the inputs is good, we have $\eta|\mathcal{I}|\varepsilon^{1/3}|\mathcal{P}| \leq \varepsilon^{2/3}|\mathcal{P}||\mathcal{I}|$ and thus $\eta \geq \varepsilon^{1/3}$. Hence, at least an $(1-\varepsilon^{1/3})$-fraction of all inputs $I \in \mathcal{I}$ is good for H.

We next show that if an input I is good for H, then there is an output label x that is dominant for almost all graphs $H_a \in \mathcal{H}$ and further, x is also the output value of $\mathcal{A}_{\mathsf{LOCAL}}$ for H on input I. We first define a graph $G_{I,\mathcal{H}} = (\mathcal{H}, E_{I,\mathcal{H}})$ on the node set $\mathcal{H} = \{H_1, \ldots, H_q\}$. The graph contains an edge $\{H_a, H_b\} \in E_{I,\mathcal{H}}$ if input I is good for H_a and H_b. Recall that in this case (if H_a and H_b are connected by an edge in $G_{I,\mathcal{H}}$), there is an output label x that is dominant for I and H_a and for I and H_b. This implies that for each connected component of $G_{I,\mathcal{H}}$, there is an output label x that is dominant for I and all graphs H_a in the connected component. If I is good for H, the graph $G_{I,\mathcal{H}}$ contains a $(1-\varepsilon^{1/3})$-fraction of all the $\binom{q}{2}$ possible edges. This implies that $G_{I,\mathcal{H}}$ has a connected component of size at least $(1-\varepsilon^{1/3})q$. We call this component the giant component for $G_{I,\mathcal{H}}$. Clearly, if ε is chosen below a sufficiently small constant, the dominant output value of this giant component is also the output value of the constructed LOCAL algorithm $\mathcal{A}_{\mathsf{LOCAL}}$. Note that if $\mathcal{H}$ consists only of a single graph $H_1 = H$, I is good for H_1 if and only if it is good for H and in this case, the giant component consists of the single graph H_1.

We next put the above definitions together and we define an input $I \in \mathcal{I}$ to be well-behaved for a graph $H_a \in \mathcal{H}$ if a) I is good for H_a, b) I is good for H, and c) H_a is part of the giant component of $G_{I,\mathcal{H}}$. We next show that if $\varepsilon^{1/3} < 1/2$, for every $H_a \in \mathcal{H}$, at least a $(1-\varepsilon^{1/3}-3\varepsilon^{2/3})$-fraction of all inputs $I \in \mathcal{I}$ are well-behaved for H_a. To show this, let $\deg_I(H_a)$ be the degree of node H_a in graph $G_{I,\mathcal{H}}$. We know that for every $H_b \in \mathcal{H} \setminus \{H_a\}$, at least a $(1-\varepsilon^{2/3})$-fraction of all inputs I, I is good for the pair (H_a, H_b). We therefore have

$$\sum_{I\in\mathcal{I}} \deg_I(H_a) \geq \sum_{H_b \neq H_a} (1-\varepsilon^{2/3})|\mathcal{I}| = (1-\varepsilon^{2/3})(q-1)|\mathcal{I}|.$$

Let γ be the fraction of inputs $I \in \mathcal{I}$ for which $\deg_I(H_a) < \varepsilon^{1/3}q - 1 \leq \varepsilon^{1/3}(q-1)$. We have

$$\gamma|\mathcal{I}| \cdot \varepsilon^{1/3}(q-1) + (1-\gamma)|\mathcal{I}|(q-1) \geq (1-\varepsilon^{2/3})(q-1)|\mathcal{I}|,$$

which together with $\varepsilon^{1/3} < 1/2$ implies that $\gamma < 2\varepsilon^{2/3}$. Note that if I is good for H, the giant component has size at least $(1-\varepsilon^{1/3})q$ and thus if $\deg_I(H_a) \geq \varepsilon^{1/3}q - 1$, H_a is part of the giant

component. Hence, the fraction of inputs that are not good for H_a is at most $\varepsilon^{2/3}$, the fraction of inputs that are not good for H is at most $\varepsilon^{1/3}$ and among the inputs that are good for H, the fraction where H_a is not in the giant component is at most $2\varepsilon^{2/3}$. Overall, the fraction of inputs for which H_a is well-behaved is thus at least $1 - \varepsilon^{1/3} - 3\varepsilon^{2/3}$.

It finally remains to show that the LOCAL algorithn $\mathcal{A}_{\mathsf{LOCAL}}$ succeeds with sufficiently large probability. It suffices to show that $\mathcal{A}_{\mathsf{LOCAL}}$ has a large success probability if each node ID is chosen independently and uniformly at random from a sufficiently large domain. A randomized LOCAL algorithm can always do this as a first step. Let us therefore assume that we have a graph $G = (V, E)$ of size at most k on which we want to solve problem $\mathcal{P}$. If each node $v \in V$ independently chooses a uniformly random node ID from a domain of size at least n^3, the probability that two nodes choose the same ID is less than $1/n$ (even for $k = n$). We now show that the algorithm $\mathcal{A}_{\mathsf{LOCAL}}$ correctly solves the graph problem $\mathcal{P}$ on G if the IDs are chosen independently and uniformly at random and if the shared randomness $\mathcal{S}_0$ and $\mathcal{S}_1$ is also chosen uniformly at random. Let $\mathrm{ID}(G)$ be the given random ID assignment. Consider some node $v \in V$ of G and let H_v be the centered graph induced by the R-hop neighborhood of v. Let $\mathcal{H}_v$ be the set of centered graphs with at most k nodes that are R-hop-isomorphic to H_v. Note that G is one such graph. Assume that for the input I_v induced by the given ID assignment to H_v and the given shared randomness $\mathcal{S}_0$ and $\mathcal{S}_1$, G is well-behaved. Then, the output of v by the LOCAL algorithm $\mathcal{A}_{\mathsf{LOCAL}}$ is the same as the output of the given MPC algorithm $\mathcal{A}_{\mathsf{MPC}}$ with probability at least $1 - \varepsilon^{2/3}$ because the input I_v is good for G and good for H_v and G is in the giant component of $G_{I_v,\mathcal{H}_v}$. The probability that the input I_v is well-behaved is at least $1 - \varepsilon^{1/3} - 3\varepsilon^{2/3}$ and thus, the probability that v outputs the same value in $\mathcal{A}_{\mathsf{LOCAL}}$ and in $\mathcal{A}_{\mathsf{MPC}}$ is at least $1 - \varepsilon^{1/3} - 4\varepsilon^{2/3}$ (when running $\mathcal{A}_{\mathsf{MPC}}$ by using the shared randomness of $\mathcal{A}_{\mathsf{LOCAL}}$ as described above). By using a union bound over all at most k nodes of G, we thus get that $\mathcal{A}_{\mathsf{LOCAL}}$ and $\mathcal{A}_{\mathsf{MPC}}$ output the same solution with probability at least $1 - k(\varepsilon^{1/3} + 4\varepsilon^{2/3})$. The MPC algorithm $\mathcal{A}_{\mathsf{MPC}}$ outputs a correct solution with probability $1 - 1/n$ if the random ID assignment was successful in assigning unique IDs. The success probability of $\mathcal{A}_{\mathsf{MPC}}$ is thus at least $1 - 2/n$ and thus the success probability of $\mathcal{A}_{\mathsf{LOCAL}}$ is at least $1 - k(\varepsilon^{1/3} + \varepsilon^{2/3}) - 2/n$. Recall that $k = \lfloor n^{\kappa\delta} \rfloor$ and that $\varepsilon = n^{-\delta}$ for sufficiently small constants $\delta > 0$ and $\kappa > 0$. By choosinig δ and κ appropriately, the success probability of $\mathcal{A}_{\mathsf{LOCAL}}$ is thus better than $1 - 1/k$, which is a contradition to the assumption that no such LOCAL algorithm exists. □

4 Farsighted MPC Algorithms Imply Super Fast Connectivity

In Section 4.1, we discuss our conditional hardness assumption for the connectivity problem. Then, in Section 4.2, we explain how to combine this with the result of Section 3, which shows that distributed lower bounds imply that MPC algorithms that are too fast must be farsighted, in order to obtain our conditional hardness results.

4.1 Hardness Assumption for the Connectivity Problem

As the base assumption of our conditional hardness results, we assume that any MPC algorithm with local memory n^α for a constant $\alpha \in (0, 1)$ and global memory $\mathrm{poly}(n)$ needs $\Omega(\log D)$ rounds to solve the following *D-diameter s-t connectivity* problem:

Definition 4.1 (The D-diameter s-t connectivity problem)**.** *Given two special nodes s and t in the graph, the algorithm should determine whether they are connected or not, in a way that provides the following (rather weak) guarantee: If s and t are in the same connected component and that component is a path of length at most D with s and t at its two endpoints, then the algorithm should output*

YES, *with probability* $1 - 1/\operatorname{poly}(n)$. *If* s *and* t *are in different connected components, the algorithm should output* NO, *with probability* $1 - 1/\operatorname{poly}(n)$. *In other cases, the output can be arbitrary.*

We remark that the above problem can be solved in $O(\log D)$ rounds, using the algorithm of Andoni et al. [ASS+18], using local memory n^α for any constant $\alpha \in (0, 1)$ and global memory $\operatorname{poly}(n)$. Furthermore, we also note that, if we take an $\Omega(\log D)$ hardness for the above problem for granted, we can relax the probability guarantee to $2/3$ and still have an $\Omega(\log D)$ round conditional hardness. This is because any algorithm that guarantees a $2/3$ probability of success can be amplified to provide success probability of at least $1 - 1/\operatorname{poly}(n)$, with no asymptotic round complexity overhead: simply use $O(\log n)$ parallel repetitions and then take the majority output.

We conjecture and more concretely assume $\Omega(\log D)$ to be a lower bound on the time needed for D-diameter s-t connectivity. We think that this itself is a robust and reliable hardness assumption. To add to the justification of this assumption, in the remainder of this subsection, we relate this assumption to the more standard $\Omega(\log n)$ round hardness for the problem of distinguishing the case where the input graph is one cycle from the case where it is two cycles:

Lemma 4.1. *Suppose that every* MPC *algorithm with local memory* n^α *for a constant* $\alpha \in (0, 1)$ *and global memory* $\operatorname{poly}(n)$ *that can distinguish one* n*-node cycle from two* $n/2$*-node cycles requires* $\Omega(\log n)$ *rounds. Then, any* MPC *algorithm with local memory* n^α *for a constant* $\alpha \in (0, 1)$ *and global memory* $\operatorname{poly}(n)$ *that solves* D*-diameter* s*-*t *connectivity for* $D \leq \log^\gamma n$ *for a constant* $\gamma \in (0, 1)$ *requires* $\Omega(\log D)$ *rounds.*

Proof. Let us start with a proof outline. For the sake of contradiction, suppose that there is an MPC algorithm $\mathcal{A}$ with local memory n^α for a constant $\alpha \in (0, 1)$ and global memory $\operatorname{poly}(n)$ that solves D-diameter s-t connectivity in $o(\log D)$ rounds. We use this to argue that there is such an MPC algorithm $\mathcal{B}$ that solves the one cycle vs. two cycles problem in $o(\log n)$ rounds. This argument has two steps. In the first step, we build an $o(\log D)$ round algorithm $\mathcal{A}'$ for the following connected component identification problem: for any component that is a path of length at most D, the algorithm should produce the same label for all of the nodes of this component, with high probability. On the other hand, with high probability, no two nodes of different components should receive the same label. In the second step, we build the algorithm $\mathcal{B}$ for the one cycle vs. two cycles problem by repeated applications of algorithm $\mathcal{A}'$ on certain inputs.

First Step: We now build algorithm $\mathcal{A}'$ for connected component identification using $\operatorname{poly}(n)$ parallel repetitions of algorithm $\mathcal{A}$ for D-diameter s-t connectivity in carefully devised input graphs. Our parallel repetitions are divided into at most n groups, where in each of them we set t to be one of the nodes in the graph with degree exactly 1. Then, in each group, we have $O(n)$ tests, one for each possibility of node s that has degree 1 or 2 in the graph. The test consists of $O(\log n)$ runs of $\mathcal{A}$, on graphs defined as follows: In each run, we make s keep exactly one of its edges (chosen randomly if it has 2 edges), and then run $\mathcal{A}$ on this instance. If in any of the $O(\log n)$ runs algorithm $\mathcal{A}$ answers YES, we can conclude that s is in the same component as t. Moreover, if s is in the same component as t and this component is a path of length at most D, then, with high probability, at least one of the runs makes s and t the two endpoints of a path of length at most D and we get a YES answer. Hence, we conclude that, over the groups for different t, each node s in a path of length at most D *learns* the identifiers of the two endpoints of its path (each identifier is indicated by the index of the group in which the answer was YES). Any node s outputs the minimum of the identifiers that it has learned, as the label of its component. If node s does not learn any identifiers in this way, it outputs its own identifier as its component label. With high probability, the identifiers learned by each node s are degree-1 nodes in the component that holds s and if the component is a path of length at most D, then the node s learns the identifiers of both of the endpoints of its path. Hence, in any path of

length at most D, all nodes output the same label (the minimum ID of the two endpoints). Moreover, any two nodes that output the same label must necessarily be in the same component. Algorithm $\mathcal{A}'$ runs in the same round complexity as algorithm $\mathcal{A}$, because the former is simply $\text{poly}(n)$ parallel repetitions of the latter.

Second Step: We build $\mathcal{B}$ for solving the one cycle vs. two cycles problem using algorithm $\mathcal{A}'$ as follows: We have $O(\log n/\log D)$ sequential iterations. In the first iteration, we remove each edge with probability $1/\sqrt{D}$ and then run algorithm $\mathcal{A}'$. Hence, $\mathcal{A}'$ assigns a label to each vertex. Notice that each connected component after the random removal of edges has length $O(\sqrt{D}\log n)$ with high probability. Since $\mathcal{A}'$ is guaranteed to not assign the same label to nodes of different components, each label is held by at most $O(\sqrt{D}\log n) \ll n^{\alpha}$ nodes. In one round, we move all nodes of the same label to one machine (along with their edges), using one machine for each label. Notice that due to the way $\mathcal{A}'$ is defined, it might label two nodes v and u the same even though they are further than D hops away, and it might even label some node w that is on the shortest path between v and u differently than the (common) label of v and u. Each machine relabels the nodes that it receives with new labels, in a way that each new label is exactly a connected component of the nodes that it receives. We then contract all nodes that have the same label (i.e., all edges whose both endpoints have the same label). That is, we now have a graph with one vertex for each of the labels and two labels are connected iff there are two connected vertices that had those two labels. Since $\mathcal{A}'$ with high probability does not label nodes of different components with the same label, the result of the contraction keeps each input cycle separate (if there are two cycles). It also keeps them cycles — that is, one cycle if we had one input cycle, and two cycles if we had two input cycles — because each label corresponds to a connected component after the edge removals, and thus each contracted part is an induced connected subgraph of the cycle, that is, either a path in it or the entire cycle. The other iterations run similarly, and we continue until all edges are contracted. Eventually, we have either one label or two labels, which corresponds to the input being one cycle or two cycles.

We now analyze the number of iterations needed until termination. Define the *excess number of labels* to be the number of labels (i.e., nodes) in the graph minus the number of connected components. We can see that the expected number of excess labels in the resulting new graph is at most $O(n/D)$. The reason is as follows: let's call a node bad if after the sampling of edges, it is in a component that is longer than D. The probability of each node being bad is at most $2(1-\frac{1}{\sqrt{D}})^D = \exp(-\Theta(\sqrt{D})) \ll 1/D^2$. Let's call a component short if it has fewer than $D^{1/4}$ vertices. The probability of a node being in a short component is at most $D^{1/4}/\sqrt{D} = O(\frac{1}{D^{1/4}})$. The number of new labels is at most the number of bad nodes or those in short components, plus the number of components that have length at least $\sqrt{D}$ (and no more than D). In expectation, the former is at most $O(n/D^{1/4})$, and the latter is necessarily at most $n/\sqrt{D}$ —because each component of length at least $\sqrt{D}$ consumes at least $\sqrt{D}$ nodes. Hence, in expectation, the number of new labels is $O(n/D^{1/4})$. That is, in expectation, in one iteration, the number of excess labels goes down by a factor of $O(D^{1/4})$. After $O(\log n/\log D)$ repeated application of the same procedure, the number of excess labels is $1/\text{poly}(n)$. Hence, by Markov's inequality, with probability $1-1/\text{poly}(n)$, there is no excess label. That is, we have either one label or two labels, depending on whether the input was one cycle or two cycles. Hence, we can solve the one cycle vs. two cycles problem via $O(\log n/\log D)$ repetitions of algorithm $\mathcal{A}'$. Since $\mathcal{A}'$ runs in $o(\log D)$ rounds, the algorithm $\mathcal{B}$ that we have built runs in $o(\log n)$ rounds. □

4.2 Solving Connectivity, Assuming a Farsighted MPC Algorithm

From Lemma 3.1, we know that for every MPC algorithm $\mathcal{A}_{\mathsf{MPC}}$ that solves $\mathcal{P}$ in $o(\log t(n))$ rounds, there are two $t(n)$-hop-isomorphic centered graphs G and G' of size less than n^{δ} such that $\mathcal{A}_{\mathsf{MPC}}$

is $(t(n), 1/n^\delta)$-sensitive w.r.t. G and G'. In this section, we leverage that to obtain an $o(\log T(n))$ round algorithm for D-diameter s-t where $D = T(n)$, hence putting us in contradiction with the assumed hardness for the latter. Having arrived at this contradiction, we can conclude that any MPC algorithm for $\mathcal{P}$ needs at least $\Omega(\log T(n))$ rounds.

Throughout this argument, we limit $T(n)$ to be at most $\Theta(\log n/\log\log n)$, which suffices for all our hardness results. Indeed, limiting it to any $T(n) \leq \log^\gamma n$ for any positive constant $\gamma > 0$ would also suffice, as our final hardness bound is $\log T(n)$ and thus polynomial changes in $T(n)$ appear as constant factor changes in our final hardness result.

Lemma 4.2. *Let $\mathcal{P}$ be a graph problem that has a $D = T(n)$-round lower bound in the randomized* LOCAL *model with shared randomness. Suppose that $T(n) \leq \log^\gamma n$ for a constant $\gamma \in (0,1)$ and that that there is an $o(\log D)$-round component-stable* MPC *algorithm $\mathcal{A}_{\mathsf{MPC}}$ that solves $\mathcal{P}$ with local memory n^α for a constant $\alpha \in (0,1)$. Then, we obtain an* MPC *algorithm $\mathcal{B}_{\mathsf{st-conn}}$ with local memory n^β for an arbitrary constant $\beta \in (0,1)$ with round complexity $o(\log D)$ for D-diameter s-t connectivity.*

Proof. From Lemma 3.1, we know that for a $t(n)$ such that $\log t(n) = \Theta(\log T(n))$, there are two $t(n)$-hop-isomorphic centered graphs G and G' of size less than n^δ such that $\mathcal{A}_{\mathsf{MPC}}$ is $(t(n), 1/n^\delta)$-sensitive w.r.t. G and G'. We use this to conclude that then there must be an MPC algorithm $\mathcal{B}_{\mathsf{st-conn}}$ with round complexity $o(\log T(n))$ for D-diameter s-t connectivity, where $D = T(n)$.

We now describe how we build the algorithm $\mathcal{B}_{\mathsf{st-conn}}$. This algorithm is made of $O(n^{\delta+o(1)})$ parallel repetitions of some weaker algorithm $\mathcal{B}'_{\mathsf{st-conn}}$, which we will describe later, and that when s and t are endpoints of a path of length at most D it outputs YES with probability at least $n^{-(\delta+o(1))}$ and when s and t are in different components, it outputs NO with high probability. The output of $\mathcal{B}_{\mathsf{st-conn}}$ is YES if an only if the output of at least one of the parallel repetitions of $\mathcal{B}'_{\mathsf{st-conn}}$ is YES. This parallel repetition amplifies the success probability to $1 - 1/\operatorname{poly}(n)$.

Before describing how we build $\mathcal{B}'_{\mathsf{st-conn}}$, let us recall the meaning of the aforementioned sensitivity of $\mathcal{A}_{\mathsf{MPC}}$ w.r.t G and G': We consider two dependent runs of $\mathcal{A}_{\mathsf{MPC}}$, one on G and one on G'. In both runs, each node in G or G' has an independently and uniformly chosen ID from the ID space of algorithm $\mathcal{A}_{\mathsf{MPC}}$, where for $D = t(n)$, the nodes in the isomorphic r-hop neighborhoods $G[N_D(v)]$ and $G'[N_D(v')]$ are assigned the same random IDs. Each edge $\{u, w\}$ of G or G' is labeled by the set of IDs $\{\mathrm{ID}(u), \mathrm{ID}(w)\}$. For each edge label, we independently choose a machine uniformly at random and we give all edges with that label to that machine. That is, in particular, the corresponding edges of $G[N_D(v)]$ and $G'[N_D(v')]$ are given to the same machines in both runs. Then, if the MPC algorithm $\mathcal{A}_{\mathsf{MPC}}$ is run on G and G' with the same shared randomness, the probability that the two center nodes v and v' output different values is at least $1/n^\delta$.

We now describe how we build $\mathcal{B}'_{\mathsf{st-conn}}$. This will be by simulating $\mathcal{A}_{\mathsf{MPC}}$ on G and G'. By the definition of the problem, we only need to say YES, when s and t are endpoints of a path of length at most D. In all other cases, we can output NO. If we are in the case that s-t is a path of length at most D, then we would like that each of the nodes on this path "simulates" exactly one layer of the BFS of G or G', in our run of $\mathcal{A}_{\mathsf{MPC}}$: For instance, node s will simulate node v in the run on G and node v' in the run on G'. Moreover, node t will simulate all of the nodes outside $G[N_r(v)]$ and $G'[N_r(v)]$, in the respective runs. This should allow us to detect that v has a different output in the two runs, with probability at least $1/n^\delta$. However, there are two issues to handle: (1) the length of the s-t path might be less than D so we cannot give exactly one layer to each node, (2) nodes do not know which layer to simulate. We next discuss how we handle these two issues:

Let us first discuss the first issue: If node s has more than one edge, we output NO. Otherwise, node s samples an integer random variable ℓ uniformly distributed in $[1, D]$ and it virtually replaces its edge $\{s, w\}$ in G — or its edge $\{s, w'\}$ in the run on G' — with a path of length ℓ starting at s and ending at v. This length ℓ is a guess for the length to be added to the path so that it becomes

a path of length exactly D. Notice that the guess is correct with probability $1/D \gg n^{-o(1)}$. Then, node s is responsible for simulating the behavior of all the nodes in this path of length ℓ. Let us now think that we are in the good event and we have a path of length exactly D.

We now discuss the second issue: We make each node of degree 2 (including those simulated by s, in the description given above, when replacing one edge with a path of length P_ℓ) guess its BFS layer number from $[1, D-1]$ at random. With probability at least $(1/D)^D \geq n^{-o(1)}$, as $D \leq \log^\gamma n$, the nodes on the s-t path have guessed layers exactly from 1 to D. Now, two consecutive nodes on the path with consecutive layer numbers can communicate with each other and simulate the behavior of each edge that is between the corresponding two consecutive layers of BFS. Node t is responsible for simulating all nodes outside $G[N_D(v)]$ and $G'[N_D(v')]$, in the runs on G and G' respectively. We randomly fix the labels of nodes in $G[N_D(v)]$ and $G'[N_D(v')]$ and the distribution of the related edges on different machines, using the same randomness. Then, we conduct the following two step experiment: in the first step, node t samples the outside topology according to G. In the second step, node t samples the outside topology according to G'. In each case, we then run algorithm $\mathcal{A}_{\mathsf{MPC}}$ on this simulated graph. Algorithm $\mathcal{B}'_{\mathsf{st-conn}}$ outputs YES if and only if the output of s, which is set to be v in the first run and v' in the second run, in algorithm $\mathcal{A}_{\mathsf{MPC}}$ is different in the two steps.

Now we analyze the behavior of algorithm $\mathcal{B}'_{\mathsf{st-conn}}$. If t is in a different component than s, then the fact that t is taking different topologies in different steps does not change the topology, ID assignments, and the distribution of the edges of the component of node s. Hence, the component-stable algorithm $\mathcal{A}_{\mathsf{MPC}}$ will output the same result at node v in both steps, with high probability. Thus, in this case, algorithm $\mathcal{B}'_{\mathsf{st-conn}}$ outputs NO, with high probability. On the other hand, suppose that t is in the same component as s and the component is a path of length at most D. Then our guesses of path length addition ℓ and numbering of the nodes on the path have been correct with probability at least $n^{-o(1)}$ and thus we have a correct simulation of algorithm $\mathcal{A}_{\mathsf{MPC}}$ on G and G'. Then, since $\mathcal{A}_{\mathsf{MPC}}$ is $(t(n), 1/n^\delta)$-sensitive w.r.t. G and G' as we showed in Lemma 3.1, this algorithm would output different values in the two steps with probability at least $n^\delta n^{-o(1)}$. Hence, $\mathcal{B}'_{\mathsf{st-conn}}$ outputs YES, with probability at least $n^{-(\delta+o(1))}$, as desired. □

We can now provide a proof for our main technical theorem.

Proof of Theorem 1.4. The statement of Lemma 4.2 is equivalent to the first claim of Theorem 1.4 (the existence of an $o(\log D)$ strongly sublinear local memory MPC algorithm to solve D-diameter s-t connectivity). By Lemma 4.1, this also implies a strongly sublinear local memory MPC algorithm that distinguishes a single cycle of length n from two cycles of length $n/2$ in time $o(\log n)$. □

5 Distributed Lower Bounds with Shared Randomness

In this section, we prove that existing distributed LOCAL model lower bounds also hold if the nodes have access to shared randomness. We start with the lower bounds of [KMW16] for approximating minimum vertex cover and maximum matching.

Theorem 5.1. *Any randomized* LOCAL *model algorithm to compute an* $\operatorname{poly}\log(n)$*-approximate solution for the minimum vertex cover, maximum matching, or the minimum dominating set problem in* n*-node graphs requires at least* $\Omega\big(\sqrt{\frac{\log n}{\log\log n}}\big)$ *rounds. This holds even if the nodes of the graph know* n *and if they have access to an unlimited amount of shared randomness.*

Proof. The proof follows by using the same argument as in [KMW16]. In [KMW16], for the minimum vertex cover problems, it is shown that there exists an n-node graph $G = (V, E)$ that contains an induced bipartite subgraph $B = (U_0 \dot\cup U_1, E_B)$ such that $|U_0| \geq (1 - 1/(1+2\alpha))n$ and every node

in U_0 has exactly one neighbor in U_1 and such that for some $r = \Theta(\sqrt{\log n / \log\log n})$, the r-hop neighborhoods of all nodes in $u \in U_0 \cup U_1$ are isomorphic. Note that since U_0 is an independent set, the set $V \setminus U_0$ is a vertex cover and thus G has a vertex cover of size at most $n/(1+2\alpha)$. Assume that assignment of node IDs is done uniformly at random. Then all nodes $u \in U_0 \cup U_1$ have the same distribution over the possible r-hop views and thus, in any r-round distributed vertex cover algorithm with a random ID assignment, all nodes in $U_0 \cup U_1$ join the vertex cover with the same probability. This is clearly also true if public randomness is available. Because the edges between U_0 and U_1 need to be covered, this probability has to be at least $n/2$. In expectation, any r-round distributed vertex cover algorithm will therefore compute a vertex cover of size at least $|U_0|/2 \geq \alpha n/(1+2\alpha)$ and thus, the expected approximation ratio is at least α. The lower bound for the minimum dominating set problem follows because a dominating set of size d on the line graph of a graph G directly implies a vertex cover of size $2d$ on G and a vertex cover of size s on G directly implies a dominating set of size s on the line graph of G.

For the maximum matching problem, [KMW16] shows that there is an n-node graph $G = (V, E)$ that contains two disjoint sets of edges E_0 and E_1: For a given $\alpha \leq \operatorname{poly}\log n$ and a suitably chosen constant $c > 1$, E_0 is a matching of size at least $(1 - 1/(c\alpha))n/2$, any matching that does not contain edges of E_0 has size at most $n/(c\alpha)$, and every edge in E_1 shares a vertex with at least $c\alpha$ other edges in E_1. Further, it is shown that for some $r = \Theta(\sqrt{\log n / \log\log n})$, the r-hop neighborhoods of all edges in $E_0 \cup E_1$ are identical. Thus, as before, if the IDs are chosen uniformly at random, even with shared randomness, for all r-round distributed matching algorithms, all edges in $E_0 \cup E_1$ need to join the matching with the same probability. As edges in E_1 share an endpoint with $c\alpha$ other edges in E_1, this probability can be at most $1/(c\alpha)$. However, this means that at most $n/(2c\alpha)$ edges of the matching E_0 of size $(1 - 1/(c\alpha))n/2$ are added to the matching in expectation. By choosing c appropriately, this again implies that the expected approximation ratio cannot be better than α. □

As a maximal matching is a 2-approximation for the maximum matching problem, the above theorem directly also implies the same lower bound for computing a maximal matching. More generally, we obtain the following corollary.

Corollary 5.2. *Any randomized* LOCAL *model algorithm to compute maximal matching or a maximal independent set (MIS) in n-node graphs requires at least $\Omega\big(\sqrt{\frac{\log n}{\log\log n}}\big)$ rounds. This holds even if the nodes of the graph know n and if they have access to an unlimited amount of shared randomness. Furthermore, the maximal matching lower bound even holds on n-node trees.*

Proof. The lower bound for maximal matching on general graphs follows directly because a maximal matching is a 2-approximation for the maximum matching problem (and also for the minimum vertex cover problem). The lower bound holds on trees because in the constructions of [KMW16], the isomorphic local neighborhoods are trees. For maximal matching, where the validity of a solution is defined by a local condition, any violation of this condition at a node v in a graph G also occurs if we only run the algorithm on the local neighborhood of v (and thus on a tree). The lower bound on the MIS problem follows because a maximal matching on G is an MIS on the line graph of G. □

We next discuss a conditional randomized LOCAL lower bound that is particularly interesting for the distributed $(\Delta+1)$-coloring problem. Chang, Kopelowitz, and Pettie [CKP16] showed that for every so-called locally checkable labeling (LCL) problem $\mathcal{P}$, the randomized complexity of $\mathcal{P}$ in the LOCAL model on graphs of size n is at least the deterministic complexity of $\mathcal{P}$ in the LOCAL model on graphs of size $\sqrt{\log n}$. We next show that the argument of [CKP16] also works if the randomized LOCAL algorithm has access to shared randomness.

Theorem 5.3. *Let $\mathcal{G}_{n,\Delta}$ be the family n-graphs with maximum degree Δ and assume that the best deterministic* LOCAL *algorithm for solving a locally checkable graph problem $\mathcal{P}$ on graph in $\mathcal{G}_{n,\Delta}$ has a time complexity of $T(n,\Delta)$. Then, every randomized* LOCAL *algorithm for solving $\mathcal{P}$ on graphs of size n from $\mathcal{G}$ has a time complexity of at least $T(\sqrt{\log n},\Delta)$. This holds even if the nodes of the randomized algorithm know n and if they have access to shared randomness.*

Proof. The proof works in the same way as the proof of [CKP16], we therefore only sketch it here. Assume that we are given a shared memory randomized LOCAL algorithm $\mathcal{A}_R$ for $\mathcal{P}$ that runs in time $t(n,\Delta)$ and succeeds with probability at least $1-1/n$. When applying $\mathcal{A}_R$ on a graph G of size n, each node initially has an $c\log n$-bit unique ID for some constant $c\geq 1$ and access to a shared random bit string $\mathcal{S}$ of sufficient length. Given the ID assignment and $\mathcal{S}$, the behavior of $\mathcal{A}_R$ is deterministic. Let $\mathcal{H}_{n,\Delta}$ be the set of labeled graphs in $\mathcal{G}_{n,\Delta}$, where each node is labeled with a unique $O(\log n)$-bit ID. We say that a shared bit string $\mathcal{S}$ is successful for a labeled graph $G\in\mathcal{H}_{n,\Delta}$ if $\mathcal{A}_R$ computes a correct solution on G when using $\mathcal{S}$ as the shared randomness. If there is a shared bit string $\mathcal{S}^*$ that is successful for all $G\in\mathcal{H}_{n,\Delta}$, we can always run $\mathcal{A}_R$ with $\mathcal{S}^*$ and obtain a deterministic algorithm $\mathcal{A}_D$.

If for each bit string $\mathcal{S}$, there is a graph $G\in\mathcal{H}_{n,\Delta}$ for which $\mathcal{S}$ is not successful, the success probability of $\mathcal{A}_R$ cannot be better than $1-1/|\mathcal{H}_{n,\Delta}|$ (because for some $G\in\mathcal{H}_{n,\Delta}$, at least a $1/|\mathcal{H}_{n,\Delta}|$-fraction of all strings $\mathcal{S}$ are not successful). We have $|\mathcal{H}_{n,\Delta}|\leq 2^{\binom{n}{2}+cn\log n}<2^{n^2}$ as long as $n\geq n_0$ for a sufficiently large constant n_0. Thus, if the success probability of $\mathcal{A}_R$ is $1-2^{-n^2}$, we can derandomize it to a deterministic algorithm. In order to obtain a randomized algorithm with such a high success probability, we define $N:=2^{n^2}$ and we lie to $\mathcal{A}_R$ about the number of nodes n and pretend it is actually N. We can do this by running $\mathcal{A}_R$ on a graph $G'\in\mathcal{G}_{N,\Delta}$, such that the original n-node graph G is a connected component of G'. To obtain the output on the component G, we only need to run the algorithm on G and because $\mathcal{P}$ is a locally checkable labeling problem, any valid output on G' also implies that the output is valid when considering just the connected component G. The running time of the resulting deterministic algorithm is $t(N,\Delta)=t(2^{n^2},\Delta)$. □

The $(\Delta+1)$-coloring problem is a locally checkable problem for which the above argument holds. Recall that the fastest known deterministic LOCAL algorithm for solving $(\Delta+1)$-coloring in general graphs of size n has a time complexity of $2^{O(\sqrt{\log n})}$ [PS92] and showing the existence of a deterministic $2^{o(\sqrt{\log n})}$-round LOCAL algorithm for $(\Delta+1)$-coloring would be considered a major breakthrouh in the area of distributed graph algorithms (see, e.g., [BE13, GKM17]). For $\Delta=2$, i.e., on paths and rings, the deterministic time complexity of $(\Delta+1)$-coloring is known to be $\Omega(\log^* n)$ [Lin87].

Corollary 5.4. *Unless the deterministic complexity of $(\Delta+1)$-coloring in the* LOCAL *model can be improved to $2^{o(\sqrt{\log n})}$, any randomized* LOCAL *model algorithm to compute $(\Delta+1)$-coloring in n-node graphs requires at least $2^{\Omega(\sqrt{\log\log n})}$ rounds. Further, any randomized* LOCAL *model algorithm to compute a $O(1)$-coloring of n-node rings requires at least $\Omega(\log^* n)$ rounds. Both results hold even if the nodes of the graph know n and if they have access to an unlimited amount of shared randomness.*

Proof. The results follow directly from Theorem 5.3 and in the case of the unconditional lower bound for ring networks, from Linial's $\Omega(\log^* n)$ lower bound for deterministically coloring n-node rings [Lin87]. □

We remark that in [Nao91], Naor showed that the randomized time complexity of coloring an n-node ring with $O(1)$ colors is $\Omega(\log^* n)$. However, the result does not trivially extend to algorithms that have access to shared randomness and we therefore used Theorem 5.3 to prove this lower in a straightforward way. As the following corollary shows, we can also use Theorem 5.3 to lift the $\Omega(\log\log n)$ randomized lower bound for sinkless orientation of Brandt et al. [BFH+16] to a setting with shared randomness.

Corollary 5.5. *Any randomized* LOCAL *model algorithm to compute a sinkless orientation requires* $\Omega(\log \log n)$ *rounds. This holds even if the nodes of the graph know* n *and if they have access to an unlimited amount of shared randomness.*

Proof. In [CKP16], Chang et al. show that computing a sinkless orientation deterministically in the LOCAL model requires at least $\Omega(\log n)$ rounds. By Theorem 5.3, we therefore obtain an $\Omega(\log \log n)$ lower bound for randomized algorithms even if shared randomness is available. □

References

[ABB+19] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs. In *the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1616–1635, 2019.

[ACK19] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear Algorithms for (Δ+ 1) Vertex Coloring. In *the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 767–786. SIAM, 2019.

[AG15] Kook Jin Ahn and Sudipto Guha. Access to Data and Number of Iterations: Dual Primal Algorithms for Maximum Matching Under Resource Constraints. In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 202–211, 2015.

[AGM12a] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing Graph Structure via Linear Measurements. In *the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467. SIAM, 2012.

[AGM12b] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph Sketches: Sparsification, Spanners, and Subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 5–14. ACM, 2012.

[ANOY14] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel Algorithms for Geometric Graph Problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 574–583, 2014.

[Ass17] Sepehr Assadi. Simple Round Compression for Parallel Vertex Cover. *arXiv preprint arXiv:1709.04599*, 2017.

[ASS+18] Alexandr Andoni, Clifford Stein, Zhao Song, Zhengyu Wang, and Peilin Zhong. Parallel Graph Connectivity in Log Diameter Rounds. In *the Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 674–685, 2018.

[ASW18] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs. *arXiv preprint arXiv:1805.02974*, 2018.

[BBD+19] Soheil Behnezhad, Sebastian Brandt, Masha Derakhshan, Manuela Fischer, Mohammad-Taghi Hajiaghayi, Richard M. Karp, and Jara Uitto. Massively parallel computation of matching and MIS in sparse graphs. *Manuscript*, 2019.

[BE13] L. Barenboim and M. Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers, 2013.

[BEG+18] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating Edit Distance in Truly Subquadratic Time: Quantum and Mapreduce. In *the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1170–1189, 2018.

[BFH+16] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A Lower Bound for the Distributed Lovász Local Lemma. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 479–488. ACM, 2016.

[BFU18] Sebastian Brandt, Manuela Fischer, and Jara Uitto. Breaking the linear-memory barrier in mpc: Fast mis on trees with n^ϵ memory per machine. *arXiv preprint arXiv:1802.06748*, 2018.

[BHH19] Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G Harris. Exponentially faster massively parallel maximal matching. *arXiv preprint arXiv:1901.03744*, 2019.

[BKS13] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 273–284, 2013.

[BKS14] Paul Beame, Paraschos Koutris, and Dan Suciu. Skew in parallel query processing. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 212–223, 2014.

[CFG+19] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The Complexity of $(\Delta + 1)$ Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. *Manuscript*, 2019.

[CKP16] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An Exponential Separation Between Randomized and Deterministic Complexity in the LOCAL Model. In *the Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2016.

[CLM+18] Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Round Compression for Parallel Matching Algorithms. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 471–484, 2018.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation (OSDI)*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[GGK+18] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *the Proceedings of the Symposium on Principles of Distributed Computing (PODC)*. arXiv:1802.08237, 2018.

[GKM17] M. Ghaffari, F. Kuhn, and Y. Maus. On the complexity of local distributed graph problems. In *Proc. 49th ACM Symp. on Theory of Computing (STOC)*, pages 784–797, 2017.

[GKMS18] Buddhima Gamlath, Sagar Kale, Slobodan Mitrović, and Ola Svensson. Weighted matchings via unweighted augmentations. *arXiv preprint arXiv:1811.02760*, 2018.

[GSZ11] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the MapReduce framework. In *the Proceedings of the International Symposium on Algorithms and Computation (ISAAC)*, pages 374–383. Springer, 2011.

[GU19] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1636–1653. SIAM, 2019.

[HLL18] Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. Greedy and Local Ratio Algorithms in the MapReduce Model. In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 43–52, 2018.

[HP15] James W Hegeman and Sriram V Pemmaraju. Lessons from the congested clique applied to mapreduce. *Theoretical Computer Science*, 608:268–281, 2015.

[IBY+07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Operating Systems Review*, 41(3):59–72, 2007.

[IMS17] Sungjin Im, Benjamin Moseley, and Xiaorui Sun. Efficient Massively Parallel Methods for Dynamic Programming. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 798–811, 2017.

[JN18] Tomasz Jurdziński and Krzysztof Nowicki. MST in O(1) Rounds of Congested Clique. In *the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2620–2632. SIAM, 2018.

[KMW16] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local Computation: Lower and Upper Bounds. *J. ACM*, 63(2):17:1–17:44, 2016.

[KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010.

[Lin87] Nathan Linial. Distributive Graph Algorithms-Global Solutions from Local Data. In *the Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 331–335, 1987.

[LMSV11] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a Method for Solving Graph Problems in MapReduce. In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 85–94, 2011.

[Nao91] Moni Naor. A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. *SIAM J. Discrete Math.*, 4(3):409–412, 1991.

[Ona18] Krzysztof Onak. Round Compression for Parallel Graph Algorithms in Strongly Sublinear Space. *CoRR*, abs/1807.08745, 2018.

[PS92] Alessandro Panconesi and Aravind Srinivasan. Improved Distributed Algorithms for Coloring and Network Decomposition Problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 581–592. ACM, 1992.

[RVW16] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and Circuits: (On Lower Bounds for Modern Parallel Computation). In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–12, 2016.

[Whi12] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2010.