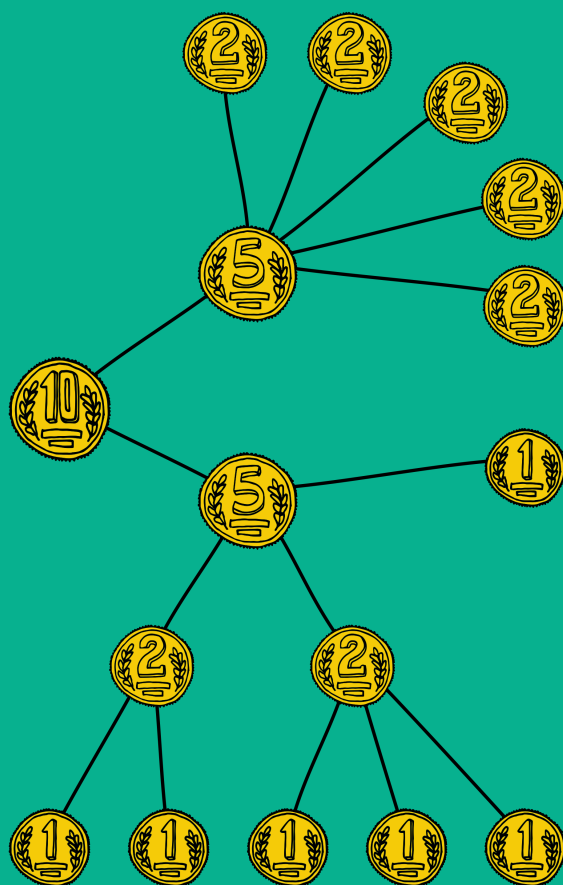


Massively parallel algorithms for sparse graphs

Rustam Latypov



Aalto University publication series
Doctoral Theses 82/2026

Massively parallel algorithms for sparse graphs

Rustam Latypov

A doctoral thesis completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall M232 M1 of the school on the 10th of April 2026 at 12:00.

Aalto University
School of Science
Department of Computer Science

Supervising professor

Assistant Professor Jara Uitto, Aalto University, Finland

Preliminary examiners

Professor Artur Czumaj, University of Warwick, United Kingdom

Associate Professor Sepehr Assadi, University of Waterloo, Canada

Opponent

Professor Artur Czumaj, University of Warwick, United Kingdom

Aalto University publication series

Doctoral Theses 82/2026

© Rustam Latypov

Image on the cover: Cover illustration by Rustam Latypov, based on licensed artwork by Nikolai Shitov / Adobe Stock.

ISBN 978-952-64-3082-9 (soft cover)

ISBN 978-952-64-3081-2 (PDF)

ISSN 1799-4934 (printed)

ISSN 1799-4942 (PDF)

<https://urn.fi/URN:ISBN:978-952-64-3081-2>

Unigrafia Oy, Helsinki 2026

PunaMusta Oy, Joensuu, 2026

Author Rustam Latypov

Name of the doctoral thesis Massively parallel algorithms for sparse graphs

Article-based thesis

Number of pages 167

Keywords Massively parallel computation model, uniformly sparse graphs, connectivity, graph coloring, maximal matching, maximal independent set

Most real-life graphs are large and sparse, consisting of billions of nodes connected by billions of edges. Single-machine computation for such graphs is often slow and inefficient. The modern approach is to deploy a fleet of machines that work on a given graph problem in parallel. The Massively Parallel Computation (MPC) model captures the strengths and limitations of such parallel systems and is a widely accepted theoretical framework for analyzing modern parallel algorithms. This thesis considers fundamental graph problems for uniformly sparse graphs in the MPC model, focusing on memory-optimal solutions. Computation is performed on arbitrarily many machines simultaneously while using roughly the same amount of memory that is needed to store the input itself.

We resolve open problems and advance the state of the art by developing novel deterministic algorithms for graph problems such as connectivity, vertex coloring, maximal matching, and maximal independent set. Our focus is on uniformly sparse graphs, technically known as low-arboricity graphs. These graph families form the setting for most known lower bounds in MPC, and designing memory-optimal algorithms for them is notoriously difficult.

Algorithm complexity is measured in communication rounds and is typically expressed as a function of the total number of nodes n or the diameter of the graph D . We establish the first $O(\log D)$ -round algorithm for connectivity, as well as the first $O(\log \log n)$ - and $O(\log D)$ -round algorithms for 3-coloring, maximal matching, and maximal independent set on forest graphs. We also show that strengthening the MPC model, by allowing adaptive queries to a distributed data store, enables a constant-round $O(\alpha)$ -coloring algorithm for graphs with bounded arboricity α .

One major technical contribution of this thesis is a novel graph exploration technique called balanced exponentiation, on which many of our results rely. This technique has been developed into a parameterized, standalone tool that allows nodes to explore surrounding sparse regions of the graph without prior knowledge of their location, while incurring minimal memory overhead.

Tekijä Rustam Latypov

Väitöskirjan nimi Massively parallel algorithms for sparse graphs

Artikkeliväitöskirja

Sivumäärä 167

Avainsanat Massiivisen rinnakkaislaskennan malli, tasaisen harvat verkot, yhtenäisyys, verkon väritys, maksimaalinen pariutus, maksimaalinen riippumaton joukko

Useimmat tosielämän verkot ovat suuria ja harvoja, koostuen miljardeista solmuista, joita yhdistävät miljardit kaaret. Yksittäisen tietokoneen suorittama laskenta tällaisille verkoille on usein hidasta ja tehontonta. Nykyaikainen lähestymistapa on käyttää useita tietokoneita ratkaisemaan annettua verkko-ongelmaa rinnakkain. Massiivisen rinnakkaislaskennan (MPC) -malli on vakiintunut teoreettinen viitekehys modernien rinnakkaisalgoritmien analysointiin, kuvaten rinnakkaisjärjestelmien vahvuuksia ja rajoituksia. Tämä väitöskirja tarkastelee perustavanlaatuisia ongelmia MPC-mallissa, keskittyen muistioptimaalisiin ratkaisuihin tasaisen harvoissa verkoissa. Laskenta suoritetaan samanaikaisesti mielivaltaisella määrällä koneita, käyttäen suunnilleen yhtä paljon muistia kuin itse syötteen tallentamiseen tarvitaan.

Ratkaisemme avoimia ongelmia ja parannamme alan aiempia huipputuloksia uusilla deterministisillä algoritmeilla yhtenäisyydelle, väritykselle, maksimaaliselle pariutukselle sekä maksimaaliselle riippumattomalle joukolle. Keskitymme tasaisen harvoihin verkkoihin, jotka tunnetaan teknisesti matalan puumaisuuden verkkoina. Suurin osa tunnetuista alarajoista MPC-mallissa on osoitettu juuri näissä verkoissa ja muistioptimaalisten algoritmien suunnittelu niille on tunnetusti haastavaa.

Algoritmien kompleksisuus mitataan kommunikaatiokierrosten kokonaismääränä ja se ilmaistaan tyypillisesti solmujen määrän n tai verkon halkaisijan D funktiona. Kehitämme ensimmäisen $O(\log D)$ -kierroksisen algoritmin yhtenäisyydelle sekä ensimmäiset $O(\log \log n)$ - ja $O(\log D)$ -kierroksiset algoritmit 3-väritykselle, maksimaaliselle pariutukselle ja maksimaaliselle riippumattomalle joukolle metsäverkoissa. Vahvistamalla MPC-mallia siten, että adaptiiviset kyselyt hajautettuun tietokantaan ovat sallittuja, kehitämme vakiokierroksisen $O(\alpha)$ -väritys-algoritmin verkoille, joiden puumaisuus α on vakio.

Yksi tämän väitöskirjan merkittävimmistä teknisistä kontribuutioista on uusi verkon kartoitusmenetelmä, nimeltään tasapainotettu eksponointi, johon monet väitöskirjan tulokset perustuvat. Menetelmä on jalostettu erilliseksi parametrisoiduksi työkaluksi, jonka avulla solmut voivat kartoittaa niitä ympäröiviä verkon harvoja alueita ilman ennakkotietoa niiden sijainnista, aiheuttamatta merkittävää muistikuormaa.

Preface

What originally drew me to the theoretical study of graph problems was their striking duality — they are often simple to state but difficult to solve. One does not need to be a math graduate to understand, for example, the graph coloring problem, which I can easily explain to my friends, my parents, or even my daughter.

First and foremost, I want to thank Jara for being an exceptional supervisor. You have always been patient and generous with your time. Our frequent whiteboard marathons and debugging sessions were often the most exciting part of any project. Your calming presence in the face of broken proofs and impending deadlines made me feel that everything would work out. In addition to genuinely caring for my development as a researcher, you understood my need for a healthy work-life balance. I could not have asked for a better mentor.

I've been fortunate to have collaborated extensively with Yannic, whose green comments tend to appear all over the manuscript just minutes after resolving the previous ones. The color green will haunt me long after my graduation. Your rigorous attention to detail and precision has shaped the way I approach research.

I'm deeply grateful to Bernhard for hosting me at ETH. Your endless enthusiasm for research and your sweet personality made our stay in Zürich truly memorable. During my visit, I had the pleasure of spending time with Christoph, Václav, Nick, Antti R., and Goran. Thank you for the epic research sessions and late nights over Hanabi and Wizard.

Thank you to all my co-authors: Alkida Balliu, Sebastian Brandt, Karl Bringmann, Manuela Fischer, Nick Fischer, Christoph Grunau, Chetan Gupta, Bernhard Haeupler, Jakub Łącki, Yannic Maus, Dennis Olivetti, Shreyas Pai, Antti Röyskö, Simo Särkkä, Jan Studený, Aurelio Sulser, Jukka Suomela, Jara Uitto, and Hossein Vahidi. I want to thank the Academy of Finland, the Nokia Foundation, and the Finnish Foundation for Technology Promotion for funding my research. I'm very grateful to Artur Czumaj and Sepehr Assadi for serving as the pre-examiners for my thesis, and to Artur for also serving as my opponent.

I've been lucky to have amazing companions throughout this journey. Thank you to the wonderful people I've met at various summer and winter schools and conferences over the years: Giovanna, Marc, Maxime, Zahra, Cem, Zurab, and many others. Every event was made better by the jokes and puzzles we shared. To my longtime friends Saara, Olli, Alisa, Visa, Veera, Sampo, Kalle, Juho, Otto, and Jani, thank you for the years of friendship that had nothing to do with research. To my officemate Mélanie, thank you for the peer support and a mellow workspace.

Climbing is one of my greatest passions, one I've been lucky enough to pursue alongside research. I'm grateful for all the wild vertical adventures I've shared with Joshua, Antti S., Arttu, Paavo, Simon, Tim, Martin, Gavin, and Olga. I could not have asked for a better crew to be routinely scared to death with.

My deepest gratitude goes to my family. My parents' courage to immigrate to Finland from post-Soviet Russia is the only reason I had access to quality education and was free to pursue international academia. Finally, to my beloved partner Olga and our daughter Liana. You were always certain of me, even when I was not, and your "permission" to keep working was the only reason deadlines were met and papers were published. Becoming a father mid-studies was far from easy, but I would not have had it otherwise.

Helsinki, March 8, 2026,

Rustam Latypov

Contents

Preface	1
Contents	3
List of Publications	5
Author's Contribution	7
List of Figures	9
1. Introduction	11
1.1 The Graph Coloring Problem	11
1.2 Local Problems	13
1.3 Global Problems	15
1.4 How to Solve Graph Problems?	16
1.4.1 Distributed Computing Models	17
1.4.2 Modern Parallel Computing Models	17
1.5 Research Output and Roadmap	18
2. Preliminaries	21
2.1 Graph Notations	21
2.2 Nash-Williams Decomposition	23
2.3 Locally Checkable Labeling (LCL) Problems	23
2.4 The LOCAL Model	24
2.5 The Massively Parallel Computation (MPC) Model	26
2.5.1 Node-Centric Approach	27
2.5.2 Graph Exponentiation	27
2.5.3 Lower Bounds and the 1 vs. 2 Cycles Conjecture	27
2.5.4 Component Stability	28
2.6 The Adaptive MPC Model	28
3. Contributions and Related Work	31
3.1 Exponential Speedup Over Locality	32

3.2	Forest Connectivity	33
3.3	Coloring, MIS and Maximal Matching	35
3.4	Coloring Sparse Graphs in AMPC	36
4.	Exponential Speedup Over Locality	39
4.1	Tiny-Regime $f(n) = \Theta(1)$ and $f(n) = \Theta(\log^* n)$	40
4.2	High-Regime $f(n) = \Theta(n^{1/k})$, for all $k \in \mathbb{N}$	41
4.3	Mid-Regime $f(n) = \Theta(\log n)$	41
4.4	Low-Regime $g(n) = \Theta(\log \log n)$	42
5.	The Balanced Exponentiation Technique	43
5.1	Forest Connectivity	43
5.2	Coloring, MIS and Maximal Matching	45
6.	Coin Dropping Game via Adaptive Queries	49
6.1	Sublinear LCA	49
6.1.1	Where Should One Query?	50
6.1.2	Our Solution	52
6.2	Using Our LCA Recursively	53
	References	55
	Publications	63

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

- I** Alkida Balliu, Sebastian Brandt, Manuela Fischer, Rustam Latypov, Yannic Maus, Dennis Olivetti, and Jara Uitto. Exponential Speedup Over Locality in MPC with Optimal Memory. In *International Symposium on Distributed Computing (DISC) 2022 (also in Journal of Distributed Computing, February 2025)*, Volume 246, 9:1-9:21, doi.org/10.4230/LIPIcs.DISC.2022.9, Augusta, USA, October 2022.
- II** Alkida Balliu, Rustam Latypov, Yannic Maus, Dennis Olivetti, and Jara Uitto. Optimal Deterministic Massively Parallel Connectivity on Forests. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) 2023*, pages 2589-2631, doi.org/10.1137/1.9781611977554.ch99, Florence, Italy, January 2023.
- III** Christoph Grunau, Rustam Latypov, Yannic Maus, Shreyas Pai, and Jara Uitto. Conditionally Optimal Parallel Coloring of Forests. In *International Symposium on Distributed Computing (DISC) 2023*, Volume 281, 23:1-23:20, doi.org/10.4230/LIPIcs.DISC.2023.23, L'Aquila, Italy, October 2023.
- IV** Rustam Latypov, Yannic Maus, Shreyas Pai, and Jara Uitto. Adaptive Massively Parallel Coloring in Sparse Graphs. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC) 2024*, pages 508-518, doi.org/10.1145/3662158.3662821, Nantes, France, June 2024.

Author's Contribution

Publication I: “Exponential Speedup Over Locality in MPC with Optimal Memory”

The problem of exponential speedup over locality was suggested by Jara Uitto. The author of this thesis served as the main author of the paper, writing approximately 50% of it. The mid-regime result was developed and written by the thesis author. The high-, low-, and tiny-regime results were a collaborative effort by all authors, with most proof verification carried out by the thesis author. Writing the high-regime result was primarily the work of Sebastian Brandt and Jara Uitto. The tiny-regime result was mostly written by Alkida Balliu and Dennis Olivetti.

Publication II: “Optimal Deterministic Massively Parallel Connectivity on Forests”

The author of this thesis served as the main author of the paper, writing approximately 80% of it. The thesis author developed the core technique of balanced exponentiation. Formulating the result as a connectivity algorithm without a runtime dependence on the number of nodes was a collaborative effort by all authors. The thesis author derived all technical proofs, with guidance from Yannic Maus. Alkida Balliu and Dennis Olivetti were responsible for reformulating the LCL solver, originally written by the thesis author in an unpublished manuscript.

Publication III: “Conditionally Optimal Parallel Coloring of Forests”

The idea of utilizing balanced exponentiation to solve the coloring problem was conceived jointly by Christoph Grunau and the author of this thesis.

The solution required generalizing balanced exponentiation, which was done by the thesis author, who wrote approximately 60% of the paper. The techniques for constructing and merging the necessary graph decompositions were developed collectively by all authors, with most of the writing carried out by Christoph Grunau. The introduction and motivation sections were written by Jara Uitto and Yannic Maus.

Publication IV: “Adaptive Massively Parallel Coloring in Sparse Graphs”

The author of this thesis served as the main author of the paper, writing approximately 80% of it. The thesis author developed most of the algorithmic ideas, including the key technique of coin-dropping along with its critical forwarding rules. Formulating the technique as an LCA was a joint effort by Jara Uitto and Yannic Maus. The section on various trade-offs between the number of colors and runtime was written by Shreyas Pai.

List of Figures

1.1	A colored map of South African provinces and its graph representation.	12
1.2	Graph G_1 is colored incorrectly and graph G_2 is colored correctly.	13
1.3	The immediate neighborhoods of every node in G_1 . The upper left-hand corner corresponds to the neighborhood of node 1, the upper right-hand corner to node 2, and so on. The neighborhoods with conflicting colors are highlighted.	14
1.4	The immediate neighborhoods of every node in G_2 . The upper left-hand corner corresponds to the neighborhood of node 1, the upper right-hand corner to node 2, and so on. .	15
1.5	Two different maximal matchings for the same graph, where dashed edges represent matched neighbors. The matching on the left is maximum, while that on the right is not. . . .	16
5.1	(Publication II, Figure 1) Light subtrees highlighted in green.	44
6.1	(Publication IV, Figure 1) An illustration of a β -partition where most nodes are assigned a layer.	50
6.2	(Publication IV, Figures 2 & 3) Examples of different dependency graphs.	50

1. Introduction

A *graph* is a mathematical structure consisting of *nodes* (or *vertices*) connected by *edges*, used to represent relationships between objects. If two nodes share an edge, they are called *neighbors*. The *topology* of a graph is the arrangement of its elements, and can be derived from structures such as social networks, transportation systems, molecular structures, or maps.

1.1 The Graph Coloring Problem

The *graph coloring problem* asks to color all nodes with a distinct color (e.g., green, blue, or yellow) such that all neighbors have different colors, while using as few colors as possible. When deriving a graph topology from a map, every country is represented by a node, and if two countries share a border, the corresponding nodes are joined by an edge. If a graph represents a map, its coloring corresponds to coloring every country on the map such that countries sharing a border have different colors. As an example, we color a map of South African provinces and show that it is equivalent to coloring its graph representation in Figure 1.1.

A trivial solution to the coloring problem would be to use a unique color for every node, in which case it is obvious that all neighboring nodes will have different colors. While this solution would be correct, it would use an excessive number of colors — the same number as there are nodes in the graph. Hence, the real challenge lies in using as few colors as possible, regardless of the number of nodes in the graph.

Coloring is often encountered in practice, without even thinking about it in terms of nodes and edges. The last time you saw a colored world map, neighboring countries were most likely colored with different colors for clarity. But how many colors were used? An even more intriguing question is, what is the smallest number of colors one has to use? An old graph theoretical result from 1890 by [Hea90] shows that 5 colors suffice for any imaginable map. This result was later improved to just 4 colors by [AH76], being the first major result with a computer-aided proof.

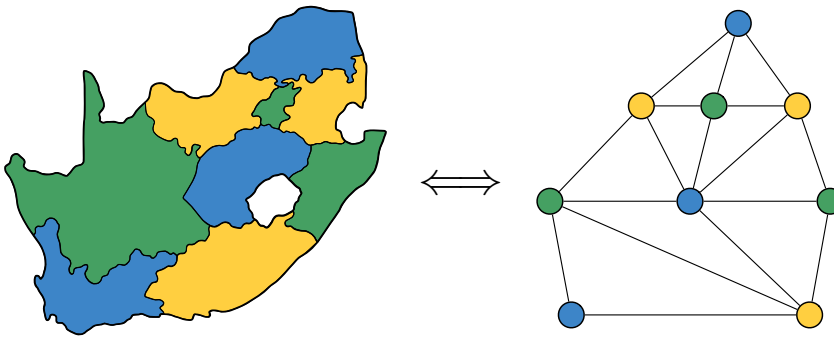


Figure 1.1. A colored map of South African provinces and its graph representation.

While coloring maps is great fun, it is not particularly useful in the grand scheme of things. Luckily, the coloring problem makes perfect sense for all kinds of graph structures, not only for those derived from maps. This makes graph coloring an important subroutine for solving advanced problems across computer science and other disciplines such as statistical mechanics, bioinformatics, and resource scheduling.

From a theoretical point of view, graph coloring is a fascinating problem in itself. The focus is often on minimizing the number of colors used. Let Δ denote the maximum number of neighbors any node has. A trivial greedy approach shows that every graph can be colored with $(\Delta + 1)$ colors: fix a color palette $1, \dots, \Delta + 1$, then visit every node, one by one, and color it with some color from the palette that is not used by any of its neighbors. This approach relies heavily on having exactly $\Delta + 1$ colors, and it fails when only Δ colors are available. This restriction is addressed by Brooks' Theorem, which gives a clean characterization of graphs that admit a Δ -coloring [Bro41; Lov75]. In general, coloring a graph with the minimum number of colors, whatever that number may be, is shown to be an NP-hard (essentially 'extremely difficult') problem by [Kar72].

In the context of distributed or parallel computing (defined formally in Chapter 2), finding a Δ -coloring is significantly more difficult than finding a $(\Delta + 1)$ -coloring; the latter can be solved by local greedy choices, whereas the former requires navigating global constraints.

Besides restricting the number of colors, one could also focus on certain types of graphs. How difficult is it to color a *path*, a *cycle*, or a *tree* graph? We will define these different graphs in Chapter 2.

Graph coloring is a *local problem*, in the sense that the correctness of a solution can be checked by inspecting only the immediate neighborhoods of every node in the graph. Problems that follow this general line of thought are called *locally checkable labeling problems* (LCLs), formally defined in Section 2.3. In addition to coloring, this umbrella term encompasses problems such as *maximal matching* and *maximal independent set*, which

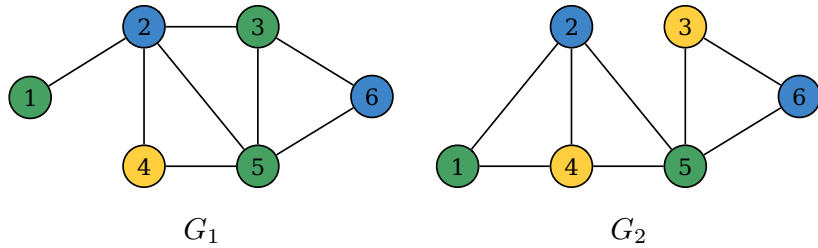


Figure 1.2. Graph G_1 is colored incorrectly and graph G_2 is colored correctly.

we will define in Section 1.2. What these problems have in common is that their solutions consist of labeling all nodes and/or edges with some label, and that the validity of any proposed solution can be verified by inspecting the local neighborhoods of every node.

On the other hand, if a problem has to satisfy some condition that depends on the whole topology of the graph, inspecting only the local neighborhoods of every node is not sufficient to validate a solution. Such problems are called *global*. In Section 1.3, we will define problems such as *maximum matching*, *maximum independent set*, and *connectivity*, which are global and whose candidate solutions may look valid in the local neighborhoods of every node, but could nevertheless turn out to be invalid when considering the whole graph.

This thesis is dedicated to solving problems from both ends of the locality spectrum by developing novel techniques in a modern parallel computing framework called the *Massively Parallel Computation (MPC) model*, defined formally in Section 2.5.

1.2 Local Problems

The protagonist of this thesis is graph coloring, which is the canonical example of an LCL problem. A coloring is correct, or is proper, if the color of every node differs from the color of its neighbors. Otherwise, the coloring is incorrect. Figure 1.2 presents a concrete example: a graph G_1 that is colored incorrectly and a graph G_2 that is colored correctly.

A candidate solution can be verified by only checking the immediate neighborhoods surrounding every node in the graph. If the solution is “locally” correct, meaning that the solution looks correct around every node in the graph, then the solution is also “globally” correct, and vice versa. Conversely, if there exists at least one node around which the solution looks invalid, then the whole solution is invalid. More precisely, we can validate a solution by inspecting and approving, one by one, the immediate neighborhood of every node in the graph. We illustrate this explicitly for graph G_1 in Figure 1.3 and for graph G_2 in Figure 1.4. We can do

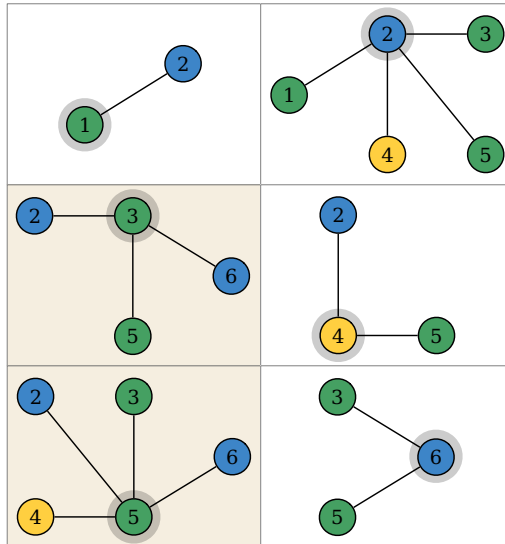


Figure 1.3. The immediate neighborhoods of every node in G_1 . The upper left-hand corner corresponds to the neighborhood of node 1, the upper right-hand corner to node 2, and so on. The neighborhoods with conflicting colors are highlighted.

this because the correctness requirement of coloring is local: if a node is colored blue, we only care that its neighbors are not colored blue, and we don't care about how the rest of the graph is colored (for the sake of this particular node). Figure 1.3 shows that the coloring of G_1 is incorrect, since the neighborhoods of nodes 3 and 5 reveal a conflicting coloring. Using the same verification method, Figure 1.4 shows that the coloring of G_2 is correct.

Another example of an LCL problem is the *maximal independent set* problem. Here, the goal is to select a set of nodes such that there are no neighboring nodes within this set, and no nodes can be added to the set without violating the aforementioned condition. In other words, every node in the graph either belongs to the maximal independent set or one of its neighbors belongs to it.

Much like the maximal independent set problem, but defined over edges rather than nodes, is the *maximal matching* problem. In this setting, the goal is to match neighboring nodes together, such that every node takes part in only one matching, and no additional matches can be made. In other words, neighboring nodes are either matched together or at least one of them is matched with some other node. See Figure 1.5 for an example.

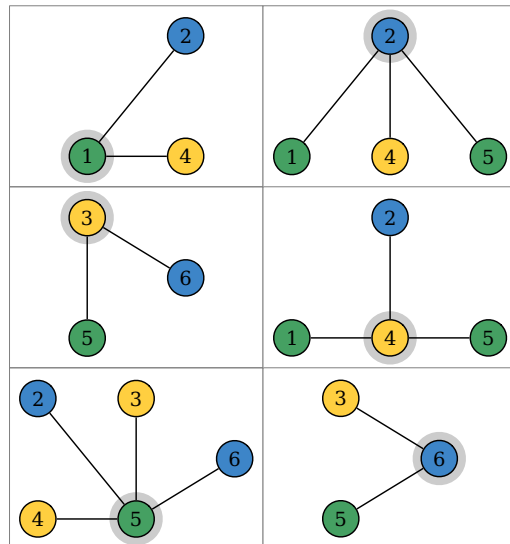


Figure 1.4. The immediate neighborhoods of every node in G_2 . The upper left-hand corner corresponds to the neighborhood of node 1, the upper right-hand corner to node 2, and so on.

1.3 Global Problems

Global problems have some inherently global conditions baked in that they are meant to satisfy, making it impossible to verify a candidate solution solely based on the local neighborhoods of the nodes.

Let us dissect a popular global problem — *maximum matching*. Note that there is a huge difference between maximal matching, which is an LCL problem, and maximum matching, which is a global problem. A maximal matching is maximum if the number of matches in the graph is the most you can have for a given graph. In Figure 1.5 we see two maximal matchings for the same graph. The left solution has 3 matches, while the right side has 2. One can prove that for this graph, 3 matches is the most you can have. Hence, the matching on the left side is maximum, while the one on the right is simply maximal. It is straightforward to realize that a candidate solution to the maximum matching problem can only be verified by looking at the whole graph, and not by inspecting the local neighborhoods.

Similarly to maximum matching, the *maximum independent set* problem is a global problem, whereas the maximal independent set problem is an LCL problem. A maximum independent set is a maximal independent set that is the largest you can have for a given graph.

Another example of a global problem is *connectivity*, where the goal is to assign a unique identifier to each *connected component* of the graph, which is a part of the graph where every node is connected via some series of edges. Specifically, two nodes must receive the same identifier if and only if

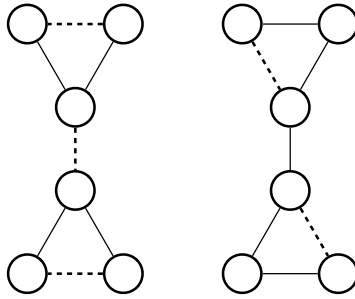


Figure 1.5. Two different maximal matchings for the same graph, where dashed edges represent matched neighbors. The matching on the left is maximum, while that on the right is not.

they belong to the same connected component. Connectivity is not an LCL problem: since a local neighborhood only ever contains nodes from the same connected component, it is impossible to verify (using local neighborhoods) whether nodes in different components have received different identifiers.

1.4 How to Solve Graph Problems?

So far, we have introduced graph problems and their solutions, without really diving into the algorithmic means to obtain them. What even is an *algorithm*? An algorithm is a precise sequence of instructions that tells a computer how to solve a problem. It entails a computational model, which is a set of ground rules to be followed when working on the solution, and some metric based on which the quality of an algorithm is evaluated. This metric is referred to as *computational complexity* or simply *complexity*, and it measures the amount of resources required to execute an algorithm.

The resource that is most commonly considered is *time*, which in itself is quite ambiguous. The earthly units of time, such as seconds and minutes, are not used to measure time complexity, since they depend on the choice of a specific computer and the evolution of hardware. For instance, a specific algorithm will run much faster on a modern computer than on one from the 90s. This, however, is not a virtue of the algorithm itself, but rather a consequence of technological advances in computer architecture and software. What time complexity aims to capture is the intrinsic time requirements of algorithms — the basic, immutable time units an algorithm would require on any computer. This is measured by tracking how many elementary operations, like summation, multiplication, and comparison, are executed until the algorithm terminates. Other complexities of interest besides time include *memory complexity* (computer memory that is required) and *communication complexity* (the necessary amount of communication between computational entities when there are multiple).

The model of computing that people unconsciously resort to when faced with a graph problem is *centralized*, which is indeed very intuitive. Here, the solver has a God’s eye view of the whole graph, and the complexity measure is time complexity: the number of elementary operations it takes to reach a solution. However, this is by far not the only computational model that makes sense to study.

1.4.1 Distributed Computing Models

In *distributed computing* models like *LOCAL* and *CONGEST*, which we define formally in Section 2.4, the approach to algorithm design is fundamentally different from the centralized setting. Here, every node is thought of as a processor with a unique identification number, and every edge is a communication link over which the endpoints can communicate. The goal is to design a communication protocol (algorithm) that is distributed to every node (processor) to follow and eventually reach a solution. One round of communication allows all nodes to communicate with all of their neighbors once. The complexity measure is communication: the number of communication rounds required to reach a solution.

For example, when solving a graph coloring problem, nodes communicate with their neighbors to exchange information, and eventually every node concludes that for the graph to be properly colored, they have to be labeled with a certain color. In fast *LOCAL* algorithms, nodes can only communicate with nodes that reside in close vicinity, highlighting the central theme of this framework — *locality*. What kind of problems can be solved based only on the information from nearby nodes, and how fast?

There are parallels to be drawn between distributed algorithms and nature. Ant colonies and schools of fish seemingly consist of identical independent entities that communicate only locally, while still achieving complex goals efficiently.

1.4.2 Modern Parallel Computing Models

Modern *parallel computing* models like *Massively Parallel Computation (MPC)* [KSV10; BKS17] and *Adaptive MPC* [Beh+19c] share some similarities with distributed models, but differ fundamentally in their communication capabilities. We define them formally in Sections 2.5 and 2.6. These parallel systems consist of multiple machines, each with a restricted amount of *local space* (or *local memory*), which refers to the amount of memory they have to store data. The amount of memory used by the whole system is called *global space* (or *total space*). Graph problems are solved in such a way that the input is first divided among the machines, so that each machine knows parts of the graph, but no machine knows the topology of the whole graph. One round of communication allows every machine to

communicate with all other machines, as long as the message sizes are small and machines do not exceed their local space. In this setting, there are two complexity measures of interest: how many rounds of communication are required to reach a solution (communication complexity) and how much local and global space is required throughout the algorithm execution (memory complexity).

Note the difference in communication capabilities between these parallel and distributed models. In parallel models, machines can communicate with all other machines, whereas distributed models allow communication only across the edges of a given graph. The main challenge of modern parallel models is hence related to the communication complexity: how to efficiently utilize this all-to-all communication? Another major challenge is related to the memory complexity: how to solve problems space-efficiently, without needing to store too much auxiliary data?

By allowing adaptive queries to a distributed data store on top of the MPC model, we arrive at the Adaptive MPC model, AMPC for short. This model is a practically motivated extension of the MPC model, which takes into account the more advanced capabilities of real-world data centers, while still serving as a clean theoretical framework for analysis.

The MPC model and its variants are inspired by real-life data centers, where communication is the most time-consuming part of the operation, and where memory is a limited resource. They are used as abstractions of modern frameworks of parallel computing, such as Hadoop [Whi09], Spark [Zah+10], MapReduce [DG08], and Dryad [Isa+07].

1.5 Research Output and Roadmap

In this thesis, we consider fundamental graph problems for uniformly sparse graphs in the MPC model, focusing on memory-optimal solutions. We resolve open problems and advance the state of the art by developing novel deterministic algorithms for graph problems such as connectivity, vertex coloring, maximal independent set, and maximal matching.

Most of our algorithms are memory-optimal in two ways. First, the local space of individual machines is strongly sublinear in the size of the input. Second, the total space used by the whole system remains linear in the size of the input — the best one can hope for.

By focusing on uniformly sparse graphs, we are forced to develop novel algorithmic techniques, as current approaches fundamentally do not extend to memory-optimal algorithms for these graphs. Even when not striving for memory-optimal algorithms, these graph families are hard instances to solve, since they form the setting for most known lower bounds in MPC.

Chapter 2 is dedicated to basic graph theoretical notations and the formal definitions of the LOCAL model, the MPC model, and the AMPC model. In

Chapter 3, we give an overview of the contributions of this thesis and the related work. In Chapters 4 to 6 we dive deeper into the individual results of this thesis.

In Chapter 4, we present the results of Publication I, which include exponentially faster algorithms for LCL problems on forests in the MPC model compared to the LOCAL model. In Chapter 5, we present the results of Publication II and Publication III, which give the first deterministic $O(\log D)$ -round algorithm for connectivity, as well as the first $O(\log \log n)$ - and $O(\log D)$ -round algorithms for 3-coloring, maximal independent set, and maximal matching on forests. Here, n denotes the total number of nodes in the graph, and D denotes the maximum diameter of the graph. Publication II and Publication III are grouped because both rely on our novel graph exploration technique called *balanced exponentiation*, which is one of the major technical contributions of this thesis. This technique has been developed into a parameterized, standalone tool that allows nodes to explore surrounding sparse regions of the graph without prior knowledge of their location, while incurring minimal memory overhead. Lastly, in Chapter 6, we present the results of Publication IV, which include AMPC algorithms for *arboricity-dependent* colorings. Arboricity is a general measure of the density of a graph, and is defined formally in Chapter 2. We present several algorithms with varying trade-offs between the number of colors and runtime. Most notably, we achieve a constant-round $O(\alpha)$ -coloring algorithm for graphs with bounded arboricity α .

2. Preliminaries

This chapter is not meant to serve as an exhaustive survey of distributed and parallel computing, but rather as prerequisite information for understanding the contributions of this thesis.

The theory of computation focuses on the asymptotic complexity of functions, ignoring constant factors and lower-order functions. Big- O , Little- o , Big- Ω , Little- ω , and Θ are the basic formal notations for stating the growth of a function. Formally,

- $T(n)$ is $O(f(n))$, iff¹ for some constants c and n_0 , $T(n) \leq c \cdot f(n)$, $\forall n \geq n_0$.
- $T(n)$ is $\Omega(f(n))$, iff for some constants c and n_0 , $T(n) \geq c \cdot f(n)$, $\forall n \geq n_0$.
- $T(n)$ is $\Theta(f(n))$, iff $T(n)$ is $O(f(n))$ and $T(n)$ is $\Omega(f(n))$.
- $T(n)$ is $o(f(n))$, iff $T(n)$ is $O(f(n))$ and $T(n)$ is not $\Theta(f(n))$.
- $T(n)$ is $\omega(f(n))$, iff $T(n)$ is $\Omega(f(n))$ and $T(n)$ is not $\Theta(f(n))$.

Informally, Big- O describes the upper bound, Big- Ω describes the lower bound, Θ describes the exact bound, and Little- o (Little- ω) describes the upper (lower) bound that can never be reached.

2.1 Graph Notations

A graph $G = (V, E)$ is a pair of sets V and E , such that $E \subseteq [V]^2$. The elements of V are called nodes or vertices, and the elements of E are called edges. The number of nodes in the graph is denoted by $n = |V|$ and the number of edges in the graph is denoted by $m = |E|$. A common way to picture a graph is by drawing a circle for each node and joining two of these

¹iff stands for “if and only if”

circles with a line if the corresponding two nodes form an edge. For this thesis, we assume that graphs are undirected, finite, and simple.

The set of neighbors of a node $v \in V$ in a graph G is defined as $N_G(v) := \{u \mid \{u, v\} \in E\}$. The degree of a node v refers to the number of neighbors v has in G and is defined as $\deg_G(v) := |N_G(v)|$. The maximum degree of any node in G is denoted by Δ_G . The distance $\text{dist}_G(u, v)$ between two vertices $v, u \in V$ is the length of a shortest path between nodes v and u in G . If no such path exists, we set $\text{dist}_G(v, u) = \infty$. Note that in undirected graphs, it holds that $\text{dist}_G(v, u) = \text{dist}_G(u, v)$. The largest distance between any two vertices in G is the diameter of G , denoted by $\text{diam}(G)$. The r -hop neighborhood of a node v , or in other words an r -radius ball around node v , is the subgraph $B_G(v, r) = (V(v, r), E(v, r))$, where

$$V(v, r) = \{u \in V \mid \text{dist}(u, v) \leq r\}, \text{ and}$$

$$E(v, r) = \{\{u, w\} \in E \mid \text{dist}(v, u) \leq r \text{ and } \text{dist}(v, w) \leq r\}.$$

If the underlying graph G is clear from the context, we may omit the subscript G from all of the notation above.

Consider graphs $G = (V, E)$ and $H = (V', E')$. If $V' \subseteq V$ and $E' \subseteq E$, we say that H is a subgraph of G . If $V' \subseteq V$ and $E' = \{\{u, v\} \in E \mid u, v \in V'\}$, we say that H is a subgraph induced by the node set V' , and it can be written as $G[V']$.

A graph $G = (V, E)$ is called *connected* if $\text{dist}(v, u) \neq \infty, \forall v, u \in V$. If the graph is not connected, it is *disconnected* and consists of multiple *connected components*. A connected graph where all nodes have degree 2 is a *cycle*. A connected graph where two of its nodes have degree 1, and all other nodes have degree 2, is a *path*. A connected graph without cycles as subgraphs is called a *tree*. A disconnected graph consisting of multiple paths/cycles is called a collection of paths/cycles. A disconnected graph consisting of multiple trees is called a *forest*.

Arboricity is a general measure of density. The arboricity α of a graph is defined as the smallest number of forests into which its edge set can be partitioned. More precisely, for any subgraph S of a graph G , let n_S and m_S denote the number of vertices and edges in S . Then the arboricity of G equals

$$\max_S \lceil m_S / (n_S - 1) \rceil, \quad n_S \geq 2.$$

Low-arboricity graphs are uniformly sparse, in the sense that they cannot contain dense subgraphs. Conversely, any graph with high arboricity is guaranteed to contain a dense subgraph.

2.2 Nash-Williams Decomposition

Due to Nash-Williams [Nas64], it is well known that the edge set E of any graph $G = (V, E)$ with arboricity α can be decomposed into α edge-disjoint forests. This fundamental theorem has many applications in graph theory and combinatorics.

The first efficient distributed algorithm for computing such decompositions was given by Barenboim and Elkin [BE08], who coined the term *H-partitions* for the resulting constructions. An *H-partition* entails partitioning the vertex set of the graph into $l = \lfloor 2/\varepsilon \cdot \log n \rfloor$ disjoint subsets H_1, H_2, \dots, H_l that satisfy that every vertex $v \in H_i, i \in \{1, 2, \dots, l\}$, has at most $(2 + \varepsilon)\alpha$ neighbors in the vertex set $\cup_{j=i}^l H_j$. It is worth noting that Miller and Reif introduced a related concept in the parallel setting for trees [MR85], where a logarithmic number of so-called *rake* and *compress* operations remove all nodes from a tree.

Many algorithms of this thesis will involve computing *H-partitions*, or suitable variants thereof, for low-arboricity graphs efficiently and in parallel.

2.3 Locally Checkable Labeling (LCL) Problems

In their seminal work [NS93], Naor and Stockmeyer introduced the notion of *locally checkable labeling problems* (LCLs, for short). Their definition focuses on problems where nodes are labeled (e.g., vertex coloring), though they remark that an analogous definition applies to problems where edges are labeled (e.g., edge coloring). A modern formulation that unifies both perspectives and their combinations labels *half-edges*, i.e., pairs (v, e) where e is an edge incident to vertex v . Furthermore, it is known that on trees every LCL problem can be expressed in a special form called *node-edge-checkable LCL problems* [Bal+21; GRB22]. Next, we define half-edge labelings formally and then present this modern definition of LCL problems.

Definition 2.3.1 (Publication I, 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 types of half-edge labelings: *input labelings*, which are part of the problem instance, and *output labelings*, which are produced by an algorithm executed on the instance. For our purposes, we assume that any input graph G is equipped with an input labeling $g_{\text{in}}: H(G) \rightarrow \Sigma_{\text{in}}$, where Σ_{in} denotes the *set of input labels*. If an LCL problem has no input labels, we simply take $\Sigma_{\text{in}} = \perp$ and assume that each node is labeled with \perp .

Although the formal definition of a node-edge-checkable LCL (see below) may seem complicated, the underlying intuition is simple: we specify (i) a set of allowed output label combinations around nodes, (ii) a set of allowed output label combinations on edges, and (iii) a set of allowed input-output label combinations. A correct solution to the LCL must satisfy all of these constraints.

Definition 2.3.2 (Publication I, 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 .*

An algorithm \mathcal{A} is said to *solve* an LCL problem Π on a graph class \mathcal{G} if it produces a correct solution for Π on every $G \in \mathcal{G}$. Note that the definitions of LCL problems above implicitly require that class \mathcal{G} has constant degree.

2.4 The LOCAL Model

Distributed computing [Pel00] studies the capabilities and limitations of a network, where every node is occupied by a computing entity (computer, processor) and messages can be exchanged only over the links of this network. The primary focus of this research is *locality*: what problems can be solved when each processor makes decisions based only on information from nearby nodes.

The LOCAL model [Lin92] has been the standard model for studying locality in distributed computing for over 30 years, see surveys by [Suo13; Roz24]. In this model, a given network is modeled as a connected simple graph $G = (V, E)$, with nodes being processors that can communicate with each other only over the edges of the graph G . Every node has a unique identification number (ID) of size $O(\log n)$ bits, where n is the number of nodes in G . Initially, every node is only aware of its immediate neighbors. The model is synchronous, so the computation proceeds in sequential rounds with all nodes starting at the same time. In each round, every node sends messages to its neighbors, receives messages from its neighbors, and performs some unlimited computation. The message sizes are unbounded, and nodes are assumed to be infinitely powerful, as in any computation finishes instantly. Every node aims to output its part of the solution, e.g., in the case of coloring, nodes only need to output their own color. The communication complexity, or round complexity, of an algorithm is the number of rounds until all nodes have finished. The system is fault-free, meaning that nodes never crash and messages never get corrupted.

There is an important alternative way to view LOCAL algorithms as pointed out by [Lin92]. Any t -round algorithm \mathcal{A} can be simulated by another t -round algorithm \mathcal{B} which proceeds in two distinct phases. First, every node gathers all the information about the network at a distance t around it. Second, every node simulates the behavior of the original algorithm \mathcal{A} without communicating with other nodes. In other words, any t -round LOCAL algorithm is simply a function from the ball $B_G(v, t)$ of radius t around every node $v \in V$, to an output set.

There are two randomized variants of the LOCAL model. One allows private randomness, as in every node is given access to an infinite string of random bits that is guaranteed to be independent of the strings of other nodes in the graph. Another allows shared randomness, where nodes are given simultaneous access to the same infinite string. Both variants have been proven to be stronger models than the deterministic LOCAL model for many problems [Bal+20b; Bal+25].

The LOCAL model is a particularly clean and streamlined model that has been the medium for a very active line of research for over 30 years. However, despite all of its merits, the model has been subject to criticisms as far as practical applications are concerned. Being fault-free, synchronous, and having unbounded message sizes and computational power is not something one can assume of a practical computing system. However, it's worth emphasizing that this apparent weakness is simultaneously a strength: this ensures that any LOCAL lower bounds also hold in strictly weaker models of computing. If some problem is impossible to solve faster than, say, $O(\log n)$ time using unbounded message sizes and computational power in LOCAL, it must also be impossible in models that limit message sizes and computing power.

2.5 The Massively Parallel Computation (MPC) Model

The Massively Parallel Computation (MPC) model [KSV10; BKS17] is a mathematical abstraction of modern frameworks of parallel computing such as Hadoop [Whi09], Spark [Zah+10], MapReduce [DG08], and Dryad [Isa+07]. In the MPC model, there are M machines, each with local space S . Local space refers to the amount of memory each machine has to store data. Machines can communicate with each other in an all-to-all fashion, in synchronous rounds. Within a round, every machine sends messages to any other machine, receives messages from other machines, and then performs some unbounded computation on local data. Initially, an input graph $G = (V, E)$ consisting of n nodes and m edges is arbitrarily and equally distributed among the machines. Every node has a unique identification number (ID) of size $O(\log n)$ bits. The time complexity, or round complexity, of an algorithm is the number of rounds it takes until every machine knows the output for each node and/or edge it holds in memory, e.g., whether or not an edge belongs to the maximal matching.

The MPC model is typically divided into three regimes according to the size of the local space S . The memory unit used is a *word* of size $O(\log n)$ bits, which is enough to store a node or a machine identifier from a polynomial (in n) domain. The *superlinear* regime allows for $S = n^{1+\Omega(1)}$ words of space, and the *linear* regime allows for $S = \tilde{O}(n)$ words². The local space of a machine restricts the amount of data a machine can store during the execution of an algorithm, and hence also restricts the amount of data a machine is allowed to send and receive within one round. Both linear and superlinear regimes allow for very fast algorithms because machines can get a “global view” of the graph in the sense that they can store some information about every node in the graph. Due to the size of ever-growing real-world graphs, it is impractical to assume that a single machine can get such a global view. Hence, the research in recent years has focused on the most challenging *low-space (scalable, sublinear)* regime with $S = O(n^\delta)$, for some constant $\delta \in (0, 1)$. In this regime, it is possible that a machine cannot even store its immediate neighborhood in its local space. As each machine can only gather a small amount of local graph information around a node, the low-space regime is closely connected to the study of locality and the LOCAL model of distributed computing.

The total space T that the collection of M machines uses has historically been an overlooked metric of the model. However, it has gained traction in recent years, partly due to the contributions of this thesis. The aim is to use as little total space as possible, with linear $T = \Theta(n + m)$ being the gold standard, as $\Omega(n + m)$ words are *required* to even store the input graph.

²The \tilde{O} notation hides polylogarithmic factors, so $\tilde{O}(f(n))$ equals $O(f(n) \cdot \log(f(n)))$.

2.5.1 Node-Centric Approach

To make graph algorithm design simpler in the MPC model, one often adopts a node-centric framework, which has some conceptual connections to the LOCAL model. This approach is common in literature, but is rarely written down explicitly. In this approach, every machine is assumed to simulate several virtual machines, such that every node in the graph is assigned to a unique virtual machine. Now, an algorithm can be designed from a node's perspective, like in the LOCAL model, except that it can communicate with any other node in the graph directly. This approach comes with one major caveat with regard to total space.

We would like to assume that during an algorithm, every virtual machine hosting a node has S local space available. However, if we were to reserve this amount of space for every virtual machine, it would defeat the purpose of having virtual machines, since every node would require its own physical machine, and the total space would always be $\Theta(n \cdot S)$. The workaround for obtaining better total space bounds is as follows. During every round of an algorithm, we bound the total space by $O(\sum_{v \in V} S_v)$, where $S_v = O(S)$ is how much local space the virtual machine hosting a node v is actively using. We assume that the underlying architecture does load balancing for us and re-shuffles the virtual machines between physical machines so that the local space S of every physical machine is not violated.

2.5.2 Graph Exponentiation

A technique called *graph exponentiation* [LW10] is so integral to the MPC model that it is worth defining as a preliminary.

Lemma 2.5.1 (Graph exponentiation). *Let $G = (V, E)$ be a graph. In the low-space MPC model, every node $v \in V$ can collect its 2^r -hop neighborhood $B(v, 2^r)$ in $O(r)$ rounds as long as for every node $v \in V$ it holds that $|B(v, 2^r)| \leq n^\delta$.*

Proof. Initially, every node $v \in V$ knows its 1-hop neighborhood $B(v, 1)$. For the induction step, consider a node u in some round $t < r$ that knows $B(u, 2^t)$. Node u can communicate with every node $v \in B(u, 2^t)$ and acquire their neighborhoods $B(v, 2^t)$, effectively collecting $B(u, 2^{t+1})$. By induction, node u collect $B(u, 2^r)$ after $O(r)$ rounds. \square

2.5.3 Lower Bounds and the 1 vs. 2 Cycles Conjecture

All current lower bounds in low-space MPC are conditional, meaning that they hold if some other commonly assumed statement is true. The conjecture, on which essentially all lower bounds in this model condition, is the widely believed 1 vs. 2 cycles conjecture. It states that $\Omega(\log n)$ rounds

are required to distinguish between an n -node cycle and two $n/2$ -node cycles [GKU19].

It is worth noting that establishing unconditional lower bounds appears unattainable. Any unconditional non-constant lower bound in the low-space MPC model for any problem in P would imply a breakthrough result in circuit complexity, namely a separation between NC^1 and P [RVW16], which seems far beyond the reach of current techniques [RR97; AB09].

2.5.4 Component Stability

The concept of component-stable low-space MPC algorithms was introduced by [GKU19] as the first broad class of MPC algorithms for which meaningful conditional lower bounds can be derived. This is achieved by lifting lower bounds in the LOCAL model to the MPC model via a sophisticated lifting framework. In essence, algorithms in this class require that the outputs of nodes belonging to different connected components are independent.

After its introduction, the notion of component stability has been refined by [CDP21a], encompassing many existing randomized MPC algorithms, and allowing for more robust conditional lower bounds. These refinements extend the original framework to also cover deterministic algorithms as well as algorithms whose runtime depends on the maximum degree Δ .

Definition 2.5.2 (Component-stability, [CDP21a]). *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 S . That is, the output of A_{MPC} at v can be expressed as a deterministic function $A_{\text{MPC}}(CC(v), v, n, \Delta, S)$. A deterministic MPC algorithm A_{MPC} is component-stable under the same definition, but omitting dependency on the random seed S .*

This thesis adopts the revised version of component stability by [CDP21a].

2.6 The Adaptive MPC Model

The Adaptive MPC (AMPC) model [Beh+19c] is designed to reflect the challenges faced by modern large-scale data processing platforms. It extends the standard MPC model in a practically motivated way: communication takes place through distributed key-value storage rather than all-to-all message passing. In this model, there are P machines, each equipped with $S = O(n^\delta)$ local space where $\delta \in (0, 1)$ and n denotes the number of vertices in the input graph $G = (V, E)$. The total space across all machines is $T = S \cdot P$. The basic unit of measure is a word consisting of $O(\log n)$ bits.

Communication among machines is represented by a sequence of distributed data stores (DDS), denoted by D_0, D_1, \dots, D_f . Each DDS supports key-value semantics: it maintains collections of key-value pairs, where querying with a key retrieves its associated value. Both keys and values are of constant size. The initial input resides in D_0 , which is indexed by a set of keys known to all machines, such as consecutive integers. The final solution to the problem must be written into the last data store D_f . If a key x is associated with $k > 1$ values in some DDS, then these values can be accessed via extended keys $(x, 1), \dots, (x, k)$, where the indices from 1 to k are assigned arbitrarily. A query to a key not present in the DDS returns an empty result.

The computation proceeds in synchronous rounds. In round i , each machine reads from D_{i-1} and writes to D_i . During a round, a machine may issue $O(S)$ queries (reads) and perform $O(S)$ writes, in addition to carrying out unrestricted computation on local data. A query with key x corresponds to reading the value linked with x in D_{i-1} , and each write represents storing a single key-value pair in D_i .

The defining feature of the model is *adaptivity*, which distinguishes it from the standard MPC model. Specifically, queries within the same round may depend on one another. For example, suppose g is a function mapping a set X to itself, and for every $x \in X$, the store D_{i-1} contains a pair $(x, g(x))$. Then, in round i , a machine can compute $g^k(y)$ for any $y \in X$, provided $k = O(S)$. A more detailed discussion of the practical realism of the AMPC model can be found in [Beh+19c].

3. Contributions and Related Work

There is a successful line of research on MPC algorithms that are exponentially faster than the best LOCAL algorithms. Most such results are achieved with *superlinear global space*, that is, $\omega(m+n)$ words [Beh+19a; GGJ20; CDP21b; CDP21a], where n is the number of nodes and m the number of edges in the input graph. One of the main themes of this thesis is that we deviate from the usual approach of using superlinear global space, and instead strive for strictly *linear global space*, forcing us to develop novel algorithmic techniques. We restrict ourselves to optimal MPC parameters across the board: we only allow $O(n^\delta)$ words of local space and $O(m+n)$ words of global space. A simple example illustrates why minimizing global space matters: for an input of one million data points, would you prefer storing 10^6 values (linear global space) or 10^{12} (quadratic global space)?

Most of our algorithms are for forests and other low-arboricity graphs. At first glance, it may seem that forests are an easy graph class for which to develop algorithms because they are sparse. We would like to point out that, when striving for linear global space, the converse is actually true. When the input graph is dense, it requires a huge amount of global space to store it, which can be leveraged by an algorithm via *sparsification*. Indeed, a common algorithm design pattern for dense graphs is to first sparsify the input graph, i.e., make the graph less dense, and then solve the problem in this sparser instance [And+18; GU19; CDP21a; CDP20] using superlinear global space with respect to the current graph size, while remaining linear with respect to the original graph size. An additional step is then needed to show how a solution for the original input can be derived from a solution for the sparsified graph. Such techniques are fundamentally incompatible with low-arboricity graphs such as forests, as they are already maximally sparse by definition.

Another compelling reason to focus on trees and forests is their central role in understanding problem complexity in the LOCAL model: many problems of interest are challenging even on trees, and often even in regular balanced trees of small degree. In fact, most known lower bounds in the LOCAL model are established in this setting. Through lifting techniques, these lower

bounds naturally extend to forests in the MPC model. Therefore, decreasing the significance of trees and forests would require a fundamental shift in techniques: either the development of entirely new lower bound frameworks for the LOCAL model, along with new lifting theorems, or novel lower bound techniques tailored directly for the MPC model.

3.1 Exponential Speedup Over Locality

The central result of Publication I is that for any LCL problem (see Section 2.3) on trees with deterministic (respectively, randomized) complexity T in the LOCAL model, we can systematically derive an exponentially faster MPC algorithm with deterministic (respectively, randomized) complexity $O(\log T)$ on forests. While there have been prior works focusing on individual LCL problems in the MPC model, this kind of automated procedure that solves all LCL problems is the first of its kind. In particular, we establish the following.

Theorem 3.1.1. *[Publication I, 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.*

A rich line of work [NS93; Cha+19b; CP17; Bal+19; Bal+18; Bal+20a; Cha20; GRB22] recently came to an end, showing that a problem in the LOCAL model can only have deterministic complexity

$$f(n) \in \{\Theta(1), \Theta(\log^* n), \Theta(\log n)\} \cup \{\Theta(n^{1/k}) : k \in \mathbb{N}\} .$$

We show that knowing the asymptotic value of $f(n)$ is enough to obtain a deterministic MPC algorithm with complexity

$$O(\log(f(n))) \in \{O(1), O(\log \log^* n), O(\log \log n), O(\log n)\} .$$

Furthermore, it is known that for all complexities $f(n) \notin \Theta(\log n)$, the randomized and deterministic LOCAL complexities coincide. However, when $f(n) \in \Theta(\log n)$, the randomized LOCAL complexity $g(n)$ can be either $\Theta(\log n)$ or $\Theta(\log \log n)$. In the latter case, we present an MPC algorithm with randomized complexity $O(\log \log \log n)$. If the component-stability (see Section 2.5.4) condition is dropped, the same $O(\log \log \log n)$ complexity can be achieved deterministically in the MPC model.

Theorem 3.1.2. *[Publication I, 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.

The work of [GKU19] (Theorem I.4) and [CDP21a] (Theorem 14) show that Theorem 3.1.1 is optimal for LCL problems with complexity $O(\log n)$ in the LOCAL model: if a problem requires T rounds in the LOCAL model, then any component-stable algorithm in the low-space MPC model requires $\Omega(\log T)$ rounds — assuming the widely believed 1 vs. 2 cycles conjecture holds (see Section 2.5.3).

Consider an LCL problem \mathcal{P} with deterministic complexity $\Theta(\log n)$ and randomized complexity $\Theta(\log \log n)$ in the LOCAL model. By [CDP21a], problem \mathcal{P} has a low-space MPC lower bound of $\Omega(\log \log n)$ for deterministic component-stable algorithms. In Theorem 3.1.2, we show that a component-unstable algorithm can solve problem \mathcal{P} deterministically in $O(\log \log \log n)$ rounds. In other words, we show that the lower bounds obtained by the lifting framework of [GKU19] can be bypassed for all LCL problems in the aforementioned complexity class with component-unstable algorithms. Obtaining the same improvement for even a single problem while preserving component stability would constitute a breakthrough, as it would refute the 1 vs. 2 cycles conjecture mentioned in Section 2.5.3.

As a key subroutine for speeding up problems to $O(\log n)$ complexity in the MPC model, we provide a component-stable MPC algorithm for rooting a forest in $O(\log n)$ rounds. This algorithm supports arbitrary degrees and may be of independent interest.

3.2 Forest Connectivity

The primary contribution of Publication II is a deterministic algorithm that detects the connected components of a forest in time logarithmic in the maximum diameter of any component. Crucially, without any dependence on the total number of nodes n .

The problem of connectivity has attracted considerable attention in recent years. Most notably by [And+18] who developed a randomized $O(\log D \cdot \log \log n)$ algorithm for general graphs and by [Beh+19b] who achieved a randomized $O(\log D + \log \log n)$ algorithm for general graphs. Later, [CC22] derandomized the work of [Beh+19b], achieving the same runtime. Another notable result is by [ASW19] who designed an $O(\log \log n + \log(1/\lambda))$ -round algorithm that finds all connected components with spectral gap at least λ .

Our result contrasts these prior approaches, where the runtime seems to inherently depend on n . Our algorithm is deterministic and has a runtime of $O(\log D)$ for forests. Moreover, we prove that our algorithm is asymptotically optimal under the 1 vs. 2 cycles conjecture. Formally, we prove the following statement.

Theorem 3.2.1. [Publication II, Theorem 1.1] *Consider the family of forests. There is a deterministic low-space MPC algorithm to detect the connected components on this family of graphs. In particular, each node learns the maximum ID of its component. The algorithm runs in $O(\log D)$ rounds, where D is the maximum diameter of any component. The algorithm requires $O(n + m)$ words of global memory, it is component-stable, and it does not need to know D . Under the 1 vs. 2 cycles conjecture, the runtime is asymptotically optimal.*

We can extend the techniques from Theorem 3.2.1 to also obtain a rooting algorithm with the same time and space complexity guarantees.

Theorem 3.2.2. [Publication II, Theorem 1.2] *Consider the family of forests with component-wise maximum diameter D . There is a deterministic low-space MPC algorithm that roots the forest in $O(\log D)$ rounds using $O(n + m)$ words of global memory, and it is component-stable.*

While rooting a forest is not particularly groundbreaking, it does provide a convenient handle for algorithm design and symmetry breaking in low-space MPC. As a concrete example, consider the 2-coloring problem. Without a rooting in place, this is a daunting task. However, if you can invoke a $O(\log D)$ time rooting algorithm as a black-box, the task of 2-coloring in $O(\log D)$ time becomes almost trivial. Without going into technical details, this can be achieved using a rather simple and efficient algorithm: each node is colored based on the parity of its distance to the root, which can be computed via *pointer jumping*, which is a similar technique to graph exponentiation of Lemma 2.5.1.

Observe that the rooting algorithm in Theorem 3.2.2 supports *arbitrary* degrees, making it the only known $O(\log D)$ time forest rooting algorithm. Prior connectivity algorithms for general graphs by [Beh+19b; CC22] yield slightly slower rooting algorithms, achieving a runtime of $O(\log D \cdot \log \log n)$.

Our final contribution is a component-stable algorithm for solving any LCL problem (which includes 2-coloring) deterministically in $O(\log D)$ time. Our result employs the rooting algorithm of Theorem 3.2.2 as a subroutine. Furthermore, we show that if $D \in \Omega(\log n)$ and $D \in n^{o(1)}$, there cannot exist (under the 1 vs. 2 cycles conjecture) an MPC algorithm that solves all LCL problems in time $o(\log D)$ on forests with component-wise maximum diameter D . This claim holds even for component-unstable algorithms and even when allowing $\text{poly}(n)$ global space. This implies that our $O(\log D)$ time LCL solver is conditionally optimal.

Theorem 3.2.3 (Publication II, Theorem 1.3). *Consider an LCL problem Π on forests and let D be the component-wise maximum diameter. There is a deterministic low-space MPC algorithm that solves Π in $O(\log D)$ rounds using $O(n + m)$ words of global memory. Under the 1 vs. 2 cycles conjecture, the runtime is asymptotically optimal.*

3.3 Coloring, MIS and Maximal Matching

Since LCL problems are limited to graphs with only constant degree by definition, a very natural question to ask is how fast can we solve classical symmetry-breaking problems such as coloring, maximal independent set (MIS), and maximal matching on forests when degrees are arbitrary?

A long line of research has studied this question [BFU19; Beh+19a; LU21; GGJ20; FGG23]. In Publication III, we consolidate and unify these works by presenting a conceptually simple deterministic algorithm that simultaneously solves 3-coloring, MIS, and maximal matching. Our approach is based on computing a variant of the classical H -partition (see Section 2.2), which provides a unified framework for addressing these problems. While H -partitions are a natural tool for trees and forests, efficiently computing them in the MPC model had remained beyond the reach of prior techniques.

Theorem 3.3.1 (Publication III, Theorem 1). *There are deterministic $O(\log \log n)$ -round low-space MPC algorithms for 3-coloring, maximal matching, and maximal independent set (MIS) on forests. These algorithms use $O(n)$ global space.*

Our algorithms are conditionally optimal under the 1 vs. 2 cycles conjecture, at least when restricted to the class of component-stable algorithms.

Algorithms achieving similar guarantees for maximal matching and maximal independent set on forests were recently obtained by [FGG23]. However, their methods are highly technical, and they rely on sophisticated derandomization machinery. Moreover, their techniques inherently cannot extend to coloring with a small number of colors. Our main result, which is 3-coloring, is widely considered to be harder than MIS and maximal matching. For example, once a graph is 3-colored, an MIS can be trivially computed in constant time. Furthermore, prior MPC algorithms for MIS and maximal matching crucially exploit the property that any partial solution can be extended to a full solution for the entire graph — a property that does not hold for 3-coloring.

The best previously known deterministic algorithm for 3-coloring trees, due to the author of this thesis in an unpublished manuscript [LU21], achieves a runtime of $O(\log^2 \log n)$. Allowing one extra color and randomization makes the problem substantially easier: the best such algorithm [GGJ20] uses 4 colors and runs in $O(\log \log n)$ rounds using a divide-and-conquer strategy. Each node joins one of two partitions uniformly at random, creating induced connected components of logarithmic diameter. Due to their small diameter, each component can then be 2-colored independently in $O(\log \log n)$ rounds in a straightforward manner.

How is this different to LCLs? It is worth emphasizing that the problems addressed in Theorem 3.3.1 are not LCL problems because the degrees of the graphs can be arbitrary, whereas LCL problems are only defined

for constant-degree graphs. This seemingly minor change in the initial conditions leads to a fundamentally harder algorithmic setting.

One might ask whether the techniques developed in Publication I can be ported to this setting? Unfortunately, the answer is no. The underlying techniques in Publication I are either too slow or heavily exploit the fact that nodes have constant degree.

Another avenue of approach could be to employ the LCL solver from Publication II (Theorem 3.2.3), since under the hood, this solver can handle arbitrary degrees. However, this approach is also too slow. The solver runs in $O(\log D)$ rounds, where D is the component-wise maximum diameter, which in the worst case, can result in an $O(\log n)$ runtime.

3.4 Coloring Sparse Graphs in AMPC

The problem of finding arboricity-dependent colorings has received significant attention across various settings, including LOCAL [BE08; BE10; Bar15; FHK16; Mau21; MT20], MPC [GS19; FGG23; BCG20; GG25], dynamic algorithms [HNW20; CNR23; CR22; Bha+24], and streaming algorithms [BCG20]. Obtaining an $O(\alpha)$ -coloring on graphs with arboricity α efficiently remains a challenging open problem in low-space MPC. A step towards solving this open problem was taken by [GG25], who gave a $\text{poly}(\log \log n)$ algorithm for $O(\alpha \log \log n)$ -coloring. The solution relies on computing an edge orientation such that the maximum out-degree of any node is $O(\alpha \log \log n)$.

By shifting from the MPC to the AMPC model, we gain a new avenue of attack. While a randomized constant-time¹ $(\Delta + 1)$ -coloring algorithm in the AMPC model is already known [Cha+19a], this number of colors is often far from optimal for sparse graphs, where Δ can be significantly larger than α . In Publication IV, we present deterministic AMPC algorithms that in constant, or near-constant, time give $\text{poly } \alpha$ - and $O(\alpha)$ -colorings for sparse graphs.

Theorem 3.4.1 (Publication IV, Theorem 1.3). *For any constant $\varepsilon > 0$ on graphs with possibly non-constant arboricity α , there are deterministic low-space AMPC algorithms to compute:*

1. An $O(\alpha^{2+\varepsilon})$ -coloring in $O(1/\varepsilon)$ rounds.
2. An $O(\alpha^2)$ -coloring in $O(\log \alpha)$ rounds.
3. A $((2 + \varepsilon)\alpha + 1)$ -coloring in $\tilde{O}(\alpha/\varepsilon)$ rounds.

¹Under the assumption that Δ is a small enough polynomial in n .

Although Theorem 3.4.1 offers several trade-offs between the number of colors and runtime, its main strength is captured in the following corollary.

Corollary 3.4.2 (Publication IV, Corollary 1.4). *For any constant $\varepsilon > 0$, there is a constant time AMPC algorithm to compute a $((2+\varepsilon)\alpha+1)$ -coloring on any graph with bounded arboricity $\alpha = O(1)$.*

A standard and powerful approach for computing arboricity-dependent colorings is based on H -partitions (see Section 2.2), which define an acyclic edge orientation where each node has bounded out-degree. Our main technical contribution is designing efficient deterministic algorithms to construct such orientations, and showing how to leverage them to obtain colorings in AMPC.

Why using less than $\Delta + 1$ colors is hard? The standard approaches for $(\Delta + 1)$ -coloring are essentially oblivious to the graph's topology. They rely on the fact that *any* partial coloring can always be extended to a valid coloring of the entire graph. However, this property no longer holds when aiming for colorings with only $O(\alpha)$ (or similarly small in terms of α) colors, which is typically far fewer than $\Delta + 1$. In such cases, extending partial colorings may be impossible, giving rise to intricate dependencies that must be carefully resolved.

4. Exponential Speedup Over Locality

In this chapter, we give an overview of the main techniques used to address the challenges encountered in Publication I. Earlier work has established that in the LOCAL model, the deterministic complexity of a problem on trees can only take the following forms:

$$f(n) \in \{\Theta(1), \Theta(\log^* n), \Theta(\log n)\} \cup \{\Theta(n^{1/k}) \mid k \in \mathbb{N}\}.$$

We show that knowing the asymptotic behavior of $f(n)$ is sufficient to design a deterministic MPC algorithm with runtime

$$O(\log(f(n))) \in \{O(1), O(\log \log^* n), O(\log \log n), O(\log n)\}.$$

It is known that for all $f(n) \notin \Theta(\log n)$, the randomized and deterministic LOCAL complexities coincide. In contrast, when $f(n) \in \Theta(\log n)$, the randomized complexity $g(n)$ in the LOCAL model can be either $\Theta(\log n)$ or $\Theta(\log \log n)$. In the latter case, we design an MPC algorithm with randomized complexity $O(\log \log \log n)$.

Our central result can be summarized as follows.

Theorem 3.1.1. *[Publication I, 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.*

We group these runtimes into four distinct regimes:

- the tiny-regime: $f(n) \in \{\Theta(1), \Theta(\log^* n)\}$,
- the low-regime: $f(n) = \Theta(\log n)$,
- the mid-regime: $g(n) = \Theta(\log \log n)$,
- the high-regime: $f(n) = \Theta(n^{1/k}), \forall k \in \mathbb{N}$.

The tiny-regime is the simplest case, which still illustrates the difficulties in staying within the optimal global space bound. The most technically demanding case is the high-regime, where we establish that every LCL problem admits a deterministic $O(\log n)$ -round MPC algorithm.

Graph exponentiation. A central obstacle across all regimes is the restriction to linear global space, which in practice means that each node is limited to using only a constant amount of space on average. This is particularly problematic because nearly all recent advances in MPC algorithms, especially those achieving exponential speedups, depend heavily on the memory-intensive technique of graph exponentiation (see Lemma 2.5.1). Informally, this method allows a node to collect its 2^k -hop neighborhood within k communication rounds. However, performing this in parallel for all nodes incurs a global space overhead of Δ^{2^k} . Almost all effective algorithms need $k = \omega(1)$, which results in a super-constant multiplicative increase in global space. In order to use this technique effectively while using only linear global space, we develop new techniques outlined in the following sections.

4.1 Tiny-Regime $f(n) = \Theta(1)$ and $f(n) = \Theta(\log^* n)$

For $f(n) = \Theta(1)$, the approach is straightforward: any LOCAL algorithm for LCL problems can be directly simulated in the MPC model. For $f(n) = \Theta(\log^* n)$, prior work has shown that such problems in the LOCAL model can be solved by reducing them to computing a distance- k coloring, where k is a constant depending on the specific problem. In a distance- k c -coloring, each vertex is assigned a color from $\{1, \dots, c\}$ such that no two vertices within distance k share the same color. This coloring can be obtained in $O(\log^* n)$ rounds in the LOCAL model. Using graph exponentiation, the same can be achieved in the MPC model in $O(\log \log^* n)$ rounds, but this requires an additional factor of $O(\log^* n)$ in global space.

We show that this space overhead is not necessary. It is known that, in the LOCAL model, coloring general graphs can be reduced to coloring directed pseudoforests, i.e., graphs in which every edge is oriented and each node has at most one outgoing edge. We give a memory-efficient MPC algorithm for coloring such structures using a variant of graph exponentiation that only requires keeping track of a constant number of IDs. As a result, each node uses constant memory, and the overall space requirement remains linear.

4.2 High-Regime $f(n) = \Theta(n^{1/k})$, for all $k \in \mathbb{N}$

For every solvable LCL problem, we design a novel algorithm with runtime $O(\log n)$. The algorithm processes each tree in the input forest independently, so it suffices to consider the case of trees. On a high level, we first root the tree in $O(\log n)$ rounds, and then solve the problem in two phases:

1. Working bottom-up from the leaves, we compute, for a substantial number of nodes v , the set of output labels that could be assigned to v such that any assignment would extend to a valid solution within the subtree rooted at v .
2. Using the information from Phase 1, we construct a valid solution top-down, starting from the root.

While this outline appears simple, there are several technical difficulties. For example, the input tree may have depth $\omega(\log n)$, preventing a purely sequential execution of the above idea. Furthermore, storing the required compatibility information using standard graph exponentiation causes an exponential blowup in memory requirements, violating the linear global space constraint. The key technique for resolving these challenges is a method for interleaving graph exponentiation with graph compression, while also storing compatibility information in a reversible way (so it can be reused in Phase 2). The most difficult aspect of this approach is that exponentiation processes on different subtrees must be merged — sometimes simultaneously, sometimes at different times — into a single coherent process. In order to analyze the resulting complex and highly non-sequential algorithm, we develop a finely tuned potential function.

4.3 Mid-Regime $f(n) = \Theta(\log n)$

Ideally, one would hope to use the algorithm of Chang and Pettie [CP17] as a black box. On a high level, their LOCAL algorithm constructs a rake-and-compress decomposition of depth $O(\log n)$, resembling the classical decomposition introduced by Miller and Reif [MR85]. Compatibility information is then propagated upward layer by layer, labels are fixed at the top, and the assignments are propagated downward.

However, directly applying this algorithm in the MPC model does not work out of the box. First, the compatibility information grows exponentially, which causes congestion in MPC communication. Second, a straightforward application of graph exponentiation would exceed the linear global space limit. We overcome these barriers as follows. We show that the compatibility information can be reduced to a constant size at each iteration, preventing

congestion. We interleave graph exponentiation with memory-releasing steps in a balanced fashion, ensuring that the overall space usage remains linear.

4.4 Low-Regime $g(n) = \Theta(\log \log n)$

For this regime, prior work by [Bal+21] shows a constant-time reduction to instances of size $N = \log n$, and gives a LOCAL algorithm with runtime $\text{poly}(\log N) = \text{poly}(\log \log n)$. With a naïve application of graph exponentiation, this reduction translates into an MPC algorithm with runtime $O(\log \log \log n)$ and an $O(\log n)$ multiplicative factor in global space.

Without using additional global space, however, solving these reduced instances within triple-logarithmic time is more challenging. Our solution is to apply the algorithm developed for the mid-regime to the reduced instances, yielding a memory-efficient algorithm with runtime $O(\log \log \log n)$. To the best of our knowledge, no prior work is capable of efficiently handling such small instances under strict global space constraints.

5. The Balanced Exponentiation Technique

This chapter introduces the *balanced exponentiation* technique, explaining its origins, motivation, and applications. The method was first developed for solving connectivity in Publication II, was later adapted for coloring forests with arbitrary degree in Publication III, and was the inspiration for the coin-dropping game used in arboricity-based colorings (see Chapter 6) in Publication IV.

5.1 Forest Connectivity

Our goal is to design a deterministic algorithm for connectivity on forests with runtime $O(\log D)$, where D denotes the maximum diameter of any connected component. Prior algorithms for general graphs achieve runtimes with an unavoidable dependence on n , the number of nodes, which we are determined to eliminate.

A natural approach for solving connectivity on forests is to root each tree and then identify it by the ID of its root. Rooting, however, already illustrates the inherent difficulty of the problem. A standard rooting method is to iteratively perform rake operations: at each step, every leaf chooses its unique neighbor as their parent. This procedure roots a tree in $O(D)$ parallel rounds, and in a forest, each tree can be rooted independently. If we disregard space constraints, this process could be accelerated to $O(\log D)$ rounds using graph exponentiation: in $O(\log D)$ rounds, every node could collect its D -hop neighborhood, effectively the entire graph. However, in the low-space MPC model, where local and global space are restricted to $O(n^\delta)$ and $O(n + m)$ respectively, this is infeasible. Graph exponentiation requires every node to simultaneously expand its neighborhood, which, under the strict space limit, means that each node can only afford to gather its constant-radius neighborhood. On graphs with arbitrary degrees, even this is not possible, since the degree of a single node could be $\omega(n^\delta)$. As a result, only a constant number of rake steps can be simulated in an MPC round, preventing the desired speedup.

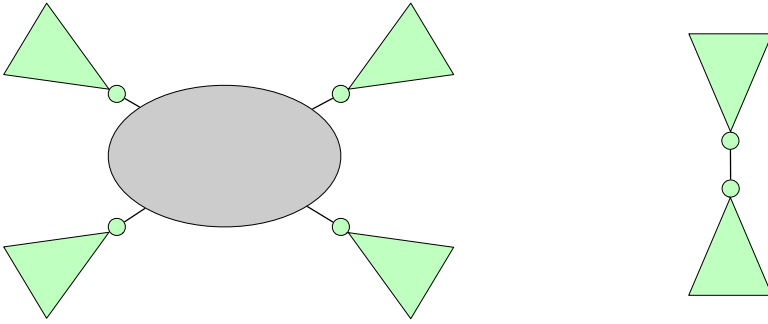


Figure 5.1. (Publication II, Figure 1) Light subtrees highlighted in green.

One might hope to exploit the fact that the ratio of the “original” total space to the number of active nodes increases as we rake the graph, an observation made in earlier work [And+18; Beh+19b; CDP21a]. If each round shrinks the graph by a constant factor, then the average available space per node also increases, potentially enabling nodes to gather progressively larger neighborhoods. Yet, even with such a guarantee, the runtime would remain dependent on n , since progress would still be measured using the size of the graph. This suggests that a fundamentally different approach is needed, rather than one that gradually sparsifies the graph.

The algorithm. The core idea of our algorithm in Publication II is to iteratively compress parts of the graph while preserving connectivity, ensuring that the maximum identifier within each compressed part is retained in the reduced graph. This process is repeated until only a single node remains, which holds the maximum identifier (ID) of the whole connected component. We then reverse the process by iteratively decompressing the graph and propagating ID back to all nodes. Eventually, the original graph is restored, with every node aware of ID.

The two types of structures we compress are light subtrees and paths. The definition of a light subtree (highlighted in green in Figure 5.1) is a bit convoluted, because we are in an unrooted setting. Intuitively, a subtree T_v at a node v is the graph connected to v if we ignore exactly one edge adjacent to v . This subtree is light if $|T_v| \leq n^{\delta/8}$. We refer to v as the root of T_v , and we refer to the nodes in T_v as the descendants of v , even though we do not have any global rooting in place. To make the definition more formal, one needs to include the ignored edge in the notation, which is precisely what we do in the technical parts of the paper.

The algorithm proceeds in $\ell = O(1)$ phases, followed by the same number of reversal phases. In each phase, all light subtrees are compressed into single nodes, and paths are compressed into single edges, yielding a sequence of graphs G_0, G_1, \dots, G_ℓ . During the reversal phases, the compression steps are undone in reverse order, spreading ID across the graph.

Balanced exponentiation. The main technical challenge lies in compressing light subtrees, or in other words, ensuring that every root of a light subtree can gather the subtree. Tackling this challenge led to the development of the balanced exponentiation technique, which ensures that every root of a light subtree gathers a part of its neighborhood that is balanced in the following sense. If a node u is the root of a light subtree T_u of size $\gamma \leq n^{\delta/8}$, we ensure that u gathers T_u , as well as only $O(\gamma)$ nodes in the direction of the edge that was ignored when defining T_u . Thus, the memory requirement at u is $O(\gamma)$.

A crucial step in our analysis is showing that even if every node gathered its γ descendants and some other $O(\gamma)$ additional nodes, the total space increases by only a multiplicative D factor. The following lemma is central to this argument.

Lemma 5.1.1 (Publication II, Lemma 4.5). *Consider an n -node tree T with diameter D that is rooted at node r . Let $T(v, r)$ denote the subtree rooted at v (including v) when T is rooted at r . It holds that $\sum_{v \in V} |T(v, r)| \leq (D+1) \cdot n$.*

Proof. Consider the unique path P_{rv} from the root r to a node v . Observe that v appears only in the subtrees of the nodes in P_{rv} . Since $|P_{rv}| \leq D+1$, node v is overcounted at most D times, and $\sum_{v \in V} |T(v, r)| \leq (D+1) \cdot n$. \square

By the lemma above, if every node u would gather $\min\{n^\delta, O(|T_u|)\}$ information, where T_u is a light subtree rooted at u , the global space would be bounded by $O(m \cdot D)$. With some simple preprocessing, we can eliminate the extra factor of D , ensuring that the algorithm operates within linear global space.

The major crux of this approach is the fact that the input graph is not rooted, so nodes do not know in advance who their descendants are and whether or not they are roots of light subtrees. Nevertheless, we show that, without asymptotic slowdown, each node can deterministically identify its “worst-case” parent choice and gather information accordingly. Using this strategy, nodes with light subtrees can identify them in $O(\log D)$ time, independent of n . Since the number of phases is constant, the overall runtime is $O(\log D)$.

Another issue is that executing balanced exponentiation requires knowing the diameter D , which is not given as input. This issue is resolved by running $O(\log \log n)$ parallel instances of the algorithm with exponentially increasing diameter guesses, one of which will be sufficiently close to the correct value.

5.2 Coloring, MIS and Maximal Matching

In Publication III, we present a unified algorithmic framework that solves 3-coloring, maximal independent set (MIS), and maximal matching in

$O(\log \log n)$ rounds. The core technical ingredient enabling this unification is an efficient procedure for computing H -partitions (see Section 2.2).

Theorem 5.2.1 (Publication III, Theorem 2). *There is a deterministic $O(\log \log n)$ -round low-space MPC algorithm that computes a strict H -decomposition with $O(\log n)$ layers on forests in $O(n)$ global space.*

On forests, an H -partition entails partitioning the vertex set into layers such that each node has at most two neighbors in the same or higher layers. It is known that every forest admits an H -partition with $O(\log n)$ layers. In the LOCAL model, such a decomposition immediately yields an $O(\log n)$ -round algorithm for 3-coloring. Essentially, one can process the layers in reverse order sequentially, coloring all nodes in a layer while avoiding conflicts with already colored neighbors in higher layers.

The novelty of our work does not lie in the use of H -partitions to obtain a 3-coloring — this approach is already well known and even taught in graduate courses — but rather in how we compute the decomposition efficiently. We extend the balanced exponentiation technique, originally developed for our connectivity algorithm in Publication II, and adapt it to this setting. Our method steers each node to explore portions of the graph (largely in an uncoordinated manner) so that it can locally construct a *partial* H -partition. These partial partitions are then combined in a non-trivial way into a consistent global decomposition. To the best of our knowledge, this is the first MPC algorithm for H -partitions that follows such an approach.

Furthermore, we formulate balanced exponentiation as a standalone technique, making it available as a general tool that can be applied to a broader range of problems.

Balanced exponentiation. Our balanced exponentiation procedure enables nodes to explore meaningful, by some metric, parts of the graph, while keeping space usage minimal. To describe this method, we first need to introduce the notion of a *subtree*: a connected subgraph of a tree whose removal does not disconnect the graph. A node is considered *important* if it lies within a subtree of size at most $n^{\delta/8}$. Informally, balanced exponentiation can be summarized as follows (the precise formulation is given in Lemma 5.2.2).

Publication III, page 23:5: *Let $0 < k \leq n^{\delta/8}$ be a parameter. There is a deterministic low-space MPC algorithm that, given an n -node forest F , uses $O(\log k)$ rounds in which every important node $v \in F$ discovers its k -hop neighborhood in every direction of the graph, except for at most one.*

For a node $v \in F$, each neighbor $x \in N(v)$ defines a direction relative to v . The portion of the forest that v can explore in direction x is simply the subgraph of F that is connected to v via x . This subgraph is unique because F is a forest.

The above technique may be of independent interest and could find applications in other graph problems. We obtain it by generalizing the exponentiation technique introduced in Publication II. In that earlier work, the parameter k was fixed to be the maximum diameter of the graph, which enabled an $O(\log D)$ -round algorithm for solving connectivity on forests. The same technique has since been adapted for certain dynamic programming tasks on tree-structured data by [Gup+23] (a publication by the author not included in this thesis), also achieving runtimes logarithmic in the diameter of the graph.

The key advantage of our generalized result is its flexibility: parameter k can be tuned so that both runtime and space remain small whenever nodes require only a limited “view” of the graph. This property plays a central role in our algorithm for computing H -partitions. For completeness, we include the formal statement of this generalized balanced exponentiation procedure below.

Lemma 5.2.2 (Full version of Publication III [Gru+23], Lemma 8.4). *Let $0 < k \leq n^{\delta/8}$ be a parameter that may or may not be constant. Given an n -node forest F , there is a deterministic $O(\log k)$ time low-space MPC algorithm after which the following holds. For every important node $v \in V(F)$, there is a node $z \in N(v)$ and a machine that for all $x \in N(v) \setminus z$ holds $N^k(v) \cap F_{v \rightarrow x}$ in memory. The algorithm requires $O(n \cdot \text{poly}(k))$ total space.*

Our algorithm in a nutshell. As outlined previously, the central technical component of our approach is computing an H -partition of the input forest F . Concretely, we seek a partition

$$V(F) = V_1 \sqcup V_2 \sqcup \dots \sqcup V_L$$

of $L = O(\log n)$ layers such that each node in V_i has at most two neighbors in $\bigcup_{j \geq i} V_j$. A straightforward way to obtain such a decomposition is via an iterative peeling process: in each step, remove all nodes of degree at most 2 and define V_i as the set of nodes removed in the i -th round. A simple counting argument shows that at least half of the remaining nodes are removed in every iteration, which yields a decomposition of $O(\log n)$ layers. Moreover, one can determine the layer of a node solely based on its $O(\log n)$ -hop neighborhood. Hence, if we could store the $O(\log n)$ -neighborhood of each node, the layer assignment could be decided locally without further communication.

However, the space constraints of the low-space MPC model prevent us from gathering the full $O(\log n)$ -neighborhood for every node. To overcome this, we employ the balanced graph exponentiation technique for forests described earlier. When setting parameter k to $O(\log n)$, balanced exponentiation provides a slightly weaker guarantee than collecting the complete $O(\log n)$ neighborhood of every node:

1. It only applies to nodes within small subtrees, specifically of size $\leq n^{\delta/8}$.
2. For such nodes, it gathers the $O(\log n)$ -neighborhood in all directions except one.

We first address the limitation of handling only small subtrees. Suppose we repeatedly remove, from F , all nodes that either belong to a subtree of size at most x or have degree 2 outside said subtrees. Within $O(\log_x n)$ iterations, the entire forest is removed using a simple argument presented in Publication II. Consequently, if we assign nodes in subtrees of size at most $n^{\delta/8}$ together with nodes of degree 2 to one of $O(\log n)$ layers in each round, then after $O(1/\delta)$ such rounds every node has been assigned to some layer. This results in a decomposition of $O((1/\delta) \log n)$ layers. Hence, it suffices to restrict our attention to nodes contained in subtrees of size at most $n^{\delta/8}$ (nodes of degree 2 are trivial to handle).

Addressing the second challenge is more involved. In general, no machine stores the full $O(\log n)$ neighborhood of any node, making it impossible to directly replicate the peeling process. Instead, each node v runs a *conservative* peeling algorithm based only on the portion of the graph stored locally. In this conservative peeling algorithm, not all nodes are assigned a layer — some remain unlayered and must be handled in subsequent iterations. Consequently, each iteration yields only a *partial* H -partition, consisting of both layered and unlayered nodes. An additional complication is that these partial decompositions may not be globally consistent because a node might get assigned to different layers by different nodes. This discrepancy arises because balanced exponentiation can give different nodes very different perspectives on the graph. Fortunately, this inconsistency does not pose a real obstacle. We prove a key structural property of (partial) H -partitions: given multiple (partial) decompositions, one can construct another valid (partial) decomposition by assigning each node to the smallest layer it is assigned to across all decompositions. Using this property, we can merge the various locally computed partial H -partitions into a single globally consistent partial H -partition, ensuring that every node in a sufficiently small subtree is assigned to one of the $O(\log n)$ layers.

6. Coin Dropping Game via Adaptive Queries

Balanced exponentiation makes an appearance in the AMPC model in the form of the so-called *coin-dropping*. This technique is used to develop a deterministic *local computation algorithm* (LCA) [Rub+11; Alo+12] that enables us to compute $\text{poly}(\alpha)$ - and $O(\alpha)$ -colorings in constant, or near-constant, time on graphs with arboricity α in the AMPC model. On a high level, the approach is very similar to the tree-coloring algorithm in the MPC model from Publication III, in that the core task reduces to computing an H -partition of the graph. The key difference, however, is that the input graphs are no longer trees. This alone renders the generalized balanced exponentiation technique from Publication III useless because it fundamentally relies on a property unique to trees: the subgraph connected to a node v via its neighbor $u \in N(v)$ is disconnected from the subgraph connected to a node v via some other neighbor $w \in N(v), w \neq u$. However, we show that we can bypass this limitation in the AMPC mode by designing an LCA that computes (partial) H -partitions efficiently, even on graphs with cycles.

6.1 Sublinear LCA

We develop a deterministic LCA that produces an acyclic orientation with small out-degree $\beta = \Omega(\alpha)$ for the vast majority of nodes on a graph with arboricity α , using a sublinear number of queries. In particular, we construct a β -*partition*, i.e., a layering of the vertex set where each node has at most β neighbors in its own or higher layers, while minimizing the total number of layers. Such a partition directly induces the desired acyclic orientation: inter-layer edges are oriented from lower to higher layers, and intra-layer edges are oriented from lower to higher ID (or, if a layer is colored, from lower to higher color). Previous work suggests that achieving this for all nodes with sublinear query complexity is highly unlikely. Instead, our algorithm focuses on computing layers for *most* of the vertices, thereby achieving an LCA with sublinear-query complexity that produces a near-complete β -partition (see Figure 6.1).

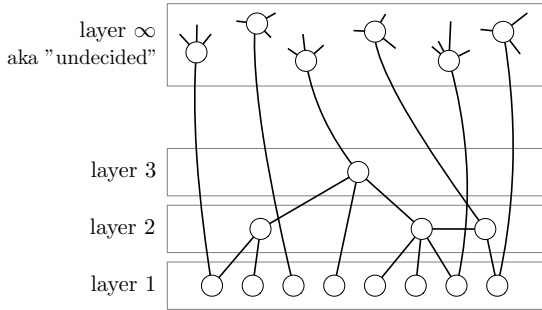
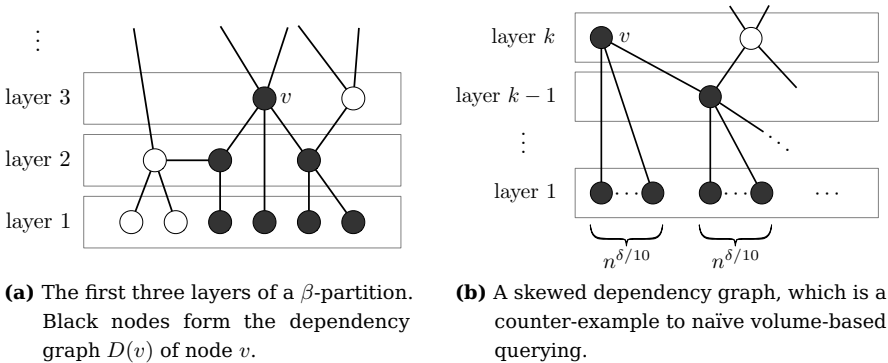


Figure 6.1. (Publication IV, Figure 1) An illustration of a β -partition where most nodes are assigned a layer.



(a) The first three layers of a β -partition. Black nodes form the dependency graph $D(v)$ of node v .

(b) A skewed dependency graph, which is a counter-example to naive volume-based querying.

Figure 6.2. (Publication IV, Figures 2 & 3) Examples of different dependency graphs.

The informal guarantee we achieve is summarized below.

Lemma 6.1.1 (Publication IV, Lemma 1.1). *For any constant $\delta > 0$ there is deterministic LCA that uses at most $O(n^\delta)$ queries per node on a graph G with arboricity α and assigns each node a layer from $\mathbb{N} \cup \{\infty\}$ such that the following holds:*

- ▷ There exists a subset $S \subseteq V$ containing at least a $1 - 1/n^{O(\delta)}$ fraction of vertices such that the layering of S forms a β -partition of $G[S]$ with $O(\log_\beta n)$ layers. ◁

6.1.1 Where Should One Query?

A fundamental question to answer is which part of the graph a node needs to query to determine its layer? The answer is straightforward to answer if we assume that the target β -partition ℓ is already known, because this partition naturally defines the *dependency graph* of each node v . The dependency graph $D(v)$ is the set of all nodes u that can be reached from v along a path of strictly decreasing layers (see Figure 6.2a). Every node in $D(v)$ has at most β neighbors outside of $D(v)$. The dependency graph certifies the layer

of v , in the sense that the layer of v in ℓ can be determined solely based on the nodes in $D(v)$ and their degrees (see Figure 6.2a for an example). Once a node has discovered its dependency graph through queries, it can correctly compute its layer. While this imagined partition ℓ is not available to guide the algorithm, simply because it is not known beforehand, it is precisely what we aim to compute, and it provides valuable intuition and a useful framework for analysis.

Observe that if a node's dependency graph is too large, then learning it with a sublinear number of queries is impossible. Thus, our strategy relies on the fact that many nodes have *small* dependency graphs (of size $\leq n^\delta$). For such nodes, we hope to explore their dependency graphs and determine their layers.

Everything boils down to solving what appears to be an impossible task: guiding the exploration of a node v so that most queries are concentrated within its dependency graph, despite having no prior knowledge of where this region lies. Deterministic exploration is particularly difficult, as the local neighborhood of v can look identical in every direction. However, only a fraction of these directions actually lead to $D(v)$, and there is no obvious way to identify them in advance. Let us briefly explain why the classic search paradigms fail in this setting.

- *Depth-First Search (DFS)*: A node v may have up to β neighbors outside its dependency graph. If the algorithm initiates DFS from one of these, all queries may end up being wasted on irrelevant parts of the graph.
- *Breadth-First Search (BFS)*: To avoid the pitfalls of DFS, one might try BFS in hopes of balancing exploration. At first glance, this seems promising: after the first round of queries, at least a $1/(\beta+1)$ fraction of queried nodes belong to $D(v)$. However, the gains of this approach quickly deteriorate. Suppose v has a neighbor u outside $D(v)$ with very high degree, say $\deg(u) = n^\delta$. Then, most second-level queries are spent on exploring u 's neighbors, which most likely lie outside $D(v)$ and provide no useful information.

Clearly, a more refined exploration strategy is needed. Our solution is to explore the graph in a *volume-balanced* manner, rather than in a *distance-balanced* way like BFS or DFS. This method draws direct inspiration from the balanced exponentiation technique of Publication II and Publication III.

Volume-based querying. To guide the queries, consider the following coin-dropping *game*. Suppose node v is given n^δ coins. A naïve strategy (which, as we will see, still fails) is for v to distribute its coins evenly among its neighbors, so that each neighbor receives the same share. Each neighbor then repeats this process recursively, forwarding the coins it received to its own neighbors, and so on, until the coins can no longer be divided.

Conceptually, this procedure acts like a “volume-based” version of BFS. The act of passing a coin corresponds to querying a node, and any node that receives at least one coin is thereby “discovered” by v . Ideally, this process would succeed in forwarding a coin to every node in $D(v)$. Unfortunately, this approach breaks down in the presence of skewed dependency graphs, an example of which is depicted in Figure 6.2b: a long “descending” path where each node has $n^{\delta/10}$ neighbors in the lowest layer. In this case, the initial n^δ coins given out by node v are exhausted already after ten rounds of distribution, discovering only around $10 \cdot n^{\delta/10}$ nodes of the dependency graph — far fewer than desired. The exploration stalls because most of the coins get “stuck” at the lowest layer very fast and cannot be utilized where they are truly needed. Repeating this coin-dropping game yields no improvement, since the coins will always get stuck in the same way.

6.1.2 Our Solution

Our approach builds on the coin-dropping game described earlier, but instead of naively distributing coins, we design more refined *forwarding rules* that dictate how coins are passed along. These rules are changed dynamically as more of the graph is revealed by the queries of node v , allowing us to steer exploration more effectively and fully exploit the adaptivity of the queries.

Let us think about how to guarantee that a coin-dropping game will always reveal new parts of the dependency graph. The previous approach failed because nodes distributed coins to all of their neighbors blindly. What if nodes only forwarded coins to a carefully chosen subset of their neighbors? Recall that every node in $D(v)$ has at most β neighbors outside of $D(v)$. Consequently, if every node $u \in D(v)$ forwards coins to $\beta + 1$ of its neighbors (chosen by some rule), then at least one of those neighbors must belong to $D(v)$. Hence, at least a $1/(\beta + 1)$ fraction of u 's coins is guaranteed to stay inside the dependency graph. After k forwarding rounds, at least $x/(\beta + 1)^k$ of the original x coins distributed by v remain within $D(v)$, which sounds promising. Does this approach work? Is it possible to distribute coins long enough to learn new parts of the dependency graph? Crucially, does performing multiple of these coin-dropping games lead to v learning the whole dependency graph?

Everything boils down to how we choose, for every node $u \in D(v)$, the $\beta + 1$ neighbors that u forwards coins to. It seems that any fixed choice is insufficient: after some number of coin-dropping games, coins will begin accumulating within already discovered regions of $D(v)$. In such cases, the process does not lead to v to querying new, undiscovered parts of the dependency graph. Hence, the rules for selecting *which* neighbors receive coins must be defined adaptively.

Our coin forwarding rules. We aim to define forwarding rules such that a sufficient fraction of coins flows into unexplored regions of $D(v)$ while minimizing leakage outside of the dependency graph or stagnation in areas already known. To achieve this, node v maintains an approximation of the β -partition (called the *induced* β -partition) of the subgraph it has discovered so far (denoted S_v). In this approximation, every node in S_v is assigned a tentative layer (or ∞ if no layer is yet assigned). Every time a coin-dropping game is initiated by node v , a new approximation is computed and every node $u \in D(v)$ forwards coins to its $\beta + 1$ neighbors with the highest tentative layer, with ties broken arbitrarily. These adaptive rules guarantee that enough coins flow toward undiscovered parts of $D(v)$ in every coin-dropping game.

Interestingly, the process cannot simply stop once node v is assigned a tentative layer different from ∞ . Even if v has already uncovered most of $D(v)$, its tentative layer might still be an overestimate: for example, $n^{\delta/10}$, while its true layer (had the entire dependency graph been explored) could be much smaller. Such inaccuracies are unacceptable, since we aim for $O(\log n)$ layers overall. The success of our method, therefore, hinges on how many times we reinitiate the coin-dropping game and how long individual games are allowed to run.

6.2 Using Our LCA Recursively

Our LCA computes the β -partition layers for a large majority of the nodes, but not for all of them. To obtain a complete β -partition, one would hope to simply recurse on the subgraph that remains after removing all nodes assigned a finite layer. Unfortunately, this step turns out to require more care. From Lemma 6.1.1, we only know that there exists a subset $S \subseteq V$ of size at least $(1 - 1/n^{O(\delta)}) \cdot |V|$ such that the layering of S forms a valid β -partition of $G[S]$. Utilizing this property is difficult because we do not know which nodes belong to this “good” set S , since there may be nodes outside of S that also compute a finite layer. Including such nodes in the partition risks creating a *globally inconsistent* layering. For example, consider nodes $w, u \notin S$. It can happen (because the nodes are not in S) that w assigns itself to layer 10 because it has computed u to be in layer 9, while u independently assigns itself to layer 11. Which result to accept, and could this create an unwanted domino effect? By choosing poorly, nodes could end up with more than β neighbors in higher layers, violating the β -partition property. To overcome this issue, we strengthen our LCA beyond the informal guarantee of Lemma 6.1.1. In addition to outputting its own layer, each node $u \in V$ also provides a *proof* in the form of a β -partition¹ ℓ_u . Intuitively, this proof

¹We actually require this partition to be *partial*, analogous to partial H -partitions introduced in Chapter 5.

acts as a local certificate that justifies why u can be safely assigned to layer $\ell_u(u)$: it describes how to layer parts of the graph so that u 's assignment is valid. With access to these proofs, we no longer need to decide between different layer assignments or identify which finite-layer nodes belong to S . Instead, we can merge all discovered β -partitions into a single partition by defining

$$\ell(v) = \min_{u \in V} \ell_u(v)$$

to be the layer of a node v . This construction yields a valid and globally consistent β -partition of the subgraph induced by all finite-layer nodes. This consistency guarantee is sufficient for applying our LCA recursively in the AMPC model.

References

- [AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. doi: 10.1017/CB09780511804090 (cit. on p. 28).
- [AH76] Kenneth Appel and Wolfgang Haken. “Every planar map is four colorable”. In *Bulletin of the American Mathematical Society* 82 (1976), pp. 711–712. doi: 10.1090/S0002-9904-1976-14122-5 (cit. on p. 11).
- [Alo+12] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. “Space-Efficient Local Computation Algorithms”. In *the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2012, pp. 1132–1139. doi: 10.1137/1.9781611973099.89 (cit. on p. 49).
- [And+18] Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. “Parallel Graph Connectivity in Log Diameter Rounds”. In *the Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 2018, pp. 674–685. doi: 10.1109/FOCS.2018.00070 (cit. on pp. 31, 33, 44).
- [ASW19] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. “Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs”. In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 2019, pp. 461–470. doi: 10.1145/3293611.3331596 (cit. on p. 33).
- [Bal+18] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. “Almost Global Problems in the LOCAL Model”. In *the International Symposium on Distributed Computing (DISC)*. 2018, 9:1–9:16. doi: 10.4230/LIPIcs.DISC.2018.9 (cit. on p. 32).
- [Bal+19] Alkida Balliu, Juho Hirvonen, Dennis Olivetti, and Jukka Suomela. “Hardness of Minimal Symmetry Breaking in Distributed Computing”. In *the Proceedings of the ACM Symposium on Prin-*

- ciples of Distributed Computing (PODC)*. 2019, pp. 369–378. doi: 10.1145/3293611.3331605 (cit. on p. 32).
- [Bal+20a] Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. “Classification of Distributed Binary Labeling Problems”. In *the International Symposium on Distributed Computing (DISC)*. 2020, 17:1–17:17. doi: 10.1145/3382734.3405703 (cit. on p. 32).
- [Bal+20b] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. “How much does randomness help with locally checkable problems?” In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 2020, pp. 299–308. doi: 10.1145/3382734.3405715 (cit. on p. 25).
- [Bal+21] Alkida Balliu, Keren Censor-Hillel, Yannic Maus, Dennis Olivetti, and Jukka Suomela. “Locally Checkable Labelings with Small Messages”. In *the International Symposium on Distributed Computing (DISC)*. 2021, 8:1–8:18. doi: 10.4230/LIPIcs.DISC.2021.8 (cit. on pp. 23, 42).
- [Bal+25] Alkida Balliu, Mohsen Ghaffari, Fabian Kuhn, Augusto Modanese, Dennis Olivetti, Mikaël Rabie, Jukka Suomela, and Jara Uitto. “Shared randomness helps with local distributed problems”. In *the Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*. 2025, 16:1–16:18. doi: 10.4230/LIPIcs.ICALP.2025.16 (cit. on p. 25).
- [Bar15] Leonid Barenboim. “Deterministic $(\Delta + 1)$ -Coloring in Sub-linear (in Δ) Time in Static, Dynamic and Faulty Networks”. In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 2015, pp. 345–354. doi: 10.1145/2767386.2767410 (cit. on p. 36).
- [BCG20] Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. “Graph Coloring via Degeneracy in Streaming and Other Space-Conscious Models”. In *the Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*. 2020, 11:1–11:21. doi: 10.4230/LIPIcs.ICALP.2020.11 (cit. on p. 36).
- [BE08] Leonid Barenboim and Michael Elkin. “Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition”. In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 2008, pp. 25–34. doi: 10.1145/1400751.1400757 (cit. on pp. 23, 36).
- [BE10] Leonid Barenboim and Michael Elkin. “Deterministic Distributed Vertex Coloring in Polylogarithmic Time”. In *the Proceedings of the ACM Symposium on Principles of Distributed Comput-*

- ing (PODC). 2010, pp. 410–419. doi: 10.1145/1835698.1835797 (cit. on p. 36).
- [Beh+19a] 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 ACM Symposium on Principles of Distributed Computing (PODC)*. 2019, pp. 481–490. doi: 10.1145/3293611.3331609 (cit. on pp. 31, 35).
- [Beh+19b] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Łącki, and Vahab Mirrokni. “Near-Optimal Massively Parallel Graph Connectivity”. In *the Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 2019, pp. 1615–1636. doi: 10.1109/FOCS.2019.00095 (cit. on pp. 33, 34, 44).
- [Beh+19c] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Łącki, Vahab Mirrokni, and Warren Schudy. “Massively Parallel Computation via Remote Memory Access”. In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2019, pp. 59–68. doi: 10.1145/3323165.3323208 (cit. on pp. 17, 28, 29).
- [BFU19] 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 Colloquium on Structural Information and Communication Complexity (SIROCCO)*. 2019, pp. 124–138. doi: 10.1007/978-3-030-24922-9_9 (cit. on p. 35).
- [Bha+24] Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. “Arboricity-Dependent Algorithms for Edge Coloring”. In *the Proceedings of the Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*. 2024, 12:1–12:15. doi: 10.4230/LIPIcs.SWAT.2024.12 (cit. on p. 36).
- [BKS17] Paul Beame, Paraschos Koutris, and Dan Suciu. “Communication Steps for Parallel Query Processing”. In *the Journal of the ACM* 64.6 (2017). doi: 10.1145/3125644 (cit. on pp. 17, 26).
- [Bro41] Rowland L. Brooks. “On colouring the nodes of a network”. In *the Mathematical Proceedings of the Cambridge Philosophical Society* 37 (1941), pp. 194–197. doi: 10.1017/S030500410002168X (cit. on p. 12).
- [CC22] Sam Coy and Artur Czumaj. “Deterministic Massively Parallel Connectivity”. In *the Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*. 2022, pp. 162–175. doi: 10.1145/3519935.3520055 (cit. on pp. 33, 34).

- [CDP20] Artur Czumaj, Peter Davies, and Merav Parter. “Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space”. In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2020, pp. 175–185. doi: 10.1145/3350755.3400282 (cit. on p. 31).
- [CDP21a] Artur Czumaj, Peter Davies, and Merav Parter. “Component Stability in Low-Space Massively Parallel Computation”. In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 2021, pp. 481–491. doi: 10.1145/3465084.3467903 (cit. on pp. 28, 31, 33, 44).
- [CDP21b] Artur Czumaj, Peter Davies, and Merav Parter. “Improved Deterministic $(\Delta + 1)$ Coloring in Low-Space MPC”. In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 2021, pp. 469–479. doi: 10.1145/3465084.3467937 (cit. on p. 31).
- [Cha+19a] 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 ACM Symposium on Principles of Distributed Computing (PODC)*. 2019, pp. 471–480. doi: 10.1145/3293611.3331607 (cit. on p. 36).
- [Cha+19b] 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”. In *ACM Trans. Algorithms* (2019), pp. 1–51. doi: 10.1145/3365004 (cit. on p. 32).
- [Cha20] Yi-Jun Chang. “The Complexity Landscape of Distributed Locally Checkable Problems on Trees”. In *the International Symposium on Distributed Computing (DISC)*. 2020, 18:1–18:17. doi: 10.4230/LIPIcs.DISC.2020.18 (cit. on p. 32).
- [CNR23] Aleksander B. G. Christiansen, Krzysztof Nowicki, and Eva Rotenberg. “Improved Dynamic Colouring of Sparse Graphs”. In *the Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*. 2023, pp. 1201–1214. doi: 10.1145/3564246.3585111 (cit. on p. 36).
- [CP17] Yi-Jun Chang and Seth Pettie. “A Time Hierarchy Theorem for the LOCAL Model”. In *the Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 2017, pp. 156–167. doi: 10.1109/FOCS.2017.23 (cit. on pp. 32, 41).
- [CR22] Aleksander B. G. Christiansen and Eva Rotenberg. “Fully-Dynamic $\alpha + 2$ Arboricity Decompositions and Implicit Colouring”. In *the Proceedings of the International Colloquium on*

- Automata, Languages, and Programming (ICALP)*. 2022, 42:1-42:20. doi: 10.4230/LIPIcs.ICALP.2022.42 (cit. on p. 36).
- [DG08] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In *Communications of the ACM* (2008), pp. 107–113. doi: 10.1145/1327452.1327492 (cit. on pp. 18, 26).
- [FGG23] Manuela Fischer, Jeff Giliberti, and Christoph Grunau. “Deterministic Massively Parallel Symmetry Breaking for Sparse Graphs”. In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2023, pp. 89–100. doi: 10.1145/3558481.3591081 (cit. on pp. 35, 36).
- [FHK16] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. “Local Conflict Coloring”. In *the Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 2016, pp. 625–634. doi: 10.1109/FOCS.2016.73 (cit. on p. 36).
- [GG25] Mohsen Ghaffari and Christoph Grunau. “Density-Dependent Graph Orientation and Coloring in Scalable MPC”. In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 2025, pp. 349–359. doi: 10.1145/3732772.3733501 (cit. on p. 36).
- [GGJ20] Mohsen Ghaffari, Christoph Grunau, and Ce Jin. “Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond”. In *the International Symposium on Distributed Computing (DISC)*. 2020, 34:1–34:18. doi: 10.4230/LIPIcs.DISC.2020.34 (cit. on pp. 31, 35).
- [GKU19] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. “Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds”. In *the Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 2019, pp. 1650–1663. doi: 10.1109/FOCS.2019.00097 (cit. on pp. 28, 33).
- [GRB22] Christoph Grunau, Václav Rozhoň, and Sebastian Brandt. “The Landscape of Distributed Complexities on Trees and Beyond”. In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 2022, pp. 37–47. doi: 10.1145/3519270.3538452 (cit. on pp. 23, 32).
- [Gru+23] Christoph Grunau, Rustam Latypov, Yannic Maus, Shreyas Pai, and Jara Uitto. “Conditionally Optimal Parallel Coloring of Forests”. In *arXiv*. 2023. doi: 10.48550/arXiv.2308.00355 (cit. on p. 47).

- [GS19] Mohsen Ghaffari and Ali Sayyadi. “Distributed Arboricity-Dependent Graph Coloring via All-to-All Communication”. In *the Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*. 2019, 142:1–142:14. doi: 10.4230/LIPIcs.ICALP.2019.142 (cit. on p. 36).
- [GU19] Mohsen Ghaffari and Jara Uitto. “Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation”. In *the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2019, pp. 1636–1653. doi: 10.1137/1.9781611975482.99 (cit. on p. 31).
- [Gup+23] Chetan Gupta, Rustam Latypov, Yannic Maus, Shreyas Pai, Simo Särkkä, Jan Studený, Jukka Suomela, Jara Uitto, and Hossein Vahidi. “Fast Dynamic Programming in Trees in the MPC Model”. In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2023, pp. 443–453. doi: 10.1145/3558481.3591098 (cit. on p. 47).
- [Hea90] Percy J. Heawood. “Map-Colour Theorem”. In *Quarterly Journal of Pure and Applied Mathematics* 24 (1890), pp. 332–338. url: <https://babel.hathitrust.org/cgi/pt?id=inu.30000050138159> (cit. on p. 11).
- [HNW20] Monika Henzinger, Stefan Neumann, and Andreas Wiese. “Explicit and Implicit Dynamic Coloring of Graphs with Bounded Arboricity”. In *arXiv*. 2020. doi: 10.48550/arXiv.2002.10142 (cit. on p. 36).
- [Isa+07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. “Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks”. In *ACM SIGOPS Operating Systems Review* (2007), pp. 59–72. doi: 10.1145/1272996.1273005 (cit. on pp. 18, 26).
- [Kar72] Richard M. Karp. “Reducibility among Combinatorial Problems”. In *the Proceedings of the Symposium on the Complexity of Computer Computations*. 1972, pp. 85–103. doi: 10.1007/978-1-4684-2001-2_9 (cit. on p. 12).
- [KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. “A Model of Computation for MapReduce”. In *the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2010, pp. 938–948. doi: 10.1137/1.9781611973075.76 (cit. on pp. 17, 26).
- [Lin92] Nathan Linial. “Locality in Distributed Graph Algorithms”. In *SIAM Journal on Computing* (1992), pp. 193–201. doi: 10.1137/0221015 (cit. on p. 25).

- [Lov75] László Lovász. “Three short proofs in graph theory”. In *the Journal of Combinatorial Theory, Series B* 19.3 (1975), pp. 269–271. doi: 10.1016/0095-8956(75)90089-1 (cit. on p. 12).
- [LU21] Rustam Latypov and Jara Uitto. “Coloring Trees in Massively Parallel Computation”. In *arXiv*. 2021. doi: 10.48550/arXiv.2105.13980 (cit. on p. 35).
- [LW10] Christoph Lenzen and Roger Wattenhofer. “Brief Announcement: Exponential Speed-up of Local Algorithms Using Non-Local Communication”. In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 2010, pp. 295–296. doi: 10.1145/1835698.1835772 (cit. on p. 27).
- [Mau21] Yannic Maus. “Distributed Graph Coloring Made Easy”. In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2021, pp. 362–372. doi: 10.1145/3409964.3461804 (cit. on p. 36).
- [MR85] Gary L. Miller and John H. Reif. “Parallel tree contraction and its application”. In *the Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 1985, pp. 478–489. doi: 10.1109/SFCS.1985.43 (cit. on pp. 23, 41).
- [MT20] Yannic Maus and Tigran Tonoyan. “Local Conflict Coloring Revisited: Linial for Lists”. In *the International Symposium on Distributed Computing (DISC)*. 2020, 16:1–16:18. doi: 10.4230/LIPICS.DISC.2020.16 (cit. on p. 36).
- [Nas64] Crispin Nash-Williams. “Decomposition of Finite Graphs Into Forests”. In *Journal of the London Mathematical Society* s1–39 (1964), p. 12. doi: 10.1112/jlms/s1-39.1.12 (cit. on p. 23).
- [NS93] Moni Naor and Larry J. Stockmeyer. “What can be computed locally?” In *the Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*. 1993, pp. 184–193. doi: 10.1145/167088.167149 (cit. on pp. 23, 32).
- [Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi: 10.1137/1.9780898719772 (cit. on p. 24).
- [Roz24] Václav Rozhoň. “Invitation to Local Algorithms”. In *arXiv*. 2024. doi: 10.48550/arXiv.2406.19430 (cit. on p. 25).
- [RR97] Alexander A. Razborov and Steven Rudich. “Natural Proofs”. In *Journal of Computer and System Sciences* 55 (1997), pp. 24–35. doi: 10.1006/jcss.1997.1494 (cit. on p. 28).

- [Rub+11] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. “Fast Local Computation Algorithms”. In *the Proceedings of Innovations in Computer Science (ICS)*. 2011, pp. 223–238. doi: 10.48550/arXiv.1104.1377 (cit. on p. 49).
- [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)*. 2016, pp. 1–12. doi: 10.1145/2935764.2935799 (cit. on p. 28).
- [Suo13] Jukka Suomela. “Survey of local algorithms”. In *ACM Computing Surveys* (2013), 24:1–24:40. doi: 10.1145/2431211.2431223 (cit. on p. 25).
- [Whi09] Tom White. *Hadoop: The Definitive Guide*. O’Reilly Media, Inc., 2009. isbn: 0596521979 (cit. on pp. 18, 26).
- [Zah+10] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. “Spark: Cluster Computing with Working Sets”. In *the Proceedings of the USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*. 2010, p. 10. doi: 10.5555/1863103.1863113 (cit. on pp. 18, 26).

Business, Economy
Art, Design, Architecture
Science, Technology
Crossover

Doctoral Theses

Aalto DT 82/2026

ISBN 978-952-64-3082-9
ISBN 978-952-64-3081-2 (pdf)

Aalto University
School of Science
Department of Computer Science
aalto.fi