




Exponential Speedup over Locality in MPC with Optimal Memory

Alkida Balliu  
Gran Sasso Science Institute, L'Aquila, Italy

Sebastian Brandt  
CISPA Helmholtz Center for Information Security,
Saarbrücken, Germany

Manuela Fischer  
ETH Zürich, Switzerland

Rustam Latypov  
Aalto University, Espoo, Finland

Yannic Maus  
TU Graz, Austria

Dennis Olivetti  
Gran Sasso Science Institute, L'Aquila, Italy

Jara Uitto  
Aalto University, Espoo, Finland

Abstract

Locally Checkable Labeling (LCL) problems are graph problems in which a solution is correct if it satisfies some given constraints in the local neighborhood of each node. Example problems in this class include maximal matching, maximal independent set, and coloring problems. A successful line of research has been studying the complexities of LCL problems on paths/cycles, trees, and general graphs, providing many interesting results for the LOCAL model of distributed computing. In this work, we initiate the study of LCL problems in the low-space Massively Parallel Computation (MPC) model. In particular, on forests, we provide a method that, given the complexity of an LCL problem in the LOCAL model, automatically provides an exponentially faster algorithm for the low-space MPC setting that uses optimal global memory, that is, truly linear.

While restricting to forests may seem to weaken the result, we emphasize that all known (conditional) lower bounds for the MPC setting are obtained by lifting lower bounds obtained in the distributed setting *in tree-like networks* (either forests or high girth graphs), and hence the problems that we study are challenging already on forests. Moreover, the most important technical feature of our algorithms is that they use optimal global memory, that is, memory linear in the number of edges of the graph. In contrast, most of the state-of-the-art algorithms use more than linear global memory. Further, they typically start with a dense graph, sparsify it, and then solve the problem on the residual graph, exploiting the relative increase in global memory. On forests, this is not possible, because the given graph is already as sparse as it can be, and using optimal memory requires new solutions.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed computing, Locally checkable labeling problems, Trees, Massively Parallel Computation, Sublinear memory

Digital Object Identifier 10.4230/LIPIcs.DISC.2022.9

Related Version *Full Version*: <https://arxiv.org/abs/2208.09453>

Funding *Rustam Latypov*: Supported in part by the Academy of Finland, Grant 334238.

1 Introduction

The Massively Parallel Computation (MPC) model, introduced in [42] and later refined by [2, 12, 39], is a mathematical abstraction of modern data processing platforms such as MapReduce [28], Hadoop [55], Spark [56], and Dryad [41]. Recently, tremendous progress has been made on fundamental *graph problems* in this model, such as maximal independent set (MIS), maximal matching (MM) [37, 25], and coloring problems [19, 27]. All these



© Alkida Balliu, Sebastian Brandt, Manuela Fischer, Rustam Latypov, Yannic Maus, Dennis Olivetti, and Jara Uitto;

licensed under Creative Commons License CC-BY 4.0

36th International Symposium on Distributed Computing (DISC 2022).

Editor: Christian Scheideler; Article No. 9; pp. 9:1–9:21



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problems, and many others, fall under the umbrella of *Locally Checkable* problems, in which the feasibility of a solution can be checked by inspecting local neighborhoods. They also serve as abstractions for fundamental primitives in large-scale graph processing and have recently gained a lot of attention [9, 7, 27, 29, 3, 18, 32]. *Locally checkable labelings (LCLs)* are locally checkable problems restricted to constant degree graphs. A more formal definition of LCLs is deferred to Section 2.

LCLs have been a rich source of research in various models of computation, because they can be seen as a starting point to understand locally checkable problems in general, and this holds independently of the model. For example, in the distributed setting, techniques developed to understand LCLs [16] have then been used to prove lower bounds in the unbounded degree setting, which the LCL setting does not include, e.g., for the maximal independent set problem, or the Δ -coloring problem [4, 6, 5]. In the distributed LOCAL model of computing, a lot is known about LCLs: for example, if the graph on which we want to solve the problem is a tree, then there is a discrete set of possible complexities, and in some cases, given an LCL, we can even automatically decide its distributed time complexity. Our goal is to bring to the parallel setting, and in particular to the MPC model, the knowledge that researchers developed about LCLs in the distributed setting, while also developing new techniques that can be used in the parallel setting. We show that, on forests, the mere knowledge of what is the distributed complexity of a problem is enough to obtain blazingly fast algorithms in the MPC setting. In particular, we obtain MPC algorithms that are *exponentially faster* than the best distributed ones. We summarize our main result.

The complexity of any LCL problem on forests in the MPC model is exponentially lower than its distributed complexity, even when using optimal memory bounds.

More in detail, in our work, we solve LCL problems in forests in the most restrictive *low-space* MPC model with *linear* total memory, which is the most *scalable* variant of the MPC model. Our results provide an automatic method that, for all LCL problems, yields an algorithm that solves the given problem *exponentially* faster than its optimal distributed counterpart. The resulting algorithms are *component-stable* [35, 26], which implies that the solutions in individual connected components are independent of the other components. Our results are in some sense optimal: for problems that in the LOCAL model can be solved in $n^{o(1)}$, finding more-than-exponentially faster component-stable algorithms would violate the widely-believed 1 vs. 2 cycle conjecture in the MPC setting.

Why do we care about trees and forests? All known conditional lower bounds¹ for problems in the MPC setting are derived by lifting lower bounds that hold in the LOCAL model of distributed computing [35, 26]. Most of the lower bounds known in the LOCAL model are actually proved either on trees or on high-girth graphs (where the neighborhood of each node corresponds to a tree): see, e.g., [44, 5, 4, 6, 16]. It follows that essentially all the conditional lower bounds known in the MPC setting already hold on forests². Despite this fact, with a few exceptions, there is no work on upper bounds on forests in the MPC model – a gap we aim to fill.

¹ Proving unconditional lower bounds for the MPC model would imply a major breakthrough in circuit complexity and seems out of reach [53].

² As lifting lower bounds from the LOCAL model to the MPC model requires hereditary graph classes one cannot immediately lift a lower bound in the LOCAL model that holds on trees. Instead, a lower bound in the LOCAL model on trees implies the same lower bound in the LOCAL model for forests which can then be lifted to a lower bound for MPC algorithms on forests.

Moreover, understanding the complexity of problems on trees has been already shown to be essential in the LOCAL model: it is typically the case that interesting problems are already challenging on trees, and often even in regular balanced trees of small degree. In fact, most lower bounds known in the LOCAL model hold exactly in this setting. Due to the lifting, the same statement adapted to forests is true for all recent MPC lower bounds. Hence, to decrease the relevance of trees and forests, we either need completely new lower bound techniques in the LOCAL model coupled with completely new lifting theorems, or completely new lower bound techniques for the MPC model.

At first glance it may seem that our results are easy to achieve, because we restrict to forests. Conversely, we would like to emphasize that many state-of-the-art algorithms for problems like MIS and coloring work as follows [37, 25]: start with a dense graph which requires a lot of memory to store, sparsify it, and then use the freed global memory to solve the problem faster on the sparsified part. On forests, this is not possible, because the given graph is already as sparse as it can be.

The MPC Model. In the MPC model, we have M machines who communicate in an all-to-all fashion. We focus on problems where the input is modeled as a graph with n vertices, m edges and maximum degree Δ ; we call this graph the *input graph*. Each node has a unique ID of size $b = O(\log n)$ bits from a domain $\{1, 2, \dots, N\}$, where $N = \text{poly}(n)$. Each node and its incident edges are hosted on a machine(s) with $S = O(n^\delta)$ *local memory*, where $\delta \in (0, 1)$ and the units of memory are *words* of $O(\log n)$ bits. When the local memory is bounded by $O(n^\delta)$, the model is called *low-space* (or *sublinear*). The number of machines is chosen such that $M = m/S = \Theta(m/n^\delta)$. For trees, where $m = \Theta(n)$, this results in $\Theta(n^{1-\delta})$ machines, that is, a *total memory* (or *global memory*) of $M \cdot S = \Theta(n)$. For simplicity³, we assume that each machine i simulates one virtual machine for each node and its incident edges that i hosts, such that the local memory restriction becomes that no virtual machine can use more than $O(n^\delta)$ memory.

During the execution of an MPC algorithm, computation is performed in synchronous, fault-tolerant rounds. In each round, every machine performs some (unbounded) computation on the locally stored data, then sends/receives messages to/from any other machine in the network. Each message is sent to exactly one other machine specified by the sending machine. All messages sent and received by each machine in each round, as well as the output, have to fit into local memory. The time complexity is the number of rounds it takes to solve a problem. Upon termination, each node (resp. its hosting machine) must know its own part of the solution. For example in the case of node-coloring, the machine hosting node u must decide on the color of u upon termination of the algorithm.

Unlike in most other works, our algorithms employ $O(m + n)$ words of total memory, which is the strictest possible as it is only enough to store a constant number of copies of the input graph. Note that if we were to allow superlinear $O(m^{1+\delta})$ global memory in our constant-degree setting, many LOCAL algorithms with complexity $O(\log n)$ could be trivially sped up exponentially in the low-space MPC model by applying the well-known graph exponentiation technique by Lenzen and Wattenhofer [45]. A crucial challenge that comes with the linear global memory restriction is that only a small fraction of $n^{1-\delta}$ of the (virtual) machines can simultaneously utilize all of their available local memory. Thus, with

³ In practice, it is assumed that the virtual machines can be shuffled between physical machines, such that the sum of the memory of the virtual machines hosted on any single physical machine is $O(n^\delta)$.

strictly linear global memory we are forced to develop new techniques which must avoid gathering local neighborhoods, i.e., fundamentally divert from direct simulations of message passing algorithms.

1.1 The Distributed Complexity Landscapes

In the last decade, there has been tremendous progress in understanding the complexities of LCLs in various models of distributed and parallel computing. A prime example is the LOCAL model [46], where the input graph corresponds to a message passing system, and the nodes must output their part of the solution according only to local information about the graph. Another example is the CONGEST model, which is a LOCAL model variant where the message size is restricted to $O(\log n)$ bits [52]. A curious fact about LCLs in the distributed setting is the existence of *complexity gaps*, that is, some complexities are not possible at all. For example, it is known that there are no LCLs with a distributed time complexity in the LOCAL and CONGEST model that lies between $\omega(\log^* n)$ and $o(\log n)$. In these two models, the *whole* complexity landscape of LCL problems is now understood for some important graph families. For instance, a rich line of work [50, 20, 22, 11, 8, 3, 18] recently came to an end when a complexity gap between $\omega(1)$ and $o(\log^* n)$ was proved [40], completing the randomized/deterministic complexity landscape of LCL problems in the LOCAL model for trees. In the CONGEST model, the authors of [9] showed that, on trees, the complexity of an LCL problem is asymptotically equal to its complexity in the LOCAL model, whereas the same does not hold in general graphs. In the randomized/deterministic LOCAL and CONGEST models, recent work showed that the complexity landscapes of LCL problems for rooted regular trees are fully understood [7], while the complexity landscapes of LCL problems in the LOCAL model for rings and tori have already been known for some while [17]. Even for general (constant-degree) graphs, the LOCAL complexity landscape of LCL problems is almost fully understood [50, 16, 21, 22, 36, 30, 34, 10, 8, 54, 33], only missing a small part of the picture related to the randomized complexity of Lovász Local Lemma (LLL).

In the case of trees, for deterministic algorithms in the LOCAL model, it is known that there is a discrete set of possible complexities, that we divide into four categories:

- **Tiny regime:** contains the complexities $O(1)$ and $\Theta(\log^* n)$.
 - Example problems: maximal independent set, maximal matching, $(\Delta + 1)$ -vertex coloring⁴, $(2\Delta - 1)$ -edge coloring, and trivial problems (e.g., all nodes must output 0).
- **Mid regime:** contains the complexity $\Theta(\log n)$.
 - Example problems: sinkless orientation [16], 3-coloring, and Δ -coloring.
- **High regime:** contains the complexities $\Theta(n^{1/k})$, for all $k \in \mathbb{N}$.
 - Example problems: 2-coloring and $2\frac{1}{2}$ -coloring [22].

Moreover, it is known that randomness can help only in the mid regime, and in particular that *some* problems requiring $\Theta(\log n)$ for deterministic algorithms have randomized complexity $\Theta(\log \log n)$, which constitutes our fourth category – **Low regime**. Problems residing in the low regime include sinkless orientation and Δ -coloring.

On forests, the complexity landscape in the LOCAL model is the same as on trees. While this is intuitively evident, it can also be shown formally using an analogous approach to the one used in the proof of [40, Lemma 3.3].

⁴ We denote the maximum degree of the graph by Δ .

1.2 Our Contributions

Our main contribution is showing that, given any LCL problem (see Definition 4) on trees that has deterministic (resp. randomized) complexity T in the LOCAL model, we can automatically obtain an MPC algorithm with deterministic (resp. randomized) complexity $O(\log T)$ on forests. In particular, we prove the following.

► **Theorem 1.** *Consider an LCL problem on trees with deterministic time complexity $f(n)$ and randomized time complexity $g(n)$ in the LOCAL model. This problem has deterministic time complexity $O(\log f(n))$ and randomized time complexity $O(\log g(n))$ in the low-space MPC model on forests using optimal $O(m+n)$ words of global memory. The provided algorithms are component-stable.*

Put differently, a problem in the LOCAL model can only have a deterministic complexity $f(n) \in \{\Theta(1), \Theta(\log^* n), \Theta(\log n)\} \cup \{\Theta(n^{1/k}) \mid k \in \mathbb{N}\}$, and we show that it is enough to know the asymptotic value of $f(n)$ in order to obtain a deterministic MPC algorithm with complexity $O(\log(f(n))) \in \{O(1), O(\log \log^* n), O(\log \log n), O(\log n)\}$.

Moreover, it is known that for all $f(n) \notin \Theta(\log n)$, the LOCAL randomized complexity of the problem is the same as the deterministic one. Instead, for $f(n) \in \Theta(\log n)$, the LOCAL randomized complexity $g(n)$ can be either $\Theta(\log n)$ or $\Theta(\log \log n)$. If it is $\Theta(\log \log n)$, then we provide an MPC algorithm with randomized complexity $O(\log \log \log n)$. If we dismiss the component-stability requirement, we can obtain the same $O(\log \log \log n)$ runtime with a deterministic MPC algorithm.

► **Theorem 2.** *Consider an LCL problem on trees with randomized time complexity $g(n) = \Theta(\log \log n)$ in the LOCAL model. This problem has deterministic time complexity $O(\log \log \log n)$ in the low-space MPC model on forests using optimal $O(m+n)$ words of global memory. This algorithm is component-unstable.*

By [35, 26], we know that Theorem 1 is in some sense optimal: if a problem requires T deterministic rounds in the LOCAL model, then it requires $\Omega(\min\{\log T, \log \log n\})$ rounds in the low-space MPC setting for component-stable algorithms, assuming that the infamous 1 vs. 2 cycle conjecture holds [12, 35, 53]. In contrast, Theorem 2 shows that one can break the conditional lower bound of $\Omega(\log \log n)$ for deterministic MPC algorithms for all LCL problems in the aforementioned class by diverting to component-unstable algorithms. Achieving the same result even for a single problem without dismissing the component-stability requirement would be a major breakthrough, as it would falsify the conjecture.

As a subroutine for solving all problems that belong to the high regime in $O(\log n)$ MPC rounds, we also develop an $O(\log n)$ round MPC algorithm for rooting a forest. This rooting algorithm is component-stable, and may be of independent interest, since it is also compatible with arbitrary degrees.

Additional observations. There is a long line of research that provided algorithms for MPC that are exponentially faster than the best algorithms for the LOCAL model. Most existing results achieved these speedup results by using additional global memory, that is, $\omega(m)$ words [13, 32, 27, 26]. We emphasize that, deviating from the usual approach, all of our results use *optimal* MPC parameters, in the sense that we work in the low-space setting with $O(n^\delta)$ words of local memory and $O(m+n)$ words of global memory.

Hence, our contribution is twofold, on the one hand we prove that we can indeed achieve this exponential speedup for all LCLs, while on the other hand we show that this exponential speedup can be achieved without requiring any additional memory. Furthermore, graph

problems in trees and forests are widely unexplored, despite their central role that we have already elaborated on. It is known that a 4-coloring, MIS, and maximal matching can be found in $O(\log \log n)$ rounds [32]. However, the coloring result heavily relies on randomness and the MIS and matching results require a (small) overhead in the total memory. To compare, our results deterministically yield a 3-coloring in $O(\log \log n)$ rounds with linear total memory. It is not clear whether randomness can even help in the case of 3-coloring, which is a significant difference to the case of 4-coloring. Furthermore, it is not clear whether the previous approaches to MIS and matching can be extended to work deterministically with the same runtime and with linear total memory. While the previous work is designed for arbitrary degree graphs, it is not clear whether the algorithms could be tuned to work faster with constant degrees.

Open Questions. In the tiny regime, our results extend to general graphs (see Theorem 5). In the low regime, our results extend to general graphs if we allow slightly more global memory (see Theorem 6.5 in the full version). Once we reach the mid regime, i.e., logarithmic distributed complexities, we do not know the behaviour in general graphs. This leads to an interesting open question. As mentioned, the asymptotic complexity of any problem on trees is identical in the LOCAL and CONGEST model, and the same is true (modulo the exact complexity of the LLL in both models) on general graphs as long as the complexity is sublogarithmic [9]. However, there is an exponential separation between the models for complexities that are at least logarithmic [9]. Does such a separation between the complexity of an LCL in the LOCAL model and the MPC model also hold for large complexities? Here, of course, we would want to have a doubly exponential separation.

Interestingly, current conditional lower bounds for the MPC model cannot prove MPC lower bounds that are $\omega(\log \log n)$. So, while our results in the high regime show that any problem on forests can be solved in $O(\log n)$ rounds in the MPC model, it remains unclear whether we cannot improve on this bound, even without falsifying the 1 vs. 2 cycle conjecture.

Component-stability. The term of a *component-stable* MPC algorithm has been introduced in [35] in the context of lifting distributed lower bounds to the MPC setting. By their definition, informally, an algorithm is component-stable if the output of a node does not change if other connected components in the graph are altered (see Definition 11).

While initially believed that it might be an artifact of their lifting techniques, Czumaj, Davies and Parter [26] showed the contrary, i.e., they showed that *component-unstable* algorithms can beat the conditional lower bounds of [35]. Their results hold assuming their revised definition of component-stability, which is argued to be more robust (see Definition 13). Under their definition, it is not strictly easier nor harder to design algorithms to be component-stable, as compared to the definition of [35]. The main difference is that they allow the output of component-stable algorithms to depend on the total number of nodes in the graph and the maximum degree. In our work, we adopt the revised definition of component-stability [26]. See Appendix A and the discussion therein for further details.

1.3 Challenges & Key Techniques

We now provide an overview of the challenges that we had to tackle in order to prove our results, and a very high level explanation of the key techniques that we used to solve them.

The tiny regime serves as a good warm-up to see why using an optimal amount of global memory is difficult. The most technically involved part is the high regime, where we obtain an $O(\log n)$ -time MPC algorithm for any LCL problem.

Graph Exponentiation. A reoccurring challenge for all regimes lies in respecting the linear global memory, which roughly means that on average, every node can use only a constant amount of memory. This is particularly unfortunate because almost all recent MPC results – and in particular all that achieve exponential speedups – rely on the memory-intense *graph exponentiation* technique [45]. Informally, this technique enables a node to gather its 2^k -hop neighborhood in k communication rounds. Doing this in parallel for every node in the graph results in a Δ^{2^k} overhead in global memory. For this technique to be useful, k has to be $\omega(1)$, yielding a non-constant multiplicative increase in the global memory requirement. In order to use this technique but not violate linear global memory, we develop new solutions that are discussed in the following paragraphs.

Tiny regime $f(n) = \Theta(1)$ and $f(n) = \Theta(\log^* n)$. Handling the $\Theta(1)$ complexity is trivial, since any LOCAL algorithm for LCLs can be simulated in the MPC setting. For the $\Theta(\log^* n)$ class, it is known from prior work that all problems can be solved in the LOCAL model in a very specific way: reduce to the problem of computing a distance- k coloring with a small enough number of colors, where k is a constant that depends on the problem. In a distance- k c -coloring, each node is assigned a color in $\{1, \dots, c\}$ such that nodes at distance at most k have different colors. Such a coloring can be computed in $O(\log^* n)$ rounds in the LOCAL model, and it could be computed easily in the MPC setting in $O(\log \log^* n)$ rounds, by exploiting the graph exponentiation technique, if we allow an additional $O(\log^* n)$ factor overhead in the amount of global memory.

We show that this overhead is not required, by developing a novel MPC algorithm for coloring. The algorithm that we provide reduces the problem of coloring a general graph to coloring directed pseudoforests, that is, graphs where all edges are oriented and every node has at most one outgoing edge. Then, we show that in directed pseudoforests, it is possible to solve the coloring problem through a variant of graph exponentiation that only requires keeping track of a constant number of IDs. This way, the memory use of each node is constant, and the global memory is linear.

High regime $f(n) = \Theta(n^{1/k})$, for all $k \in \mathbb{N}$. We explicitly provide, for any solvable LCL, a novel algorithm that has a runtime of $O(\log n)$. Essentially, we solve each tree in the forest separately, hence we will consider trees in the following argumentation. On a high level, our algorithm first roots the tree using our $O(\log n)$ -time tree rooting algorithm, and then proceeds in two phases. In the first phase, roughly speaking, the goal is to compute, for a substantial number of nodes v , the set of possible output labels that can be output at v such that the label choice can be extended to a (locally) correct solution in the subtree hanging from v . This is done in an iterative manner, proceeding from the leaves towards the root. The second phase consists of using the computed information to solve the given LCL from the root downwards.

While this outline sounds simple, there are a number of intricate challenges that require the development of novel techniques, both in the design of the algorithm and its analysis. For instance, the depth of the input tree may be $\omega(\log n)$ (which prevents us from performing the above ideas in a sequential manner), and the storage of the required completeness information grows exponentially when using graph exponentiation, exceeding the available global memory. Our key technical contributions are the following.

- The design of a process that allows for interleaving graph exponentiation steps and compressing the graph (and compatibility information) such that the process is also reversible (second phase of the algorithm). The main challenge here is that multiple graph exponentiation processes executed on individual parts of the tree have to be merged, simultaneously or at different times, into one process during the execution.

- The design of a fine-tuned potential function for the analysis of the complex algorithm resulting from addressing the aforementioned issues and the highly non-sequential behavior arising from interleaving graph exponentiation steps.

Mid regime $f(n) = \Theta(\log n)$. We would wish to use the algorithm of Chang and Pettie [22] as a black box. On a very high level idea, their LOCAL algorithm uses $O(\log n)$ rounds to compute a rake-and-compress decomposition of size $O(\log n)$, which is essentially the classic H -partition by Miller and Reif [49]. Then, compatibility information of the given LCL problem is propagated layer by layer to the top, and then labels are fixed at the top and propagated down.

Applying known MPC techniques like graph exponentiation to speed up this process does not work out of the box for several reasons. First, the compatibility information they propagate grows exponentially, which creates congestion in the MPC model. Secondly, since the input graph is as sparse as it could possibly be, the direct application of graph exponentiation would violate the optimal global memory bounds we are striving for. We resolve the first issue by first observing that the compatibility information can be reduced to constant size in every iteration. The second issue is remedied by interleaving exponentiation steps with memory freeing steps in a balanced way.

Low regime $g(n) = \Theta(\log \log n)$. With an additional $O(\log n)$ factor of global memory, this result is easy to obtain. Previous work [9] has a constant time reduction to instances of size $N = \log n$, resulting in a LOCAL algorithm with runtime $\text{poly}(\log N) = \text{poly}(\log \log n)$. A straightforward application of graph exponentiation would yield an MPC algorithm with runtime $O(\log \log \log n)$. Exploiting additional global memory in this manner has been used in a similar setting in [26]. However, without the additional memory it is harder to solve the small instances in triple logarithmic time. The work around for this memory issue is to use our mid regime algorithm on the small instances, yielding a memory efficient algorithm with runtime $O(\log \log \log n)$. To the best of our knowledge there is no other paper that can efficiently deal with such occurring small instances – small instances occur also in many other problems like MIS and graph coloring – with optimal global memory.

1.4 Further Related Work

For many of the classic graph problems, simple $O(\log n)$ -time MPC algorithms follow from classic literature in the LOCAL model and PRAM [1, 46, 48]. In particular in the case of bounded degree graphs, it is often straightforward to simulate algorithms from other models. However, it is usually desirable to get algorithms that run *much faster* than their LOCAL counterparts. If the MPC algorithms are given *linear* $\Theta(n)$ or even *superlinear* $\Theta(n^{1+\delta})$ local memory, fast algorithms are known for many classic graph problems.

In the sublinear (or low-space) model, [19] provided a randomized algorithm for the $(\Delta+1)$ -coloring problem that, combined with the new network decomposition results [54, 33], yields an $O(\log \log \log n)$ MPC algorithm, that is exponentially faster than its LOCAL counterpart. A recent result by Czumaj, Davies, and Parter [27] provides a deterministic $O(\log \log \log n)$ -time algorithm for the same problem using derandomization techniques. For many other problems, the current state of the art in the sublinear model is still far from the aforementioned exponential improvements over the LOCAL counterparts, at least in the case of general graphs. For example, the best known MIS, maximal matching, $(1 + \epsilon)$ -approximation of maximum matching, and 2-approximation of minimum vertex cover algorithms run in $\tilde{O}(\sqrt{\log \Delta} + \sqrt{\log \log n})$ time [37], whereas the best known LOCAL algorithm has a

logarithmic dependency on Δ [31]. For restricted graph classes, such as trees and graphs with small arboricity⁵ α , better algorithms are known [15, 13]. Through a recent work by Ghaffari, Grunau and Jin, the current state of the art for MIS and maximal matching are $O(\sqrt{\log \alpha} \cdot \log \log \alpha + \log \log n)$ -time algorithms using $\tilde{O}(n+m)$ words of global memory [32].

As for lower bounds, [35] gave conditional lower bounds of $\Omega(\log \log n)$ for component-stable sublinear MPC algorithms for constant approximation of maximum matching and minimum vertex cover, and MIS. In addition, the authors provided a lower bound of $\Omega(\log \log \log n)$ for LLL. Their hardness results are conditioned on a widely believed conjecture in MPC about the complexity of the connectivity problem, which asks to detect the connected components of a graph. It is argued that disproving this conjecture would imply rather strong and surprising implications in circuit complexity [53]. When assuming component-stability, they also argue that all known algorithms in the literature are component-stable or can easily be made component-stable with no asymptotic increase in the round complexity. However, recent work [26] gave a separation between stable and unstable algorithms, and that some particular problems (e.g., computing an independent set of size $\Omega(n/\Delta)$) can be solved faster with unstable algorithms than with stable ones.

It is also worth discussing the complexity of rooting a tree, as it is an important subroutine in our high regime. On the randomized side, [15] gave an $O(\log d \cdot \log \log n)$ time algorithm, where d is the diameter of the graph. On the deterministic side, Coy and Czumaj [24] gave an $O(\log n)$ time algorithm using (component-unstable) derandomization methods, which is the current state of the art. In the full version we provide a totally different rooting algorithm that is also deterministic and takes $O(\log n)$ time, but is component-stable. We note that [43] uses similar techniques in a more general setting, but in $\omega(\log n)$ time.

1.5 Outline

After the formal introduction of LCL problems and other notations in Section 2, we start proving the exponential speedup for the different regimes in Theorem 1 in separate sections. In Section 3, we warm up with the tiny regime. In Section 4, we present the high level ideas for our most involved result, the exponential speedup for the high regime. We provide full proofs and handle the remaining regimes in the full version of the paper.

Some of our speedup results use a description of a distributed algorithm with the claimed runtime to obtain the speedup. In the full version of the paper, we show that such a description can be inferred merely by knowing the distributed complexity class in which the problem resides.

2 Definitions and Notation

We work with undirected, finite, simple graphs $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges such that $E \subseteq [V]^2$ and $V \cap E = \emptyset$. Let $\deg_G(v)$ denote the degree of a node v in G and let Δ denote the maximum degree of G . The distance $d_G(v, u)$ between two vertices v, u in G is the length of a shortest $v - u$ path in G ; if no such path exists, we set $d_G(v, u) := \infty$. The greatest distance between any two vertices in G is the diameter of G , denoted by $\text{diam}(G)$. For a subset $S \subseteq V$, we use $G[S]$ to denote the subgraph of G induced by nodes in S . Let G^k , where $k \in \mathbb{N}$, denote the k :th power of a graph G , which is another graph on the same

⁵ The arboricity of a graph is the minimum number of disjoint forests into which the edges of the graph can be partitioned.

vertex set, but in which two vertices are adjacent if their distance in G is at most k . In the context of MPC, G^k is the resulting virtual graph after performing $\log k$ steps of graph exponentiation [45].

For each node v and for every radius $k \in \mathbb{N}$, we denote the k -hop (or k -radius) neighborhood of v as $N^k(v) = \{u \in V : d(v, u) \leq k\}$. The topology of a neighborhood $N^k(v)$ of v is simply $G[N^k(v)]$. However, with slight abuse of notation, we sometimes refer to $N^k(v)$ both as the node set and the subgraph induced by node set $N^k(v)$. Neighborhood topology knowledge is often referred to as vision, e.g., node v sees $N^k(v)$. In trees and forests, the number n of nodes and the number m of edges are asymptotically equal, and we may use them interchangeably throughout the paper when reasoning about global memory.

2.1 LCL Definitions

In their seminal work [50], Naor and Stockmeyer introduced the notion of a locally checkable labeling problem (LCL problem or just LCL for short). The definition they provide restricts attention to problems where nodes are labeled (such as vertex coloring problems), but they remark that a similar definition can be given for problems where edges are labeled (such as edge coloring problems). A modern way to define LCL problems that captures both of the above types of problems (and combinations thereof) labels *half-edges* instead, i.e., pairs (v, e) where e is an edge incident to vertex v . Moreover, on trees it is known that all LCL problems can be rephrased in a special form, called *node-edge-checkable LCL problems* [9, 40]. Let us first define a half-edge labeling formally, and then provide this modern LCL problem definition.

► **Definition 3 (Half-edge labeling).** *A half-edge in a graph $G = (V, E)$ is a pair (v, e) , where $v \in V$ is a vertex, and $e \in E$ is an edge incident to v . A half-edge (v, e) is incident to some vertex w if $v = w$. We denote the set of half-edges of G by $H = H(G)$. A half-edge labeling of G with labels from a set Σ is a function $g: H(G) \rightarrow \Sigma$.*

We distinguish between two kinds of half-edge labelings: *input labelings* that are part of the input and *output labelings* that are provided by an algorithm executed on input-labeled instances. Throughout the paper, we will assume that any considered input graph G comes with an input labeling $g_{\text{in}}: H(G) \rightarrow \Sigma_{\text{in}}$ and will refer to Σ_{in} as the *set of input labels*; if the considered LCL problem does not have input labels, we can simply assume that $\Sigma_{\text{in}} = \{\perp\}$ and that each node is labeled with \perp .

While the formal definition of a node-edge-checkable LCL (see below) appears complicated, the intuition behind it is simple: essentially, we have a list of allowed output label combinations around nodes, a list of allowed output label combinations on edges, and a list of allowed input-output label combinations, all of which a correct solution for the LCL has to satisfy.

► **Definition 4 (Node-edge-checkable LCL).** *Let Δ be some non-negative integer constant. A node-edge-checkable LCL is a quintuple $\Pi = (\Sigma_{\text{in}}, \Sigma_{\text{out}}, \mathcal{N}, \mathcal{E}, g)$ where Σ_{in} and Σ_{out} are finite sets, $\mathcal{N} = \{\mathcal{N}_1, \dots, \mathcal{N}_\Delta\}$ consists of sets \mathcal{N}_i of cardinality- i multisets with elements from Σ_{out} , \mathcal{E} is a set of cardinality-2 multisets with elements from Σ_{out} , and $g: \Sigma_{\text{in}} \rightarrow 2^{\Sigma_{\text{out}}}$ is a function mapping input labels to sets of output labels. We call $\mathcal{N}_1 \cup \dots \cup \mathcal{N}_\Delta$ and \mathcal{E} the node constraint and edge constraint of Π , respectively. Furthermore, we call each element of \mathcal{N} a node configuration, and each element of \mathcal{E} an edge configuration. For a node v , denote the half-edges of the form (v, e) for some edge e by $h_1^v, \dots, h_{\deg(v)}^v$ (in arbitrary order). For an edge e , denote the half-edges of the form (v, e) for some node v by h_1^e, h_2^e (in arbitrary order).*

A correct solution for Π on a graph G is a half-edge labeling $g_{\text{out}}: H(G) \rightarrow \Sigma_{\text{out}}$ such that

1. for each node v , the multiset of outputs assigned by g_{out} to $h_1^v, \dots, h_{\deg(v)}^v$ is an element of $\mathcal{N}_{\deg(v)}$,
2. for each edge e , the cardinality-2 multiset of outputs assigned by g_{out} to h_1^e, h_2^e is an element of \mathcal{E} , and
3. for each half-edge $h \in H(G)$, we have $g_{\text{out}}(h) \in g(\iota)$, where $\iota = g_{\text{in}}(h)$ is the input label assigned to h .

We say that an algorithm \mathcal{A} solves an LCL problem Π on a graph class \mathcal{G} if it provides a correct solution for Π for every $G \in \mathcal{G}$. Note that the LCL definitions above implicitly require that graph class \mathcal{G} has constant degree.

3 The Tiny Regime

In this section, we show that any LCL problem on general graphs that can be solved in the LOCAL model in $O(\log^* n)$ rounds, can be solved in the MPC model in $O(\log \log^* n)$ rounds. By combining this result with known gaps in the landscape of possible complexities in the LOCAL model [21], we obtain the following result.

► **Theorem 5.** *Let Π be an LCL problem on general graphs. Assume that there is a deterministic algorithm for the LOCAL model that solves Π in $o(\log n)$ rounds, or a randomized algorithm that solves it in $o(\log \log n)$ rounds. Then, the problem Π can be solved deterministically in $O(\log \log^* N)$ rounds in the low-space MPC model using $O(m + n)$ words of global memory, where $N = \text{poly}(n)$ is the size of the ID space. The algorithm works even if the graph consists of disconnected components, and it is components-stable.*

The rest of this section is devoted to proving Theorem 5.

A Universal Algorithm. In the LOCAL model, it is known that, if an LCL can be solved with an algorithm A in $o(\log n)$ deterministic rounds, or in $o(\log \log n)$ randomized rounds, then it can also be solved with a deterministic algorithm A' that requires just $O(\log^* n)$ rounds [21]. In order to prove this result, [21] shows how to convert any such algorithm A into an algorithm A' that works as follows (for some constant k that depends on the problem Π and the algorithm A):

1. Compute a distance- k $O(\Delta^{2k})$ -coloring of the graph;
 2. Run a k -round algorithm B that uses the computed coloring to produce the final output.
- In [21] is shown that the constant k , and the k -round algorithm B , can be mechanically determined from the original algorithm A . The runtime of algorithm A' is $O(\log^* n)$ rounds since this is the runtime for the first step, while the second step only requires constant time.

Why it Works. The high-level purpose of computing the coloring in Item 1 is to provide new identifiers at the nodes that are unique up to distance k and come from a much smaller space than the original identifiers (that are part of the setting in the LOCAL model). Roughly speaking, this ensures that the k -hop view of any node that interprets the computed colors as identifiers is consistent with the node living in a constant-sized graph (with a constant-sized identifier space).

In [21], it is argued why this approach works, and on a high level, the reason can be summarized as follows. For some sufficiently large constant k , algorithm A can be executed on all graphs of a suitable constant size with a runtime of just k rounds. Since each node of the original graph executing this k -round algorithm cannot distinguish between living in

the original graph with the generated new identifiers and living in (a suitable) one of these constant-sized graphs (on all of which the algorithm is correct), the k -round algorithm must also be correct on the (much larger) original graph. This is just a high-level sketch of the proof presented in [21]; there are a number of intricate details that have to be taken care of and are explained in [21].

How We Proceed. For our purpose, we do not actually need to know the details of [21] on how A' is constructed as a function of A , and we just use the following statement that comes from [21]: if the problem Π can be solved in $o(\log n)$ deterministic rounds or $o(\log \log n)$ randomized rounds, then it can also be solved in $O(\log^* n)$ deterministic rounds using an algorithm that first applies Item 1 and then applies Item 2. In fact, in our case, we are not even given the algorithm A as input: we just know that the problem can be solved in $o(\log n)$ deterministic or $o(\log \log n)$ randomized rounds, but we are not given an algorithm A with such a complexity. Hence, we cannot apply the construction of [21] directly.

In Section 7 of the full version, we show that this is not an issue, in the sense that, if an algorithm exists, then it can be found by brute force. To show that, we use the following two important ingredients presented in [50]:

- Any constant time algorithm that solves an LCL in the LOCAL model can be transformed into an algorithm that does not require nodes to have IDs.
- For every k , it is decidable whether there exists a k -round algorithm that solves a given problem in a setting where we do not have IDs and we are given a (suitable) distance- k coloring. The reason is that, in this setting, there are only a finite number of possible algorithm candidates (and they can be enumerated), and given a candidate, it is possible to check if it constitutes a correct algorithm by using a centralized offline procedure.

We use the above ingredients as follows. If we just know that Π can be solved in $o(\log n)$ deterministic rounds or $o(\log \log n)$ randomized rounds, even if no algorithm is given, we can use [21] to claim that there exists a k for which there is a k -round algorithm B that solves Π given a distance- k coloring, and then use the first ingredient to claim that this algorithm does not need the presence of IDs. Finally, we use the second ingredient to say that if we try increasing values of k , we are going to find the algorithm B that we need.

From the above discussion, in order to prove Theorem 5, we only need to show how to compute a distance- k $O(\Delta^{2k})$ -coloring in $O(\log \log^* n)$ deterministic MPC rounds.

3.1 LOCAL Algorithm

We start by presenting an algorithm for computing such a coloring in the LOCAL model. While computing such a coloring in the LOCAL model is easy, we present an algorithm amenable to be converted into a faster MPC algorithm. This algorithm is not new: it has been already presented in [38, 51], and we report it here, with minor modifications, for completeness.

► **Lemma 6.** *For any constant k , the distance- k $O(\Delta^{2k})$ -coloring problem on general graphs can be solved in the LOCAL model with a deterministic algorithm running in $O(\log^* n)$ rounds.*

Proof. We present an algorithm that is able to compute an $O(\Delta^2)$ coloring of a given graph G , where Δ is the maximum degree of G , in $O(\log^* n)$ rounds. By simulating such an algorithm on G^k , the k -th power of G , which has maximum degree Δ^k , we obtain the claimed result. Note that the running time is also asymptotically the same, since k is a constant.

The algorithm works as follows. At the beginning, each edge is oriented arbitrarily. Then, each node marks its incident outgoing edges with different numbers from $\{1, \dots, \Delta\}$. In this way, we decomposed our graph G into Δ edge-disjoint directed subgraphs G_1, \dots, G_Δ ,

where each G_i is the graph induced by edges marked i . Also, notice that by construction, for each i , each node in G_i has at most a single outgoing edge, and hence each G_i is a directed pseudoforest.

Assume we can color each directed pseudoforest with 3 colors in $O(\log^* n)$ rounds. Then, we can obtain a proper coloring for the nodes of G with 3^Δ colors, by letting each node construct the tuple $c(v) = (c_1(v), \dots, c_\Delta(v))$, where $c_i(v)$ is the color of v in G_i . In fact, consider two neighboring nodes u and v connected through an edge e . Assume that e is oriented from u to v , and that u marked e with value i . Then, in G_i , u and v are neighbors, and hence they obtained different colors $c_i(u)$ and $c_i(v)$, implying that $c(u) \neq c(v)$. Once a 3^Δ -coloring is obtained, we can then spend $O(3^\Delta)$ rounds to reduce the number of colors to $O(\Delta^2)$, by using a simple greedy algorithm.

We now show that each pseudoforest can be 3-colored efficiently. Let P be an arbitrary pseudotree. At first, we can use the IDs of the nodes to produce a $\text{poly}(n)$ -coloring of P . Then we apply 1 round of Linial's coloring algorithm [47] in order to obtain an $O(\log n)$ -coloring of P . While this step of coloring is not necessary for the LOCAL algorithm, it allows us to reduce the amount of information that we will later need to transmit in the MPC algorithm. Nodes can then spend $T = O(\log^* n)$ rounds to gather the color of their successors in P at distance at most T , and it is known that, with this information, nodes can compute a proper coloring of P , by simulating $O(\log^* n)$ steps of a color reduction algorithm for directed paths [38, 23]. ◀

3.2 MPC Implementation

We now show how to convert the LOCAL algorithm into an exponentially faster low-space MPC algorithm. The LOCAL algorithm consists of two main steps: The distance- k $O(\Delta^{2k})$ -coloring and the k -round algorithm. Since k and Δ are constant, the latter step is trivial, and the former step can be computed efficiently using graph exponentiation, where nodes keep track of the IDs of the two outermost nodes, and the colors of all nodes in between. Lemma 9 of the following paragraph proves the former step, completing the proof for Theorem 5. Component-stability and compatibility with disconnected components follows directly from the fact that all arguments are local, i.e., nodes in separate components never communicate, and that the runtime depends only on N .

Distance- k Coloring. We show that the initial distance- k coloring can be computed in $O(\log \log^* n)$ low-space MPC rounds, while respecting linear global memory. First, we observe that using the standard graph exponentiation technique, we can compute the k th power of a graph; for constant k , the memory overhead is only a constant. Then, we will apply techniques similar to the ones used in the LOCAL model in Lemma 6.

▶ **Observation 7.** *For an input graph G with n nodes, m edges, and maximum degree Δ , the power graph G^k can be computed deterministically in $O(\log k)$ low-space MPC rounds with $O(\Delta^k)$ words of local and $O(m + n \cdot \Delta^k)$ words of global memory, as long as $\Delta^k < n^\delta$.*

▶ **Observation 8.** *Every k -round LOCAL algorithm can be simulated in $O(\log k)$ low-space MPC rounds with $O(\Delta^k)$ words of local and $O(m + n \cdot \Delta^k)$ words of global memory, as long as $\Delta^k < n^\delta$. If the LOCAL algorithm is deterministic, then the MPC algorithm is deterministic as well.*

Proof. Using Observation 7, we can collect the k -hop neighborhood of each node and hence, simulate a k -round LOCAL algorithm in an additional $O(1)$ low-space MPC rounds. Observe that this also holds for general graphs. ◀

► **Lemma 9.** *The distance- k $O(\Delta^{2k})$ -coloring problem on general graphs can be solved in the low-space MPC model with a $O(\log \log^* n + \log k)$ -time deterministic algorithm, as long as $\Delta^k < n^\delta$. The algorithm requires $O(\Delta^k)$ words of local and $O(m + n \cdot \Delta^k)$ words of global memory. If k and Δ are constants, the runtime reduces to $O(\log \log^* n)$ and we require $O(1)$ words of local and $O(m + n)$ words of global memory.*

Proof. Using Observation 7, we can first compute G^k in $O(\log k)$ rounds, and operate on G^k instead of the input graph G henceforth. The application of Observation 7 requires $O(\Delta^k)$ words of local memory and $O(m + n \cdot \Delta^k)$ words of global memory. Then, similarly to Lemma 6, we can reduce the coloring problem to $O(1)$ -coloring of directed pseudoforests that are initially colored with $O(\log n)$ colors.

Next, our goal is to use the graph exponentiation technique such that each node can collect the topology and the colors of its $O(\log^* n)$ successors in its pseudoforest in $O(\log \log^* n)$ time. Here, we have to take care of the subtle detail that the *color* of a successor is not enough to determine the machine on which this successor lies. Suppose that each node is initially labeled with its $O(\log \log n)$ -bit color and its $O(\log n)$ -bit identifier that encodes both the identity (color) of the node and the machine containing the node. Then, in round 1, each node knows the identifier and the color of its successor. For an inductive argument, suppose that each node u knows the identifier the successor v_i in distance i and the vector of colors of all nodes in between u and v_i , on the directed path from u to v_i . Then, in $O(1)$ MPC rounds, u can learn the identifier of the $2i$:th successor v_{2i} and the colors of all nodes between u and v_{2i} . After learning the identifier of v_{2i} , node u can forget about the identifier of v_i and hence, u only keeps track of one identifier. By induction, node u learns the colors of its $O(\log^* n)$ successors in $O(\log \log^* n)$ MPC rounds.

Using the vector of colors of the successors, in $O(1)$ MPC rounds, each node can simulate the $O(\log^* n)$ -time LOCAL algorithm to obtain an $O(\Delta^{2k})$ -coloring. This requires $O(\log^* n \cdot \log \log n + \log n) = O(\log n)$ bits of memory per node per pseudoforest that the node belongs to, counting the colors of the successors and the identifier of the furthest successor. Altogether, this results in a global memory requirement of $O(n \log n \cdot \Delta^k)$ bits which fits $O(n \cdot \Delta^k)$ words. ◀

4 The High Regime

In this section, we will prove that all solvable LCL problems on forests, i.e., *all LCL problems that have a correct solution on every forest*, can be solved deterministically in $O(\log n)$ time in the low-space MPC model using $O(m + n)$ words of global memory. Our proof is constructive: we explicitly provide, for any solvable LCL, an algorithm that has a runtime of $O(\log n)$. In fact, our construction can be used to find an $O(\log n)$ -time algorithm *even for unsolvable LCLs*, with the guarantee that on any instance that admits a correct solution the given output will be correct (while the algorithm detects it if no solution exists). We show the following theorem.

► **Theorem 10.** *For any solvable LCL problem Π on a forest, there is an $O(\log n)$ -time deterministic low-space MPC algorithm that is component-stable and uses $O(m + n)$ words of global memory.*

In the full version, we provide a method to solve any LCL on forests if we can solve it on trees. Hence, we can restrict attention to trees.

4.1 High-level Overview of the Algorithm and Its Analysis

Consider an arbitrary solvable LCL problem Π on trees. In the following, we will give a slightly simplified view of the algorithm we will use to solve Π in $O(\log n)$ time. First, we root the input tree by using the $O(\log n)$ -round rooting algorithm described in the full version. Then, on a high level, the rest of the algorithm proceeds in 2 phases.

In the first phase, which we will refer to as the *leaves-to-root phase*, roughly speaking, the goal is to compute, for a substantial number of edges $e = (u, v)$, the set of output labels that can be output at half-edge (v, e) such that the label choice can be extended to a (locally) correct solution in the subtree hanging from v via e . This is done in an iterative manner, proceeding from the leaves towards the root. When, at last, the root has computed this set of output labels for each incident half-edge, it can, on each such half-edge, select an output label from the computed set such that the obtained node configuration is contained in the node constraint of Π and the input-output constraints of Π (given by the function g in the definition of Π) are satisfied. Such a selection must exist due to the fact that Π has a correct solution on the considered instance. We refer to these sets as the *completeness information*.

The second phase, which we will refer to as the *root-to-leaves phase*, consists of completing the solution from the root downwards, by iteratively propagating the selected solution further towards the leaves. With the same argumentation as at the root, certain nodes v can select an output label at the half-edge leading to its parent and output labels from the sets computed on its incident half-edges leading to its children such that the obtained node configuration is contained in the node constraint of Π , the obtained edge configuration on the edge from v to its parent is contained in the edge constraint of Π , and the input-output constraints of Π are satisfied. The fact that the selected labels come from the sets computed in the first phase ensures that after each choice the current partial solution is part of a correct global solution. While this outline sounds simple, there are a number of intricate challenges to make the mentioned ideas work in $O(\log n)$ rounds while staying within the memory bound of $O(m + n)$.

Unfortunately, if the depth of the input tree is $\omega(\log n)$ the outlined approach has $\omega(\log n)$ steps and running them sequentially is insufficient for an $O(\log n)$ -time algorithm. In order to mitigate this issue, we will not only process the leaves of the remaining unprocessed tree in each iteration, but also the nodes of degree 2, inspired by the rake-and-compress decomposition by Miller and Reif [49] which guarantees that after $O(\log n)$ iterations of removing all degree-1 and degree-2 nodes all nodes have been removed. The advantage of degree-2 nodes over higher-degree nodes w.r.t. storing completeness information (as in the above outline) is that they form paths, which by definition only have two endpoints; the idea, when processing such a path, is to simply store in the two endpoints the information for which pairs of labels at the two half-edges at the ends of the path there exists a correct completion of the solution inside the path. This allows to naturally add processing degree-2 nodes to the leaves-to-root phase, while for the root-to-leaves phase, the information stored at the endpoints s, t of a path essentially allows us to start extending the current partial solution on the path itself (and thereafter on the subtrees *hanging* from nodes on the path) one step after the output labels at s and t are selected. Note that the degrees of nodes change throughout the process due to the removal of nodes and hence new nodes might become degree-2 nodes after every step of the algorithm.

Unfortunately, there are further challenges in obtaining an $O(\log n)$ runtime. In the leaves-to-root phase, even when using graph exponentiation, processing a path of degree-2 nodes of length L involves coordination between its endpoints and takes $\Omega(\log L)$ time, whereas the $O(\log n)$ time guarantee of the rake-and-compress technique crucially relies on

the fact that each iteration (optimally, an iteration would remove all leaves and all degree-2 nodes) can be performed in constant time. Hence, essentially, we will only perform one step of graph exponentiation on paths in each iteration. Here, a new obstacle arises: before the graph exponentiation is finished, new nodes (that just became degree-2 nodes due to all except one of their remaining children being conclusively processed in the most recent iteration) might join the path. Nevertheless, we will show that this process still terminates in logarithmic time by designing a fine-tuned potential function that is inspired by the idea of counting how many nodes from certain groups of degree-2 nodes are contained in any fixed “pointer chain” from some leaf to the root.

Another issue is that we have to be able to store the completability information that we compute in the leaves-to-root phase until we use it (again) in the root-to-leaves phase. Recall that the graph exponentiation technique adds new edges/pointers. Even on paths their number can be up to logarithmic in n per node (even on average), yielding a logarithmic overhead in global memory.

In order to remedy this problem, we perform preprocessing before the leaves-to-root phase, and, as a result thereof, postprocessing after the root-to-leaves phase. The preprocessing can be thought of as a more memory-efficient (hence relatively slower) version of (a few iterations in) the leaves-to-root phase. It differs by processing the degree-2 nodes, i.e., paths, in a way that guarantees that the number of new edges introduced by the graph exponentiation (which we should rather call pointer forwarding at this point) on each path in each iteration is only a constant fraction of the length of the respective path. This is achieved by finding, in each iteration, a maximal independent set (MIS) on each path, letting only MIS nodes forward pointers, and removing the MIS nodes afterwards. The preprocessing runs for $\Theta(\log \log n)$ iterations, and computing an MIS on paths in each of them takes $O(\log^* N)$ time, where N is the size of the ID space. Note that due to the removal of vertices and the way we treat paths, new paths can appear in each iteration and we need to pay the $O(\log^* N)$ runtime in each iteration, yielding a runtime of $O(\log \log n \cdot \log^* N)$ for the preprocessing, which is much less than the target runtime of $O(\log n)$ rounds.

We will show that the number of remaining nodes is $O(n/\log n)$ after the preprocessing. This property ensures that the memory overhead of $O(\log n)$ edges per node introduced in the leaves-to-root phase does not exceed the desired global memory of $O(m+n)$ words. The postprocessing runs for $\Theta(\log \log n)$ iterations and is conceptually very similar to the preprocessing. We simply iteratively extend the partial solution (computed so far) on the edges that were processed during preprocessing, analogous to the approach in the root-to-leaves phase. Lastly, we also have to ensure that the local memory restrictions of low-space MPC are not exceeded.

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*, pages 567–583, 1986. doi:10.1016/0196-6774(86)90019-2.
- 2 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. doi:10.1145/2591796.2591805.
- 3 Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Classification of Distributed Binary Labeling Problems. In *DISC*, pages 17:1–17:17, 2020. doi:10.1145/3382734.3405703.

- 4 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower Bounds for Maximal Matchings and Maximal Independent Sets. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 481–497, 2019. doi:10.1109/FOCS.2019.00037.
- 5 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed Δ -Coloring Plays Hide-and-Seek. In *Proceedings of the Symposium on Theory of Computing (STOC)*, 2022. arXiv:2110.00643.
- 6 Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. Distributed Lower Bounds for Ruling Sets. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 365–376, 2020. doi:10.1109/FOCS46700.2020.00042.
- 7 Alkida Balliu, Sebastian Brandt, Dennis Olivetti, Jan Studeny, Jukka Suomela, and Aleksandr Tereshchenko. Locally Checkable Problems in Rooted Trees. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 263–272, 2021. doi:10.1145/3465084.3467934.
- 8 Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and J. Suomela. Almost Global Problems in the LOCAL Model. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 9:1–9:16, 2018. doi:10.4230/LIPIcs.DISC.2018.9.
- 9 Alkida Balliu, Keren Censor-Hillel, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Locally Checkable Labelings with Small Messages. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 8:1–8:18, 2021. doi:10.4230/LIPIcs.DISC.2021.8.
- 10 Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New Classes of Distributed Time Complexity. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 1307–1318, 2018. doi:10.1145/3188745.3188860.
- 11 Alkida Balliu, Juho Hirvonen, Dennis Olivetti, and Jukka Suomela. Hardness of Minimal Symmetry Breaking in Distributed Computing. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 369–378, 2019. doi:10.1145/3293611.3331605.
- 12 Paul Beame, Paraschos Koutris, and Dan Suci. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):40, 2017. doi:10.1145/3125644.
- 13 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 *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 481–490, 2019. doi:10.1145/3293611.3331609.
- 14 Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Łącki, and Vahab Mirrokni. Near-Optimal Massively Parallel Graph Connectivity. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1615–1636, 2020. doi:10.1109/FOCS.2019.00095.
- 15 Sebastian Brandt, Manuela Fischer, and Jara Uitto. Breaking the Linear-Memory Barrier in MPC: Fast MIS on Trees with Strongly Sublinear Memory. In *the Proceedings of the International Colloquium on Structural Information and Communication Complexity*, pages 124–138, 2019. doi:10.1007/978-3-030-24922-9_9.
- 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 Press, 2016. doi:10.1145/2897518.2897570.
- 17 Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R.J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemysław Uznański. LCL Problems on Grids. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 101–110, 2017. doi:10.1145/3087801.3087833.
- 18 Yi-Jun Chang. The Complexity Landscape of Distributed Locally Checkable Problems on Trees. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 18:1–18:17, 2020. doi:10.4230/LIPIcs.DISC.2020.18.

- 19 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The Complexity of $(\Delta + 1)$ Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 471–480, 2019.
- 20 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. Distributed Edge Coloring and a Special Case of the Constructive Lovász Local Lemma. *ACM Transactions on Algorithms (TALG)*, pages 1–51, 2019. doi:10.1145/3365004.
- 21 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. *SIAM Journal on Computing*, 48(1):122–143, 2019. doi:10.1137/17M1117537.
- 22 Yi-Jun Chang and Seth Pettie. A Time Hierarchy Theorem for the LOCAL Model. *SIAM Journal of Computing*, 48(1):33–69, 2019. doi:10.1137/17M1157957.
- 23 Richard Cole and Uzi Vishkin. Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking. *Inf. Control.*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.
- 24 Sam Coy and Artur Czumaj. Deterministic massively parallel connectivity. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, 2022. doi:10.1145/3519935.3520055.
- 25 Artur Czumaj, Peter Davies, and Merav Parter. Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 175–185, 2020. doi:10.1145/3350755.3400282.
- 26 Artur Czumaj, Peter Davies, and Merav Parter. Component Stability in Low-Space Massively Parallel Computation. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 481–491, 2021. doi:10.1145/3465084.3467903.
- 27 Artur Czumaj, Peter Davies, and Merav Parter. Improved Deterministic $(\Delta + 1)$ Coloring in Low-Space MPC. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 469–479, 2021. doi:10.1145/3465084.3467937.
- 28 Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Communications of the ACM*, pages 107–113, 2008.
- 29 Michal Dory, Orr Fischer, Seri Khoury, and Dean Leitersdorf. Constant-Round Spanners and Shortest Paths in Congested Clique and MPC. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 223–233, 2021. doi:10.1145/3465084.3467928.
- 30 Manuela Fischer and Mohsen Ghaffari. Sublogarithmic Distributed Algorithms for Lovász Local Lemma, and the Complexity Hierarchy. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 18:1–18:16, 2017. doi:10.4230/LIPIcs.DISC.2017.18.
- 31 Mohsen Ghaffari. An Improved Distributed Algorithm for Maximal Independent Set. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 270–277, 2016. doi:10.1137/1.9781611974331.ch20.
- 32 Mohsen Ghaffari, Christoph Grunau, and Ce Jin. Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 34:1–34:18, 2020. doi:10.4230/LIPIcs.DISC.2020.34.
- 33 Mohsen Ghaffari, Christoph Grunau, and Václav Rozhon. Improved Deterministic Network Decomposition. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2904–2923, 2021. doi:10.1137/1.9781611976465.173.
- 34 Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On Derandomizing Local Distributed Algorithms. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 662–673, 2018. doi:10.1109/FOCS.2018.00069.
- 35 Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1650–1663, 2019. doi:10.1109/FOCS.2019.00097.

- 36 Mohsen Ghaffari and Hsin-Hao Su. Distributed Degree Splitting, Edge Coloring, and Orientations. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2505–2523, 2017. doi:10.1137/1.9781611974782.166.
- 37 Mohsen Ghaffari and Jara Uitto. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1636–1653, 2019. doi:10.1137/1.9781611975482.99.
- 38 Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM J. Discret. Math.*, 1(4):434–446, 1988. doi:10.1137/0401044.
- 39 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, Searching, and Simulation in the Mapreduce Framework. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 374–383, 2011. doi:10.1007/978-3-642-25591-5_39.
- 40 Christoph Grunau, Václav Rozhon, and Sebastian Brandt. The landscape of distributed complexities on trees and beyond. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 37–47. ACM, 2022. doi:10.1145/3519270.3538452.
- 41 Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *SIGOPS Operating Systems Review*, pages 59–72, 2007. doi:10.1145/1272996.1273005.
- 42 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010. doi:10.1137/1.9781611973075.76.
- 43 Raimondas Kiveris, Silvio Lattanzi, Vahab Mirrokni, Vibhor Rastogi, and Sergei Vassilvitskii. Connected Components in MapReduce and Beyond. In *ACM Symposium on Cloud Computing*, pages 18:1–18:13, 2014. doi:10.1145/2670979.2670997.
- 44 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *Journal of the ACM (JACM)*, 63:17:1–17:44, 2016. doi:10.1145/2742012.
- 45 Christoph Lenzen and Roger Wattenhofer. Brief Announcement: Exponential Speed-Up of Local Algorithms Using Non-Local Communication. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 295–296, 2010. doi:10.1145/1835698.1835772.
- 46 Nathan Linial. Distributive Graph Algorithms - Global Solutions from Local Data. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 331–335, 1987. doi:10.1109/SFCS.1987.20.
- 47 Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.
- 48 Michael Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 1–10, 1985. doi:10.1145/22145.22146.
- 49 Gary L. Miller and John H. Reif. Parallel Tree Contraction Part 1: Fundamentals. *Adv. Comput. Res.*, 5:47–72, 1989.
- 50 Moni Naor and Larry Stockmeyer. What Can Be Computed Locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.
- 51 Alessandro Panconesi and Romeo Rizzi. Some Simple Distributed Algorithms for Sparse Networks. *Distributed Computing*, 14(2):97–100, 2001. doi:10.1007/PL00008932.
- 52 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.
- 53 Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and Circuits (On Lower Bounds for Modern Parallel Computation). *J. ACM*, pages 1–12, 2018. doi:10.1145/3232536.
- 54 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-Time Deterministic Network Decomposition and Distributed Derandomization. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 350–363, 2020. doi:10.1145/3357713.3384298.

- 55 Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012. doi:10.5555/1717298.
- 56 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2010. doi:10.5555/1863103.1863113.

A Component-stability

The notion of a *component-stable* MPC algorithm has been introduced in [35] in the context of lifting distributed lower bounds to the MPC setting. It was later revised by Czumaj, Davies and Parter [26] and argued to be made more robust.

► **Definition 11** (Component-stability, [35]). *An MPC algorithm is component-stable if 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 .*

In their revised definition, [26] assume the setting where all input graphs are legal.

► **Definition 12** (Legal graph). *A graph G is called legal if it is equipped with functions ID , $name: V(G) \rightarrow [\text{poly}(n)]$ providing nodes with IDs and names, such that all names are fully unique and all IDs are unique in every connected component.*

► **Definition 13** (Component-stability (revised), [26]). *A randomized MPC algorithm A_{MPC} is component-stable if its output at any node v is entirely, deterministically, dependent on the topology and IDs (but independent of names) of v 's connected component (which we will denote $CC(v)$), v itself, the exact number of nodes n and maximum degree Δ in the entire input graph, and the input random seed \mathcal{S} . That is, the output of A_{MPC} at v can be expressed as a deterministic function $A_{MPC}(CC(v), v, n, \Delta, \mathcal{S})$. A deterministic MPC algorithm A_{MPC} is component-stable under the same definition, but omitting dependency on the random seed \mathcal{S} .*

As opposed to [35], [26] allow the output of component-stable algorithms to depend on the total number of nodes in the graph and the maximum degree of the graph. Additionally, they assume the following setting: all input graphs are *legal* (see Definition 12), i.e., all nodes have an ID that is unique in every connected component, and a *name* that is unique across the whole input graph. Assuming the above setting, the output of a component-stable algorithm is allowed to depend on the IDs of all nodes in the same components, but not the names.

In our work, we adopt the revised definition of component-stability [26]. In all of our algorithms, nodes from different components only communicate in order to maintain a certain global synchrony. This synchrony influences *when* certain steps are executed and hence the *execution* of our algorithms. However, the output at each node is not influenced by the global communication.

Theorem 2 shows that the lower bounds for component-stable algorithms can be beaten for a large class of problems on trees and forests even with optimal memory. The long term effect of the term component-stable in this setting is unclear, but it provides room for many interesting open questions. One interesting aspect would be to see under which circumstances one can obtain algorithms with stronger component dependent guarantees, e.g., one may

want to develop algorithms for which not just the output of a node, but also the time until it has computed its output can only depend on the size of its component. Our algorithms do not meet this stronger definition. Besides an ID space dependence our algorithms have the following runtime behaviour. In the low and mid regime the time until we know the output of a node depends on the number of nodes in the largest connected component. In the high regime this time depends on the number of nodes in the whole graph. Going from trees to forests in the high regime relies on the recent beautiful (deterministic) connected components algorithm by Czumaj and Coy [24, 14].