DISS. ETH NO. 30098

Local Complexity: New Results and Bridges to Other Fields

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES

(Dr. sc. ETH Zürich)

presented by

Václav Rozhoň
M.Sc. ETH in Computer Science
born on 03.01.1996

accepted on the recommendation of

Prof. Dr. Angelika Steger
Prof. Dr. Anton Bernshteyn
Prof. Dr. Mohsen Ghaffari
Prof. Dr. Bernhard Haeupler
Prof. Dr. Seth Pettie

2024

To my beloved Hanička & Toníček

0.1 CV

BIRTHDAY:	January 3, 1996
NATIONALITY:	Czech Republic
EMAIL:	[name][surname]@gmail.com
WEB PAGE:	https://n.ethz.ch/~rozhonv/
MARITAL STATUS:	married, one child

Education:

JUNE 2024	PhD student in the group of Mohsen Ghaffari **ETH Zurich**
JANUARY 2020	Master's Degree Program in Theoretical CS **ETH Zurich**
JUNE 2018	Bachelor's Degree Program in CS **Charles University in Prague**

Publications: The list of publications is available on my Google Scholar page.

Other: I write scripts for a YouTube channel Polylog.

Abstract

This thesis presents new results for the theory of local complexity in distributed computation and builds bridges to other areas of theoretical computer science and discrete mathematics, such as parallel, distributed, and sublinear algorithms, descriptive combinatorics, and finitary factors.

We begin the thesis with an extended introduction to the area of local algorithms, particularly focusing on recent complexity-theoretic developments. We hope this text could serve as a useful resource for a broader audience, especially newcomers to the field and researchers in adjacent areas.

Next, the thesis presents our new technical results in two parts. The first part contributes to our understanding of the possible complexities of local problems. Specifically, we classify the complexities of local problems on concrete graph families, with a focus on trees and grids. We also present new algorithms for fundamental problems, such as network decomposition and the Lovász local lemma.

The second part of the thesis builds and explores connections from local complexity to other fields: We apply techniques from local algorithms to classify the complexities of problems in the local computation model of sublinear algorithms. We extend local algorithms for network decompositions to weighted graphs, leading to new parallel and distributed algorithms for several graph problems. Finally, we explore the connection between local algorithms, descriptive combinatorics, and finitary factors, yielding new results in these fields.

Abstrakt

Diese Dissertation präsentiert neue Ergebnisse zur Landschaft der lokalen Komplexität in der verteilten Berechnung und baut Brücken zu anderen Bereichen der theoretischen Informatik und diskreten Mathematik, wie parallele, verteilte und sublineare Algorithmen, beschreibende Kombinatorik und endliche Faktoren.

Wir beginnen die Dissertation mit einer ausführlichen Einführung in den Bereich der lokalen Algorithmen, insbesondere unter Berücksichtigung der jüngsten komplexitätstheoretischen Entwicklungen. Wir hoffen, dass dieser Text als nützliche Ressource für ein breiteres Publikum dient, insbesondere für Neulinge auf dem Gebiet und Forscher in angrenzenden Bereichen.

Als Nächstes präsentiert die Dissertation unsere neuen technischen Ergebnisse in zwei Teilen. Der erste Teil trägt zu unserem Verständnis der möglichen Komplexitäten lokaler Probleme bei. Insbesondere klassifizieren wir die Komplexitäten lokaler Probleme an konkreten Graphenfamilien, mit einem Fokus auf Bäume und Gitter. Wir präsentieren auch neue und verbesserte Algorithmen für grundlegende Probleme, wie Netzwerkzerlegung und das Lovász'sche lokale Lemma.

Der zweite Teil der Dissertation baut Verbindungen von der lokalen Komplexität zu externen Feldern auf und erforscht diese: Wir wenden Techniken aus lokalen Algorithmen an, um die Komplexitäten von Problemen im lokalen Berechnungsmodell sublinearer Algorithmen zu klassifizieren. Wir erweitern lokale Algorithmen für Netzwerkzerlegungen auf gewichtete Graphen, was zu neuen parallelen und verteilten Algorithmen für verschiedene Graphenprobleme führt. Schließlich erforschen wir die Verbindung zwischen lokalen Algorithmen, beschreibender Kombinatorik und endlichen Faktoren, was zu neuen Ergebnissen in diesen Bereichen führt.

Acknowledgments

First and foremost, I would like to thank Mohsen for being an absolutely amazing mentor during my PhD studies. Conversations with Mohsen often felt a bit magical: oftentimes, I came to him with what looked like a hard problem (mostly mathematical but sometimes also personal) and came away understanding that everything is actually straightforward. It's a kind of experience that is really hard to explain in words.

I am also fortunate to have met Bernhard who was to me at once a great mentor, collaborator, friend, and living proof that less organized people than me actually exist. Bernhard taught me many valuable lessons, including the importance of keeping a close watch on my phone and keys, as they have a habit of mysteriously disappearing when he is around.

I am very lucky to have met Christoph who has been at once a great collaborator, friend, and running coach. I am not sure if I will ever find as much fun in complaining about grants, bureaucracy, LaTeX, and referees as I have had in conversations with him.

I am also very grateful to Emo Welzl for accepting me into his research group during my final year and helping me along the way. I would also like to thank Angelika Steger for mentoring me since I started my Master's studies at ETH and for her help in many aspects of my studies. Many thanks also go to Andrea Salow, for assisting me with all the formal requirements associated with my doctoral studies. I am very grateful to Anton Bernshteyn and Seth Pettie for serving as co-examiners.

I encountered so many amazing people during my PhD studies, and some of them ended up as my coauthors: Marcel Bezdrighin, Sebastian Brandt, Yi-Jun Chang, Davin Choo, Austin Conner, Martin Doležal, Michael Elkin, Tom Gavenčiak, Mohsen Ghaffari, Jan Grebík, Rachel Greenfeld, Christoph Grunau, Bernhard Haeupler, Richard Hladík, Jan Hladký, Saeed Ilchi, Rajesh Jayaram, Tereza Klimošová, Matěj Konečný, Jakub Łącki, Jason Li, Anders Martinsson, Ahmet Alper Özüdoğru, Jakub Pekárek, Diana Piguet, Julian Portmann, Israel Rocha, Štěpán Šimsa, Terence Tao, Robert Tarjan, Jakub Tětek, Mikkel Thorup, Zoltán Vidnyánszky, and Goran Žužić. I have learned so much from you while having a lot of fun doing research. Thank you!

Many more people deserve to be mentioned, and I cannot do justice to all of them. I fondly remember amazing discussions with all our group members (thanks, Manuela and Jara!), Juka Suomela and Aalto group members, Paul Christiano and researchers at ARC, researchers at Charles University, and many, many more people.

Lastly, none of this would have been possible without my amazing family, especially my beloved wife Hanička. Your support, encouragement, love, and especially near-infinite patience are absolutely essential to every aspect of my life. I love you!

Contents

Preface

The thesis is organized into two parts. The first part – Chapters 1 to 3 – contains a survey of the past work in the area of local algorithms, also containing some proofs. I wrote it in the hope that I would later add a few pictures and post it online so that it could be helpful, e.g., as a material for reading groups on local algorithms. I would greatly appreciate any feedback on this part of the thesis.

This first part is divided into three chapters: In Chapter 1, we explain how the theory of local algorithms is extremely clean if we disregard polylogarithmic factors. In Chapter 2, we put together a number of results in the area to give a more or less self-contained proof of the remarkable classification theorem of local problems with sublogarithmic local complexity on bounded degree graphs. In Chapter 3, we briefly discuss connections and applications of local algorithms to other areas of computer science and discrete mathematics.

The next part of the thesis begins with Chapter 4 which overviews some new results related to the area of local algorithms. These results are small pieces fitting into the big picture presented in the first three chapters and in Chapter 4, we discuss how exactly those small pieces fit there.

The presented technical results are divided into two parts. First, in Chapters 5 to 7, we focus on results that are related to the fundamental theory of local algorithms. We present three results based on the following three papers:

Chapter 5 presents results from the following paper:

> Václav Rozhoň, Bernhard Haeupler, and Christoph Grunau. A simple deterministic distributed low-diameter clustering. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 166–174. SIAM, 2023

Chapter 6 presents results from the following paper:

> Sebastian Brandt, Christoph Grunau, and Václav Rozhoň. Generalizing the sharp threshold phenomenon for the distributed complexity of the Lovász local lemma. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 329–338, 2020

Chapter 7 presents results from the following paper:

> Christoph Grunau, Václav Rozhoň, and Sebastian Brandt. The landscape of distributed complexities on trees and beyond. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 37–47, 2022

The remaining Chapters 8 to 10 can be seen as applying the principles and techniques of local algorithms to other related areas.

Chapter 8 presents results from the following paper:

> Sebastian Brandt, Christoph Grunau, and Václav Rozhoň. The randomized local computation complexity of the Lovász local lemma. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 307–317, 2021

Additionally, Chapter 8 contains a proof of a result proven in the paper covered in Chapter 7 that more closely fits into the topic of Chapter 8.

Chapter 9 presents results from the following paper:

> Václav Rozhoň, Michael Elkin, Christoph Grunau, and Bernhard Haeupler. Deterministic low-diameter decompositions for weighted graphs and distributed and parallel applications. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1114–1121. IEEE, 2022

Chapter 10 presents results from the following paper:

> Sebastian Brandt, Yi-Jun Chang, Jan Grebík, Christoph Grunau, Václav Rozhoň, and Zoltán Vidnyánszky. Local problems on trees from the perspectives of distributed algorithms, finitary factors, and descriptive combinatorics. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022

During his doctoral studies, the author has also coauthored other papers related to the local/distributed/parallel algorithms [170, 184, 185, 80, 274, 69, 129, 172, 273, 173] and some other papers [101, 271, 187, 190, 229, 189, 196, 186].

CHAPTER 1

Local Complexity Fundamentals

In the first chapter, we introduce local algorithms and local problems in Section 1.1. We then carefully discuss the appropriate formal definitions in Section 1.2. The following sections Sections 1.3 to 1.5 discuss the basic theory of local algorithms and aim to convey that we are after a very clean, fundamental, and robust concept. Finally, Section 1.6 surveys some known results for concrete local problems.

1.1 First Example

Consider a very long, oriented cycle that we want to properly color with as few colors as possible. Two colors are enough if the number of vertices n is even, otherwise we need 3 colors. However, there is something uneasy about the 2-coloring solution even when it is possible – the solution lacks any flexibility. A decision to color any particular vertex red implies how all other vertices are going to be colored. This can be bad for all kinds of reasons, typically when we want to design a coloring algorithm that is in some way parallel or distributed. This lack of flexibility goes away if we try to color the circle with 3 colors. Now, coloring one vertex red still implies that its neighbors are not red, but other vertices can have an arbitrary color.

Imagine that there is a computer in every vertex of the cycle and neighboring computers can communicate. The computers are trying to solve the coloring problem together. How many rounds of communication are needed until each computer outputs its color?

It is possible to convince oneself that in the case of 2-coloring, at least around $n/2$ rounds are necessary even if n is even. But what about the 3-coloring scenario? Can we solve it in 10 rounds of communication? Or $O(\log n)$? Or is it similarly hard to 2-coloring?

Before we answer this question, let us note that we need to be a bit careful about how we specify the computational model. There are two reasonable possibilities:

1. *Randomized algorithms:* Every computer has a coin that it can use to sample independent random bits.

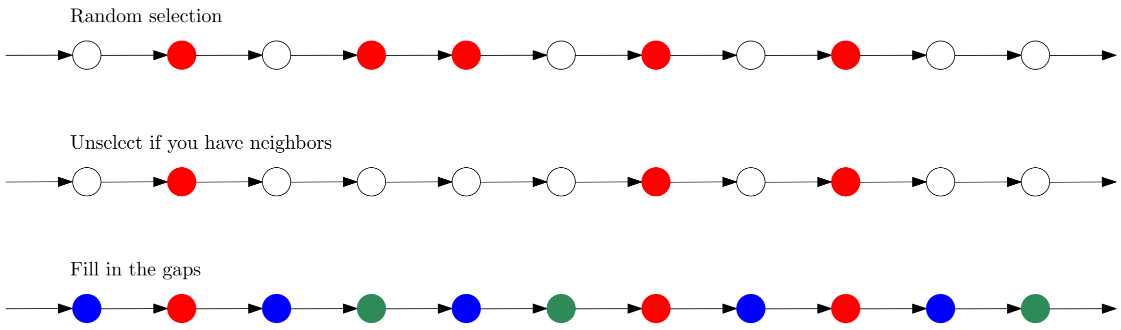

Figure 1.1: An example local algorithm that colors a long cycle, a small part of which is shown. First, every vertex flips a coin and selects itself with probability 1/2. Second, a vertex unselects itself whenever a neighbor is selected. Third, selected vertices color themselves red and each selected vertex is then responsible for coloring the subsequent vertices until the next selected one with alternating colors.

2. *Deterministic algorithms:* Every computer starts with a unique identifier (think of a MAC address) that has $O(\log n)$ bits.

The unique identifiers in case of deterministic algorithms are needed since otherwise deterministic algorithms would be extremely weak – in our example of a cycle, every computer sees the same local neighborhoods and without an additional way of breaking the symmetry all vertices then have to output the same color just because of the symmetry of the cycle.

Going back to our problem, let's see a randomized algorithm for 3-coloring a cycle with round complexity $O(\log n)$: That is, it finishes after $O(\log n)$ rounds of exchanging messages. This algorithm serves as an example that nontrivial algorithms are indeed possible, though we will see a better one for this problem later.

Our algorithm has two phases. In the first phase, every computer flips a coin and selects itself with probability $1/2$ (top picture in Figure 1.1). Subsequently, it asks its neighbors whether they are also selected. If at least one neighbor is selected, the vertex unselects itself (middle picture).

What are the lengths of the runs of unselected vertices between consecutive surviving selected vertices? They are only $O(\log n)$, with $1 - 1/\operatorname{poly}(n)$ probability (we will call this guarantee as "with high probability" later on). This is because if we split the cycle into pieces of consecutive triples of vertices, every node in the middle of a triple has a probability $1/8$ of being a surviving selected node, independently of what happens outside its triple. Using Chernoff bounds, we thus conclude that the probability of a run of ℓ consecutive vertices surviving decreases exponentially with ℓ.

We are now ready for the second phase of the algorithm. In this second phase, every selected vertex is responsible for coloring the part of the cycle until the next selected vertex. Specifically, it colors itself red and then colors the following vertices by alternating blue and green colors, until we reach the next selected vertex (bottom picture in Figure 1.1). Our algorithm properly colors the oriented cycle with 3 colors in $O(\log n)$ rounds.

Surprisingly, the best algorithm for the 3-coloring problem has a much better, albeit not constant complexity of $O(\log^* n)$.[1] However, instead of focusing on specific algorithms, this text is trying to give a bit more general understanding of what is going on here. For example, it turns out that if you construct any algorithm with complexity $O(\log n)$ for any reasonable problem defined on the cycle as we just did, there is a general theory that turns this algorithm into a new deterministic $O(\log^* n)$-round algorithm for the very same problem (Theorem 2.30). Clearly, something interesting is going on here!

1.2 Formal Definitions

In this section we formally define local problems and algorithms.

Local problems: We will be mostly interested in the so-called local problems. These are the problems on graphs such that if the solution is incorrect, we can find out by looking at a small neighborhood of one vertex.

Given a graph G and its node $u \in V(G)$, the *ball* $B_G(u, r)$[2] around u of radius r is the subgraph of nodes around u up to distance r. More generally, an *r-hop neighborhood* is a graph with one highlighted node v such that the radius of that graph measured from v is at most r.

Definition 1.1 (A local problem)**.** *Local problem*[3] *Π with checkability radius r is formally a triplet $(S, r, \mathcal{P})$. Here, S is a finite set of allowed labels and each $\mathcal{P}$ is a set of "allowed" S-colored r-hop neighborhoods. A solution to Π in a graph G is an assignment of a color from S to every vertex of G such that for every $u \in V(G)$ we have $B_G(u, r) \in \mathcal{P}$.*

For example, 3-coloring is a local problem for $S = \{\mathtt{R}, \mathtt{G}, \mathtt{B}\}$, $r = 1$, and $\mathcal{P}$ containing all properly colored 1-hop neighborhoods. On the other hand, a non-example of a local problem is coloring an input graph on n vertices with n colors: the local problem should not have different constraints for graphs of different size. Of course, while the theory of local algorithms is simplest for local problems as we defined them, the applications of local algorithms are not limited to local problems.

Local algorithms: There are two equivalent ways of thinking about local algorithms[4]

[1] The function $\log^* n$ measures how many times we need to take the logarithm of n until we get a value of size at most 2, i.e., $\log^* 2^2 = 1, \log^* 2^{2^2} = 2$ and so on.

[2] We sometimes write $B(u, r)$ when G is clear from context.

[3] Our definition is a simplified variant of the definition of the so-called locally checkable labeling problem by Naor and Stockmeyer [258], discussed later in Section 2.5.

[4] In the literature, these algorithms are often referred to as "distributed algorithms in the LOCAL

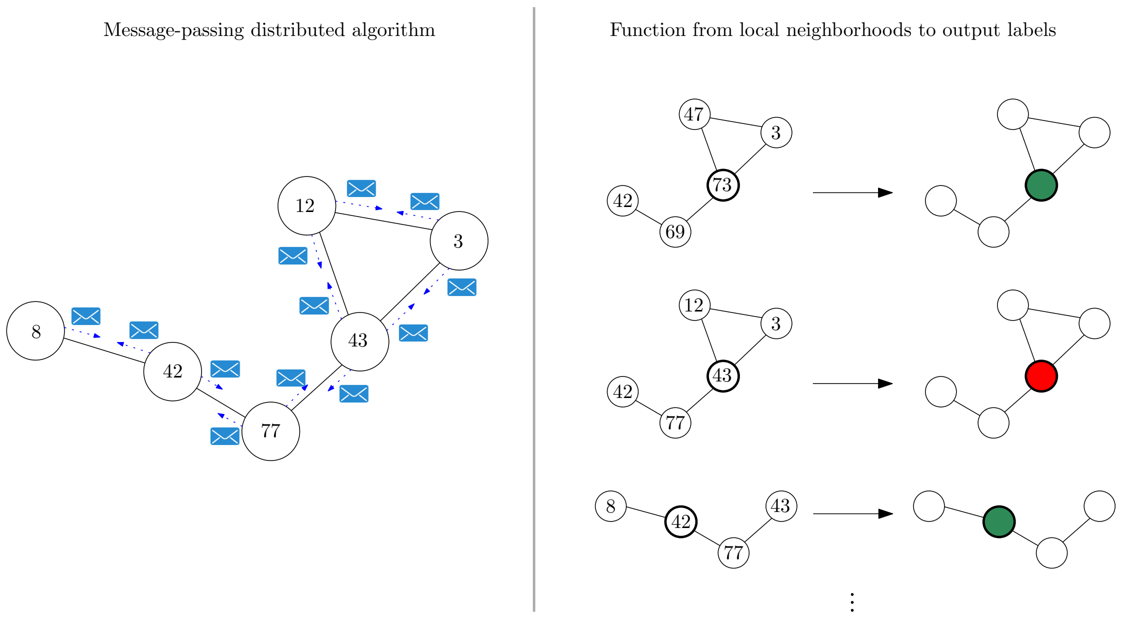
Message-passing distributed algorithm
12
3
8
42
43
77
Function from local neighborhoods to output labels
47
3
73
42
69
12
3
43
42
77
8
42
43
77
⋮

Figure 1.2: This picture shows the two fundamentally different, yet equivalent ways of understanding local algorithms.
Left: A $t(n)$-round local algorithm is a distributed protocol where in each round, each node can send any message to any of its neighbors. The computers starts with the knowledge of their unique identifier (or a random string).
Right: A local algorithm with round complexity $t(n)$ is a function that maps various $t(n)$-hop neighborhoods into output labels. Applying this function to every vertex of the input graph always has to solve our problem. For example, if our problem is a coloring problem, the first two local neighborhoods in above table need to be mapped to different colors, as the identifier labelings are compatible.

and both of them are important (see Figure 1.2). An intuitive, algorithmic definition was already sketched in Section 1.1: We assume that there is a computing device at every node. For simplicity, these devices are assumed to have unbounded computational power, thus excluding Turing machines from the definitions. A $t(n)$-round local algorithm is a protocol where these devices communicate for $t(n)$ rounds using the edges of the input graph to send messages. At the beginning, the device in each node starts only with the information about its identifier/random string and the size of the graph, n. When the protocol finishes, each device outputs its part of the solution (e.g., its color).

It will be also helpful to understand an alternative, equivalent definition that extracts the essence of what we are measuring with local algorithms. In this alternative definition, a local algorithm with round complexity $t(n)$ is simply a function that we can apply to every ball $B(u, t(n))$ of the input graph to compute the output at a given node u. Let us state it now formally.

Definition 1.2 (Local algorithm)**.** *A local algorithm $\mathcal{A}$ with a round complexity of $t(n)$ is a function that accepts two inputs: firstly, the value n, and secondly, a labeled $t(n)$-hop neighborhood of a certain node u.*

When we use this second definition, *running* a local algorithm on an input graph G simply means coloring each node $u \in V(G)$ with the output of $\mathcal{A}_n(B_G(u, t(n)))$. *Solving* a problem Π on G simply means that after running $\mathcal{A}$ on G, all output colors satisfy constraints $\mathcal{P}$.

Moreover, in the case of *deterministic* local algorithms, we assume that the nodes of the input graph are additionally labeled with unique identifiers from the range $[n^{O(1)}] = \{1, 2, \ldots, n^{O(1)}\}$.[5] In the case of *randomized* local algorithms, we assume that the nodes of the input graph are labeled with infinite bit strings. Solving a problem then means solving it with overall error probability at most $1/n^{O(1)}$, if the bit strings are sampled independently randomly.[6]

We notice that if there is a deterministic local algorithm solving some problem with round complexity $t(n)$, there is also a randomized local algorithm solving the same problem with the same round complexity. This is because in the randomized algorithm, each node can simply generate a random identifier from range $[n^C]$: The probability that these identifiers are not unique, i.e., some two nodes have the same identifier, is at most $n^2 \cdot \frac{1}{n^C}$. Choosing C large enough, this error probability can made as small as any polynomial function of n.

model of computing". We use the shorter and less formal term "local algorithm" for better readability.

[5]While assuming polynomial-range identifiers may look a bit arbitrary, we will see in Chapter 2 that the notion of deterministic algorithms is very robust. We simply need a way of breaking the symmetry of the input graph.

[6]Formally-minded readers may feel uneasy about the definitions not specifying the constant in the $n^{O(1)}$ expressions. We will see later in Theorem 2.20 that the exact constant in the definition typically does not matter. Formally, when we say that there is a local algorithm, it means that for every C there is an algorithm in the setup where the size of identifiers/error probability is n^C.

Finally, we remark that we can talk about local algorithms solving problems on graphs with additional structure (e.g. directed graphs) or on concrete graph classes. For example, in our introductory example from Section 1.1, it makes sense to think of all definitions relative not to the class of all graphs but to the class of graphs that are oriented paths. One interesting setup that we discuss mostly in Chapter 2 is the class of bounded-degree graphs. There, we fix some constant Δ and analyze the class of graphs of degree at most Δ.

Equivalence of the two definitions: Let's see a proof sketch of why the two definitions are equivalent. On the one hand, let's say we are given a function $\mathcal{A}$ that looks at $t(n)$-hop neighborhoods and we want to construct a $t(n)$-round message-passing protocol. Consider running a protocol where in the i-th round, each vertex u sends its neighbors everything there is to know about the ball $B(u, i)$: How the graph looks like and what are the identifiers/random strings at every node. Each node v can then internally use this information to learn everything there is to know about the ball $B(u, i+1)$. After $t(n)$ rounds of communication, each vertex v knows its $t(n)$-hop neighborhood $B(v, t(n))$. The vertices then stop communicating, and each one applies the function $\mathcal{A}$ locally to its ball.

On the other hand, assume that we have a $t(n)$-round communication protocol and want to turn it into a function $\mathcal{A}$ that takes $t(n)$-hop neighborhood as inputs. We notice that if we know the $t(n)$-hop neighborhood $B(u, t(n))$ of a node u, we can simulate the first round of the protocol in that ball and get to know the state of all vertices in $B(u, t(n)-1)$ after the first round. Continuing like this inductively, we conclude that starting with the knowledge of $B(u, t(n))$, we can learn the state of the protocol at the center node u after $t(n)$ rounds.

There are two different but equivalent ways of understanding local algorithms.
1. They are message-passing protocols running for some number of rounds.
2. The solution is such that the output at each node is a function of its neighborhood.

Importantly, it will be very helpful for us to keep *both* definitions in our mind: When we design local algorithms, the message-passing definition is more helpful as it is natural to think as "first, we run the protocol $\mathcal{A}_1$, then the protocol $\mathcal{A}_2$". On the other hand, when we prove lower bounds, the formal definition of Theorem 1.2 is easier to use. When we think of applications to distributed/parallel algorithms, the protocol-design definition is preferable since this is how the actual parallel/distributed algorithms are implemented. In some other applications, like applications to descriptive combinatorics, the formal definition is perhaps a bit more natural.

1.3 Sequential vs Distributed Local Complexity

This section presents one of the most fundamental results for local algorithms. Currently, it may be very unclear what kind of problems can be solved with a local algorithm of round complexity, say, $\operatorname{poly}\log n$. This will become much clearer, since we will next see that, up to $\operatorname{poly}\log n$, the model of local algorithms is the same as the model of so-called *sequential local algorithms* that are much easier to understand.

The case of maximal independent set: As an example, let's think of a concrete problem known as the *maximal independent set* problem. In this local problem, every node must be labeled either `selected` or `unselected`. The constraint is that each `selected` node should not neighbor any other `selected` node, while each `unselected` node should neighbor at least one `selected` node[7].

Is there a local algorithm constructing a maximal independent set in a polylogarithmic number of rounds? This is not clear at all! The answer to this question is positive, and perhaps the simplest algorithm is the randomized algorithm of Luby [239, 6]. This algorithm in fact served as the foundational example that later led Linial [236] to define local algorithms. But Luby's algorithm is a non-trivial algorithm[8] and just by staring at the maximal independent set problem, it is quite unclear whether a fast local algorithm exists, or not.

This stands in stark contrast with the "sequential" world: If we do not care about all vertices outputting the answer "at once", we can compute a maximal independent set with the following simple algorithm: Choose any order of vertices and iterate over them in that order. Whenever you consider a vertex u, look at its neighbors, and if at least one of them is already `selected`, mark u as `unselected`. Else, mark u as `selected`.

Here is a curious property of the above algorithm: We can still think of it as a "local" algorithm. Indeed, each node makes its decision by examining its 1-hop neighborhood. The only difference is that the algorithm is a *sequential local algorithm* where we iterate over nodes in an arbitrary order, not a *distributed local algorithm*[9] as we defined it in Theorem 1.2 where all nodes have to output the answer at once.

A fundamental result of local complexity is the fact that these two definitions are equally powerful, up to polylogarithmic factors. Hence, in the concrete example of the maximal independent set, you should not think of this problem as "easy" because of a clever algorithm like Luby's, you should think of it as easy because of the above simple sequential algorithm.

[7]This is a much easier problem than the *maximum independent set* problem where we additionally maximize the number of `selected` nodes.

[8]We can briefly describe the algorithm: It has $O(\log n)$ phases and in each phase, every vertex chooses a random number. If its number is the largest among its neighbors, the vertex goes in the independent set and is removed from future iterations, together with its neighbors.

[9]We sometimes use the name *distributed local algorithm* to stress that we are talking about a local algorithm and not a sequential local one.

The distributed round complexity of any problem equals its sequential local complexity, up to $\operatorname{poly}\log(n)$*.*

Formal definition of sequential local algorithms: We next make this principle formal. We need to start by defining a general sequential local algorithm. Here is a definition of a deterministic algorithm, made slightly more powerful than the maximal-independent-set algorithm by allowing the output of each node to be not just the final color, but also additional information that future vertices can read off the node.

Definition 1.3 (Sequential local algorithms)**.** *A sequential local algorithm of local complexity $t(n)$ is a function $\mathcal{A}$ defined on labeled $t(n)$-hop neighborhoods. Its output for a neighborhood $B(u, t(n))$ around a node u is a pair (s, t), with s being the output at u and t being additional information stored at u. An input neighborhood to $\mathcal{A}$ has some nodes labeled by these pairs.*

Running a sequential local algorithm means iterating over the vertices in some order and each time applying $\mathcal{A}$ to produce the answer at that vertex. When we run $\mathcal{A}$ on a node v, the algorithm has access to all already produced pairs (s, t) at the vertices of $B(u, t(n))$ on which $\mathcal{A}$ has already been run. Solving a problem with a sequential local algorithm means that regardless of the order *in which we choose the vertices, this process results in a solution to the problem.*

Notice that we do not require unique identifiers in the definition; we will see later in Theorem 2.7 that they are not needed for local problems. We can also define (oblivious) randomized algorithms where first an adversary chooses an order in which we iterate over vertices; then we sample random bits in each vertex and run our sequential local algorithm. We will next prove the following theorem by Ghaffari, Kuhn, and Maus [165]. [10]

Theorem 1.4 (Ghaffari et al. [165])**.** *Let $\mathcal{A}$ be a deterministic (randomized) sequential local algorithm with local complexity $t(n)$. Then, there is a deterministic (or randomized, respectively) distributed local algorithm simulating $\mathcal{A}$ with round complexity $\widetilde{O}(t(n) \cdot \log^3(n))$.*[11]

We note that it is known that there are local problems such that their sequential and distributed local complexity differ by a factor of $\Omega(\log n / \log\log n)$. [148]

Network decompositions: A crucial tool that we will rely on in this section and the next one is the concept of a *network decomposition.* Network decomposition is a clustering of the input graph into clusters of small diameter[12] (see Figure 1.3).

[10] Some version of this theorem was implicitly understood in the late 80s and led to the development of algorithms for network decompositions [13, 237].

[11] We use $\widetilde{O}(t(n))$ to denote the complexity $O(t(n) \cdot \log^{O(1)} t(n))$.

[12] A diameter of a graph G is defined as $\max_{u,v} d_G(u, v)$ where $d_G(u, v)$ is the distance between u and

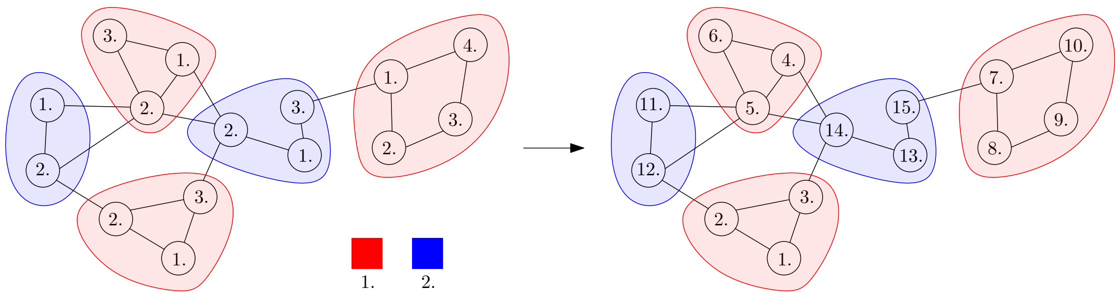

Figure 1.3: This picture shows a network decomposition with $c = 2$ color classes and $d = 2$ diameter. It also shows how network decomposition is used to convert an input sequential local algorithm (of local complexity 1) into a distributed local algorithm in Theorem 1.4.
Left: The color classes of the network decompositions are ordered as (red, blue). We iterate over the color classes and in one iteration, we consider each cluster separately and simulate an input sequential local algorithm in it (see the node ordering). When the algorithm is simulated in blue clusters, it has access to the output of neighboring red vertices.
Right: The partial simulations of the sequential local algorithm in each cluster are consistent with a single run of that algorithm over all vertices.

Definition 1.5 (Network decomposition)**.** *A (c, d)-network decomposition of a graph G is a coloring of G with c colors. We require that vertices of each color induce a graph such that each of its connected components (that we call* clusters*) has diameter at most d.*

We defer the discussion about the existence of network decompositions to Section 1.5. For now, we will simply state the guarantees of the currently best deterministic network decomposition construction.

Theorem 1.6 ([172])**.** *There is a deterministic local algorithm that outputs a $(O(\log n),$ $O(\log n \cdot \log\log\log n))$-network decomposition in $\widetilde{O}(\log^3 n)$ rounds.*

Proof of Theorem 1.4: Armed with the above algorithm for network decomposition, let us prove Theorem 1.4.

Proof of Theorem 1.4. An important concept employed throughout this text is working within the power graph: Given a graph G and a parameter r, we define the power graph G^r to be the graph with $V(G^r) = V(G)$ where two vertices are connected if their distance in G^r is at most r.

We start with a sequential local algorithm $\mathcal{A}$ of complexity $t(n)$. We will work in the power graph $G^{t(n)}$ and construct a network decomposition in it with $c, d' = \widetilde{O}(\log n)$ via

v in G.

Theorem 1.6. Consider this network decomposition in the context of the original graph G: We constructed clusters of actual diameter $d \leq t(n) \cdot d' = \widetilde{O}(t(n) \cdot \log n)$ in G. Moreover, two clusters from the same color class have distance at least $t(n) + 1$ in G. Finally, since every communication round in G^r can be simulated in r communication rounds in G, the round complexity of constructing our network decomposition is $\widetilde{O}(t(n) \cdot \log^3 n)$ by Theorem 1.6.

We will now use our clustering to simulate $\mathcal{A}$. We will simulate an order of iterating over the vertices where we first iterate over all the vertices in the first color class, then all the vertices in the second color class, and so on. For a fixed color class, we will arbitrarily simulate the algorithm $\mathcal{A}$ in each cluster C independently of all other clusters of the same color.

Notice that every two clusters C_1, C_2 of the same color i are far enough so that $\mathcal{A}$ run on a vertex $u \in C_1$ never has a vertex $u' \in C_2$ in its $t(n)$-hop neighborhood. Hence, simulations in different clusters of i-th color do not interact in any way. Our simulation is thus a faithful simulation of iterating over the vertices of G in a certain order and running the sequential algorithm $\mathcal{A}$ on them.

Finally, let us discuss how the simulation of $\mathcal{A}$ is implemented with a local algorithm. Think of a message-passing protocol with c phases where in the i-th phase, each cluster C of color i first chooses a leader node, e.g., the node with the smallest identifier. This node collects all information about the output of $\mathcal{A}$ so far up to the distance $t(n)$ from C. Then, the leader node internally simulates $\mathcal{A}$ on C and sends the result of that simulation back to all nodes in C. All this can be done in number of rounds proportional to the diameter of C and $t(n)$. Thus, the overall round complexity of the simulation is $c \cdot \widetilde{O}(t(n) \cdot \log n) = \widetilde{O}(t(n) \cdot \log^2 n)$. □

1.4 Derandomization

By now, we understand that the sequential local complexity is closely related to the distributed round complexity. However, we still do not understand the power of randomness. There might be scenarios where a problem's randomized (sequential or distributed) local complexity is significantly lower than its deterministic (sequential or distributed) complexity. Interestingly, this never happens for local problems. Distributed local algorithms for them can be derandomized with $\operatorname{poly}\log(n)$ slowdown in round complexity[13].

> *Any local problem has the same deterministic and randomized round complexity, up to* $\operatorname{poly}\log(n)$.

[13]It is important to restrict ourselves to local problems. Otherwise, consider the following silly counterexample problem: We are to mark some vertices of the input graph so that at least $n/3$ but at most $2n/3$ vertices are marked. Using randomness, this can be solved in 0 round complexity: every vertex simply flips a coin. But imagine trying to solve the problem deterministically: If the input graph has no edges, we are pretty screwed!

Before proving this result, let us contemplate how it fits into the big picture. Thus far, we have seen *six* plausible definitions of how to measure the local complexity of a problem. There are the following three different ways of thinking about it, and for each one of them, we can define both the deterministic and the randomized complexity:

1. (*distributed protocol*) There are computers at nodes, we design a message-passing protocol, and we measure the number of rounds of this protocol.

2. (*distributed local complexity*) Output at each node is a function of its local neighborhood.

3. (*sequential local complexity*) We iterate over the nodes in an arbitrary order and settle each output at a node by looking at its local neighborhood.

We now understand that all of these definitions are equivalent, up to $\operatorname{poly}\log(n)$ and for local problems.

Formal statement of derandomization: Formally, we will prove the following statement by Ghaffari, Harris, and Kuhn [166], using the derandomization method of conditional expectations.

Theorem 1.7 (Ghaffari et al. [166]). *Let Π be any local problem of randomized round complexity $t(n)$. Then, its deterministic sequential local complexity is $O(t(n))$.*

Proof. We are given a randomized local algorithm $\mathcal{A}$ with round complexity $t(n)$ for a local problem $\Pi = (S, \mathcal{P})$ with checkability radius r. We will next describe a deterministic sequential local algorithm that writes an infinite sequence of bits into each node u of the input graph G so that if we then simulate $\mathcal{A}$ with these bits, it solves Π.

For a vertex $u \in V(G)$, define the failure indicator $X(u)$ as the indicator random variable for the event that if we run $\mathcal{A}$ with truly random bits, it fails at u at solving the local problem $\Pi = (S, \mathcal{P})$. By failure at u we mean that $B(u, r) \notin \mathcal{P}$. We notice that $X(u)$ depends only on the output of $\mathcal{A}$ at an r-hop neighborhood of u, and thus it ultimately depends only on random bits in the $(r + t(n))$-hop neighborhood of u. Moreover, the probability of failure at u is less than $1/n$ by $\mathcal{A}$ being a randomized algorithm solving Π. This implies that $\mathrm{E}\left[\sum_{u \in V(G)} X(u)\right] < 1$.

Next, we will consider the following sequential local algorithm. We iterate over the nodes in an arbitrary order, and whenever it is a node u_k's turn, we fix the random bits $B(u_k)$ at this node to a value $b(u_k)$ such that

$$\begin{aligned}
&\mathrm{E}\left[\sum_{v \in V(G)} X(v) \;\middle|\; \forall i \in [k] : B(u_i) = b_i\right] \\
&\le \mathrm{E}\left[\sum_{v \in V(G)} X(v) \;\middle|\; \forall i \in [k-1] : B(u_i) = b_i\right].
\end{aligned}$$

In words, we set the random bits so that the expected number of errors does not increase.

First, such a value $b(u_k)$ of random bits at u_k definitely exists, since the right-hand side of the above inequality simply averages over many possible instantiations of $B(u_k)$ (that is, we rely on the law of total expectation). Second, we can compute this value of random bits by looking only at the $(r + t(n))$-hop neighborhood of u_k, since the values X_v for v outside of $B(u_k, r + t(n))$ are independent of the choice of random bits at u_k.

After this sequential algorithm with local complexity $(r + t(n))$ finishes, we have set the random bits at every vertex $u \in V(G)$ in a way that makes

$$\mathrm{E}\left[\sum_{u\in V(G)} X(u) \,\middle|\, \forall i \in [n] : B(u_i) = b_i\right] < 1.$$

But all values $X(u)$ are now deterministic, so we conclude that no failure occurs if we run $\mathcal{A}$ with these bits.

Finally, after this derandomization procedure is run, we also have to run $\mathcal{A}$. We will defer the discussion of how to combine two sequential local algorithms into one to Theorem 1.9 that shows how to construct a sequential local algorithm with local complexity $O(t(n))$ that simulates first running the derandomization procedure and then running $\mathcal{A}$ with the bits computed by that procedure. □

Putting Theorems 1.4 and 1.8 together, we get the following derandomization theorem for (distributed) local algorithms.

Theorem 1.8 (Ghaffari et al. [166]). *Let Π be any local problem of checkability r and randomized round complexity $t(n)$. Then, its deterministic distributed local complexity is $\widetilde{O}(t(n) \cdot \log^3 n)$.*

Conversely, it's known that for some local problems, the gap between randomized and deterministic local complexity can be as large as $\Omega(\log n/\log\log n)$ [28].

Leftover: composing sequential algorithms: We will briefly discuss how two sequential local algorithms run one after the other can be composed into a single one of larger local complexity.

Lemma 1.9 (Ghaffari et al. [165, Observation 2.1, Lemma 2.2]). *Let $\mathcal{A}_1, \mathcal{A}_2$ be two deterministic (randomized) sequential local algorithms with local complexities $t_1(n), t_2(n)$. Then, there is a single deterministic (randomized, respectively) sequential local algorithm $\mathcal{A}$ of complexity $2(t_1(n)+t_2(n))$[14] that simulates the output of first running $\mathcal{A}_1$, and then running $\mathcal{A}_2$ on the output of $\mathcal{A}_1$.*

[14]In general, k local sequential algorithms can be simulated with complexity $2\sum_{i=1}^{k} t_i(n)$.

Proof. We would like $\mathcal{A}$ to work as follows: We iterate over the nodes and when it is the turn of a node u, we first simulate $\mathcal{A}_1$ for all nodes in $B(u, t_2(n))$. Then, we use the computed information to simulate $\mathcal{A}_2$ at u to compute the final output at u.

The only difficulty is that once u simulates $\mathcal{A}_1$ for a node $v \in B(u, t_2(n))$, we cannot simulate $\mathcal{A}_1$ at v again in the future since we want to have the guarantee that all simulations of $\mathcal{A}_1$ taken together correspond to a consistent iteration over the nodes of the input graph and running $\mathcal{A}_1$ on them.

To implement this idea, $\mathcal{A}$ additionally stores at u the output of simulations of $\mathcal{A}_1$ within $B(u, t_2(n))$. The final local complexity of $\mathcal{A}$ will be $\max(t_1(n) + t_2(n), 2t_2(n))$: $\mathcal{A}$ starts by looking at its $2t_2(n)$-hop neighborhood and fixing the answers of $\mathcal{A}_1$ for nodes $v \in B(u, t_2(n))$ at which $\mathcal{A}_1$ was already simulated in the past. Only then we simulate $\mathcal{A}_1$ for the rest of the nodes in $B(u, t_2(n))$ and run $\mathcal{A}_2$ after that. □

1.5 Network Decompositions

Let us recall the definition of a network decomposition:

Definition 1.10 (Network decomposition)**.** *A (c, d)-network decomposition of a graph G is a coloring of G with c colors. We require that vertices of each color induce a graph such that each of its connected components (that we call* clusters*) has diameter at most d.*

Let us now discuss constructions of network decompositions[15]: the missing piece in proof of Theorems 1.4 and 1.8.

Existence of network decompositions: First of all, it is unclear whether network decomposition of the input graph, say with parameters $c, d = O(\log n)$ always exists. Let's first confirm this by constructing it using the following folklore sequential *ball-carving* algorithm.

Theorem 1.11 (Ball-carving algorithm)**.** *An $(O(\log n), O(\log n))$-network decomposition exists for any graph G.*

Proof. We will show how to construct the first color class of the network decomposition. In particular, we will find a family of vertex-sets $\mathcal{C} = \{C_1, C_2, \ldots, C_t\}$, where we call each $C_i \subseteq V(G)$ a *cluster*, such that:

1. The clusters are not adjacent; i.e., there is no edge $uv \in E(G)$ with $u \in C_i, v \in C_j$, $i \neq j$,
2. each cluster C_i has diameter $O(\log n)$,

[15] In the literature, one can very often encounter numerous variants of network decompositions with names like low-diameter clusterings, padded decompositions, or sparse neighborhood covers.

3. at least $n/2$ vertices are clustered, i.e., $\left|\bigcup_{i=1}^{t} C_i\right| \geq n/2$.

We can then construct the desired network decomposition by simply repeating this process $\log_2 n$ times and always removing the clustered vertices from the graph until every vertex is clustered.

To construct $\mathcal{C}$ in one phase, iterate over the vertices of G in an arbitrary order. Each time we are at a vertex u, we consider the balls $B(u, 0), B(u, 1), \ldots$ of increasing radius around it. In particular, think of gradually growing larger and larger balls around u, and once the size of the ball does not at least double, i.e., once we have $|B(u, i+1)| \leq 2 \cdot |B(u, i)|$ for the first time, we let $B(u, i)$ be the next cluster of $\mathcal{C}$. Additionally, we remove all vertices from the boundary $B(u, i+1) \setminus B(u, i)$ from G until the end of the phase. These vertices then survive to the next clustering phase.

To analyze this algorithm, first notice that no two clusters are adjacent, since after growing each cluster, we delete its boundary (Item 1).

Moreover, all balls have diameter $O(\log n)$: This is because the volume of each ball $B(u, i)$ grows exponentially as $|B(u, i)| \geq 2^i$ which implies that $B(u, 1 + \log_2 n)$ would have to contain more than n vertices (Item 2).

Finally, we cluster at least half of the vertices, because by definition, whenever a ball stops growing, we have $|B(u, i+1) \setminus B(u, i)| \leq |B(u, i)|$, so the number of vertices unclustered because of the ball around u can be charged to the number of vertices clustered around the same vertex (Item 3). □

Deterministic distributed algorithms: There is a long line of work on deterministic algorithms for constructing network decompositions [13, 262, 276, 170, 87, 272, 275, 172]. The currently fastest deterministic algorithm for network decomposition of Akbari, Coiteux-Roy, d'Amore, Gall, Lievonen, Melnyk, Modanese, Pai, Renou, Rozhoň, et al. [4] that we stated in Theorem 1.6.

Perhaps the simplest poly $\log n$-round algorithm is the algorithm of Rozhoň and Ghaffari [276] and its variant by Rozhoň et al. [275]. They both need $O(\log^7 n)$ rounds and construct clusters with diameter $O(\log^3 n)$. We will next explain the construction from [276] that additionally provides clusters with only a weaker guarantee of small *weak-diameter*. A weak-diameter of a cluster C is defined as $\max_{u,v \in C} d_G(u, v)$, as opposed to the (strong-) diameter which is defined as $\max_{u,v \in C} d_{G[C]}(u, v)$. That is, a cluster with small weak diameter can be even disconnected. Small weak diameter is sufficient for our applications and we will discuss after the proof how one can turn it into a strong-diameter guarantee in a black box way.

Theorem 1.12 (Distributed ball-carving algorithm)**.** *There is a local algorithm with $O(\log^7 n)$ round complexity that constructs a network decomposition with $c = O(\log n)$ colors and $d = O(\log^3 n)$ weak-diameter.*

Proof. Similarly to Theorem 1.11, we show how to construct a family of clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_t\}$ such that:

1. No two clusters are adjacent,
2. each cluster C_i has weak-diameter $O(\log^3 n)$,
3. at least $n/2$ vertices are clustered.

Recall that in deterministic local algorithms, every vertex starts with a unique $b = O(\log n)$-bit identifier. Our algorithm has b phases. At the beginning of the algorithm, we start with a clustering $\mathcal{C}_0$ that contains each node of G as a trivial one-vertex cluster. Each cluster C in our clustering is assigned an identifier $id(C)$ which is a unique b-bit string; at the beginning, the identifier of a cluster is set to be the identifier of its unique vertex.

This clustering evolves during the following b phases, with some vertices changing a cluster and some vertices being deleted from $\mathcal{C}_i$. We will prove that after each phase $i \in [b]$, the clustering $\mathcal{C}_i$ has the following guarantees:

1. Consider the graph $G_i = G[\bigcup_{C \in \mathcal{C}_i} C]$, i.e., the graph induced by vertices that were not deleted. Consider any connected component of G_i. All clusters in this connected component agree on the first i bits of their unique identifier.
2. Each cluster $C \in \mathcal{C}_i$ has weak-diameter $O(i \cdot \log^2 n)$.
3. $|V(G_i)| \geq n - i \cdot \frac{n}{2b}$.

Plugging $i = b$, the inductive guarantees of Items 1 to 3 reduce to the final desired guarantees of Items 1 to 3. Thus, we only need to show how one phase of the algorithm makes sure that if we start with the guarantees for some i, they also hold for $i + 1$.

At the beginning of each phase $i + 1 \in [b]$, we classify each cluster in $\mathcal{C}_i$ as either *active* or *inactive*, where a cluster C is active if and only if the $(i + 1)$-th bit in $id(C)$ is equal to zero.

Next, we run a variant of the ball-carving algorithm from Theorem 1.7 with $t = 10b \log_2 n$ steps. In each step of this algorithm, each vertex u from some inactive cluster first checks whether it neighbors with a vertex that is currently part of an active cluster. If there are more such neighboring vertices, u chooses an arbitrary such vertex $v \in C$ and *proposes to join* C.

Next, each cluster C collects how many vertices u proposed to join C. If there are at least $|C|/(2b)$ such vertices, then C decides to grow: All the vertices that proposed to C leave their original cluster and join C. On the other hand, if less than $|C|/(2b)$ proposed to join C, all these vertices are *deleted* and will not participate in the rest of the algorithm; after the deletion, the cluster C is neighboring only with nodes from other active clusters and thus it does not grow anymore in the rest of the phase. This finishes the description of the phase.

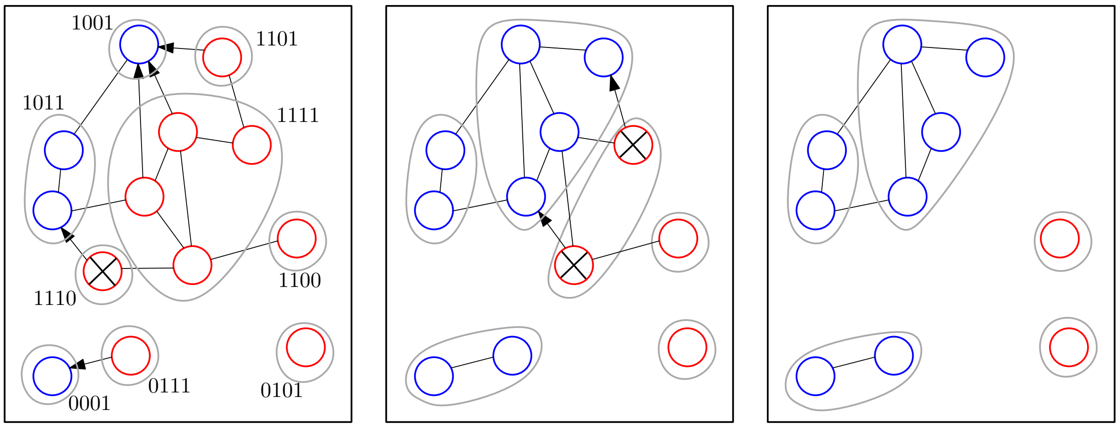

Figure 1.4: This picture illustrates the second phase of the algorithm from Theorem 1.12, in a simple example graph. Left: At the beginning of the second phase, the clusters are already separated according to their first bit, i.e., if the identifier starts with 1, the cluster is in the top connected component, while if the identifier starts with 0, the corresponding cluster is in one of the two bottom connected components. In the second phase, a cluster is active (blue) if its second bit in the identifier is 0, and inactive (red) otherwise. In the first step of this phase, each red vertex proposes to join an arbitrary neighboring blue cluster, provided that there is one (the arrows in the picture). The active cluster than either decides to grow (cluster 1001) or the vertices that proposed to it are deleted (cluster 1011).
Middle: In the second step, each red vertex again proposes to join an arbitrary neighboring blue cluster.
Right: The picture shows the resulting clustering after the second phase is finished. Note that the only two adjacent clusters (1011 and 1001) will be separated in the following, third, phase, as their identifiers differ on the third position.

To analyze the ball growing of phase $i+1$, we first check that each cluster stops growing during the algorithm. Otherwise, it would have to contain more than

$$(1+1/(2b))^{10b\log_2 n} > n$$

nodes. In particular, the weak-diameter of each active cluster grows additively by at most $t = O(\log^2 n)$, while the weak-diameter of each inactive cluster does not increase. This proves Item 2.

Next, consider any connected component K of G_i. The clusters in K already agree on the first i bits of their identifier at the beginning of the phase $i+1$. Recall that all active clusters stop growing during the phase and delete their boundary with inactive clusters. Thus in G_{i+1}, the component K further splits into connected components $K_1, K_2, \ldots$ such that for each component K_i we have that either all of its clusters are active or all are inactive. We conclude that clusters in each connected component of G_{i+1} agree on the first $i+1$ bits in their identifier as needed in Item 1.

Finally, whenever a cluster C stops growing, we delete at most $|C|/(2b)$ nodes. Thus, during the phase we delete at most $n/(2b)$ nodes which proves Item 3.

Finally, we discuss the running time of the algorithm. The clusters have diameter $O(\log^3 n)$ so each growing step can be implemented with that round complexity. There are $t = O(\log^2 n)$ steps, $b = O(\log n)$ phases, and we need to repeat the overall algorithm $c = \log_2 n$ times. We conclude that the overall round complexity is $O(\log^7 n)$. □

Next, let us show how we can combine our current knowledge of local algorithms to construct a local algorithm for network decomposition with $c, d = O(\log n)$.

Corollary 1.13. *There is a local algorithm with $O(\log^9 n)$ round complexity that constructs a network decomposition with $c = O(\log n)$ colors and $d = O(\log n)$ diameter.*

Proof. We observe that the algorithm from Theorem 1.11 can be seen as a sequence of $O(\log n)$ sequential local algorithms per Theorem 1.3, each one with local complexity $O(\log n)$. Using Theorem 1.9, we can thus view it as a single sequential local algorithm of local complexity $O(\log^2 n)$.

Thus, we can use Theorem 1.4 to convert this algorithm into a local algorithm. While we phrased the guarantees of Theorem 1.4 using the best available network decomposition algorithm, we can plug in Theorem 1.12 instead; the reduction holds even if we use a network decomposition with a weak-diameter guarantee. □

Let us briefly go back to the observation from Theorem 1.13 that Theorem 1.11 can be viewed as a sequential local algorithm. This observation helps us to appreciate network decomposition as the "complete" problem for turning sequential local algorithms into distributed ones: On the one hand, network decomposition allows us to do this task via Theorem 1.4; on the other hand, *any* way of proving that theorem leads to a distributed algorithm for network decomposition via the proof of Theorem 1.13.

Randomized distributed algorithms: There is a classic randomized algorithm by Linial and Saks [237] that constructs an $(O(\log n), O(\log n))$-network decomposition in $O(\log^2 n)$ distributed rounds. Let us sketch its beautiful variant by Miller, Peng, and Xu [255] also known as the MPX algorithm:

In their algorithm, every vertex independently chooses a random *head start* sampled from an exponential distribution; that is, a head start of each node is 0 with probability 1/2, 1 with probability 1/4, and so on. Next, we run the breadth-first-search algorithm from all nodes at once with those head starts. That is, we first choose some number $T = O(\log n)$ such that with high probability, no node samples a head start larger than T. Then, we simulate a run of breadth-first search from an additional virtual node u_0 which is connected to every other node u with an oriented edge of length $T - \text{head start}(u)$. All vertices reached by the breadth-first search from the same starting node u form one cluster. The output of the algorithm are only those vertices such that all their neighbors are from the same cluster.

One can prove that the output clustering from this algorithm contains at least a constant fraction of all vertices and uses disjoint clusters of diameter $O(\log n)$, with high probability.

1.6 Bounds for Concrete Problems

In the last section of this chapter, we briefly survey known results for some concrete well-known local problems. While the technology developed in this section often automatically provides a $\text{poly} \log n$-round deterministic algorithm for a given problem, we can try to improve the complexity further: For many problems, we can get randomized algorithms with complexities closer to $\text{poly} \log \log n$. Typically, we want to understand the complexity as the function of both the number of vertices n, and the maximum degree Δ.

Many concrete results use certain general techniques. Randomized local algorithms with $\text{poly} \log \log n$ dependency typically use the shattering framework (see Section 2.2) [48, 133, 92]. Deterministic local algorithms with $\text{poly} \log n$ rounds often rely on local rounding techniques [198, 134, 200, 158, 129]. Many state-of-the-art lower boundsare proven using the technique of round elimination (see Sections 2.1 and 2.2 or survey of Suomela [288]) [75, 73, 22, 30, 76, 36, 26].

Maximal independent set: The maximal independent set problem is one of the most prominent problems in the area of local, distributed, parallel, or sublinear algorithms. The seminal randomized Luby's algorithm [239, 6] with $O(\log n)$ round complexity initiated a long line of work on the problem [239, 6, 257, 43, 150, 22, 153]. The currently fastest deterministic algorithm by Ghaffari and Grunau [153] combines ideas related to network decompositions and local rounding techniques to achieve round complexity $\widetilde{O}(\log^2 n)$. On the lower bound side, Balliu et al. [22] used round elimination to show that the deterministic local complexity of maximal independent set is $\Omega(\log n / \log\log n)$. Thus, there is still a gap in our understanding:

Problem 1.14. *Is there a deterministic $\widetilde{\Theta}(\log n)$-round algorithm for maximal independent set?* [16]

On the randomized side, there is a randomized local algorithm for the problem by Ghaffari [150] using the shattering approach and the best deterministic algorithm for the maximal independent set. With the currently best deterministic algorithm, its round complexity is $O(\log\Delta) + \widetilde{O}(\log^3 \log n)$.

This complexity is very close to optimal: Balliu et al. [22] use round elimination to give a lower bound of $\Omega(\frac{\log\log n}{\log\log\log n})$ for randomized algorithms, on a graph with $\Delta = \Omega(\frac{\log\log n}{\log\log\log n})$. Next, a lower bound of Kuhn, Moscibroda, and Wattenhofer [227] says that the local complexity of maximal independent set is $\Omega(\frac{\log\Delta}{\log\log\Delta})$. Thus, one cannot hope for an algorithm with round complexity $O(\log\Delta)+o(\frac{\log\log n}{\log\log\log n})$, or even $o(\Delta)+o(\frac{\log\log n}{\log\log\log n})$.

Maximal Matching: Maximal matching can be seen as a special case of the maximal independent set problem on the line graph, i.e., the graph constructed from an input graph G that has $E(G)$ as the vertex set and it contains an edge whenever two original edges of G share a vertex. There is a long line of work on the problem [203, 178, 150, 22, 153, 38, 115]; the currently fastest local algorithms are a bit faster than the algorithms for maximal independent set. The deterministic complexity is $\widetilde{O}(\log^{4/3} n)$ by [153] and the fastest randomized algorithm of Ghaffari [150] has complexity $O(\log\Delta + \log^{4/3}\log n)$. On the other hand, the lower bounds of Balliu et al. [22] for maximal independent set also apply to matching.

Vertex coloring: Linial's coloring algorithm [236] that we will discuss more in-depth later in Section 2.1 provides a coloring of graphs of degree at most Δ with $\Delta + 1$ many colors in $O(\Delta^2 + \log^* n)$ time. There is a long line of work trying to improve this complexity: [239, 6, 104, 175, 236, 290, 216, 257, 261, 226, 224, 281, 45, 47, 42, 48, 201, 92, 49, 248, 247, 158, 153, 225]. The fastest known deterministic algorithm in the regime of large Δ is either the algorithm of Ghaffari and Kuhn [158] with $O(\log n \cdot \log^2\Delta)$ complexity, or the $\widetilde{O}(\log^2 n)$-round algorithm of Ghaffari and Grunau [153]. The fastest randomized algorithm in this regime is by Chang et al. [92] and uses the shattering framework: Its randomized round complexity is $\widetilde{O}((\log\log n)^2)$. If we want the dependency on n to be only $O(\log^* n)$, then the fastest algorithm is the algorithm of Fraigniaud, Heinrich, and Kosowski [136] with complexity $O(\sqrt{\Delta}\log\Delta + \log^* n)$. The following problem is well-known in the local algorithms community.

Problem 1.15. *Is there a $O(\text{poly}\log\Delta+\log^* n)$-round local algorithm for $\Delta+1$ coloring?*

[16]This text lists some open problems. For a different list, see [289].

There is a long line of work on numerous variants of the coloring problem, in particular on Δ-coloring (i.e., local Brooks' theorem) [169], defective colorings [102, 139], frugal coloring [102], fractional colorings [33], arboricity-dependent coloring [45], weak coloring [224, 23, 24] and other variants.

Edge coloring: When allowing $2\Delta - 1$ colors, the proper edge coloring problem becomes a special case of $\Delta + 1$ vertex coloring on line graphs, making the results for that problem applicable. There is a long line of work on $2\Delta - 1$ edge coloring trying to improve upon the state of the art for $\Delta + 1$ vertex coloring [29, 37, 167]. As a highlight, Balliu et al. [37] provided a $O(\operatorname{poly}\log \Delta + \log^* n)$-round algorithm for it, thus resolving Problem 1.15 in this special case.

Another natural target is $\Delta + 1$ edge coloring guaranteed by Vizing's theorem. There is a long line of work on the problem [90, 183, 182, 64, 66, 65]; the fastest algorithms of Bernshteyn [64], Bernshteyn and Dhawan [66] have deterministic $\operatorname{poly}(\Delta \log n)$ local complexity.

Spanners: Given a graph G, its subgraph H is a spanner with *stretch* α if for every two nodes u, v we have $d_G(u, v) \leq d_H(u, v) \leq d_G(u, v)$. It is known that for any k there is always a spanner with stretch $2k - 1$ and $n^{1+1/k}$ edges [9]. The task is to construct a spanner with similar parameters with a local algorithm.

This problem is a nice example of a non-local problem (certifying whether a subgraph H is a spanner has checkability radius $\Omega(n)$ in general) where it is still meaningful to ask about its local complexity. It is known that the deterministic local complexity of constructing a spanner with stretch $2k - 1$ with $O(kn^{1+1/k})$ edges is $O(k)$ [53, 117]. For $k = \operatorname{poly}\log n$, we can deterministically construct spanners with $\operatorname{poly}\log n$ stretch and $n + n/\operatorname{poly}\log n$ edges in $\operatorname{poly}\log n$ deterministic rounds [122, 69].

Other problems: There are many other problems studied in the literature: Approximation to covering or packing integer linear programs [165], approximate maximum matching and flows [259, 12, 135, 112], dominating sets and set covers [287, 177, 118], low out-degree orientations [44, 162], and many others.

CHAPTER 2

The Sublogarithmic Regime

In the previous chapter, we gained a good understanding of the fundamentals of local complexity – in essence, if we care about the complexities up to $\text{poly} \log n$, there are several equivalent definitions of it with sequential local algorithms being a particularly helpful model for designing algorithms.

In this chpater, we will restrict our attention to bounded degree graphs and local problems of sublogarithmic complexity. Something remarkable is going to happen: We will see that while, a priori, we would expect all kinds of problem complexities, there are only three distinctive classes of local problems. This *sharp threshold phenomenon* is a consequence of remarkable results known as *speedup theorems.*

As a rough roadmap for the rest of the chapter, we are going to give a more or less self-contained proof of the following Theorem 2.1 in it, with Section 2.1 focusing on the symmetry-breaking regime, Section 2.2 focusing on the Lovász-local-lemma regime, and Section 2.3 focusing on showing that there are gaps in between the regimes.

Theorem 2.1 (Classification of local problems with $o(\log n)$ complexity on bounded degree graphs)**.** *Let us fix any Δ and the class of graphs of degree at most Δ. Then, any local problem with randomized round complexity $o(\log n)$ has one of the following three round complexities.*

1. Order-invariant regime: *The problem has $O(1)$ deterministic and randomized round complexity.*
2. Symmetry-breaking regime: *The deterministic and randomized round complexity of the problem lies between $\Omega(\log \log^* n)$ and $O(\log^* n)$ (both the deterministic and the randomized complexity is the same function).*
3. Lovász-local-lemma regime: *The problem has deterministic round complexity between $\Omega(\log n)$ and $\widetilde{O}(\log^4 n)$. Its randomized round complexity is between $\Omega(\log \log n)$ and $\widetilde{O}(\log^4 \log n)$.*

2.1 The Symmetry-Breaking Regime

In the bounded-degree regime, the problems of maximal independent set or the closely related problem of $(\Delta+1)$-coloring play a bit similar role to network decomposition, as we will see in Theorem 2.5. These problems are known as *symmetry-breaking* problems. This section shows that the round complexity of these problems is $\Theta(\log^* n)$ on bounded-degree graphs.

Basic symmetry breaking problems like maximal independent set and $\Delta+1$ coloring are closely related. Their round complexity on bounded degree graphs is $\Theta(\log^* n)$.

2.1.1 Fast Coloring Algorithm and its Implications

We will first discuss a classical local coloring algorithm of Linial [236] improving upon earlier work [104, 175]. This algorithm colors a graph of degree at most Δ with $O(\Delta^2)$ many colors in $O(\log^* n)$ rounds. We will later see in Theorem 2.5 that this implies that on bounded-degree graphs, the local complexity of maximal independent set and $\Delta+1$ coloring is $O(\log^* n)$. We will only analyze a weaker version of Linial's algorithm here, see e.g. [236, 174] for the full proof.

Theorem 2.2 (Linial [236])**.** *The deterministic round complexity of constructing a coloring with $O(\Delta^2)$ many colors is $O(\log^* n)$.*

Partial proof. We will only prove a weaker result where the polynomial dependency in Δ in the round complexity is replaced by exponential dependency.

Recall that every vertex starts with a unique identifier. In particular, we view these identifiers as an input coloring with $n^{O(1)}$ many colors. Next, we will show how in one round, we can use a *color reduction* procedure to turn an input coloring that uses 2^k colors into a coloring with $2^{O(\Delta \cdot \log k)}$ many colors.

At the beginning of one color reduction procedure, each vertex u holds a k-bit string s_u that translates into its color from $[2^k]$ (see Figure 2.1). The vertex sends this color to all its neighbors. After u receives $d \le \Delta$ strings of its neighbors, $s_1, s_2, \dots, s_d$, it does the following. For each received s_i, the node u identifies an index j_i where s_i differs from its own bit string s_u. Such an index exists since we assumed that we started with proper coloring. The new color of u is defined as the concatenation $s'_u = j_1 \circ s_u[j_1] \circ \cdots \circ j_d \circ s_d[j_d]$. That is, the vertex u simply remembers only the values of bits of s_u on the positions where those values show that s_u differs from its neighbors. We observe that no two neighboring vertices can end up with the same string. Moreover, if we write s'_u in binary, it has only $O(\Delta \cdot \log k)$ many bits.

We will now iterate this procedure $O(\log^* n)$ times. A brief calculation shows that we end up with $2^{O(\Delta \log \Delta)}$ colors. □

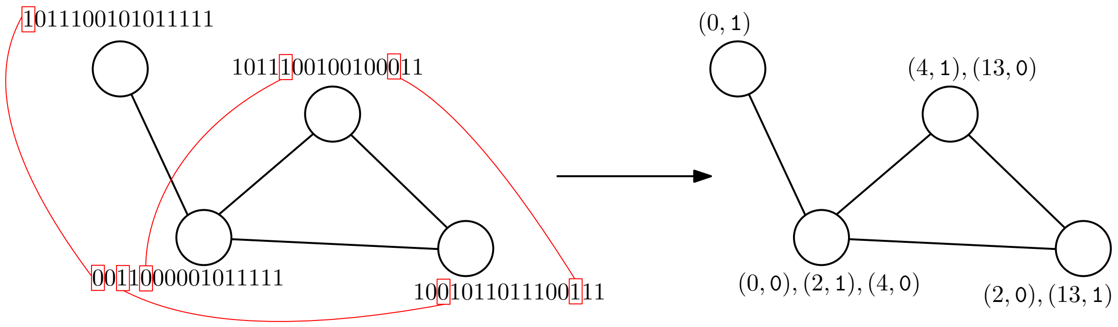

Figure 2.1: One step of the simple color-reduction algorithm from the proof of Theorem 2.2.
Left: We represent the current color of each vertex as a bit string. Each pair of vertices finds a position where the respective bit strings differ.
Right: The new color of each vertex only encodes the values of the original bit strings in these important positions.

Actual Linial's algorithm uses the same strategy as in the above proof, but it uses a more clever encoding in every color-reduction step using the so-called cover-free families that we define next.

Definition 2.3. *Given a ground set $[k']$, a family of sets $S_1, \ldots, S_k \subseteq [k']$ is called a Δ-cover free family if for each set of indices $i_0, i_1, \ldots, i_\Delta \in [k]$, we have $S_{i_0} \setminus \left(\cup_{j=1}^{\Delta} S_{i_j}\right) \neq \emptyset$. That is, no set in the family is a subset of the union of some Δ other sets.*

A generalization of our partial proof says that in one round, one can turn a proper k-coloring into a proper k'-coloring, whenever a Δ-cover free family of size k over $[k']$ exists. Theorem 2.2 then follows from the following statement on the existence of cover-free families, proof of which we omit.

Theorem 2.4 ([221, 126] or [174, Lemma 1.19, 1.20])**.** *For any k, Δ, there exists a Δ-cover free family of size k on a ground set of size $k' = O(\Delta^2 \log k)$. Moreover, if $\Delta \geq k^{1/3}$, it exists for $k' = O(\Delta^2)$.*

Simulation of sequential local algorithms: We notice that graph coloring can be seen as a special case of network decomposition discussed in Chapter 1 where each cluster has diameter 0. In particular, coloring with a small number of colors allows us to turn sequential local algorithms into distributed local ones similarly to Theorem 1.4.

Theorem 2.5. *Let $\mathcal{A}$ be a sequential local algorithm with local complexity $t(n)$. Then, we can turn it into a distributed local algorithm of round complexity $O(\Delta^{O(t(n))} + t(n) \cdot \log^* n)$. If the sequential algorithm is deterministic, so is the distributed one.*

Proof. We follow the outline of the proof of Theorem 1.4 that simulated sequential local algorithms using network decompositions. To simulate a sequential local algorithm $\mathcal{A}$ of local complexity $t(n)$, we first construct a coloring of $G^{t(n)}$ with $O\left(\Delta^2(G^{t(n)})\right)$ colors[1] using Linial's algorithm from Theorem 2.2. That algorithm needs $O\left(t(n) \cdot \log^* n\right)$ rounds where we multiply by $t(n)$ because one round of communication in $G^{t(n)}$ can be simulated in $t(n)$ rounds in G. Subsequently, we iterate over all colors and simulate the sequential algorithm $\mathcal{A}$ as in Theorem 1.4. That simulation takes additional $O(\Delta^2(G^{t(n)})) = \Delta^{O(t(n))}$ rounds. □

As a corollary of Theorem 2.5, we get that maximal independent set or $\Delta + 1$-coloring can be constructed in $O(\log^* n)$ rounds on bounded-degree graphs.

Corollary 2.6. *Sequential local algorithms with local complexity $O(1)$ (such as algorithms for the maximal independent set or $\Delta{+}1$ coloring) can be simulated with round complexity $O(\log^* n)$ on bounded degree graphs.*

Understanding unique identifiers: Let us now discuss the unique identifiers from the range $[n^{O(1)}]$ in the definition of deterministic algorithms. We will use our understanding of coloring to see that the strength of the model of deterministic algorithms remains the same even if the identifiers come from a much larger range like $[2^n]$ or $[2^{2^n}]$. Moreover, we will understand why identifiers are not needed in the definition of sequential local algorithms.

The following theorem will use an instance of a *fooling argument*, variants of which we will enjoy employing later on. To motivate it, notice that it is a bit awkward that the definition of a *local* algorithm talks about *globally* unique identifiers. We will next "fool" a given deterministic local algorithm solving a local problem by supplying to it a distance coloring (i.e., coloring of the power graph) with $n^{O(1)}$ many colors instead of unique identifiers. The algorithm still has to work since a failure of the algorithm for input distance coloring would imply a failure for input unique identifiers.

Theorem 2.7. *Let $\mathcal{A}$ be a deterministic local algorithm of round complexity $t(n)$ for a local problem Π. Then, given any $s = n^{\omega(1)}$, there is a deterministic local algorithm $\mathcal{A}'$ solving Π in $O(t(n) \cdot \log^*_n(s))$ rounds[2] and assumes that the identifiers are from range $[s]$.*

Similarly, let $\mathcal{A}$ be a sequential local algorithm of local complexity $t(n)$ for a local problem Π in a model of sequential local algorithms where each node has additionally a unique identifier from $[n^{O(1)}]$. Then there is a sequential local algorithm $\mathcal{A}'$ of complexity $O(t(n))$ in the standard model of sequential local algorithms without any identifiers.

Proof. Let $\mathcal{A}$ be a deterministic algorithm of round complexity $t(n)$ solving a local problem Π with local checkability r in the model where the unique identifiers are from $[n^{O(1)}]$

[1]The notation $\Delta(G)$ stands for the maximum degree of G.

[2]Here, $\log^*_n(s)$ returns how many times we need to take the logarithm of s, until its value drops below n, i.e., $\log^*_n(2^n) = 1, \log^*_n(2^{2^n}) = 2$ and so on.

(the constant in the exponent is assumed to be large enough). We construct a new algorithm $\mathcal{A}'$ that works in the less powerful model with identifiers from $[s]$ as follows. We first compute a coloring of the power graph $G^{2(t(n)+r)}$ with $n^{O(1)}$ many colors, then we simulate $\mathcal{A}$ with that coloring as identifiers.

In particular, the coloring is constructed by iterating color reductions of Linial's algorithm of Theorem 2.2. Recall that each color reduction reduces the range of color exponentially, thus after $\log_n^*(s)$ rounds, we reduce the input coloring with colors from $[s]$ to a coloring with colors of size at most $O(\Delta^2(G^{2(t(n)+r)})) = n^{O(1)}$.

Next, we prove that $\mathcal{A}'$ is correct. Assume that $\mathcal{A}'$ fails to solve Π at a node u. Notice that this failure depends only on the $(t(n)+r)$-hop neighborhood of u where the coloring constructed by $\mathcal{A}'$ uses unique colors. In particular, we can extend this coloring of $B(u, t(n)+r)$ to a labeling of every node of the input graph with unique identifiers. The original algorithm $\mathcal{A}$ fails to solve Π at u for these identifiers, a contradiction with $\mathcal{A}$ being correct.

The proof of the second claim in the theorem is very similar and, in fact, easier, since $\Delta+1$-coloring has constant sequential local complexity and thus Linial's color reductions are not needed. □

As a helpful corollary of the second part of Theorem 2.7, we can now see that any deterministic local algorithm $\mathcal{A}$ solving some local problem can be converted into a deterministic sequential algorithm of the same asymptotic local complexity. This was actually not clear until now since our definition of deterministic sequential algorithms in Theorem 1.3 did not contain input unique identifiers.

2.1.2 Lower bound for coloring via round elimination

Finally, let us prove that the $\log^* n$ dependency for constructing maximal independent sets or colorings is necessary.

Theorem 2.8 (Naor [257], Linial [236]). *The local complexity of computing $\Delta+1$ coloring is $\Omega(\log^* n)$, even on graphs that are oriented paths.*

There are several known proofs of this theorem [257, 236]; we will use the proof of Linial [236] framed in the language of a powerful technique known as the *round elimination* (see [288] for an introduction to it). In essence, given a local problem Π, round elimination is an automated technique that defines a problem Π' such that the round complexity of the fastest algorithm for Π' is exactly one round smaller than the round complexity of the fastest algorithm for Π (unless its complexity was already zero). This is very helpful for proving lower bounds: If we start with some problem Π and argue that even after t rounds of round elimination, the problem Π' that we ended up with is not solvable in zero rounds, we infer that the local complexity of Π is at least t.

Preparations for the lower bound: We will prove the lower bound in the extremely simple setup where we are promised that the input graph is an oriented path, i.e., the setup from Section 1.1. The lower bound will be for deterministic algorithms, so each node starts with a unique identifier. We will make a further restriction on the identifiers, we require that the input labeling with identifiers is increasing; that is, if for every oriented edge $e = uv$ going from u to v, we have $ID(u) < ID(v)$.

Next, let us discuss local algorithms. It will be helpful to think about them as functions in the spirit of Theorem 1.2. A subtlety we need to be careful about is that in one step of round elimination, we don't want to directly convert a t-round local algorithm (that sees $2t + 1$ vertices) to a $t - 1$ round algorithm (that sees $2t - 1$ vertices). Instead, we want to convert an algorithm that sees t vertices to an algorithm that sees $t - 1$ vertices. To this end, we define an *edge-centered* $(t + 1/2)$-round algorithm $\mathcal{A}$ to be a local algorithm such that for an edge $e = uv$, the input to $\mathcal{A}$ is the ball $B(uv, t - 1)$ defined as $B(u, t - 1) \cup B(v, t - 1)$. The output of $\mathcal{A}$ is a label for the edge e. For example, 1/2-round local algorithm run on e has access to the two identifiers $ID(u)$ and $ID(v)$ and it maps the two identifiers to a label of e.

One half-round reduction: Here comes the heart of the proof; we will show that any given t-round local algorithm for coloring vertices with k colors can be converted to a $((t - 1/2))$-round algorithm for coloring edges with 2^k colors.

Lemma 2.9. *Assume that for $t > 0$ we are given a node-centered t-round deterministic local algorithm $\mathcal{A}$ that outputs a proper coloring with k colors on oriented paths with increasing unique identifiers. Then, there is an edge-centered $((t - 1/2))$-round algorithm $\mathcal{A}'$ that properly colors the edges with 2^k colors in the same setup. Similarly, edge-centered $(t + 1/2)$-round algorithms can be converted into node-centered t-round algorithms.*

Proof. We will only discuss the conversion of a node-centered algorithm into an edge-centered one, the other case is very similar.

Given a t-round algorithm $\mathcal{A}$, we define a $(t - 1/2)$-round edge-centered algorithm $\mathcal{A}'$ as follows (see Figure 2.2): Given an edge $e = uv$ so that $\mathcal{A}'$ has access to $B(uv, t - 1)$, we let $\mathcal{A}'$ to consider all possible identifiers of the unique node $x \in B(v, t) \setminus B(uv, t - 1)$, i.e., the only node that $\mathcal{A}$ sees but $\mathcal{A}'$ doesn't. For each possible value of the identifier $ID(x)$ of x (we also consider the option that we are at the end of the path and the vertex x does not exist), the algorithm $\mathcal{A}'$ simulates $\mathcal{A}$ and records the color that $\mathcal{A}$ outputs. The final output of $\mathcal{A}'$ is a subset of $[k]$ that contains each color $c \in [k]$ whenever there is an identifier that makes $\mathcal{A}$ output c. We can encode the set with k bits, i.e., we can view this set of original colors as a new color from the range $[2^k]$. This finishes the description of $\mathcal{A}'$.

Our task is to prove that $\mathcal{A}'$ returns a proper coloring of edges. Here is what this question reduces to: Consider any increasing labeling of the oriented path with identifiers and consider any three consecutive nodes u, v, w (see Figure 2.3). We again let x be the

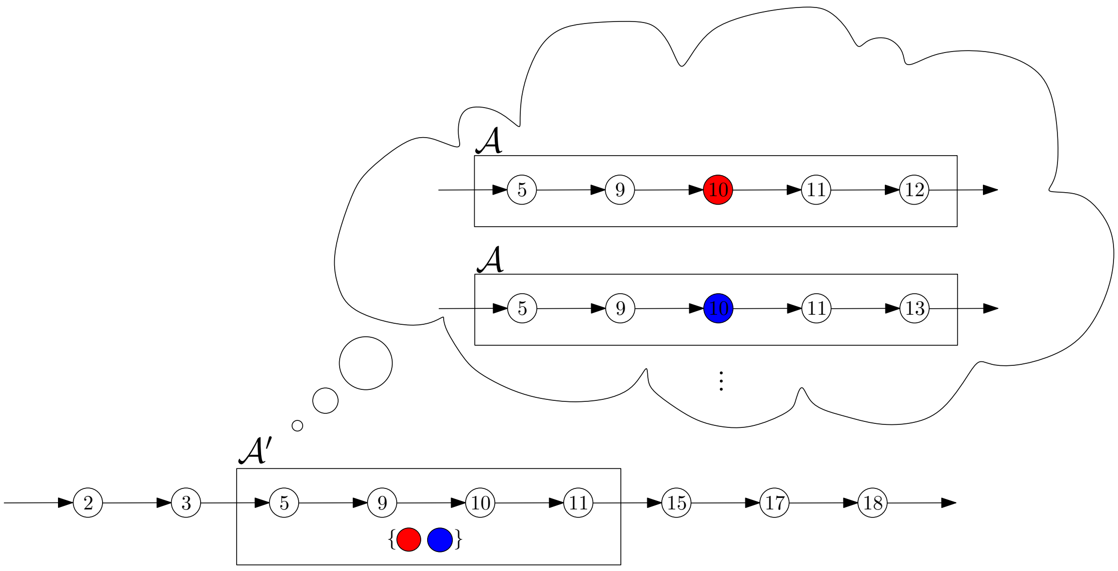

Figure 2.2: The definition of the 1.5-round algorithm $\mathcal{A}'$ derived from a 2-round algorithm $\mathcal{A}$: The algorithm $\mathcal{A}'$ considers all compatible one-node extensions of its neighborhood containing 4 nodes to 5-node-neighborhoods. It considers all possible answers that $\mathcal{A}$ returns for those neighborhoods (the picture shows that the red and the blue color are two of those possible answers). The color returned by $\mathcal{A}'$ is simply the set of all colors that $\mathcal{A}$ returns for some extension (in the picture, it is the set containing the red and the blue color).

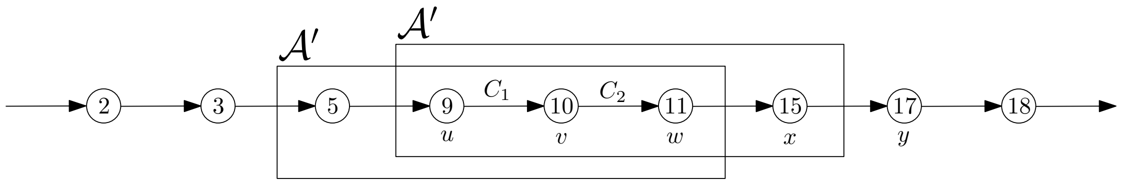

Figure 2.3: The situation from the proof of correctness of $\mathcal{A}'$: We need to argue that the two colors C_1, C_2 are different. To do so, we define c to be the color that $\mathcal{A}$ outputs on the input $(5, 9, 10, 11, 15)$. By definition, we have $c \in C_1$. On the other hand, if $c \in C_2$, we get an existence of an identifier $ID'(y)$ such that $\mathcal{A}$ returns c on the input $(9, 10, 11, 15, ID'(y))$. But then we consider the input $(5, 9, 10, 11, 15, ID'(y))$ and notice that $\mathcal{A}$ returns colors two consecutive vertices with the same color c, a contradiction with its correctness.

unique node in $B(v,t) \setminus B(uv, t-1)$ and y the subsequent neighbor of x. Consider the new colors $C_1, C_2 \in [2^k]$ that $\mathcal{A}'$ outputs on the two edges uv and vw. We need to prove that the two colors C_1, C_2 are different. We do that by focusing on the original color $c \in [k]$ that $\mathcal{A}$ outputs at v.

On one hand, notice that $c \in C_1$ because the edge-centered algorithm $\mathcal{A}'$ run at uv considers the actual identifier $ID(x)$ as a possibility and thus includes c in C_1.

On the other hand, assume for contradiction that $c \in C_2$. That would imply that there is a certain identifier $ID'(y)$ that makes the output of $\mathcal{A}$ at w to be the color c. But now consider changing the valid increasing labeling of identifiers that we started with by letting the identifier of y be equal to $ID'(y)$. Let's look at all the nodes in $B(vw, t)$. The identifiers on them are still a valid increasing identifier sequence.[3] But after extending $B(vw, t)$ and its identifiers to a path on n vertices labeled with increasing identifiers, $\mathcal{A}$ fails to solve k-coloring on that graphs since it outputs the same color c at both v and w, a contradiction. □

We can now prove Theorem 2.8 as follows: Assuming an $t = o(\log^* n)$-round algorithm for 3-coloring of oriented paths, we apply Theorem 2.9 $2t$ times until we end up with a 0-round algorithm $\mathcal{A}_0$ that colors oriented paths with less than n colors. But such $\mathcal{A}_0$ is simply a function mapping an input identifier from range $[n^{O(1)}]$ to a color in the smaller range $[n]$. Using the pigeonhole principle, we can find two identifiers that $\mathcal{A}_0$ maps to the same color and argue that if these two identifiers happen to be present at two neighboring nodes, $\mathcal{A}_0$ fails to output proper coloring. [4]

[3]This is the place in the argument where we need increasing and not just unique identifiers: With unique identifiers, the leftmost and rightmost node of $B(vw, t)$ could now have the same identifier.

[4]Round elimination may used both to prove lower bounds *and* to construct algorithms. In particular, it can be used to derive that the round complexity of 3-coloring paths in $\frac{1}{2}\log^* n \pm O(1)$ [278].

2.2 The Lovász Local Lemma Regime

We will next discuss Lovász local lemma, a very expressible local problem closely related to the third regime of problems from Theorem 2.1.

Definition of Lovász local lemma: Lovász local lemma is the following very general local problem [5]. The problem is formally defined for bipartite graphs in which nodes of one part are labeled as *random variables* and the nodes of the other part are labeled as *bad events*. We denote the maximum degree of random variable nodes as $\Delta_{r.v.}$ and the maximum degree of bad event nodes as $\Delta_{b.e.}$. Each node u labeled as a random variable is additionally labeled with a probability space Ω_u. To simplify discussions, we will without loss of generality assume that each probability space is an infinite list of random bits (i.e., it is the uniform distribution on $[0, 1]$). Next, each node v labeled as a bad event that neighbors with nodes $u_1, \dots, u_d$ with $d \leq \Delta_{b.e.}$ is additionally labeled with an event $\mathcal{E}_v$ on the space $\prod_{i=1}^d \Omega_{u_i}$.

Often, it is useful to work only with a graph induced by bad-event nodes where two bad-event nodes are connected if they share a common random variable. In that case, we will talk about the *dependency-graph* formulation of Lovász local lemma and use Δ to denote its maximum degree. On the other hand, the setup with a bipartite graph with both bad-event nodes and random-variable nodes will be denoted as the *variable-event-graph*.

The crucial ingredient to the Lovász local lemma is a requirement on the bad events that, roughly speaking, says that we can use the union bound in the dependency-graph neighborhood of every bad event $\mathcal{E}_v$ and conclude that with positive probability, neither $\mathcal{E}_v$, nor its neighboring bad events occur. Namely, in the *tight* version of Lovász local lemma, we are given a promise [6] that each bad event $\mathcal{E}_v$ has probability at most p where p is defined by the following *Lovász local lemma criterion*:

$$p \cdot (\Delta + 1) \leq 1/\mathrm{e}. \tag{2.1}$$

A foundational result of Erdős and Lovász [125] is that even in the tight formulation, it is always possible to solve any instance of the local lemma problem, by which we mean that one can instantiate random variables so that no bad event occurs.

A *C-relaxed* (or just polynomially-relaxed) version of Lovász local lemma, which is a bit more relevant to our applications, only requires that

$$p \cdot \Delta^C \leq 1 \tag{2.2}$$

This section will show both fast algorithms and lower bounds for the polynomially-relaxed Lovász local lemma.

[5]The Lovász-local-lemma problem does not quite satisfy the requirements for the local problem as we defined it in Theorem 1.1, but, morally speaking, it can be seen as such.

[6]Our definition of a local problem from Theorem 1.1 does not allow "promises" on input labels. To turn local lemma into a proper instance of a local problem, we can postulate that if the promise is not satisfied, the local solution around the node is not bound by any constraints.

Lovász local lemma is a very versatile and expressible local problem. Its deterministic round complexity on bounded degree graphs is $\Theta(\operatorname{poly}\log n)$*, and its randomized complexity* $\Theta(\operatorname{poly}\log\log n)$.

2.2.1 Fast Algorithms for the Local Lemma

We first discuss how to solve any instance of the local lemma with a fast deterministic local algorithm. To this end, we will start with a randomized algorithm that we later derandomize by Theorem 1.8. In a breakthrough result in the area of constructive algorithms for the local lemma, Moser and Tardos [256] presented an algorithm for it in the tight formulation. Moreover, they presented a parallel variant of their algorithm that can be interpreted as a local algorithm with round complexity $O(\log^2 n)$. This complexity was later improved by Chung et al. [102] to $O(\log n)$ rounds on bounded degree graphs. Plugging their result to Theorem 1.8, we obtain the following theorem.

Theorem 2.10. *The deterministic round complexity of solving any instance of the tight version of Lovász local lemma on bounded degree graphs is* $\widetilde{O}\left(\log^4(n)\right)$.

Even faster randomized algorithm: Next, we will show that there is an even exponentially faster randomized algorithm for the relaxed version of the local lemma. We will discuss the algorithm of Fischer and Ghaffari [133] with round complexity $\operatorname{poly}\log\log n$ which is based on the technique of shattering, a successful technique (cf. Section 1.6) that goes back to the work on algorithmic local lemma by Beck [54].

The idea is as follows. Our algorithm will have two phases. In the first phase, we use a sequential local algorithm with constant complexity to do the following. Given an instance of the local lemma, the algorithm fixes *most* of the random variables in such a way that *most* bad events are satisfied (i.e., all random variables relevant for that bad event are fixed and it does not occur). The fraction of fixed random variables and satisfied events is in particular $1-1/\Delta^{O(1)}$. The price for this outcome is that remaining, unfixed, bad events have their probability slightly increased from $1/\Delta^C$ to $1/\Delta^{C'}$ for some $C' < C$.

Fortunately, we have an additional guarantee on the unfixed bad events. Due to the very small locality of the algorithm in the first phase, we can use an independence-like argument to show that the size of the connected components of unfixed bad events is $O(\Delta^{O(1)}\log n)$, with high probability (see Figure 2.4). We also say that the graph *shatters* into small components. We can thus run the best deterministic algorithm from Theorem 2.10 as the second phase of the algorithm, to solve the remaining instance of the C'-relaxed local lemma. This second phase takes $\operatorname{poly}\log(\Delta\log n)$ rounds which is also the round complexity of the overall algorithm.[7] We will now make the above discussion formal.

[7]Our example algorithm from Section 1.1 can be seen as a simple shattering algorithm.

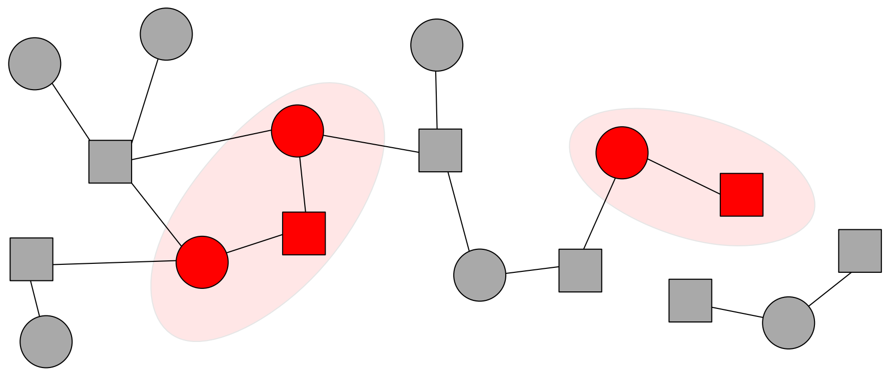

Figure 2.4: This picture shows an example variable-event graph corresponding to an instance of Lovász local lemma; the circles represent bad events and squares represent random variables. The picture shows the situation after the first phase of Fischer-Ghaffari algorithm (Theorem 2.11). Most random variables are set to a fixed value (grey squares). A small proportion of the random variables remains unset (red squares) and the bad events neighboring with an unset random variable (red circles) form connected components of diameter $O(\log n)$.

Theorem 2.11 (First phase of Fischer and Ghaffari [133])**.** *There is a randomized sequential local algorithm $\mathcal{A}$ of constant local complexity that gets as input a variable-event graph of an instance of C-relaxed Lovász local lemma for large enough C. The algorithm fixes some bits of each random variable to concrete values so that conditioned on those fixed bits, we have the following two properties:*

1. *Each bad event has probability at most $1/(3\Delta)$.*
2. *Up to $1/\operatorname{poly}(n)$ error probability, the residual dependency graph induced by bad events with non-zero probability has connected components of size $O(\Delta^3 \log n)$.*

Proof. The algorithm $\mathcal{A}$ is defined as follows. We iterate over the random variables and for each random-variable node u with neighboring bad-event nodes $v_1, \dots, v_d$, we perform the following process. We sample the random bits of Ω_u one by one. While we do that, we consider the probabilities of neighboring bad events $P(\mathcal{E}_i)$ for each $1 \le i \le d$. The first time it happens that for some i we have $P(\mathcal{E}_i) > 1/(6\Delta)$, i.e., the bad-event probability crosses the *dangerous threshold* of $1/(6\Delta)$, we stop sampling and do the following.

We label u, together with all other random variables neighboring v_i as *frozen* (if more v_i's jumped over the dangerous threshold at the same time, we do this to all of them). We never sample bits of frozen random variables in the future. In particular, we stop sampling bits of u and continue with the next unfrozen random variable in the arbitrary order of our sequential local algorithm. This finishes the description of the algorithm.

To see that each bad event has probability at most $1/(3\Delta)$ during any point throughout the algorithm, we notice that if an event $\mathcal{E}$ of probability $P(\mathcal{E}) = p$ depends on a single bit of randomness b, we have, by Markov's inequality, that $P(\mathcal{E}|b = 0), P(\mathcal{E}|b = 1) \leq 2p$. That is, the bad event probability in our process never increases multiplicatively by a factor larger than 2, which together with the definition of the dangerous threshold implies the desired bound.

To understand the size of connected components in the residual dependency graph (i.e., components of bad-event nodes of non-zero probability after we set the random bits), consider any fixed set S of bad-event nodes with the following two properties:

1. (*independence*) no two nodes $v_1, v_2 \in S$ are neighboring in the dependency graph G,
2. (*connectivity*) S is connected in G^6.

We will next prove that with high probability, no such set S of size $\Omega(\log n)$ survives to the residual graph.

First, we use the independence property to prove that the probability that all nodes in a fixed set S crossed the dangerous threshold during our process is exponentially small in the number of nodes $|S|$. Let us contemplate the behavior of the algorithm $\mathcal{A}$ with respect to any bad-event node $v \in S$ and the random variables $u_1, \ldots, u_d$ neighboring v in the variable-event graph. We note that if bits $b_1, \ldots, b_{k-1}$ were already sampled and b_k is sampled next, we have $E_{b_k}[P(\mathcal{E}_v|b_1, \ldots, b_k)] = P(\mathcal{E}_v|b_1, \ldots, b_{k-1})$. Hence, viewing $P(\mathcal{E}_v)$ as a random variable that depends on bits sampled from $\Omega_u, u \in V(G)$, we conclude that its expectation is at most $1/\Delta^C$ and we can thus use Markov's inequality to conclude that the probability of $P(\mathcal{E}_v)$ crossing the dangerous threshold is at most $\frac{6\Delta}{\Delta^C} = 6/\Delta^{C-1}$. Moreover, we notice that if we first set the randomness of nodes in $V' = V(G) \setminus (u_1 \cup \cdots \cup u_d)$ to whatever values, we can still make above argument and conclude that for all ways of setting $\Omega_u = \omega_u$ for all $u \in V'$ we have

$$P(\mathcal{E}_v | \forall u \in V' : \Omega_u = \omega_u) \leq 6/\Delta^{C-1}.$$

Since we assumed that S is an independent set of bad-event nodes in the dependency graph, we can thus inductively prove that the probability of all nodes in S crossing the dangerous threshold is at most $\left(\frac{6}{\Delta^{C-1}}\right)^{|S|}$.

We next count the number of possible sets S of size t that satisfy the connectivity property: Each such S can be specified by fixing any node $u \in S$, and then specifying how one can walk for $2(|S| - 1)$ steps in G^6 so that the walk defines a spanning tree of $G^6[S]$. Hence, the number of sets S of size t is at most $n \cdot (2\Delta^6)^{2(t-1)}$. We can thus upper bound the existence of some connected surviving set S of size t as

$$n \cdot (2\Delta)^{12t} \cdot \left(\frac{6}{\Delta^{C-1}}\right)^t.$$

Choosing $C = O(1)$ and $t = O(\log n)$ large enough, we conclude that the size of this expression is at most $1/n^{O(1)}$.

Finally, for $t = O(\log n)$ from the above argument, consider any connected subset S of G of size at least $(2\Delta)^3 \cdot t$ and assume that S survived to the residual dependency graph. Then, we can find its subset $S' \subseteq S$ of size at least $|S'| \geq t$ where S' is independent in G^3, yet S' is connected in G^4. To see this, consider a greedy algorithm that starts with $S' = \{u\}$ for arbitrary $u \in S$, iterates through vertices of S and while we still can add at least one node $v \in S$ to S' and keep the independence property, we choose any v with distance 4 to S' and add it to S'. One can see that the final set S' has to be connected in G^4. Also, $|S'| \geq t$ because adding a vertex to S' disqualifies at most $(2\Delta^3)$ vertices to be added to S' in the future.

If all nodes of S survived to the residual dependency graph, it means that every node in S' has to have a neighboring node that crossed the dangerous threshold. The set S' thus gives rise to a set S'' of size $|S''| = |S'| \geq t$ of nodes that all crossed the dangerous threshold. Moreover, by S' being independent in G^3, we conclude that S'' is independent in G. Since S' was connected in G^4, S'' is connected in G^6. The set S'' thus satisfies the requirements of a set that, as we have proven, does not occur in the residual graph with high probability and we can thus conclude the same for the original connected set S. $\square$

Putting Theorem 2.11 (simulated as a distributed algorithm by Theorem 2.5) together with Theorem 2.10, we conclude that the following result holds.

Theorem 2.12 (Fischer and Ghaffari [133]). *There exists a constant C such that the randomized round complexity of any instance of the C-relaxed Lovász local lemma on bounded-degree graphs is $\widetilde{O}\left(\log^4 \log n\right)$.*

The complexity can be improved to $O(\Delta/\log\Delta + \min(\log^4(\Delta \log n), \log^5 \log n))$ by combining results of Fischer and Ghaffari [133] with the recent result of Davies [116]. It is unknown whether there is an algorithms with better dependency on Δ.

Problem 2.13. *Is there a randomized local algorithm for relaxed Lovász local lemma with round complexity $O(\text{poly}\log\Delta + \text{poly}\log\log n)$?*

Conjecture of Chang and Pettie [88]: We note that Chang and Pettie [88, Conjecture 1] conjecture that any instance of the relaxed local lemma can be solved with randomized $O(\log\log n)$ complexity.

Problem 2.14 (Chang-Pettie Conjecture). *Is it true that the randomized round complexity of C-relaxed Lovász Local Lemma for some large enough C is $\Theta(\log\log n)$ on bounded degree graphs?*

We will see later in Section 2.3.2 that the randomized round complexity $O(\log\log n)$ implies the deterministic round complexity of $O(\log n)$ via Theorem 2.21.

Self-contained algorithm: A bit unfortunate property of Theorem 2.12 is that it relies on the randomized entropy-compression algorithms from Moser and Tardos [256] or Chung et al. [102]. However, we can prove Theorem 2.12 also in a self-contained way: First, we observe that after the first phase of the Fischer-Ghaffari algorithm finished, each surviving bad event simply collects the whole connected component of the residual graph; then we solve the problem in each residual component by applying the standard, existential, Lovász local lemma. This new randomized algorithm has complexity $O(\log n)$ on bounded degree graphs since this is the maximum diameter of residual components, with high probability. We can derandomize this algorithm using Theorem 1.8 and use the resulting deterministic algorithm instead of the algorithm of Chung et al. [102] in the second phase of our shattering algorithm. This way, we get a simpler and more self-contained algorithm. Its downside is a worse value of C and a worse dependence of round complexity on Δ.

2.2.2 Lower Bound via Round Elimination

Next, we will show that the deterministic local complexity of solving a certain specific instance of Lovász local lemma is $\Omega(\log n)$. Later, we will prove in Theorem 2.21 that this also implies a randomized lower bound of $\Omega(\log\log n)$, thus showing that the round complexity of Fischer-Ghaffari algorithm from Theorem 2.12 is close to tight.

The problem we choose for the lower bound is *sinkless orientation*. We will define the problem only on trees of degree at most Δ where it is already hard. The task is to orient all the edges of the input tree so that no node is a *sink*, which is defined as a node such that all Δ neighboring edges point towards it (nodes of degree less than Δ are not constrained in any way).

Sinkless orientation can be seen as a specific instance of the local lemma: orienting each edge randomly corresponds to a random variable at each edge. Then, a vertex becoming a sink corresponds to a bad event of probability $2^{-\Delta}$. For large enough Δ, this is much smaller than the polynomial criterion $1/\Delta^C$ from (2.2) which makes this problem a valid instance of the local lemma. We will next prove the following theorem.

Theorem 2.15 (Brandt et al. [73]). *For any constant Δ, the sinkless orientation problem has deterministic round complexity $\Omega(\log n)$ on the class of trees of degree at most Δ.*

Preparations: As in Theorem 2.8, we will want to replace the uniqueness of identifiers with local constraints that imply that they are unique. In the proof of Theorem 2.8, we worked with identifiers that were monotonically increasing (which implied they were unique), this time we will work with identifiers that are consistent with a so-called *ID graph* [222, 40, 83]. [8]

Definition 2.16 (ID graph). *Given a parameter Δ, an ID graph H is a graph on a set $[n]$ that we associate with unique identifiers. Every edge of the graph is colored with one*

[8]The usual usage of round elimination is for randomized algorithms instead of using the ID graph, but this would require a longer setup and more calculations.

of Δ many colors and we write H_i for the graph induced by the i-th color. We require that:

1. *The girth of H, i.e., the length of the shortest cycle in H, is at least $\gamma \log_\Delta n$ for some fixed $\gamma > 0$.*

2. *Each independent set of each H_i has less than n/Δ vertices.*

Such a graph exists, which can be proven using the same argument as how one proves that high-girth high-chromatic graphs exist [83, 222].

We will fix any ID graph and work in a model *relative* to that ID graph. Here is what that means. First, we will assume that the input graph G is always a tree of degree at most Δ such that its edges are, moreover, properly Δ-colored on the input (i.e., each vertex is incident to edges of different colors). Additionally, we require that if two nodes u, v neighbor in G with an edge of color i, then their identifiers $ID(u), ID(v)$ neighbor in H_i.

We notice that this is a local constraint on input identifiers that does not imply they are unique. However, notice that whenever two vertices $u, v \in V(G)$ have the same identifier, we can consider the path between u and v in G and how it maps to a walk in H that starts in $ID(u)$ and finishes in $ID(v) = ID(u)$. The proper edge-coloring of G implies that the walk never goes from $x \in V(H)$ to $y \in V(H)$ and then back to x in the subsequent step. This implies that the walk contains a cycle, thus the distance of u and v is at least as large as the girth of H. That is, the identifiers are unique up to a large distance which is pretty much the same as them being unique (cf. Theorem 2.7).

Similarly to the lower bound of Theorem 2.8, we will work with node-centered and edge-centered algorithms. For node-centered algorithms, solving sinkless orientation means that the algorithm outputs one of Δ many input edge colors at each node. Outputting a color i means that u decides that the edge of the color i goes outwards from u. The local constraint on this output vertex-coloring is that no two neighboring nodes should select the same edge going in between. We will call this variant of the problem *edge-grabbing*. On the other hand, edge-centered algorithms will solve the sinkless orientation as we described the problem: Each edge simply outputs how it is oriented and we have a local constraint at each vertex, requiring that it has at least one outgoing edge.

Round elimination: We proceed with performing the actual round elimination. Notice that the spirit of the following proof is very similar to the proof of Theorem 2.9.

Lemma 2.17. *Assume that we are given a node-centered deterministic t-round local algorithm $\mathcal{A}$ of round complexity at most $t \leq \log_\Delta n - 1$ that solves the edge-grabbing problem on trees of degree at most Δ relative to some fixed ID graph H. Then, there is a $t - 1/2$-round edge-centered deterministic local algorithm $\mathcal{A}'$ that solves sinkless orientation in the same setup.*

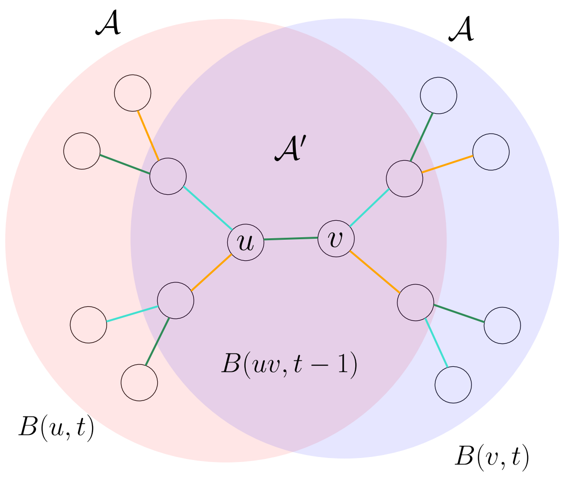

Figure 2.5: The picture shows the definition of the algorithm $\mathcal{A}'$ for $t = 2$. Notice that our setup is a large Δ-regular tree with edges colored with Δ colors. The algorithm $\mathcal{A}'$ has access only to the identifiers in $B(uv, t-1)$ (the intersection of the red and the blue ball). To decide on the orientation of the edge uv, the algorithm considers all possible identifier extensions of $B(uv, t-1)$ to $B(u, t)$ (the red ball), and if at least one such extension makes $\mathcal{A}$ run at u grab the edge uv, $\mathcal{A}'$ orients that edge towards v. We analogously orient this edge towards u if at least one extension of $B(uv, t-1)$ to $B(v, t)$ (the blue ball) makes $\mathcal{A}$ run at v grab the edge uv.
We notice that it cannot happen that $\mathcal{A}'$ wants to orient the edge uv in both directions: that would imply the existence of an identifier-labeling of $B(uv, t)$ on which $\mathcal{A}$ is incorrect.

Similarly, $t+1/2$-round edge-centered algorithm for sinkless orientation implies a t-round node-centered algorithm for edge-grabbing.

Proof. We prove just the first part of the statement, the proof of the second part is similar, and doing it is a good exercise.

We start with any node-centered algorithm $\mathcal{A}$ with round complexity t that solves the edge-grabbing problem. We define the edge-centered algorithm $\mathcal{A}'$ of round complexity $t - 1/2$ as follows. For an edge $e = uv$, the algorithm first considers all the possible extensions of the (known) ball $B(uv, t-1)$ to the ball $B(u, t)$ that is known to $\mathcal{A}$ when it is run on u. By an extension, we mean first how the graph looks like (e.g., maybe some vertices on the boundary of $B(uv, t-1)$ turn out to be leaves), and second, what the identifiers are (they have to be consistent with H). We consider all valid extensions and if at least one of them leads to $\mathcal{A}$ grabbing the edge uv from u, then $\mathcal{A}'$ orients the edge uv as going from u to v. After this is done, the algorithm makes an analogous reasoning for v; again, whenever at least one extension of $B(uv, t-1)$ to $B(v, t)$ decides to grab the edge uv, $\mathcal{A}'$ orients it from v to u. If no vertex ever decides to grab uv, $\mathcal{A}'$ decides to orient it arbitrarily. This finishes the description of $\mathcal{A}'$ (see Figure 2.5).

To make sure that $\mathcal{A}'$ is well-defined, we need to prove that it never happens that there is an extension of $B(uv, t-1)$ to both $B(u,t)$ and $B(v,t)$ such that $\mathcal{A}$ run on $B(u,t)$ decides to grab the edge uv and run on $B(v,t)$, it decides to grab the edge vu. Notice that putting the two extensions $B(u,t), B(v,t)$ together, the graph $B(uv,t)$ has at most n nodes and its identifiers respect H.[9] We observe that on this labeled graph, $\mathcal{A}$ would fail to solve edge-grabbing, a contradiction with its correctness. Thus, $\mathcal{A}'$ is well-defined.

Moreover, the algorithm $\mathcal{A}'$ solves sinkless orientation: For any node u of full degree Δ, the original algorithm $\mathcal{A}$ decided to grab a certain edge uv. When we run $\mathcal{A}'$ on uv, $\mathcal{A}'$ will by definition orient this edge from u to v, thus u cannot be a sink. $\square$

Finishing the proof: Let us finish the proof of Theorem 2.15.

Proof of Theorem 2.15. Consider an input graph G which is a branching tree with $\varepsilon \log n$ layers and every non-leaf vertex has degree Δ. Note that for small enough $\varepsilon > 0$, the girth property of the ID graph implies that any labeling of G with identifiers from $[n]$ that respects a fixed ID graph H has unique identifiers. On the other hand, G has $O\left(\Delta^{\varepsilon \log n}\right)$ nodes, so the range from which the identifiers are coming is $|V(G)|^{O(1)}$. Therefore, a deterministic local algorithm for sinkless orientation on G implies a local algorithm for that problem on G with identifiers consistent with H.

But Theorem 2.17 shows that any $o(\log n)$-round algorithm that works relative to H can be sped up to 0 round complexity. A 0-round algorithm $\mathcal{A}_0$ is simply a function that maps an input identifier to one of Δ many colors, i.e., which edge the vertex decides to grab. We can thus think of $\mathcal{A}_0$ as a coloring of H with Δ many colors. But notice that for any such vertex coloring, we can consider the largest color class i that has at least n/Δ colors. Then, we use the independence property of ID graphs from Theorem 2.16 to infer that the set of vertices of color i cannot be independent in H_i, i.e., there is an edge uv in H where both u and v, as well as the edge uv are colored by the color i. We thus found two vertices that may be neighboring in the input graph G and the algorithm $\mathcal{A}_0$ decides to grab the edge connecting them from both endpoints of that edge, a contradiction with $\mathcal{A}_0$ being correct. $\square$

Sinkless orientation as the "simplest hard problem": Notice that if we formulate sinkless orientation as an instance of Lovász local lemma, the bad event probability is equal to $2^{-\Delta}$, that is, the bad event probability is exponentially small compared to the polynomial guarantee in the relaxed criterion of (2.2). It turns out that this is the threshold where an instance of the local lemma is still "hard".

Theorem 2.18 ([73, 78, 79])**.** *On one hand, there is an instance of Lovász local lemma (namely sinkless orientation) with the criterion $p \cdot 2^{\Delta} \leq 1$ that has deterministic round complexity $\Omega(\log n)$. On the other hand, any instance of Lovász local lemma with the criterion $p \cdot 2^{\Delta} < 1$ has deterministic round complexity $O(\log^* n)$.*

[9] This part of the argument needs to work with the ID graph instead of unique identifiers.

A similar threshold phenomenon holds also if we work in the variable-event graph.

Theorem 2.19 ([133, Theorem 3.5], [61, Corollary 1.8])**.** *On one hand, there is an instance of Lovász local lemma*[10] *with the criterion $p \cdot \Delta_{r.v.}^{\Delta_{b.e.}} \le 1$ has deterministic round complexity $\Omega(\log n)$. On the other hand, any instance of Lovász local lemma with the criterion $p \cdot \Delta_{r.v.}^{\Delta_{b.e.}} < 1$ has deterministic round complexity $O(\log^* n)$.*

2.3 Speedups and Slowdowns

The following section covers speedup and slowdown theorems which are at the heart of why we understand that there are sharp thresholds in the local complexities in Theorem 2.1.

The elegant idea behind speedups and slowdowns is that we can simply "lie" to algorithms about the size of the input graph, an idea closely related to the fooling argument we have already seen in the proof of Theorem 2.7. To understand this technique, it may be useful to briefly recall that in Theorem 1.2, we defined a local algorithm as a function that takes two inputs. Firstly, it is n, the size of the input graph, and secondly, it is a $t(n)$-hop neighborhood of a vertex that is additionally labeled by unique identifiers or random strings. We will use the notation $\mathcal{A}_n$ to denote the algorithm $\mathcal{A}$ when the first input is n, i.e., we will view $\mathcal{A}$ as a sequence of functions $\mathcal{A}_1, \mathcal{A}_2, \dots$ In this section, we will contemplate what happens if we run $\mathcal{A}_n$ on a graph of size $n' \neq n$. If $n' > n$, we are "speeding up" $\mathcal{A}$, while if $n' < n$, we are "slowing it down".

2.3.1 Slowdowns

Let's first see why slowdowns may be useful. As a first application, let us recall that we defined deterministic and randomized round complexities in Theorem 1.1 by requiring unique identifiers from the range $[n^{O(1)}]$ or error probability at most $1/n^{O(1)}$. We will next see that for algorithms with sufficiently small round complexity, we can replace $n^{O(1)}$ by n in the definition without changing its strength. For deterministic algorithms, this complements Theorem 2.7 that shows how to replace very large identifiers with polynomially-sized ones. This follows by plugging in $f(n) = n^{O(1)}$ into the following slowdown theorem.

Theorem 2.20 (Chang, Kopelowitz, and Pettie [94])**.** *Let f be any increasing function with $f(n) \ge n$, let Π be any local problem, and let us use $t_{f(n)}(n)$ to denote the deterministic (randomized) round complexity of solving Π if the input identifiers are from the range $f(n)$ (the error probability is required to be $1/f(n)$, respectively). Then,*

$$t_n(n) \le t_{f(n)}(n) \le t_n(f(n)).$$

[10] The instance is sinkless orientation on trees where one color class has degree $\Delta_{r.v.}$ and the other has degree $\Delta_{b.e.}$. One formulates it as an instance of the local lemma by letting each vertex of one color class grab a random outgoing edge. The proof that this variant of sinkless orientation is still hard seems to be missing in the literature.

The theorem holds for local complexities defined with respect to any subclass of graphs closed on adding isolated vertices. [11]

Proof. We will prove just the deterministic version of the theorem. Notice that $t_n(n) \leq t_{f(n)}(n)$ follows directly from our assumption $f(n) \geq n$ and the definition: any algorithm expecting identifiers from the range $[f(n)]$ certainly works if they happen to be from the smaller range $[n]$.

Next, consider any deterministic algorithm $\mathcal{A}$ that solves Π if the identifiers are from $[n]$ in round complexity $t(n)$. Our goal is to turn it into an algorithm $\mathcal{A}'$ with round complexity $t'(n)$ that works if the identifiers are from $[f(n)]$. Think of $\mathcal{A}$ as a sequence $\mathcal{A}_1, \mathcal{A}_2, \dots$ for each $n \in \mathbb{N}$. We define $\mathcal{A}'$ by setting for each n that $\mathcal{A}'_n := \mathcal{A}_{f(n)}$. That is, we "lie" to the algorithm $\mathcal{A}$, telling it that the size of the graph is larger (namely $f(n)$) than what it actually is (namely n). Since the round complexity of $\mathcal{A}$ on $f(n)$-sized instances is $t(f(n))$, for the complexity of $\mathcal{A}'$ on n-sized instances we have $t'(n) = t(f(n))$.

Moreover, since $\mathcal{A}_{f(n)}$ assumes that the unique identifiers are from $[f(n)]$, $\mathcal{A}'_n$ also assumes that the unique identifiers from $[f(n)]$, making it a well-defined algorithm for the definition of local algorithm where identifiers are supposed to be from $[f(n)]$ on instances of size n.

Finally, we claim that $\mathcal{A}'$ is a correct algorithm. To see this, suppose that $\mathcal{A}'_n$ fails to solve Π on some graph G with n vertices labeled with unique identifiers from $[f(n)]$. Then, we go back to $\mathcal{A}$ and run it on a graph G' defined as G together with $f(n) - n$ additional isolated vertices, all labeled with unique identifiers from $[f(n)]$. We notice that since Π is a local problem, a failure in G implies a failure in G', and we get a contradiction with $\mathcal{A}$ being correct. $\square$

As an example of a non-local problem where the range of identifiers matters, consider the leader-election problem[12] where exactly one node of the input graph is to be selected. If the identifiers are from $[n]$, we can simply select the node with identifier 1. Otherwise, there is no local algorithm for it on the empty graph.

A similar argument to the proof of Theorem 2.20 can be used to prove another sleep-well-at-night result: any local algorithm $\mathcal{A}$ with round complexity $t(n)$ solving some local problem can be turned into an algorithm $\mathcal{A}'$ of round complexity $t'(n) \leq t(n)$ which is a non-decreasing function of n.

The randomized version of Theorem 2.20 turns out to be particularly interesting. Chang et al. [94] used it to show that *any* randomized algorithm can be derandomized, if we appropriately slow down its complexity. [13]

[11] Looking at the proof, it is hard to come up with a reasonable class of graphs where the theorem does *not* apply.

[12] This is a fundamental problem in the broader area of distributed computing. The maximal independent set problem can be seen as a local variant of this problem.

[13] Their technique is similar to Adelman's theorem, i.e., BPP $\subseteq$ P/poly. See [11, Theorem 7.14].

Theorem 2.21 (Chang et al. [94])**.** *If a local problem Π has randomized round complexity $t(n)$, its deterministic round complexity is $t\left(2^{O(n^2)}\right)$.*

The theorem holds for any class of graphs closed on adding isolated vertices.

Proof. Let $\mathcal{A}$ be a randomized local algorithm with error $1/n^{O(1)}$ and round complexity $t(n)$ solving Π. We start by applying Theorem 2.20 with $f(n) = 2^{O(n^2)}$ to slow down $\mathcal{A}$ to complexity $t\left(2^{O(n^2)}\right)$ while pushing the error probability down to $2^{-\Omega(n^2)}$. We still call this new algorithm $\mathcal{A}$.

The deterministic algorithm $\mathcal{A}'$ for Π will work as follows. Let C be such that the unique identifiers are coming from $[n^C]$. We will use a certain function h (a hash function that we soon specify) that maps each identifier from $[n^C]$ to an infinite bit string. We define $\mathcal{A}'$ as follows: It first uses h to map each input identifier to an infinite bit string, and next it simulates $\mathcal{A}$ using each bit string as the string of random bits.

To construct the function h that makes $\mathcal{A}'$ correct, we simply choose a random one and notice that on any concrete graph, $\mathcal{A}'$ fails to solve Π with probability at most $2^{-\Omega(n^2)}$ with the randomness over the choice of h. This is because on any concrete graph, $\mathcal{A}'$ with random h is equivalent to running $\mathcal{A}$. But notice that the number of distinct graphs on n vertices labeled with polynomial identifiers is $2^{O(n^2)} \cdot (n^C)^n = 2^{O(n^2)}$. Hence, we can union-bound over all of them and still get a positive probability that random h works. We thus conclude that for some h, $\mathcal{A}'$ is a valid deterministic algorithm for Π, as needed. $\square$

As a nice corollary, we are getting a very precise understanding of what is happening for the randomized round complexity of Lovász local lemma: On one hand, we have seen a randomized algorithm with round complexity $\operatorname{poly}\log\log(n)$ (Theorem 2.12) that used a deterministic algorithm with round complexity $\operatorname{poly}\log(n)$ (Theorem 2.10) as a subroutine. But we can now see that *any* $\operatorname{poly}\log\log(n)$ round randomized algorithm would imply a $\operatorname{poly}\log(n)$-round deterministic one via Theorem 2.21. A similar observation applies not just for the local lemma, but for many other problems with randomized algorithms based on shattering (see Section 1.6). This explains why in local complexity deterministic algorithms are a big deal – not only they are used as subroutines of randomized algorithms, but we in fact understand that to make progress in the randomized world, progress in the deterministic world is necessary.

2.3.2 Speedups

While slowdowns are very useful, the real fun starts when we try to speed up algorithms by lying to them that the size of the graph is *smaller* than it actually is. In fact, what can possibly go wrong if we tell an algorithm that the size of the graph is constant, i.e., we claim that $n = O(1)$? As it turns out, not much! One potential problem is that the algorithm may find out that its $t(n)$-hop neighborhood contains more than n nodes and "crash". The other problem is that the algorithm now requires very small identifiers (if we started with a deterministic algorithm) or that the failure probability of the algorithm increased to constant (if we started with a randomized one). This is where our story connects with our discussions of coloring and Lovász local lemma.

Theorem 2.22 (Chang and Pettie [88], Chang et al. [94]). *Let $\mathcal{A}$ be a local algorithm for a local problem Π with round complexity $t(n) = o(\log n)$. Then there is a local algorithm $\mathcal{A}'$ solving Π such that*

1. *If $\mathcal{A}$ was deterministic, so is $\mathcal{A}'$ and its round complexity is $O(\log^* n)$.*
2. *If $\mathcal{A}$ was randomized, so is $\mathcal{A}'$ and its round complexity is the same as the randomized complexity of solving relaxed Lovász local lemma, i.e., it is $\widetilde{O}\left(\log^4 \log(n)\right)$.*

The theorem holds for the class of bounded degree graphs and any of its subclasses closed on taking subgraphs and adding isolated vertices. More generally, for round complexities defined relative to a subclass of bounded degree graphs of restricted growth, the complexity $t(n)$ needs to be such that we always have $|B(u, t(n))| = o(n)$ for any node u.

Proof. Let Π be a local problem with a checkability radius of r and $\mathcal{A}$ be a deterministic (randomized) local algorithm for Π. We will assume that the identifiers are coming from a range $[n^C]$ (alternatively, the failure probability is $1/n^C$) for sufficiently large C. We will view $\mathcal{A}$ as a sequence $\mathcal{A}_1, \mathcal{A}_2, \dots$, and we start by choosing n_0 to be a large enough constant so that

$$\Delta^0 + \Delta^1 + \cdots + \Delta^{t(n_0)+r} \leq n_0. \tag{2.3}$$

We notice that such an n_0 exists by the requirement $t(n) = o(\log n)$. Our algorithm $\mathcal{A}'$ will simulate $\mathcal{A}_{n_0}$ for *any* input n (for $n < n_0$ we define $\mathcal{A}'_n = \mathcal{A}_n$).

Let us first handle the case of deterministic algorithms (see Figure 2.6). There, $\mathcal{A}'$ needs to supply identifiers to its simulation of $\mathcal{A}_{n_0}$. This is done as follows. We compute a $(\Delta(G') + 1)$-coloring of the power graph $G' = G^{2(t(n_0)+r)}$ such that the number of colors used is $\Delta(G') + 1 = \Delta^{O(t(n_0)+r)} \leq n_0^C$ where we used that C is large enough. We use those colors as identifiers for our simulation of $\mathcal{A}_{n_0}$. This finishes the description of $\mathcal{A}'$.

On one hand, the round complexity of the overall algorithm is $O(\log^* n)$ on bounded-degree graphs. On the other hand, suppose that $\mathcal{A}'$ fails to solve the problem Π at some node u. We notice that (2.3) yields that the $(t(n_0) + r)$-hop neighborhood of u contains at most n_0 nodes. Moreover, this neighborhood is colored by $\mathcal{A}'$ with unique colors from the range $[n_0^C]$. Recall that $\mathcal{A}_{n_0}$ is defined on neighborhoods with at most n_0 nodes labeled with identifiers from $[n_0^C]$. Thus the algorithm $\mathcal{A}'$ is well-defined. Additionally, a failure of $\mathcal{A}'$ at u would imply a failure of $\mathcal{A}$ on a neighborhood of size at most n_0 labeled with some unique identifiers from $[n_0^C]$. Adding isolated vertices to that neighborhood to make its size n_0, we reach a contradiction with $\mathcal{A}_{n_0}$ being correct on n_0-sized instances.

We handle the case of randomized algorithms similarly: We again try to simulate $\mathcal{A}_{n_0}$ for large enough n_0. We notice that we do not need to supply any identifiers in the simulation. However, if we simply simulate $\mathcal{A}$, we fail to solve Π at any fixed vertex u

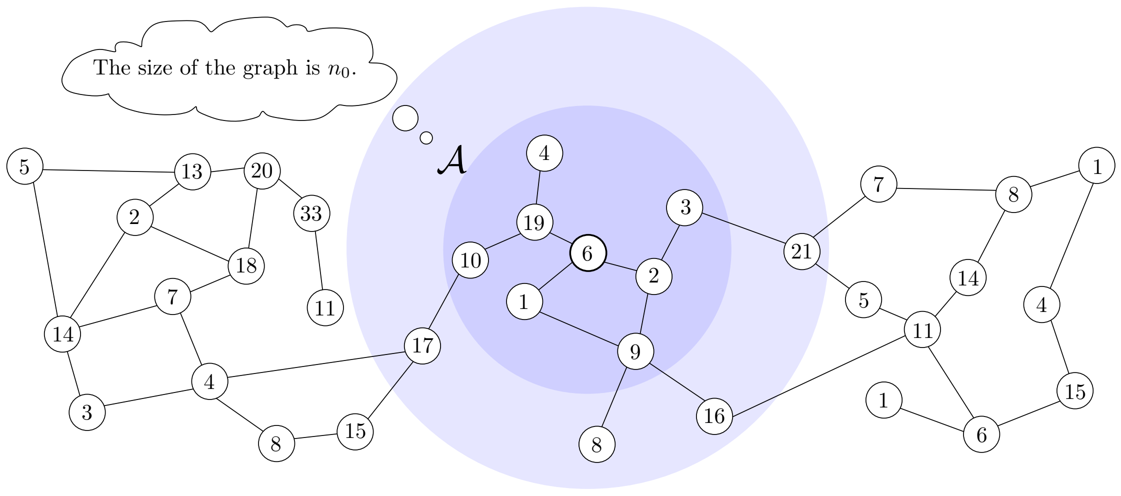

Figure 2.6: To speed up an algorithm $\mathcal{A}$, we tell it that the size of the graph is some constant n_0, whatever the true size n is. In this case, the round complexity of the algorithm becomes 2 (see the smaller ball in the picture) and we assume that the checkability radius of the local problem solved by $\mathcal{A}$ is 1.
The algorithm $\mathcal{A}$ assumes that the identifiers are unique numbers of size up to $n_0^{O(1)}$. Since $n \gg n_0$, we cannot supply such identifiers. However, we notice that if $\mathcal{A}$ is supplied with $n_0^{O(1)}$-sized identifiers that are unique only in every ball of radius 3 (see the larger ball in the picture), the algorithm still has to work since. This is because a failure of $\mathcal{A}$ around some node u implies a failure of $\mathcal{A}$ on the graph $B(u, 3)$ which has size at most n_0 and is labeled with unique identifiers.

with probability $1/n_0^C$ as this is the failure probability of $\mathcal{A}_{n_0}$. Thus, we instead formulate the process of simulating $\mathcal{A}$ as *an instance of Lovász local lemma* on a new variable-event graph G' as follows. Any original vertex u of the input graph G will be thought of as two new vertices in G'. One vertex, $u_{r.v}$, is a random-variable vertex, and it corresponds to the random string at u. The other vertex, $u_{b.e.}$, corresponds to the event that if we run $\mathcal{A}_{n_0}$ at u and use the random strings sampled from the random-variable vertices $v_{r.v.} \in B(u_{b.e.}, t(n_0))$, $\mathcal{A}_{n_0}$ fails to solve Π at u.

We notice that the event at $u_{b.e.}$ depends only on the $(t(n_0)+r)$-hop neighborhood of u in G. Thus the degree of this local-lemma instance (in the sense of its dependency-graph formulation) is at most $\Delta^0 + \Delta^1 + \cdots + \Delta^{t(n_0)+r}$ which is in turn at most n_0 by (2.3). On the other hand, the probability of each bad event is at most $1/n_0^C$. We can thus formulate the simulation of $\mathcal{A}_{n_0}$ as an instance of a C-relaxed local lemma. For large enough C, we can then apply Theorem 2.12 to solve that instance.

Finally, we notice that both in the deterministic and the randomize case, we can generalize the requirement $t(n) = o(\log n)$ to requiring that t satisfies for each u that $|B(u, t(n))| = o(n)$. □

We note that in the statement of Theorem 2.22, we do not literally require $t(n) = o(\log n)$ (or $|B(u, t(n))| = o(n)$), it should just be that for some large enough n_0, we have $t(n_0) \le \varepsilon \log n_0$, where ε is some small constant which is a function of Δ, r, and C.

Additional intuition yielded by the proof: The proof of Theorem 2.22 tells us a bit more: Basically, it allows us to think of local problems with deterministic round complexity $o(\log n)$ or, equivalently, $O(\log^* n)$, as "those problems that can be solved in a constant number of rounds, after we compute a distance coloring". Moreover, the problems with randomized round complexity $o(\log n)$ or, equivalently poly $\log\log n$, can be thought of as "those problems that we can view as an instance of the local lemma".

Problems with deterministic round complexity $o(\log n)$ can be, in fact, solved in constantly many rounds, after a suitable coloring used as "fake identifiers" is computed.
Problems with randomized round complexity $o(\log n)$ can be formulated as instances of relaxed Lovász local lemma.

Speedup below $\log^* n$: We will next sketch how one can speed up algorithms with round complexity $o(\log\log^* n)$ to complexity $O(1)$. Why the weird complexity $o(\log\log^* n)$? The following speedup argument relies more on the volume than on the radius[14], so the importance of radius $o(\log\log^* n)$ is that the number of nodes the algorithm sees is $\Delta^{o(\log\log^* n)} = o(\log^* n)$. The $\log^* n$ volume is then the right threshold for which

[14]Compare with Theorem 3.2 classifying volume complexities where we have $o(\log^* n)$ speedup instead.

we can argue that any deterministic algorithm can be turned into a drastically simpler *order-invariant* algorithm.

An order-invariant algorithm is a deterministic local algorithm with an additional restriction: it does not have direct access to the identifiers written on the nodes; it only knows their *order*, meaning that it can only compare their relative size. For example, in the setup of Theorem 2.8 where we worked with an oriented path with increasing labels, an order-invariant algorithm would see the same input order at every vertex. This already shows that order-invariant algorithms struggle to solve interesting local problems on graphs that are paths.

Theorem 2.23 (Naor and Stockmeyer [258], Chang et al. [94])**.** *Let $\mathcal{A}$ be a deterministic local algorithm for a local problem Π with round complexity $o(\log\log^* n)$. Then, there is a deterministic local algorithm $\mathcal{A}'$ solving Π with local complexity $O(1)$.*

The theorem holds for any subclass of bounded-degree graphs closed on taking subgraphs and adding isolated nodes.

Proof Sketch. We will sketch how any deterministic local algorithm $\mathcal{A}$ with round complexity $t(n) = o(\log\log^* n)$ can be turned into an order-invariant algorithm $\mathcal{A}'$, using the hypergraph Ramsey theorem[15]. Consider the set $B = [n^{O(1)}]$ of identifiers and any of its subsets S of size $(2\Delta)^{t(n)} = o(\log^* n)$. Here, the function $(2\Delta)^{t(n)}$ upper bounds the maximum number of nodes in any $t(n)$-hop neighborhood. We say that the *type* of S is the list of all possible $t(n)$-hop neighborhoods, together with their labeling with identifiers from S, where for each labeled graph on this list we record the output of $\mathcal{A}$ when being run on that labeled neighborhood.

We note that the number of types is relatively small, only roughly $2^{2^{O(\log^* n)^2}}$. For r the checkability radius of Π, we want to find a medium-sized set $M \subseteq B$ of size $(2\Delta)^{t(n)+r} = o(\log^* n)$ such that all subsets $S \subseteq M$ have the same type (i.e., the same color in the terminology of Ramsey's theorem). The known bounds for hypergraph Ramsey numbers [181, §1, Theorem 2] allow us to conclude that B is large enough to always contain such a set M.

With such a set M at hand, we next define the order-invariant algorithm $\mathcal{A}'$ for Π as follows: given an input order on vertices, the algorithm chooses an arbitrary set $S \subseteq M$ and outputs the answer that $\mathcal{A}$ would give if the vertices were labeled with identifiers from S in the following way: The order induced by the identifiers from S has to be the same as the actual input order. It does not matter which $S \subseteq M$ we choose since they all have the same type. This finishes the description of $\mathcal{A}'$.

To see that $\mathcal{A}'$ is valid, we assume for contradiction that it fails for some input order on vertices at some vertex u. Consider that input order on vertices in the $(t(n) + r)$-hop neighborhood of u. Replace this input order with identifiers from M such that they

[15]Recall that in hypergraph Ramsey theorem we have a big set B, we color its small subsets S and we want to find a medium-sized set M such that all its small subsets $S \subseteq M$ have the same color.

induce the same order. The original algorithm $\mathcal{A}$ behaves the same as the new algorithm $\mathcal{A}'$ when it sees only the identifiers from M: hence, $\mathcal{A}$ also fails. After adding isolated nodes to this neighborhood, we reach a contradiction.

Finally, let us contemplate the proof of Theorem 2.22 again. The only reason why we could speedup $o(\log n)$-round deterministic algorithm only to $O(\log^* n)$ instead of $O(1)$ was the necessity to compute a coloring that served as "fake" identifiers supplied to the simulation of an algorithm $\mathcal{A}_{n_0}$. But for order-invariant algorithms, this is not necessary. We can simply give $\mathcal{A}_{n_0}$ exactly the same order as the order that appears on input; thus order-invariant algorithms can be sped up to complexity $O(1)$, as needed. □

Tower-sized identifiers: We can wonder what would happen if we wanted to make the above proof of Theorem 2.23 work for *all* deterministic local algorithms, not just those that encounter only $o(\log^* n)$ vertices. This would force us to change the small-set size from $o(\log^* n)$ to n in the proof. Since the hypergraph Ramsey numbers grow roughly as a tower function of the small-set size, this means that to make the proof work, we would have to start with unique identifiers from the range which is contains numbers as large as the tower function of n. The rest of the proof then goes through and allows us to speed up $o(\log n)$-round algorithms to $O(1)$-round order-invariant algorithms.

This observation nicely complements Theorem 2.7: While it is true that exponentially-, doubly-exponentially-, etc. sized identifiers have the same power as polynomially-sized ones, there is a transition somewhere around tower-function-sized identifiers. From then on, deterministic algorithms of $o(\log n)$ complexity with such huge identifiers can be turned into order-invariant ones. In other words, the identifiers are so large that no algorithm can extract any meaningful information from them other than their relative order. [16]

2.4 Classification of Local Problems

We can now put all the pieces together and prove Theorem 2.1. Let's restate it here for convenience.

Theorem 2.1 (Classification of local problems with $o(\log n)$ complexity on bounded degree graphs)**.** *Let us fix any Δ and the class of graphs of degree at most Δ. Then, any local problem with randomized round complexity $o(\log n)$ has one of the following three round complexities.*

[16]Order-invariant algorithms stop being "useless" once we start considering $\Omega(\log n)$ round complexities. Problems like sinkless orientation can be solved deterministically in $O(\log n)$ rounds even in the order-invariant model since we can orient edges towards leaves. In general, at $O(\log n)$ complexity, deterministic algorithms stop being extremely reliant on identifiers for symmetry breaking since they start seeing irregularities in the input graph that can sometimes break the symmetry instead. In particular, the local neighborhood can no longer look like an expanding regular tree without leaves anymore.

1. Order-invariant regime: *The problem has $O(1)$ deterministic and randomized round complexity.*

2. Symmetry-breaking regime: *The deterministic and randomized round complexity of the problem lies between $\Omega(\log\log^* n)$ and $O(\log^* n)$ (both the deterministic and the randomized complexity is the same function).*

3. Lovász-local-lemma regime: *The problem has deterministic round complexity between $\Omega(\log n)$ and $\widetilde{O}(\log^4 n)$. Its randomized round complexity is between $\Omega(\log\log n)$ and $\widetilde{O}(\log^4 \log n)$.*

Proof. Let Π be any local problem of randomized local complexity $t(n) = o(\log n)$. We can use Theorem 2.22 to formulate Π as an instance of Lovász local lemma. This instance is then solved either with the deterministic $\widetilde{O}(\log^4 n)$-round algorithm from Theorem 2.10, or the randomized $\widetilde{O}(\log^4 \log n)$-round algorithm from Theorem 2.12.

Next, let us assume that the randomized round complexity of Π is even $o(\log\log n)$. Then, we can use the derandomization of Theorem 2.21 to conclude that the deterministic round complexity is $o(\log n)$. This allows us to use the speedup theorem of Theorem 2.23 to conclude that the deterministic round complexity of Π is $O(\log^* n)$. Moreover, we notice that for any increasing function $t(n) = O(\log^* n)$, we have $t(2^{O(n^2)}) = O(t(n))$; this means that the derandomization from Theorem 2.21 tells us that randomized and deterministic round complexities of Π are the same.

Finally, assume that the randomized local complexity of Π is even $o(\log\log^* n)$. Then, we can use Theorem 2.23 to find an order-invariant constant-round local algorithm for Π via Theorem 2.23. □

Local problems with complexities below $o(\log n)$ are of three types:
1. *Those that can be solved in constant rounds by order-invariant algorithms,*
2. *those that are roughly as hard as the basic symmetry-breaking problems such as coloring or maximal independent set,*
3. *those that can be viewed as instances of Lovász local lemma.*

Improvements to Theorem 2.1: Can we improve Theorem 2.1? The conjecture of Chang and Pettie [88] (Problem 2.14) suggests that in the local lemma regime, we may get rid of the polynomial gap between the lower and upper bounds. According to our current knowledge, there could be a local problem with deterministic and randomized round complexities $t_d(n), t_r(n)$ for any $t_d(n) = \Omega(\log n)$, $t_d(n) = \widetilde{O}(\log^4 n)$, $t_r(n) = \Omega(\log\log n)$, $t_r(n) = \widetilde{O}(\log^4 \log n)$, and $t_d(n) = O(t_r(2^n))$, where the last constraint follows from Theorem 2.20.

On the other hand, it is known [20] that the (extremely narrow) regime between $\Omega(\log\log^* n)$ and $O(\log^* n)$ is densely populated by certain (artificial) local problems.

2.4.1 Sequential local complexities

Let us now discuss how sequential local complexities fit into the picture painted by Theorem 2.1. We will first observe that the first two classes from our classification in Theorem 2.1, i.e., $O(1)$ and $O(\log^* n)$ round complexities, are equal to the class of problems solvable with the sequential local complexity is $O(1)$.

Theorem 2.24. *On bounded degree graphs, local problems with round complexity $O(\log^* n)$ are exactly those whose sequential local complexity is $O(1)$.*

Proof sketch. On one hand, consider any local problem with constant sequential local complexity. We can use Theorem 2.5 to simulate it with a distributed local algorithm in $O(\log^* n)$ rounds.

On the other hand, consider any local problem Π with local checkability r solvable in $O(\log^* n)$ round complexity. Recall that the proof of Theorem 2.22 says that Π can in fact be solved with a local algorithm $\mathcal{A}$ of round complexity $t = O(1)$ on any graph G if we are provided a proper coloring of a suitable power graph $G^{2(t+r)}$. Recall that $(\Delta + 1)$-coloring has sequential local complexity $O(1)$, meaning that we can construct a sequential local algorithm of constant complexity for Π. □

Next, we will show how our distributed speedup theorems imply speedups in the sequential world.

Theorem 2.25. *Let $\mathcal{A}$ be a sequential local algorithm solving a local problem Π on bounded degree graphs. Assume that either*

1. *$\mathcal{A}$ is deterministic and its sequential local complexity is $o(\log \log n)$,*
2. *or $\mathcal{A}$ is randomized and its sequential local complexity is $o(\log \log \log n)$.*

Then, there is a distributed local algorithm $\mathcal{A}'$ that solves Π in $O(\log^* n)$ round complexity.

Proof. If $\mathcal{A}$ is a deterministic sequential algorithm of local complexity $t(n) = o(\log \log n)$, we can simulate it with a distributed local algorithm via Theorem 2.5. We get a deterministic algorithm with local complexity $\Delta^{O(t(n))} + O(t(n) \log^* n) = \log^{o(1)} n = o(\log n)$. Such an algorithm is then sped up to $O(\log^* n)$ complexity using Theorem 2.22.

Similarly, if $\mathcal{A}$ is randomized sequential algorithm of local complexity $o(\log \log \log n)$, Theorem 2.5 implies that we get a randomized distributed local algorithm with $\log^{o(1)} \log n = o(\log \log n)$ round complexity. Such an algorithm can then be derandomized by Theorem 2.21 into a deterministic algorithm of local complexity $o(\log n)$ which is in turn sped up to $O(\log^* n)$ using Theorem 2.22. □

Finally, we sketch how one can construct very fast sequential local algorithms for instances of the local lemma.

Theorem 2.26. *Let Π be a C-relaxed Lovász local lemma for sufficiently large C. Then, its deterministic sequential local complexity is $\widetilde{O}(\log^4 \log n)$ and its randomized sequential local complexity is $\widetilde{O}(\log^4 \log\log n)$*[17]*, on bounded degree graphs.*

Proof Sketch. To prove the first part of the theorem, we consider the randomized distributed local algorithm for local lemma from Theorem 2.12. We turn it into a deterministic sequential local algorithm of the same complexity by the derandomization result of Theorem 1.7.

To prove the second part of the theorem, we notice that our randomized distributed local algorithm from Theorem 2.12 can certainly be implemented in the model of randomized sequential local algorithms. Recall that the algorithm reduces an instance of the local lemma to instances on graphs of size $O(\log n)$, as proven in Theorem 2.11. Then, the best deterministic algorithm is applied to those instances to finish the job. While Theorem 2.11 is stated as reducing from C-relaxed local lemma with some large C to local lemma with condition $p \le 1/(3\Delta)$, we can generalize it to show that starting with a very large C', we can reduce an instance of C'-relaxed local lemma to C-relaxed local lemma for C that is still arbitrarily large.

This allows us to run the deterministic sequential algorithm with complexity $\widetilde{O}(\log^4 \log n)$ constructed in the first part of the proof instead of using the deterministic $O(\text{poly}\log n)$-round algorithm from Theorem 2.10. The final randomized sequential local complexity is thus $\widetilde{O}(\log^4 \log\log n)$. □

We note that for problems like sinkless orientation for which an $O(\log\log n)$-round randomized algorithm is known [166], the proofs of above Theorems 2.25 and 2.26 imply tight bounds of $\Theta(\log\log n)$ and $\Theta(\log\log\log n)$ deterministic and randomized sequential local complexity on bounded degree graphs. In the case of sinkless orientation, the same bounds are also known even on unbounded degree graphs [166]. If the conjecture of Chang and Pettie [88] (Problem 2.14) is true, then there is no gap between Theorems 2.25 and 2.26.

We also note that Theorems 2.25 and 2.26 show that while one can turn randomized distributed local algorithms into sequential deterministic ones via Theorem 1.7, it is *not* possible to turn randomized sequential local algorithms into deterministic sequential local ones, without loss in their local complexity.

Putting everything together, we get the following refined classification theorem that also includes sequential local complexities.

Theorem 2.27 (Refined classification of local problems)**.** *Let us fix any Δ and any class of graphs $\mathcal{G}$ of degree at most Δ that is closed under taking subgraphs and adding isolated*

[17]It is an interesting exercise to open up the proof that the randomized sequential local complexity of local lemma is $\Theta(\text{poly}\log\log\log n)$ and try to think of theorems we have seen so far that do *not* go into that proof.

nodes. Let $t(n)$ be a function such that for any node u in some graph in $\mathcal{G}$ we have $|B(u, t(n))| = o(n)$.

Then, any local problem with randomized round complexity at most $t(n)$ has one of the following three complexities.

1. Order-invariant regime: *The problem has $O(1)$ deterministic (or randomized) round complexity and sequential local complexity.*
2. Maximal-independent-set / Coloring regime: *The deterministic and randomized round complexity of the problem lies between $\Omega(\log\log^* n)$ and $O(\log^* n)$ (both the deterministic and the randomized complexity is the same function). Moreover, the sequential local complexity is $O(1)$.*
3. Lovász-local-lemma regime: *The problem has the following complexities:*
   - (a) *deterministic round complexity is between $\Omega(\log n)$ and $\widetilde{O}(\log^4 n)$,*
   - (b) *randomized round complexity and deterministic sequential local complexity is between $\Omega(\log\log n)$ and $\widetilde{O}(\log^4 \log n)$,*
   - (c) *randomized sequential local complexity is between $\Omega(\log\log\log n)$ and $\widetilde{O}(\log^4 \log\log n)$.*

Moreover, if any graph from the class satisfies that for any node u, we have $|B(u, r)| = 2^{O(r^{0.249})}$, then there are no local problems in the third class of problems.

The last part of the theorem follows by applying Theorem 2.22: We know that every local problem in the local-lemma regime can be solved with a deterministic local algorithm of round complexity $\widetilde{O}(\log^4 n)$; since $2^{\left(\widetilde{O}(\log^4 n)\right)^{0.249}} = o(n)$, the speedup to $O(\log^* n)$ complexity can be applied. If the conjecture of Chang and Pettie [88] (Problem 2.14) is true, we can get rid of the polynomial slack in the local-lemma regime complexities and the whole local-lemma class of problems is then empty for the class of subexponential growth graphs (see Section 2.5).

Problem 2.28. *Can local problem solvable with deterministic sequential local complexity $o(\log n)$ be sped up to* $\operatorname{poly}\log\log n$ *deterministic sequential complexity (i.e., to the local-lemma regime)? What about problems with randomized sequential local complexity $o(\log n)$?*

2.5 Classification of Local Problems for Concrete Graph Classes

This section compiles results from a long line of work trying to extend the classification of local problems on bounded-degree graphs, i.e., Theorem 2.1, to more specific graph classes. For some simple graph classes, we now have an almost complete understanding of the local complexity classes that go even beyond the $\log n$ local complexity threshold.

Locally checkable labeling problems: The theorems below are proven for the so-called *locally checkable labeling problems.* These are local problems that additionally allow one of the finitely many input colors on each node of the input graph. For example, a list coloring is an example of a locally checkable labeling problem.

Definition 2.29. *A* locally checkable labeling problem *(LCL)* Π *is formally a quintuple* $(S_{in}, S_{out}, r, \Delta, \mathcal{P})$*. Here,* S_{in} *is the finite set of* input labels*,* S_{out} *is the finite set of* output labels*, and* $\mathcal{P}$ *is the set of allowed neighborhoods: Each neighborhood has maximum degree* Δ*, radius* r*, and each node is labeled with a label from* S_{in} *and* S_{out}*. Given an input graph of degree at most* Δ *and any input coloring of its nodes with* S_{in}*, the task is to find an output coloring by colors from* S_{out} *so that the* r*-hop neighborhood of each node is in* $\mathcal{P}$*.*

We did not have to distinguish between our definition of a local problem from Theorem 1.1 and the locally checkable labelings since, in the general graph classes like the class of all graphs or all bounded-degree graphs, it is straightforward to encode input labels as a part of the input graph. We thus chose the simpler definition that also allowed us to talk about local problems even outside of the setup of bounded-degree graphs. In the following theorems, especially for very restrictive classes like paths or grids, the difference between the two definitions matters, since the possibility of allowing inputs may substantially enlarge the number of possible local problems.

In the following theorems, we will always implicitly assume that they are for *solvable* locally checkable labelings, i.e., for problems where the solution is always guaranteed to exist. Otherwise, the problem does not have a well-defined local complexity class.

Computational aspect: We note that an additional dimension for all classification results below is the computational complexity of the classification problem, where the input is a description of locally checkable labeling, and the output is which class it belongs to. On the one hand, it is known that the classification problem is decidable on paths, albeit PSPACE-hard [21]. On the other hand, the classification problem is undecidable even for grids [77]. The decidability on trees is open; see Problem 2.34. We will not discuss this dimension in the following theorem statements for brevity.

Remarks: We list the concrete classification theorems below. Unless stated otherwise, the deterministic and the randomized round complexity of the problem is always the same. We are always citing papers proving results specific to the given graph class, for example, general speedup and slowdown theorems of Chang and Pettie [88], Chang et al. [94] are as a rule of thumb always an important part of the proof. Also, not all the papers we cite prove a part of the given theorem. Some of them only provide building blocks, analyze the computational complexity of the classification, etc.

List of known classifications: Let us now list known classification theorems on concrete graph classes.

Theorem 2.30 (Classification of local problems on paths [77, 21])**.** *Any solvable locally checkable labeling problem on oriented or unoriented paths with inputs has one of the following round complexities:*

1. $O(1)$,
2. $\Theta(\log^* n)$,
3. $\Theta(n)$.

Theorem 2.31 (Classification of local problems on grids [258, 77, 94, 188])**.** *Any solvable locally checkable labeling problem on d-dimensional oriented grids*[18] *has one of the following local complexities:*

1. $O(1)$,
2. $\Theta(\log^* n)$,
3. $\Theta(n^{1/d})$.

We note that on unoriented grids, the full classification is not known [188].

Theorem 2.32 (Classification of local problems on bounded-degree rooted trees [31, 97])**.** *Any solvable locally checkable labeling problem on bounded-degree rooted trees*[19] *has one of the following round complexities:*

1. $O(1)$,
2. $\Theta(\log^* n)$,
3. $\Theta(\log n)$,
4. $\Theta(n^{1/k})$ *for some* $k \in \mathbb{N}$.

Theorem 2.33 (Classification of local problems on bounded-degree trees [254, 88, 20, 85, 27, 188, 34, 25, 235])**.** *Any solvable locally checkable labeling problem on trees has one of the following local complexities:*

1. $O(1)$,
2. $\Theta(\log^* n)$,
3. $\Theta(\log\log n)$ *randomized,* $\Theta(\log n)$ *deterministic,*
4. $\Theta(\log n)$ *(both randomized and deterministic),*
5. $\Theta(n^{1/k})$ *for some* $k \in \mathbb{N}$.

[18]The edges of the input grid are consistently oriented.
[19]Edges are oriented such that every vertex has at most one ingoing edge.

Problem 2.34. *Given an input locally checkable labeling problem on bounded-degree trees, is it decidable which class in Theorem 2.33 it belongs to?*

More Classifications: It is an intriguing question of how far these classifications can be extended, see e.g. the recent work extending the theory to minor-closed classes [86] or extending the theory to unbounded-degree graphs [234].

Another interesting class is the class of subexponential growth graphs, i.e., the class of bounded degree graphs where for every $\varepsilon > 0$ we can find r such that $|B(u,r)| \leq (1+\varepsilon)^r$ [20]. In this class of graphs, we can apply Theorem 2.22 to speed up deterministic algorithms of complexity $O(\log n)$ (instead of $o(\log n)$) to $O(\log^* n)$. If the conjecture of [88] (Problem 2.14) is true, this has an interesting corollary: the local lemma regime from the classification theorem of Theorem 2.1 would then not be present in this class of graphs and there would only two classes of local problems there (see Theorem 2.27). This leads to the following special case of Problem 2.14:

Problem 2.35. *Is there a C such that all instances of the C-relaxed Lovász local lemma can be solved with round complexity $\Theta(\log^* n)$ on any class of graphs of subexponential growth?*

Some implications of a positive answer to this problem are known [111, 65].

[20]Formally, this is a family of classes, with one class for each possible dependency $r = r(\varepsilon)$.

CHAPTER 3

Applications

While the previous two chapters, Chapters 1 and 2, discussed the general theory of local algorithms, the following chapter considers various popular models of distributed, parallel, or sublinear algorithms, as well as the field of descriptive combinatorics, and briefly discusses how techniques from local algorithms can help there, discussing a few cherry-picked examples per section. These sections are heavily influenced by the author's particular interests; a different author would choose different applications.

3.1 Distributed Computing (CONGEST)

The CONGEST model [264] is a model of distributed computing that is extremely tightly connected to local algorithms[1] Starting from the definition of a local algorithm as a message-passing protocol between nodes in a graph, the CONGEST model adds an additional requirement that each message is supposed to be *small.* In particular, it should fit into $O(\log n)$ bits. For example, a node can send its unique identifier in one round.

Some local algorithms in the literature are stated as CONGEST algorithms since a number of local algorithms are readily implemented in the CONGEST model. However, not all of them. As an example, for local algorithms, it is often useful that each node can collect all the information from its local neighborhood and perform a local computation on it. However, in the CONGEST model, even just collecting the size of a 2-hop neighborhood of every node can take $\Omega(n)$ rounds.

For example, counting for each node the number of triangles that contain it is a trivial problem for local algorithms but a very complex problem in the CONGEST model, where it requires polynomially many rounds [95, 89]. As another example, coloring the graph G^2 with $\Delta(G)^2 + 1$ colors in the CONGEST model is a substantially more complicated problem [197] than with local algorithms and coloring the graph G^3 with a reasonable

[1]Local algorithms are often referred to as "distributed algorithms in the LOCAL model of distributed computing".

number of colors is probably hopeless in the CONGEST model (given a candidate coloring of G^3, we cannot even check quickly whether it is correct).

The general derandomization technique of Ghaffari et al. [166] from Theorem 1.8 is not directly applicable to the CONGEST model, since computing conditional expectations in a neighborhood of a node requires collecting a lot of information from that neighborhood. However, the general approach is still helpful, and it was used to derandomize concrete problems like maximal independent set [84] or $\Delta + 1$-coloring [41].

It is an interesting question for which classes of graphs the definitions of local and CONGEST algorithms coincide for local problems. It is known that this is the case for local problems on trees [32].

3.2 Local Computation Algorithms and the Volume Model

The model of local computation algorithms [277, 8], often denoted as LCA, is a model of sublinear algorithms closely related to the local model. We will first explain its variant, known as the *Volume* model by Rosenbaum and Suomela [270]. This model is very similar to local algorithms, but we measure the *volume* instead of the *radius*. A volume algorithm run at a node u starts with a set $S_0 = \{u\}$ and in the i-th step, the algorithm can pick a node $u_i \in S_i$ and an arbitrary index $1 \le j \le d$ where d is the degree of u_i. Then, it learns the identifier of the j-th neighbor v of u_i. We then set $S_{i+1} = S_i \cup \{v\}$. The volume complexity of the algorithm is the number of queries it makes until it decides to finish and output the label for u.

In particular, we note that for graphs of maximum degree Δ, any $t(n)$-round local algorithm can be turned into a volume algorithm of complexity $(2\Delta)^{t(n)}$ as the volume algorithm at u can simply query all the nodes in the $t(n)$-hop neighborhood of u [263].

A local computational algorithm may have several definitions. The stateless local computation algorithm is defined as follows: we start with the definition of the volume model and add the additional requirement that the nodes start with unique identifiers from the range $[n]$, even for randomized algorithms. In each query, the algorithm can additionally ask for the node with the identifier j for any $1 \le j \le n$. The node u_i with identifier j is then added to the set of already seen nodes S_i. That is, the algorithm can use additional global queries that are, however, “blind”.

It is unclear whether global queries can help with solutions to local problems. We know that they do not help for very fast randomized local computation algorithms [179].

Problem 3.1. *Is there a local problem that distinguishes (either randomized or deterministic) Local computation algorithms and Volume models with respect to a natural class of graphs?*

As an example problem considered in the LCA model, there is a long line of work on local computation algorithms for maximal independent set and related problems [259, 297, 260, 57, 277, 8, 232, 150, 269, 163, 152] culminating with the randomized algorithm of Ghaffari [152] with volume complexity $\mathrm{poly}(\Delta \log n)$.

For constant degree graphs, we can try to prove classification results similar to the classification of local problem complexities from Theorem 2.1. We have some understanding of the volume complexities of the three important local complexity classes from Theorem 2.1:

Theorem 3.2 ([263, 127, 270, 188, 81])**.** *In the class of bounded-degree graphs, each local problem satisfies the following:*

1. *If its round complexity is $O(1)$ (order-invariant regime), then its randomized/deterministic volume complexity is $O(1)$.*
2. *If its randomized/deterministic round complexity is $\omega(1)$ and $O(\log^* n)$ (maximal independent set / coloring regime), its randomized/deterministic volume complexity is $\Theta(\log^* n)$.*
3. *If its randomized local complexity is $\omega(\log^* n)$ but $o(\log n)$ (Lovász local lemma regime), its randomized volume complexity is between $\Omega(\sqrt{\log n})$ and $O(\log n)$.*

We note that as far as we know, all local problems in the local-lemma regime have randomized volume complexity of $\Theta(\log n)$. In particular, sinkless orientation has this randomized volume complexity [81].

Problem 3.3 (See Conjecture 1.3 in [81])**.** *Is it true on bounded degree graphs that local problems from the local-lemma regime have randomized volume complexity $\Theta(\log n)$?*

We also note that in the model of deterministic volume complexity where the identifiers come from exponential, instead of polynomial range, we can prove that only the volume complexities $O(1), \Theta(\log^* n), \Theta(n)$ are possible [81]. The same could be the case also for the standard, polynomial-range, identifiers.

Problem 3.4. *Are there any local problems such that their deterministic volume complexity (with polynomial-sized identifiers) on bounded degree graphs is $\omega(\log^* n)$ but $o(n)$?*

Equivalently, we could have asked whether there is a local problem whose deterministic volume complexity is different for polynomial-sized and exponential-sized identifiers.

3.3 PRAM

PRAM is a classical model of parallel algorithms studied extensively in the past 40 years [215]. It simplifies the complexity of practical parallel computing by assuming a simple model of a machine with multiple processors sharing a common memory. There are two complexity measures: *work*, i.e., the total number of instructions made by all processors together throughout the execution, and *depth*, i.e., the number of rounds necessary to finish the computation in the case that the machine is equipped with as many processors as the algorithm requires.

Many techniques developed for distributed and local algorithms are useful in the design of parallel algorithms. As an example application for a local problem, Ghaffari and Haeupler [156] develop a randomized algorithm for maximal independent set on the so-called EREW variant of the PRAM model with the optimal depth $O(\log n)$ and $O(m \log^2 n)$ work, building on top of the local maximal independent set algorithm of Ghaffari [150].

As another example application for a non-local problem, we note that one can generalize deterministic local algorithms for network decompositions discussed in Section 1.5 to get various clustering results for weighted undirected graphs like the following one.

Theorem 3.5 (Deterministic low-diameter clustering [272])**.** *Let G be a graph and w its nonnegative weights. Then, there is a parallel algorithm with $\widetilde{O}(m+n)$ work and $\widetilde{O}(1)$ depth such that, given a parameter $R > 0$, it splits vertices of G into clusters such that*

1. *Each cluster has weighted diameter $\widetilde{O}(R)$,*
2. *The total weight of edges that cross between different clusters is at most $\frac{1}{R}\sum_{e \in E(G)} w(e)$.*

Variants of this clustering result can be used for the design of deterministic parallel algorithms for the approximate shortest path problem or various metric embedding problems [274, 272].

3.4 MPC

The final parallel model we discuss is the model of Massively parallel computing (MPC) [217], the theoretical model behind the popular programming framework MapReduce and its variants. While the previous PRAM model focuses on the total work done by processors and ignores the cost of communication, the MPC model ignores the work done by processors and focuses on the communication cost.

Concretely, let us explain the so-called low-memory regime of MPC: At the beginning of any graph algorithm, the edges of the input graph are split into many machines, each capable of storing $O(n^\varepsilon)$ bits in its memory. In one round, each machine can perform an arbitrary computation on its edges, and then it sends arbitrary information to any of its neighbors (the total information sent and received per machine is $O(n^\varepsilon)$). We measure the number of rounds until the solution is computed.

Chang, Fischer, Ghaffari, Uitto, and Zheng [93] constructed a randomized massively parallel algorithm for $\Delta + 1$-coloring that works in $O(\log\log\log n)$ rounds by simulating the fastest randomized local algorithm for coloring of round complexity $\operatorname{poly}\log\log n$ (see Chapter 3) with exponential speedup. This result is complemented by the work of Ghaffari, Kuhn, and Uitto [168] that showed that this result cannot be improved, conditioned on a certain conjecture. Notice how understanding where the three logarithms are coming from requires an extensive understanding of coloring with local algorithms: The way we arrive at this complexity is that, first, starting with the trivial constant-round sequential local algorithm for coloring, we turn it into a deterministic distributed $\operatorname{poly}\log n$-round local algorithm using the general translation of Theorem 1.4. Next, this algorithm can be turned into an exponentially faster randomized algorithm using the shattering technique [92]. Finally, this randomized algorithm is simulated, again with exponential speedup, in the massively parallel model.

There is also a work aiming to extend the classification of local problems to the massively parallel algorithms model [82, 35].

3.5 Descriptive Combinatorics

Descriptive combinatorics [230, 245, 180, 119, 243, 140, 220, 244, 106, 108, 111, 60] is an area at the intersection of combinatorics, measure theory, and set theory (see e.g. the following surveys [219, 265, 63]). It studies combinatorial objects like graphs that arise when one manipulates uncountably-infinite large objects equipped with measure. The main object of interest is a *measurable graph.* Instead of a formal definition, let us discuss a particular example.

Consider a circle, i.e., a set of points of distance 1 from the origin of the plane $\mathbb{R}^2$. A rotation of this circle by 1 radian counterclockwise induces a graph G_0: We can draw an oriented edge from each point v on the circle to the point v' that v maps to by the rotation. This graph has uncountably many connected components, each of which is a doubly-infinite oriented path. Moreover, the Lebesgue measure on the circle is telling us which subsets of vertices of G_0 we are "allowed" to talk about. The graph G_0, together with the measure on top of it, is an example of a measurable graph.

"Solving" a local problem Π on some measurable graph G now means that we label the vertices of G with labels from Π. On one hand, we want all local constraints to be satisfied (almost) everywhere. On the other hand, the set of vertices labeled with the same label should be measurable. Specifying this idea formally leads to several possible definitions of what a solution is, like measurable solutions, Borel measurable solutions, or Baire measurable solutions, all of which are studied in descriptive combinatorics.

There is a close connection between local algorithms and the area of descriptive combinatorics. Although this was to some extent understood earlier (see Elek and Lippner [121], Lovász [238, Chapter 23.3]), it was only a breakthrough paper of Bernshteyn [60] that first realized the full power of the connection. We invite the reader to read the recent survey of Bernshteyn [63] that covers the connection in depth, here we will only mention a few known results.

For the class of bounded degree graphs, we understand that local problems solvable in $O(\log^* n)$ rounds admit Borel solutions [60]. Problems that one can formulate as an instance of the local lemma (i.e., problems with $o(\log n)$ randomized complexity) admit measurable and Baire-measurable solutions [60].

Sometimes, we understand that a class of local problems defined using the language of descriptive combinatorics exactly coincides with a class of problems defined using the language of local algorithms: On Cayley graphs, problems solvable in $O(\log^* n)$ rounds are exactly those allowing so-called continuous solutions [61, 147]. On regular trees, local problems with complexity $O(\log n)$ are exactly those allowing Baire-measurable solutions [83]. Finally, and this is perhaps the most easily understandable example, local problems solvable on the graph G_0 defined above are exactly those for which we have an $o(n)$-round local algorithm on oriented paths [184, Section 2].

When it comes to concrete problems, many recent results from descriptive combinatorics are using techniques that are, by now, familiar to readers of this text: Network decomposition [68], Lovász local lemma [60, 61, 111, 67, 68], derandomizations [61, 185], ID graphs [83, 80], and others.

This connection also led to progress for local algorithms. For example, Bernshteyn [64] used ideas from a measurable Vizing theorem of Grebík and Pikhurko [183] to construct a fast local algorithm for $(\Delta + 1)$-edge coloring problem. Brandt et al. [83] generalized a lower bound of Marks [244] to develop a new lower bound technique for local algorithms: This technique gives a different (and similarly simple) lower bound proof of Theorem 2.15 and leads to the currently only known example of a problem on trees with an unknown round-elimination-based proof.

3.6 Other Models

There are many more models of local/distributed/parallel/sublinear computation that are, in one way or another, related to local algorithms. In many of these models, it not only makes sense to try to solve concrete problems but often, one can also hope that the theory of local algorithms, such as the classification of the local problems on bounded-degree graphs, can be extended similarly to, e.g., Theorem 3.2.

Uniform algorithms: One favorite model of the author is the model of *uniform* local complexity. In this model, a randomized local algorithm does not know the size of the input graph, n. It simply looks at larger and larger neighborhoods of a given vertex, until it decides that it has seen enough to compute the output label at the node. This model was independently studied by local algorithms community [223] and community of probabilists [212, 207, 286] where it is known as *finitary factors of i.i.d.*

It can be seen as a more powerful version of classical local algorithms in the $o(\log n)$ regime. This is because there are certain local problems such that their solution requires a few nodes to see the whole graph, while most of the nodes can output their solution after seeing their $O(1)$-radius neighborhood [185]. On the other hand, local problems solvable by uniform local algorithms still often admit measurable solutions defined in descriptive combinatorics [184, 185, 83]. Thus, uniform local algorithms can be seen as interpolating between the extremely clean picture painted by the classification of $o(\log n)$-round algorithms (Theorem 2.1), and the much less well-understood complexity classes coming from descriptive combinatorics.

The connection between classical local algorithms and uniform ones would be much cleaner if the following problem was resolved:

Problem 3.6. *Is there a C and a uniform randomized local algorithm on bounded degree graphs for C-relaxed Lovász local lemma such that after* $\operatorname{poly}\log\log(1/\varepsilon)$ *rounds, at least* $1-\varepsilon$ *fraction of nodes know the solution?*

Other models: There are many more models of locality studied in the literature. Let us list a sample of them. The list includes the averaged local complexity [131, 46, 99, 39] (closely connected to uniform algorithms), online local model [2, 98, 3], dynamic local model [2, 3], local mending model [251, 252], supported local model [279, 222], quantum local model [149, 142, 103, 3], awake complexity [99, 160], energy complexity [91, 96, 161], local certification [176, 137, 138, 132], finitely dependent colorings [205, 208], or computable combinatorics [266].

Final Remarks

This text aims to present the area of local algorithms in a way that highlights what the author sees as its key aspect: Unlike typical subareas of computer science and discrete mathematics that are usually unified by a set of useful techniques and important results, the area of local algorithms extends beyond this norm. We have a clean theory of the model that leads to a clear complexity-theoretical picture. In this sense, local complexity can be viewed as a nascent counterpart to more established areas like communication complexity. Some of the results presented in the last chapter indicate that the theory has the potential for diverse extensions and applications. The author encourages the reader to identify and explore the next one.

CHAPTER 4

Overview of New Results Proven in the Thesis

This section briefly overviews the results proven in the thesis and explains how they fit into the big picture presented earlier in Chapters 1 to 3. We hope that various techniques employed in these results show the richness of techniques used in the area of local algorithms.

4.1 Theory of Local Algorithms

The next three chapters will discuss new results related to the theory of local algorithms.

4.1.1 A Simple Deterministic Distributed Low-Diameter Clustering

In Chapter 5, we discuss a simple deterministic algorithm for network decomposition, a fundamental problem discussed extensively in Sections 1.3 to 1.5. The chapter is based on a paper of Rozhoň et al. [275].

More concretely, we present a local algorithm that proves the following theorem (recall the definition of network decomposition from Theorem 1.5):

Theorem 4.1 (cf. Theorem 5.4)**.** *There exists a deterministic local algorithm that constructs a network decomposition with $O(\log n)$ color classes and diameter $O(\log^3 n)$. The algorithm has round complexity $O(\log^7 n)$.*

There are numerous other deterministic constructions of network decompositions, some of which [272, 172] improve upon Theorem 4.1. However, we believe that together with a closely related construction by [276], our construction is perhaps the simplest and hopefully useful for presentations in classes, especially if the algorithm of Miller et al. [255] is presented first. If the reader plans to read beyond this section, we invite them to read Chapter 5.

The construction can be seen as a deterministic variant of the beautiful randomized clustering algorithm by Miller et al. [255]. This randomized algorithm is sketched in

Section 1.5, simply speaking, every node chooses a head-start from an exponential distribution, then we simulate running breadth-first-search from all nodes at once.

In the deterministic variant of this algorithm, we have $O(\log n)$ phases, one for each bit in the identifier. In the very first phase, we first split the vertices based on their most significant identifier bit into red and blue nodes. Then we start increasing head-starts of red nodes. After each increase, we simulated the breadth-first-search with those head-starts and see how much the red clusters grew. If a red cluster did not grow by a sufficient amount, we stop increasing its head-start and delete the final layer by which the cluster grew. By an argument similar to ball-carving, we delete only a small fraction of nodes. By an argument similar to the analysis of the algorithm of Miller et al. [255], the red clusters are separated from the remaining blue nodes. Iterating this process for each bit in the identifier, we construct the first clustering of the final network decomposition from Theorem 4.1.

4.1.2 Generalizing the Sharp Threshold Phenomenon for the Distributed Complexity of the Lovász Local Lemma

In Chapter 6, we prove that there is a curious sharp threshold phenomenon related to the complexity of Lovász local lemma. This result is stated as Theorem 2.18 in Section 2.2 and put into context in Section 2.2. This chapter is based on the paper of Brandt et al. [79].

Let us restate Theorem 2.18 next for convenience:

Theorem 2.18 ([73, 78, 79])**.** *On one hand, there is an instance of Lovász local lemma (namely sinkless orientation) with the criterion $p \cdot 2^{\Delta} \le 1$ that has deterministic round complexity $\Omega(\log n)$. On the other hand, any instance of Lovász local lemma with the criterion $p \cdot 2^{\Delta} < 1$ has deterministic round complexity $O(\log^* n)$.*

We note that the first part of the theorem follows from the lower bound for sinkless orientation by Brandt et al. [73] discussed in Section 2.2. We prove the second part of the theorem, building on an earlier paper of Brandt et al. [78] who proved the conjecture for the case when each random variable has rank at most 3, using the following elegant sequential local algorithm (that can be turned into a distributed one by Theorem 2.5).

The algorithm iterates through random variables and fixes them to some values while preserving the following invariant. In order to define the invariant, each edge of the dependency graph is assigned two non-negative values, one for each endpoint of the edge, that sum up to 1. When fixing a random variable, the algorithm is also allowed to change these "book-keeping" values. The invariant now states that for any node v in the dependency graph, the product of the $\deg(v)$ values around v multiplied by p is an upper bound for the conditional probability of the event $\mathcal{E}_v$ associated with node v to occur (where we naturally condition on the already-fixed random variables being fixed as prescribed by the (partial) value assignments performed by the algorithm so far). If this invariant is preserved, then, after all variables are fixed, each event $\mathcal{E}_v$ occurs with probability at most $2^{\deg(v)} \cdot p \leq p2^{\Delta} < 1$, and therefore with probability 0, as desired.

Surprisingly, always being able to fix a random variable so that the invariant is preserved is equivalent to proving the convexity of a certain complicated set. Brandt et al. [78] were able to prove the convexity in the case where ranks are at most 3 where one can have an explicit understanding of the set. On the other hand, in Chapter 6 we prove that the set is convex *always* using a more subtle convex-geometrical argument.

We believe that Theorem 2.18 is an interesting example of the richness of techniques applied in the area of local algorithms. A priori, it is quite surprising that a substantial part of its proof engages in convex geometry.

4.1.3 The Landscape of Distributed Complexities on Trees and Beyond

In Chapter 7, we prove several speedup theorems that form a part of the classification theorems for trees (Theorem 2.33) and grids (Theorem 2.31). This chapter is based on the paper of Grunau et al. [188].

Let us restate the theorem classifying the local problems on trees here for convenience.

Theorem 2.33 (Classification of local problems on bounded-degree trees [254, 88, 20, 85, 27, 188, 34, 25, 235])**.** *Any solvable locally checkable labeling problem on trees has one of the following local complexities:*

1. $O(1)$,
2. $\Theta(\log^* n)$,
3. $\Theta(\log \log n)$ *randomized,* $\Theta(\log n)$ *deterministic,*
4. $\Theta(\log n)$ *(both randomized and deterministic),*
5. $\Theta(n^{1/k})$ *for some* $k \in \mathbb{N}$.

The theorem is a result of a substantial number of papers working their way through various complexity regimes. In Chapter 7 we prove what was the final missing piece in the theorem: a gap between constant and sub-iterated-logarithm complexities. This improves upon the $o(\log \log^* n)$ speedup discussed in Section 2.3.

Theorem 4.2 (Cf. Theorem 7.1)**.** *Let Δ be any fixed positive integer. Any locally checkable problem on bounded degree trees with round complexity $o(\log^* n)$ has, in fact, round complexity $O(1)$.*

Our approach is based on the round elimination technique [74], a highly successful technique for proving local lower bounds. However, there is a twist to it: The standard use case for applying round elimination has been to prove lower bounds for some *concrete, fixed* problem such as maximal matching or Lovász local lemma [73, 74, 22, 26]. We provide a novel application of round elimination by showing that, perhaps surprisingly, it can also be used to prove gap results, which are results that reason about *all* local problems on a given graph class.

More precisely, we show that with the tool of round elimination at hand, there is a way to prove Theorem 4.2, which proceeds as follows. We start with any problem Π for which there exists a randomized algorithm $\mathcal{A}$ that solves Π in $t(n) = o(\log^* n)$ rounds, with probability $1 - 1/\operatorname{poly}(n)$. We fix some sufficiently large number n_0 of nodes, and apply bullet point and apply round elimination framework $T = t(n)$ times to get a 0-round algorithm $\mathcal{A}^{(T)}$ for a certain problem $\Pi^{(T)}$. By analyzing the development of the (local) failure probabilites of the algorithms appearing during the T applications of round elimination, we can show that algorithm $\mathcal{A}^{(T)}$ still has a large probability of success, and the fact that $\mathcal{A}^{(T)}$ is a 0-round algorithm enables us to infer that $\Pi^{(T)}$ is in fact so easy that it can be solved with a *deterministic* 0-round algorithm.

Finally, we run the round-elimination framework back T times to obtain a deterministic T-round algorithm for the original problem Π. Due to fixing the number of nodes to n_0, the obtained T-round algorithm is only guaranteed to produce a correct output on n_0-node trees; however, due to the nature of 0-round algorithms and the precise definition of the round elimination process (which are both independent of the number of nodes of the input graph), the obtained algorithm can be shown to also work on trees with an arbitrary number of nodes, with precisely the same, constant, runtime $T(n_0) = O(1)$.

Similarly to trees, we also settle the final missing piece in the classification of local problems on oriented grids, restated next for convenience.

Theorem 2.31 (Classification of local problems on grids [258, 77, 94, 188]). *Any solvable locally checkable labeling problem on d-dimensional oriented grids*[1] *has one of the following local complexities:*

1. $O(1)$,
2. $\Theta(\log^* n)$,
3. $\Theta(n^{1/d})$.

Namely, we prove the following theorem, improving upon the $o(\sqrt[d]{\log^* n})$ speedup from [88].

Theorem 4.3 (Cf. Theorem 7.2). *Let d be a fixed positive constant. Any LCL problem (i.e., a local problem with inputs) on a d-dimensional oriented grid with local complexity $o(\log^* n)$ has, in fact, local complexity $O(1)$.*

[1]The edges of the input grid are consistently oriented.

4.2 Bridging Local Algorithms to Other Areas

Next, we discuss the results presented in Chapters 8 to 10. These chapters present new results related to extensions of the local algorithms theory to other models and applications of local algorithms. In particular, we will show bridges to the areas of the local computation algorithms and the related volume complexity measure (see Section 3.2), parallel algorithms and the distributed algorithms in the congest model (see Sections 3.1 and 3.3), and descriptive combinatorics and finitary factors (see Sections 3.5 and 3.6).

4.2.1 The Randomized Local Computation Complexity of the Lovász Local Lemma

In Chapter 8, we analyze the volume complexity model using the techniques from local algorithms. We prove that the volume complexity of sinkless orientation is $\Theta(\log n)$ as well as a part of the classification of local problems in the volume model in Theorem 3.2. It is based mainly on the paper of Brandt et al. [81], with a part from the paper of [188] covered in the previous chapter that fits more into this chapter.

Our main contribution is extending the classification of volume complexities that we restate here for convenience.

Theorem 3.2 ([263, 127, 270, 188, 81])**.** *In the class of bounded-degree graphs, each local problem satisfies the following:*

1. *If its round complexity is $O(1)$ (order-invariant regime), then its randomized/deterministic volume complexity is $O(1)$.*
2. *If its randomized/deterministic round complexity is $\omega(1)$ and $O(\log^* n)$ (maximal independent set / coloring regime), its randomized/deterministic volume complexity is $\Theta(\log^* n)$.*
3. *If its randomized local complexity is $\omega(\log^* n)$ but $o(\log n)$ (Lovász local lemma regime), its randomized volume complexity is between $\Omega(\sqrt{\log n})$ and $O(\log n)$.*

Our contribution is to show the following general speedup theorem.

Theorem 4.4 (Cf. Theorem 8.2)**.** *For any locally checkable labeling problem Π on constant degree graphs, if there is a randomized volume algorithm that solves Π and has a volume complexity of $o(\sqrt{\log n})$, then there is also a deterministic volume algorithm for Π with a volume complexity of $O(\log^* n)$.*

To prove that result, we consider the deterministic volume model where the identifiers can come from an exponential instead of a polynomial range. The result then follows from two observations. First, a variant of the Chang-Pettie speedup [88] presented in Theorem 2.22 shows that any volume algorithm with a probe complexity of $o(n)$ that works with exponential identifiers can be sped up to have a probe complexity of $\Theta(\log^* n)$.

Second, a variant of the Chang-Kopelowitz-Pettie derandomization [94] covered in Theorem 2.21 shows that any randomized algorithm with probe complexity $o(\sqrt{\log n})$ can be derandomized to give a deterministic $o(n)$-probe algorithm that works with exponential identifiers.

We also prove the following result by an adaptation of an argument from [258].

Theorem 4.5 (Cf. Theorem 8.3)**.** *On constant degree graphs, there does not exist a local checkable labeling problem with a deterministic or randomized volume complexity between $\omega(1)$ and $o(\log^* n)$.*

We get a bit better understanding of volume complexities for certain important local problems. In particular, we settle the volume complexity of sinkless orientation.

Theorem 4.6 (Cf. Theorem 8.1)**.** *The randomized volume complexity of the local lemma (see Section 2.2) on constant degree graphs is $\Theta(\log n)$. The upper bound holds for the polynomially-relaxed version of the problem (see Section 2.2), while the lower bound holds even for the exponential criterion $p \leq 2^{-\Delta}$ since this lower bound holds for the sinkless orientation problem.*

To prove an $\Omega(\log n)$ lower bound for the sinkless orientation problem, we use similar ideas to the proof of Theorem 8.2. However, we notice that the fact that one can prove the lower bound for sinkless orientation with respect to an ID graph (discussed in Section 2.2) allows us to conclude that any deterministic algorithm with $o(\log n)$ volume using polynomial identifiers can be sped up to $O(\log^* n)$ local complexity. However, this contradicts the local lower bound for sinkless orientation [73].

4.2.2 Deterministic Low-Diameter Decompositions for Weighted Graphs and Distributed and Parallel Applications

In Chapter 9, we use the techniques from local algorithms to construct parallel algorithms for various clustering problems (see Theorem 3.5 for an example). This work implies new parallel algorithms for metric embedding problems like the low stretch spanning tree problem.

The chapter gives deterministic parallel and distributed algorithms for various clustering results in weighted graphs. The main message is that once you can deterministically and efficiently compute $(1 + 1/\operatorname{poly}(\log n))$-approximate distances in undirected graphs in your favorite parallel/distributed model, you can also deterministically and efficiently solve various clustering problems. Since clusterings are very basic objects and approximate distances can efficiently and deterministically be computed in various parallel and distributed models, our clustering results directly imply efficient deterministic algorithms for various problems.

Let us give an example result proven in that chapter.

Theorem 4.7 (Cf. Theorem 9.1)**.** *Let G be a weighted graph. We are given a set of terminals $Q \subseteq V(G)$ and a parameter $R > 0$ such that for every $v \in V(G)$ we have $d(Q, v) \leq R$. Also, a precision parameter $0 < \varepsilon < 1$ is given. There is a deterministic distributed and parallel algorithm outputting a partition $\mathcal{C}$ of the vertices into clusters and a subset of terminals $Q' \subseteq Q$ with the following properties:*

1. *Each cluster $C \in \mathcal{C}$ contains exactly one terminal $q \in Q'$. Moreover, for any $v \in C$ we have $d_{G[C]}(q, v) \leq (1+\varepsilon)R$.*
2. *For the set E^{bad} of edges connecting different clusters of $\mathcal{C}$ we have*
$$|E^{bad}| = \widetilde{O}\left(\frac{1}{\varepsilon R}\right) \cdot \sum_{e \in E(G)} \ell(e).$$

The PRAM *variant of the algorithm has work $\widetilde{O}(m)$ and depth $\widetilde{O}(1)$. The* CONGEST *variant of the algorithm runs in $\widetilde{O}(\sqrt{n} + Diameter(G))$ rounds.*

This theorem is proven using techniques developed for deterministic local algorithms for network decompositions, a problem discussed in Section 1.5. However, the arguments used in the proof are substantially more complicated. Instead of growing balls by considering their neighborhoods, the algorithm for above theorem uses as a subroutine a parallel algorithm of Rozhoň et al. [274] for the $(1+\varepsilon)$-approximate shortest path problem on undirected graphs (that algorithm, in fact, also uses as subroutines algorithms extending local algorithms for network decompositions).

An example application of this result is a deterministic parallel algorithm for the low-stretch spanning tree problem.

Theorem 4.8 (Deterministic Low-Stretch Spanning Tree, cf. Theorem 9.2)**.** *Let G be a weighted graph. Each edge e has moreover a nonnegative importance $\mu(e)$. There exists a deterministic parallel and distributed algorithm which outputs a spanning tree T of G such that*

$$\sum_{e=\{u,v\} \in E(G)} \mu(e) d_T(u, v) = \widetilde{O}\left(\sum_{e=\{u,v\} \in E(G)} \mu(e) d_G(u, v)\right). \tag{4.1}$$

The PRAM *variant of the algorithm has work $\widetilde{O}(m)$ and depth $\widetilde{O}(1)$. The* CONGEST *variant of the algorithm runs in $\widetilde{O}(\sqrt{n} + Diameter(G))$ rounds.*

4.2.3 Local Problems on Trees from the Perspectives of Distributed Algorithms, Finitary Factors, and Descriptive Combinatorics

In Chapter 10, we use the techniques from local algorithms and descriptive combinatorics to understand how the two are connected for tree graphs. We prove a number of results connecting the two fields in the regime where the input graph is a regular tree.

First, the chapter introduces the ID graph technique that we discussed in Section 2.2. This technique was also applied in Chapter 8 to prove lower bounds in the volume model and also found further applications in descriptive combinatorics [80]. The original motivation of developing the technique was to adapt a lower bound for sinkless orientation by Marks [244]. His result in fact proves a stronger result than the well-known $\Omega(\log n)$ deterministic lower bound – it proves that there is no so-called Borel solution for it. While his technique is conceptually simple, it relies on the celebrated Borel determinacy by Martin [246], a result known to be provable only with proofs that rely on the existence of large uncountable infinities. We show in Chapter 10 that one can in fact recover a simple $\Omega(\log n)$ lower bound for sinkless orientation using his technique with the use of the ID graph. We also generalize Marks' technique to prove new local lower bounds that are interesting mainly because they are the only known examples of local problems on trees for which we miss a round-elimination lower bound.

Second, the chapter proves a fundamental link between local algorithms and descriptive combinatorics, proving the following theorem:

Theorem 4.9 (Informal version of Theorem 10.56)**.** *On regular trees, a local problem has round complexity $O(\log n)$ if and only if it admits a Baire measurable solution.*

We will postpone the formal definition of the Baire measurable solution to Theorem 10.10. Here, we recall that we have informally introduced the measurable graph in Section 3.5. A Baire measurable solution on a measurable graph is a solution that fails on a small set of vertices of that measurable graph. In particular, it fails only on a so-called meager set; this is a standard topological definition of a "small" set.

Third, the chapter proves a number of results connected to the relation of local algorithms, uniform local algorithms (see Section 3.6), and classes from descriptive combinatorics. For example, it proves that the descriptive class of so-called Borel solutions is incomparable with classes from local algorithms, or shows how uniform local algorithms can be used to refine the classification of local problems on trees from Theorem 2.33.

4.2.4 Remarks on Notation

We remark that the following chapters sometimes use a bit different notation than the informal discussions so far. They often talk about "algorithms in the LOCAL model" while so far we have discussed "local algorithms", and similarly they use the shorthands VOLUME and LCA for the volume and local computation algorithm model from Section 3.2. They also use the shorthand "LCL problem" instead of "locally checkable labeling problem" we have used so far. Sometimes, we need to somewhat refine our definitions of local algorithms, complexities, or problems; the increased precision in definitions is important e.g. when proving round-elimination lower bounds.

CHAPTER 5

A Simple Deterministic Distributed Low-Diameter Clustering

5.1 Introduction

This chapter focuses on distributed graph algorithms, particularly on the fundamental problem of deterministic and local ways to compute network decompositions and *low-diameter clusterings*, which cluster at least half of the nodes in a given graph into non-adjacent clusters with small diameter. In particular, the chapter describes a drastically simplified efficient deterministic distributed construction for computing such a low-diameter clustering with polylogarithmic diameter in polylogarithmic rounds of the distributed CONGEST model.

Starting with the seminal work of Luby [239] from the 1980's, fast and simple $O(\log n)$-round *randomized* distributed algorithms are known for many fundamental symmetry breaking problems like maximal independent set (MIS) or $\Delta + 1$ vertex coloring. For a long time, this was in stark contrast with the state-of-the-art deterministic algorithms. For multiple decades, it was a major open problem in the area of distributed graph algorithms to get deterministic algorithms with round complexity $\operatorname{poly}\log(n)$ for such problems, e.g., MIS or $\Delta + 1$ vertex coloring. A recent breakthrough of Rozhoň and Ghaffari [276] managed to resolve this open problem.

In their work, the authors presented the first polylogarithmic-round deterministic algorithm for network decompositions using a (weak-diameter version of) low-diameter clusterings. Network decomposition is the object we get by repeatedly finding a low diameter clustering and removing all the nodes in the clustering, until no node remains. See Section 5.1.1 for the formal definitions. It was long known that low-diameter clusterings is the up-to-then-missing fundamental tool required for a large class of LOCAL deterministic distributed algorithms. The clustering construction of [276] directly implied, among others, first efficient distributed algorithms for MIS (together with the work of [84]) and $\Delta + 1$ vertex coloring (together with the work of [41]) in the standard bandwidth-limited CONGEST model of distributed computing.

The main difference in the natural low-diameter clustering problem defined above and the weaker version solved in [276] is that clusters are not necessarily connected or induce a low low-diameter subgraph on their own but instead have low *weak-diameter*. A cluster has weak-diameter at most D if any two nodes in the cluster are connected by a path of length at most D *in the original graph* G *instead of within the cluster itself.* Hence, a cluster may even be disconnected. While the weak-diameter guarantee is enough for derandomizing local computations without bandwidth limitations, including MIS and $\Delta + 1$-coloring, the original – strong-diameter – clustering stated above is clearly the natural and right object to ask for: It is strictly stronger, easier to define, easier to use in applications, and requires less and simpler objects and notation. Indeed, in distributed models with bandwidth limitations, such as the standard CONGEST model in which message sizes are restricted, it is not sufficient that clusters have small weak-diameter but one also needs to guarantee that there exist so-called low-depth Steiner trees connecting the nodes of each cluster. The collection of these Steiner-trees must furthermore satisfy additional low-congestion guarantees, i.e., each edge or each node in the graph is not used by too many trees (as a Steiner node). Algorithms must also be able to compute the Steiner forest of a weak-diameter clustering efficiently. Lastly, there are several applications, e.g., low-stretch spanning trees, where strong-diameter clusterings are strictly required and the weak-diameter guarantee does not suffice [272]. This motivated the later works of [87, 272] to give low-diameter clustering algorithms with strong-diameter guarantees, typically first building a weak-diameter clustering and then using this weak-diameter clustering either for communication or using it as a starting point for building a strong-diameter clustering out of it recursively. This multi-step process still requires to define and maintain Steiner forests for weak-diameter clusterings during intermediate steps.

In this work, we show that there is a much simpler and more direct way to get strong-diameter guarantees by designing a natural clustering process that combines key ideas from [276] and [272].

5.1.1 Preliminaries: Distributed CONGEST Model and Low-Diameter Clusterings

We will now briefly introduce the standard model for distributed message-passing algorithms – the CONGEST model of distributed computing [264] (cf. Section 3.1) and also give the definitions of clustering that we use (see Chapter 9 or Section 1.5 for more discussion).

CONGEST : Throughout the chapter, we work with the CONGEST model, which is the standard distributed message-passing model for graph algorithms [264]. The network is abstracted as an n-node undirected graph $G = (V, E)$ where each node $v \in V$ corresponds to one processor in the network. Communications take place in synchronous rounds. Per round, each node sends one $O(\log n)$-bit message to each of its neighbors in G. We also consider the relaxed variant of the model where we allow unbounded message sizes, called LOCAL . At the end of the round, each node performs some computations on the data it holds, before we proceed to the next communication round.

We capture any graph problem in this model as follows: Initially, the network topology is not known to the nodes of the graph, except that each node $v \in V$ knows its own unique $O(\log n)$-bit identifier. It also knows a suitably tight (polynomial) upper bound on the number n of nodes in the network. At the end of the computation, each node should know its own part of the output, e.g., in the graph coloring problem, each node should know its own color.

Whenever we say that there is “an efficient distributed algorithm”, we mean that there is a CONGEST algorithm for the problem with round complexity $\operatorname{poly}(\log n)$.

Low Diameter Clustering: The main object of interest that we want to construct is a so-called *low diameter clustering*, which we formally define after introducing a bit of notation. Throughout the whole chapter we work with undirected unweighted graphs and write $G[U]$ for the subgraph of G induced by $U \subseteq V(G)$. We use $d_G(u,v)$ to denote the distance of two nodes $u, v \in V(G)$ in G. We also simplify the notation to $d(u,v)$ when G is clear from context and generalize it to sets by defining $d_G(U,W) = \min_{u \in U, w \in W} d_G(u,w)$ for $U, W \subseteq V(G)$. The diameter of G is defined as $\max_{u,v \in V(G)} d_G(u,v)$.

We use the term *clustering of G* to denote any set of disjoint vertex subsets of G. A low diameter clustering is a clustering with additional properties:

Definition 5.1 (Low Diameter Clustering)**.** *A low diameter clustering $\mathcal{C}$ with diameter D of a graph G is a clustering of G such that:*

1. *No two clusters $C_1 \neq C_2 \in \mathcal{C}$ are adjacent in G, i.e., $d(C_1, C_2) \geq 2$.*
2. *For every cluster $C \in \mathcal{C}$, the diameter of $G[C]$ is at most D.*

Similarly, we define a low diameter clustering with weak-diameter at most D by replacing the condition (2) with he requirement that for each cluster $C \in \mathcal{C}$ and any two nodes $u, v \in C$ we have $d_G(u,v) \leq D$.

Whenever we construct a low diameter clustering, we additionally want it to cover as many nodes as possible. Usually, we want to cover at least half of the nodes of G, or formally, we require that $\left|\bigcup_{C \in \mathcal{C}} C\right| \geq n/2$. Sometimes, it is also necessary to generalize (1) and require a larger separation of the clusters, but this is not considered in this chapter.

Let us now remind the reader of a formal definition of network decomposition (we use a bit different notation from Theorem 1.5).

Definition 5.2 (Network Decomposition)**.** *A network decomposition with C colors and diameter D is a coloring of nodes with colors $1, 2, \ldots, C$ such that each color induces a low-diameter clustering of diameter D.*

Notice that whenever we can construct a low-diameter clustering with diameter D that covers at least $n/2$ nodes, we get a network decomposition by repeatedly constructing a low diameter clustering and removing it from the graph. This way, we achieve a network decomposition with $C = O(\log n)$ and diameter D. Since virtually all deterministic constructions of network decomposition work this way, we focus on constructing low-diameter clusterings from now on.

The reason why network decomposition is a useful object is that it corresponds to the canonical way of using clusterings in distributed computing. To give an example, we show how to use it to solve the maximal independent set problem in the less restrictive LOCAL model.

Given access to a network decomposition, we iterate over the C color classes and gradually build independent sets $I_1 \subseteq I_2 \subseteq \cdots \subseteq I_C$ where I_C is maximal. In the i-th step, each cluster K of the low-diameter clustering induced by the i-th color computes a maximal independent set in the graph induced by all the nodes in K that are not neighboring a node in I_{i-1} and we define I_i by adding these independent sets to I_{i-1}. The set I_C is clearly maximal. Computing the maximal independent set inside one cluster K can be done in $O(D)$ rounds of the LOCAL model as follows: One node of the cluster collects all the information about $G[K]$ and its neighborhood in G, then locally computes a maximal independent set, and afterwards broadcasts the solution to the nodes in the cluster. Hence, the overall algorithm has round complexity $O(CD)$. Hence, given a network decomposition with $C, D = \text{poly}(\log n)$, one can compute a maximal independent set in $\text{poly}(\log n)$ rounds. Note that this brute-force approach for computing a maximal independent set critically relies on the fact that the LOCAL model does not restrict the size of messages.

In the more restrictive CONGEST model, computing a maximal independent set inside a low diameter cluster becomes nontrivial, but one can use the deterministic MIS algorithm of [84] with round complexity $O(D \cdot \text{poly}(\log n))$ where D is the diameter of the input graph.

5.1.2 Comparison with Previous Work

We summarize the work on deterministic distributed low-diameter clusterings in the CONGEST model in Table 5.1.

There are three highlighted rows in the table, besides our result we highlight the work of [276] and [272]; The algorithm of this chapter combines ideas from both of these papers.

Let us now go through the rows of the table. The first two rows, together with the related results of [262, 159, 157] represent the results before the work of [276] and are not relevant to our chapter.

Next, there is the work of [276] and an improved variant of it by [170]. These were the first deterministic efficient constructions of low diameter clusterings, however, they suffer from only providing a weak-diameter guarantee.

Paper	Frac. clustered	Diameter	Strong?	rounds
[13]	$2^{-\Omega(\sqrt{\log n \log\log n})}$	$2^{O(\sqrt{\log n \log\log n})}$	✓	$2^{O(\sqrt{\log n \log\log n})}$
[151]	$2^{-\Omega(\sqrt{\log n})}$	$2^{O(\sqrt{\log n})}$	✓	$2^{O(\sqrt{\log n})}$
[276]	$1/2$	$O(\log^3 n)$	×	$O(\log^7 n)$
[170]	$1/2$	$O(\log^2 n)$	×	$O(\log^4 n)$
[87]	$1/2$	$O(\log^2 n)$	✓	$O(\log^{10} n)$
[87]	$1/2$	$O(\log^3 n)$	✓	$O(\log^7 n)$
[272]	$1/2$	$O(\log^2 n)$	✓	$O(\log^4 n)$
[172]	$\Omega(1/\log\log n)$	$O(\log n)$	✓	$\widetilde{O}(\log^2(n))$
[172]	$1/2$	$O(\log n \cdot \log\log\log n)$	✓	$\widetilde{O}(\log^2(n))$
this	$1/2$	$O(\log^3 n)$	✓	$\log^6(n)$

Table 5.1: This table shows the previous work on distributed deterministic algorithms for low-diameter clusterings. We highlighted the three results relevant for this chapter.

Next, the work of [87] and [272] use the algorithm of [170] as a black blox and use additional ideas on top of the weak-diameter algorithm to create strong-diameter clusterings. The row with [272] is highlighted because our algorithm uses an idea similar to theirs.

Finally, a very recent algorithm of [172] manages to bring down the diameter of the clusters as well as the round complexity, with a very different technique than [276]. However, their algorithm is very complicated.

By far the simplest efficient algorithm from those in the table is the one from [276]. We show that with a small modification to their algorithm in the spirit of the algorithm of [272], we can get a very simple algorithm computing strong-diameter clusters. Formally, we show the following result.

Theorem 5.3. *There is a deterministic distributed algorithm that outputs a clustering $\mathcal{C}$ of the input graph G consisting of separated clusters of diameter $O(\log^3 n)$ such that at least $n/2$ nodes are clustered. The algorithm runs in $O(\log^6 n)$ CONGEST rounds.*

Recall that by repeatedly applying above result we get the following corollary.

Corollary 5.4. *There is a deterministic distributed algorithm that outputs a network decomposition with $C = O(\log n)$ colors and diameter $D = O(\log^3 n)$. The algorithm runs in $O(\log^7 n)$ CONGEST rounds.*

Comparison of our algorithm with [276]:

We now give a high-level explanation of the algorithm of [276] and afterwards compare it to our algorithm.

In the algorithm of [276], we start with a trivial clustering where every node is a cluster. Every cluster inherits the unique identifier from the starting node. During the algorithm, a cluster can grow, shrink and some vertices are deleted from the graph and will not be part of the final output clustering. In the end, the nonempty clusters cluster at least $n/2$ nodes and their weak-diameter is $O(\log^3 n)$.

More concretely, the algorithm consists of $b = O(\log n)$ phases where b is the number of bits in the node identifiers. In phase i, we split clusters into red and blue clusters based on the i-th bit in their identifier; the goal of the phase is to disconnect the red from the blue clusters by deleting at most $n/(2b)$ nodes in the graph.

Here is how this is done. The i-th phase consists of $O(b \log n)$ steps. In general red clusters can only grow and blue clusters can only shrink. More concretely, in each step every node in a blue cluster neighboring with a red cluster proposes to join an arbitrary neighboring red cluster. Now, for a given red cluster C, if the total number of proposing blue nodes is at least $|C|/(2b)$, then C decides to grow by adding all the proposing blue nodes to the cluster. Otherwise, the proposing nodes are deleted which results in C not being adjacent to any other blue nodes until the end of the phase.

One can see that the number of deleted nodes per phase is only $n/(2b)$ in total, as needed. On the other hand, each cluster can grow only $O(b \log n)$ times until it has more than n nodes, which implies that the weak-diameter of each cluster grows only by $O(b \log n) = O(\log^2 n)$ per phase.

This concludes the description of the algorithm of [276]. Note that the clusters from their algorithm only have small weak-diameter since the nodes in a cluster can leave it in the future and the cluster may then even disconnect.

Our strong-diameter algorithm: To remedy the problem with the weak-diameter guarantee, we change the algorithm of [276] as follows: Instead of clusters, we will think in terms of their centers that we call *terminals*. Given a set of terminals Q such that Q is R-ruling, i.e., for every $u \in V(G)$ we have $d_G(Q, u) \leq R$, we can always construct a clustering with strong-diameter R by running a breadth first search from Q. Hence, keeping a set of terminals is equivalent to keeping a set of strong-diameter clusters.

Our algorithm starts with the trivial clustering where $Q = V(G)$. During the algorithm, we keep a set of terminals Q and in each of the b phases we delete at most $n/(2b)$ nodes and make some nodes of Q nonterminals such that those remaining terminals with their i-th bit equal to 0 are in a different component than those that have their i-th bit equal to 1 (see Figure 5.2). Moreover, we want that if at the beginning of the phase the set Q is R-ruling, then it is $R + O(b \log n)$-ruling at the end of the phase (cf. the $O(b \log n)$ increase in weak-diameter in the algorithm of [276]).

At the beginning of each phase, we run a breadth first search from the set Q, which gives us a clustering with strong diameter R (see the left picture in Figure 5.3). We in fact think of each cluster as a rooted tree of radius R.

We then implement the same growing process as [276], but with a twist: whenever a blue node v proposes to join a red cluster, the whole subtree rooted at v proposes instead of just v (see the middle picture in Figure 5.3). This is because rehanging/deleting the whole subtree does not break the strong-diameter guarantee of blue clusters. If a blue node joins a red cluster, it stops being a terminal.

The only new argument that needs to be done is that the diameter of red clusters does not grow a lot, which is trivial in the algorithm of [276] and follows by a simple argument in our algorithm.

We note that the algorithm of [272] also keeps track of terminals. However, to separate the red and blue terminals in one phase their algorithm relies on computing global aggregates, which can only be done efficiently on a low-diameter input graph.

5.2 Clustering Algorithm

In this section we prove Theorem 5.5 given below, which is a more precise version of Theorem 5.3.

Theorem 5.5 (Clustering Theorem)**.** *Consider an arbitrary n-node network graph $G = (V, E)$ where each node has a unique $b = O(\log n)$-bit identifier. There is a deterministic distributed algorithm that, in $O(\log^6 n)$ rounds in the* CONGEST *model, finds a subset $V' \subseteq V$ of nodes, where $|V'| \geq |V|/2$, such that the subgraph $G[V']$ induced by the set V' is partitioned into non-adjacent disjoint clusters of diameter $O(\log^3 n)$.*

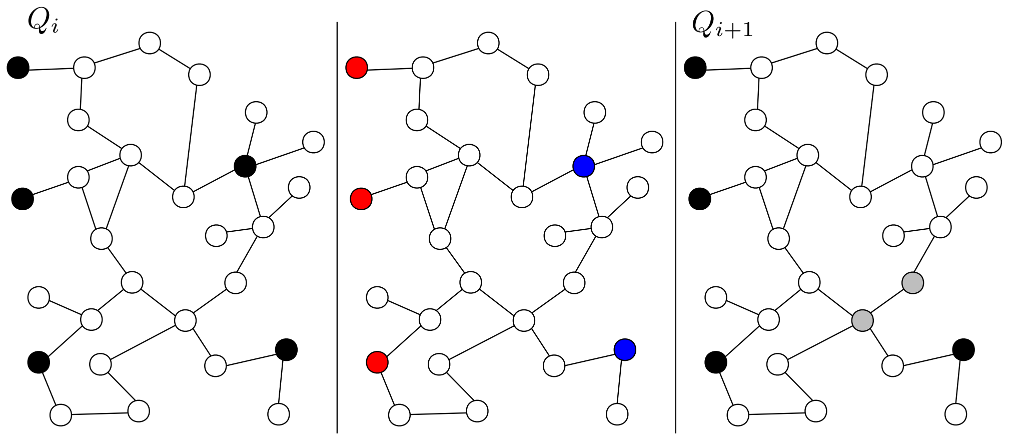

Figure 5.2: The figure shows one phase of the algorithm from Theorem 5.5. The left figure contains a 3-ruling set of terminal nodes Q_i that we start with at the beginning of phase i. We split Q_i into red and blue terminals according to the $(i+1)$-th bit of their identifiers. Then, we implement one phase of the algorithm. As a result, some of the nodes are deleted (grey) and some blue terminals stop being terminals. The set of remaining terminals Q_{i+1} is on one hand 6-ruling, on the other hand the blue terminals in Q_{i+1} are separated from the red terminals.

We start by describing the algorithm outline of Theorem 5.5. The construction has $b = O(\log n)$ phases, corresponding to the number of bits in the identifiers. For $i \in [0, b-1]$, we denote by V_i the set of living vertices at the beginning of phase i. Initially, all nodes are living and therefore $V_0 = V$. In each phase, at most $|V|/(2b)$ nodes die. Dead nodes remain dead and will not be contained in V'. Some of the alive nodes are terminals. We

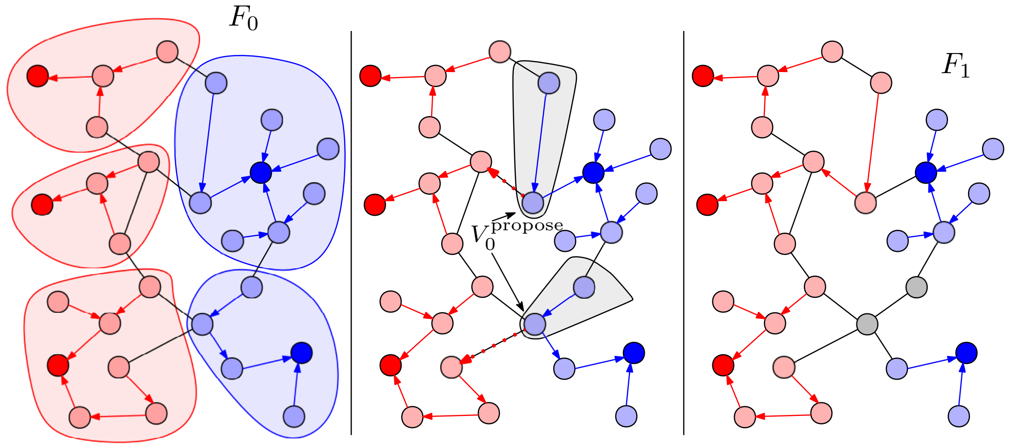

Figure 5.3: This figure explains one step of the algorithm of Theorem 5.5, namely it shows what happens between the middle and the right picture of Figure 5.2. The left picture illustrates the beginning of the phase where we compute a BFS forest F_0 from the set Q of terminals. In the first (and any other) step (the middle picture) we construct a set $V_0^{propose}$. Some proposals are accepted and the respective blue nodes join red clusters, while some proposals are rejected and respective blue nodes are deleted (the right picture).

denote the set of terminals at the beginning of phase i by Q_i. Initially, all living nodes are terminals and therefore $Q_0 = V$.

Slightly abusing the notation, we let V_b and Q_b denote the set of living vertices and terminals at the end of phase $b-1$, respectively. We define V' to be the final set of living nodes, i.e., $V' = V_b$, and each connected component of $G[V']$ will contain exactly one terminal in Q_b.

For stating the key invariants the algorithm satisfies, we need the following standard definition of a ruling set:

Definition 5.6 (Ruling set)**.** *We say that a subset $Q \subseteq V(G)$ is R-ruling in G if every node $v \in V(G)$ satisfies $d_G(Q, v) \leq R$.*

Construction invariants: The construction is such that, for each $i \in [0, b]$, the following three invariants are satisfied:

1. Ruling Invariant: Q_i is R_i-ruling in $G[V_i]$ for $R_i = i \cdot O(\log^2 n)$.
2. Separation Invariant: Let $q_1, q_2 \in Q_i$ be two nodes in the same connected component of $G[V_i]$. Then, the identifiers of q_1 and q_2 coincide in the first i bits.
3. Deletion Invariant: $|V_i| \geq \left(1 - \frac{i}{2b}\right)|V|$.

Note that setting $V_0 = Q_0 = V$ indeed results in the invariant being satisfied for $i = 0$.

In the end, we set $V' = V_b$. The deletion invariant for $i = b$ states that $|V'| \geq |V|/2$. The separation invariant implies that each connected component of $G[V']$ contains at most one node of Q_b. Together with the ruling invariant, which states that Q_b is R_b-ruling in $G[V']$ for $R_b = O(\log^3 n)$, this implies that each connected component of $G[V']$ has diameter $O(\log^3 n)$. Next, in Section 5.2.1 we present the outline of one phase. Afterwards, in Section 5.2.2 we prove the correctness of the algorithm and analyse the CONGEST complexity.

5.2.1 Outline of One Phase

In phase i, we compute a sequence of rooted forests $F_0, F_1, \ldots, F_t$ in $t = 2b^2 = O(\log^2 n)$ steps. At the beginning, F_0 is simply a BFS forest in $G[V_i]$ from the set Q_i. At the end, we set $V_{i+1} = V(F_t)$ and Q_{i+1} is the set of roots of the forest F_t.

Let $j \in \{0, 1, \ldots, t-1\}$ be arbitrary. We now explain how F_{j+1} is computed given F_j. In general, each node contained in F_{j+1} is also contained in F_j, i.e., $V(F_{j+1}) \subseteq V(F_j)$, and each root of F_{j+1} is also a root in F_j. We say that a tree in F_j is a red tree if the $(i+1)$-th bit of the identifier of its root is 0 and otherwise we refer to the tree as a blue tree. Also, we refer to a node in a red tree as a red node and a node in a blue tree as a blue node. Each red node in F_j will also be a red node in F_{j+1}. Moreover, the path to its root is the same in both F_j and F_{j+1}. Each blue node in F_j can (1) either be a blue node in F_{j+1}, in which case the path to its root is the same in both F_j and F_{j+1}, (2) be deleted and therefore not be part of any tree in F_{j+1}, (3) become a red node in F_{j+1}.

Let $V_j^{propose}$ be the set which contains each node v which (1) is a blue node in F_j, and (2) v is the only node neighboring a red node (in the graph G) in the path from v to its root in F_j. For a node $v \in V_j^{propose}$, let T_v be the subtree rooted at v with respect to F_j. Note that it directly follows from the way we defined $V_j^{propose}$ that v is the only node in T_v which is contained in $V_j^{propose}$.

Each node in $V_j^{propose}$ proposes to an arbitrary neighboring red tree in F_j. Now, a given red tree T in F_j decides to grow if

$$\sum_{\substack{v \in V_j^{propose}: \\ v \text{ proposes to } T}} |V(T_v)| \geq \frac{|V(T)|}{2b}.$$

If T decides to grow, then it accepts all the proposals it received, and otherwise T declines all proposals it received. We now set

$$V(F_{j+1}) = V(F_j) \setminus \left(\bigcup_{\substack{v \in V_j^{propose}, \\ \text{the proposal of } v \text{ was declined}}} V(T_v) \right).$$

Each node in $V(F_{j+1}) \setminus V_i^{propose}$ has the same parent in F_{j+1} and F_j, or is a root in both F_{j+1} and F_j. Each node in $V(F_{j+1}) \cap V_j^{propose}$, i.e., each node whose proposal got accepted by some red tree T in F_j, changes its parent to be an arbitrary neighboring node in the tree T. Note that if a red tree T decides to grow, then the corresponding tree in F_{j+1} contains at least $\left(1 + \frac{1}{2b}\right)|V(T)|$ vertices. Moreover, if T does not decide to grow, then T is also a tree in F_{j+1} and is not neighboring with any blue tree in F_{j+1}. This follows from the fact that each blue node neighboring a red tree either becomes a red node or gets deleted.

We now have fully specified how the rooted forests $F_0, F_1, \ldots, F_t$ are computed and recall that in the end we set $V_{i+1} = V(F_t)$ and Q_{i+1} is the set of roots of the forest F_t.

5.2.2 Analysis

For each $j \in \{0, 1, \ldots, t\}$ and $u \in V(F_j)$, we define $d_j(u)$ as the length of the path from u to its root in F_j. Note that as F_0 is a BFS forest, for any neighboring nodes $w, v \in V(F_0)$ it holds that $d_0(w) \leq d_0(v) + 1$.

Claim 5.7 (Ruling Claim)**.** *For every $i \in \{0, 1, \ldots, t\}$, the following holds:*

Blue Property: *Every blue node in F_j satisfies $d_j(u) = d_0(u)$.*

Red Property: *Every red node in F_j satisfies $d_j(u) \leq d_0(u) + 2j$.*

In particular, this implies that Invariant (I) is preserved.

Proof. The blue property directly follows from the fact that for any blue node the path to its root in F_j is the same as the path to its root in F_0. We prove the red property by induction on j. The base case $j = 0$ trivially holds.

For the induction step, consider an arbitrary $j \in \{0, 1, \ldots, t-1\}$. We show that the statement holds for $j+1$ given that it holds for j.

Consider an arbitrary red node u in F_{j+1}. We have to show that $d_{j+1}(u) \leq d_0(u) + 2(j+1)$. If u is also a red node in F_j, then we can directly use induction. Hence, it remains to consider the case u is a blue node in F_j.

In that case, there exists a node $v \in V_j^{propose}$ such that $u \in V(T_v)$ and the proposal of v was accepted. In particular, v's parent in F_{j+1} is some neighboring node w which is part of some red tree in F_j (see Figure 5.4).

The path from u to its root r in F_{j+1} can be decomposed into a path from u to v, an edge from v to w and a path from w to its root r.

The path from u to v in F_{j+1} is the same as the path from u to v in F_0 and therefore of length $d_0(u) - d_0(v)$. The path from w to r in F_{j+1} is the same as the path from w to r in F_j and therefore has a length of $d_j(w)$ with $d_j(w) \leq d_0(w) + 2j$ according to the induction hypothesis. Moreover, we noted above that because w and v are neighbors, we

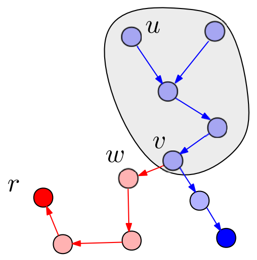

Figure 5.4: The figure shows the situation in the proof of Theorem 5.7. The path from u to r splits into three parts: from u to v, then to w, then to r. The length of each part is upper bounded separately.

have $d_0(w) \leq d_0(v) + 1$. Hence, we can upper bound the length of the path from u to its root in F_{j+1} by

$$d_{j+1}(u) \leq (d_0(u) - d_0(v)) + 1 + (d_0(w) + 2j) \leq d_0(u) + 2(j+1)$$

which finishes the induction proof. It remains to prove the last part of the claim. To that end, assume that the ruling invariant is satisfied for i, i.e., Q_i is R_i-ruling in $G[V_i]$ for $R_i = i \cdot O(\log^2 n)$. Then, every node u in $V(F_t) = V_{i+1}$ satisfies

$$d_{G[V_{i+1}]}(Q_{i+1}, u) \leq d_t(u) \leq d_0(u) + 2t \leq i \cdot O(\log^2 n) + O(\log^2 n) = (i+1)O(\log^2 n)$$

and therefore the ruling invariant is satisfied for $i+1$. □

Claim 5.8. *(Separation Claim) No red node in F_t is neighboring a blue node in F_t. In particular, this implies that Invariant (II) is preserved.*

Proof. We observed during the algorithm description that each red tree that decides to grow grows by at least a $(1 + \frac{1}{2b})$-factor in a given step. Our choice of $t = 2b^2$ implies that

$$\left(1 + \frac{1}{2b}\right)^t = \left(\left(1 + \frac{1}{2b}\right)^{2b}\right)^{(t/2b)} > 2^{t/2b} = 2^b \geq n,$$

and therefore each tree eventually stops growing. However, once a tree decides not to grow, it is not neighboring with any blue node and therefore no red node in F_t is neighboring a blue node in F_t. In particular, this implies that each connected component of $G[V_{i+1}] = G[V(F_t)]$ either entirely consists of blue nodes in F_t or entirely consists of red nodes in F_t. As the $(i+1)$-th bit of the identifier of each red root in F_t is 0 and the $(i+1)$-th bit of the identifier of each blue root in F_t is 1, we get that each connected component of $G[V_{i+1}]$ either contains no node in Q_{i+1} with the $(i+1)$-th bit of the identifier being 0 or no node in Q_{i+1} with the $(i+1)$-th bit of the identifier being 1, which implies that the separation invariant is preserved. □

Claim 5.9 (Deletion Claim)**.** *It holds that* $|V_{i+1}| = |V(F_t)| \geq \left(1 - \frac{1}{2b}\right)|V(F_0)| \geq |V_i| - \frac{|V|}{2b}$*. In particular, this implies that Invariant (III) is preserved.*

Proof. A node u got deleted in step i, i.e., $u \in V(F_j) \setminus V(F_{j+1})$, because of some tree T in F_j which decided to stop growing, as

$$\sum_{v \in V_j^{propose} \,:\, v \text{ proposes to } T} |V(T_v)| < \frac{|V(T)|}{2b}.$$

We blaim this tree T for deleting u. Note that T only receives blaim in step j and at most $\frac{|V(T)|}{2b}$ deleted nodes blaim T. During the algorithm description, we observed that T is not neighboring any blue node in F_{j+1} and therefore T is also a tree in F_t. Hence, each deleted node in $V(F_0) \setminus V(F_t)$ can blaim one tree T in F_t for being deleted in such a way that each such tree gets blaimed by at most $\frac{1}{2b}|V(T)|$ nodes, which directly proofs the claim. □

Proof of Theorem 5.5. The algorithm has $O(\log n)$ phases, with each phase consisting of $O(\log^2 n)$ steps. It directly follows from the ruling claim that each step can be executed in $O(\log^3 n)$ CONGEST rounds. Hence, we can compute V' in $O(\log^6 n)$ CONGEST rounds, which together with the previous discussion finishes the proof of Theorem 5.5. □

CHAPTER 6

Sharp Threshold Phenomenon for the Lovász Local Lemma

6.1 Introduction

This chapter discusses Lovász Local lemma (LLL). [1] The Lovász Local Lemma is a celebrated result from 1975 due to Erdős and Lovász [125], with applications in many types of problems such as coloring, scheduling or satisfiability problems [5, 54, 102, 113, 114, 124, 192, 231, 256]. While we already discussed the lemma in Section 2.2, let us state its dependency-graph variant here again and more formally.

Lovász Local Lemma (LLL) 1. *Let $\{X_1, \dots, X_m\}$ be a set of mutually independent random variables and $\mathcal{E}_1, \dots, \mathcal{E}_n$ probabilistic events that depend on the X_i. For each $\mathcal{E}_i$, let $\mathrm{vbl}(\mathcal{E}_i)$ denote the random variables $\mathcal{E}_i$ depends on. We say that $\mathcal{E}_i$ and $\mathcal{E}_j$ share a random variable if $\mathrm{vbl}(\mathcal{E}_i) \cap \mathrm{vbl}(\mathcal{E}_j) \neq \emptyset$. Assume that there is some $p < 1$ such that for each $1 \leq i \leq n$, we have $P(\mathcal{E}_i) \leq p$, and let Δ be a positive integer such that each $\mathcal{E}_i$ shares a random variable with at most Δ other $\mathcal{E}_j$ (where $j \neq i$). Then, if $4p\Delta \leq 1$, there exists an assignment of values to the random variables such that none of the events $\mathcal{E}_i$ occurs.*[2]

Brandt, Maus and Uitto [78] showed that if we restrict the random variables to affect at most 3 events each (which they call *rank* at most 3), then under the criterion $p2^{\Delta} < 1$, there is a deterministic LLL algorithm with a complexity of $O(\mathrm{poly}\,\Delta + \log^* n)$. They conjectured that this behavior also holds without their restriction on the variables.

Conjecture 6.1 ([78], rephrased)**.** *There is a (deterministic) distributed algorithm that solves the LLL problem in time $O(\Delta^2 + \log^* n)$ under the criterion $p2^{\Delta} < 1$.*

In this work, we prove Conjecture 6.1, by providing such a deterministic algorithm. This gives a first (unrestricted) answer to the aforementioned question about the relation

[1]Based on a paper by: Sebastian Brandt, Christoph Grunau, Václav Rozhoň.

[2]We note that the *LLL criterion* $4p\Delta \leq 1$ guaranteeing the existence of the desired variable assignment is not optimal and has been subject to improvements by Spencer [284] and Shearer [282].

between the LLL criterion and the complexity of the LLL: a sharp transition occurs at the criterion $p2^\Delta < 1$, where the complexity of the LLL drops from $\Omega(\log\log n)$ randomized, resp. $\Omega(\log n)$ deterministic, to $O(\Delta^2 + \log^* n)$. Moreover, our upper bound is tight on bounded-degree graphs due to the $\Omega(\log^* n)$ lower bound by Chung, Pettie and Su [102]. Finally, as is the nature of upper bounds for the LLL, our result immediately implies the same upper bound for all problems that can be phrased as an LLL problem with criterion $p2^\Delta < 1$, such as certain hypergraph edge-coloring problems or orientation problems in hypergraphs (see [78]).

Previous Techniques: Our work builds on techniques developed in [78]. In their work, Brandt, Maus and Uitto obtain an $O(\Delta^2 + \log^* n)$-round LLL algorithm under the criterion $p2^\Delta < 1$ for the case of variables of rank at most 3. In the following, we give an informal overview of their approach.

The basic idea of the algorithm is to go sequentially through all variables and fix them to some values one by one while preserving a certain invariant that makes sure that the final assignment avoids all events. In order to define the invariant, each edge of the dependency graph is assigned two non-negative values, one for each endpoint of the edge, that sum up to at most 2. When fixing a random variable, the algorithm is also allowed to change these "book-keeping" values. The invariant now states that for any node v in the dependency graph, the product of the $\deg(v)$ values around v multiplied by p is an upper bound for the conditional probability of the event $\mathcal{E}_v$ associated with node v to occur (where we naturally condition on the already-fixed random variables being fixed as prescribed by the (partial) value assignments performed by the algorithm so far). If this invariant is preserved, then, after all variables are fixed, each event $\mathcal{E}_v$ occurs with probability at most $2^{\deg(v)} \cdot p \leq p2^\Delta < 1$, and therefore with probability 0, as desired.

Brandt, Maus and Uitto do not only show that such a sequential process preserving the invariant at all times exists (even if the order in which the random variables have to be fixed are chosen adversarially), but also that it can be made to work in a local manner: in order to fix a random variable, the algorithm only needs to know the random variables and edge values in a small local neighborhood. This allows to process random variables that affect events that are sufficiently far from each other in the dependency graph in parallel. By adding an $O(\log^* n)$-round preprocessing step to the algorithm where a 2-hop node coloring with $O(\Delta^2)$ colors is computed in the dependency graph, the sequential fixing process can then be parallelized by iterating through the color classes in a standard way, yielding the desired runtime of $O(\Delta^2 + \log^* n)$ rounds. We will provide a more detailed overview of the algorithm from [78] in Section 6.1.1.

The crucial, and rather surprising, observation making the algorithm work is that in each step in which a random variable is fixed, the existence of a value for that random variable that preserves the invariant is guaranteed if a certain function is shown to be convex on some domain. Hence, proving the existence of the desired algorithm is reduced to solving an analytical problem for a fixed function f, providing a very intriguing connection between distributed algorithms and analysis. To be precise, Brandt, Maus und Uitto show that for any integer $r \geq 2$, there is a fixed function $f_r : \Delta \rightarrow \mathbb{R}$ on some domain $\Delta \subset \mathbb{R}^{r-1}$ satisfying the following property: if f_r is convex, then for any rank-r random variable, there is a value that this variable can be fixed to such that the invariant is preserved. By proving the convexity of $f_3(a, b) = 4 + 1/2 \cdot (ab - 2a - 2b - \sqrt{ab(4-a)(4-b)})$, they prove the desired upper bound for the case of variables of rank at most 3.[3]

One of the main problems with extending this proof to arbitrary ranks is that the function is only given in an indirect way, by a characterization of the set of points in $\mathbb{R}^r$ that lie below and on the function. No closed-form expression describing f_r is known for any $r > 3$, and the relatively compact form of the function for the case $r = 3$ is arguably due to the cancellation of certain terms that do not cancel out in higher dimensions. In fact, none of the ways to obtain f_3 from the characterization of the mentioned point set seems to yield *any* closed-form expression if adapted to higher dimensions, and even if a closed-form expression for all f_r were found in some way, it is far from clear that proving convexity of these functions would be feasible.

New Techniques: We overcome this obstacle by showing that, perhaps surprisingly, even without any analytical access to the functions f_r, we can infer their convexity for all r. In the following we give an informal overview of our approach. Our main idea is to prove convexity of f_r—or equivalently, convexity of the set bounded by f_r from below—by finding a so-called locally supporting hyperplane for each point q on f_r. More precisely, for each such q, we want to find a number of vectors such that the following two properties hold:

1. The affine subspace of $\mathbb{R}^r$ spanned by the vectors and containing q is a hyperplane, i.e., an affine subspace of dimension $r - 1$.
2. In an ε-ball around q, the hyperplane is contained in the set consisting of all points on and below f_r.

These properties ensure convexity of f_r in q; however, a priori it is completely unclear how to find such vectors. In order to obtain these vectors, we consider the combinatorial description of the points on and below f_r that is tightly connected to the aforementioned invariant: Consider a hyperedge of rank r and write two non-negative values that sum up to at most 2 on each edge of the skeleton of the hyperedge (i.e., a clique induced by the hyperedge) one value for each endpoint of the edge. For each endpoint of the hyperedge multiply the $r-1$ values belonging to the endpoint, and consider the r-dimensional vector obtained by collecting the resulting products. The points that can be generated in this way are exactly the (non-negative) points that lie on or below f_r.

For each such point q', call the tuple of the $\Theta(r^2)$ values written on the edges that generate q' in the above description a generator of q'; a point can have (and usually has) more than one generator. Roughly speaking, we find the desired vectors for a point q by picking an arbitrary generator and, for each edge e in the skeleton of the hyperedge,

[3]Taking care of the case of rank-1 and rank-2 variables is comparably easy.

computing the vector by which q changes if we subtract some small ε from one value on e and add it to the other. A crucial insight is that it is fine to pick such a large set of $\Theta(r^2) \gg r-1$ vectors: due to the specific construction, one can show that the affine subspace spanned by these $\Theta(r^2)$ vectors and containing q is $(r-1)$-dimensional. Moreover, the redundancy contained in this choice enables us to prove Item 2 by finding, for each q' on the hyperplane in an ε-ball around q, a way to write $q'-q$ as a linear combination of $r-1$ of these vectors that satisfies certain desirable properties.

Note that we will use terminology that does not refer to the convexity of the function f_r as we do not make use of this function from an analytical perspective. Instead, we will aim for the equivalent goal of showing that the set bounded by f_r from below is convex, by making use of its combinatorial description.

6.1.1 The Reduction

In this section, we will give a detailed explanation of the argumentation presented in [78] that reduces proving the existence of an $O(\Delta^2 + \log^* n)$-round distributed deterministic LLL algorithm under the criterion $p2^{\Delta} < 1$ to showing that a certain family of sets or functions is convex. The blueprint for such an algorithm $\mathcal{A}$ is given as follows.

Consider an instance of the LLL, given by a set $\{X_1, \ldots, X_m\}$ of mutually independent random variables and a set of events that depend on the random variables. Consider the dependency graph $G = (V, E)$ of this instance, and denote the event associated with a vertex v by $\mathcal{E}_v$, and the maximum degree of G by Δ. Let p be a parameter such that each event occurs with probability at most p, and assume that $p2^{\Delta} < 1$, i.e., fix the LLL criterion to $p2^{\Delta} < 1$. As any two events that depend on the same variable are neighbors of each other in G, we can create for each random variable X_i a hyperedge that has the nodes v such that $\mathcal{E}_v$ depends on X_i as endpoints. Technically, the hyperedges are not part of G, but for simplicity, we might consider them as such.

Algorithm $\mathcal{A}$ starts by computing a 2-hop coloring with $O(\Delta^2)$ colors in $\widetilde{O}(\Delta) + O(\log^* n)$ rounds, by applying the coloring algorithm by Fraigniaud, Heinrich and Kosowski [136] to G^2, i.e., to the graph obtained by connecting any two nodes of distance at most 2 in G by an edge. Then, it iterates through the colors one by one, and each time a color c is processed, each node v of color c fixes each incident random variable (i.e., each random variable whose corresponding hyperedge is incident to v) that has not been fixed so far. We will see that in order to fix all incident random variables of a node in a suitable way, $O(1)$ rounds suffice, and as there are $O(\Delta^2)$ colors, algorithm $\mathcal{A}$ runs in $O(\Delta^2 + \log^* n)$ rounds.

The challenging part is to fix the random variables in a manner such that the produced final assignment is correct, i.e., such that none of the events occurs under the assignment. To this end, during the fixing process the authors keep track of, roughly speaking, how favorable or unfavorable the variable fixings performed so far were for the nodes (regarding avoiding the associated event), by assigning two values to each edge. More precisely, they assign a non-negative value φ_e^v to each pair $(e, v) \in E \times V$ for which e is incident to v. We can imagine the two values φ_e^u and φ_e^v to be written on edge e; each time a random variable X_i is fixed by a node, the node also updates the values that are written on the edges in the skeleton of the hyperedge corresponding to X_i.

The purpose of these edge values w.r.t. obtaining a correct output in the end of the process is to define a property P^* that is kept as an invariant during the fixing process and guarantees that the final assignment avoids all events. Consider an arbitrary point in the fixing process where some random variables $X_1, \ldots, X_\ell$ already have been fixed to some values $x_1, \ldots, x_\ell$, respectively. Property P^* is satisfied if the following two conditions hold.

1. $\varphi_e^u + \varphi_e^v \leq 2$ for each edge $e = \{u, v\}$.
2. $P(\mathcal{E}_v \mid X_1 = x_1, \ldots, X_\ell = x_\ell) \leq p \cdot \prod_{e \ni v} \varphi_e^v$ for each node v.

If Property P^* is satisfied when all variables have been fixed, then for each event $\mathcal{E}_v$ we have a bound of $p \cdot \prod_{e \ni v} \varphi_e^v \leq p2^\Delta < 1$ for the probability that $\mathcal{E}_v$ occurs, which implies that $\mathcal{E}_v$ does not occur since the probability of it occurring can only be 0 or 1. By initializing each value φ_e^v to 1, the authors make sure that P^* is satisfied when algorithm $\mathcal{A}$ starts. The crucial insight in [78] is that there is always a way to preserve Property P^* each time a random variable is fixed if a certain function or set is convex. For the precise statement, the authors introduce the notion of a representable triple.

Definition 6.2 (Definition 3.3 of [78]). *A triple $(a, b, c) \in \mathbb{R}^3_{\geq 0}$ is called* representable *if there are values $a_1, a_2, b_1, b_3, c_2, c_3 \in [0, 2]$ such that $a_1 \cdot a_2 = a$, $b_1 \cdot b_3 = b$, $c_2 \cdot c_3 = c$, $a_1 + b_1 \leq 2$, $a_2 + c_2 \leq 2$, and $b_3 + c_3 \leq 2$. Let $S_{\text{rep}} = \{(a, b, c) \in \mathbb{R}^3_{\geq 0} \mid (a, b, c)$ is representable$\}$ denote the set of all representable triples.*

Using this definition, the authors prove the following statement for the case of rank-3 random variables (which we give in a reformulated version using the notion of convexity instead of the concept of “incurvedness" used in [78]).

If $[0, 2]^3 \setminus S_{\text{rep}}$ is a convex set, then there is a way to fix any given random variable X_i of rank at most 3 at any point in time during the algorithm (or, more generally, for any arbitrary fixing of already fixed random variables such that Property P^* is satisfied) such that Property P^* is preserved. Moreover, the only information required to fix X_i is the set of values φ_e^v written on the edges e that belong to the skeleton of the hyperedge corresponding to X_i. We refer to [78, Section 3.3] for the details of the proof.

Hence, in algorithm $\mathcal{A}$, each node v that has the task to fix all its incident unfixed random variables can simply collect all edge values written on edges between nodes in its inclusive 1-hop neighborhood, and then go through its incident random variables one by one, each time finding a value for the random variable in question that preserves Property P^*. As the sequential fixing does not require any communication after obtaining the required edge values, fixing *all* incident unfixed variables of a node can be done in $O(1)$ rounds. Moreover the local nature of P^* and the fact that the set of edge values required and rewritten by a node during the fixing does not intersect with the set of analogous edge values for a node in distance at least 3 ensures that any two nodes with the same color in the computed 2-hop coloring can perform the variable fixing in parallel. This concludes the description of the reduction.

As already noted by the authors, the definitions and proofs (for the reduction to the convexity statement) generalize straightforwardly to the case of random variables of arbitrary rank. However, showing that the convexity of the respective set indeed holds for higher dimensions remained unanswered in [78]; and indeed, even given our resolution, it remains unclear and would be interesting to see whether their analytical approach can feasibly be extended to higher dimensions than 3. To be precise, their approach extends in the following way: to prove the existence of the deterministic algorithm in the case that each random variable affects at most r events, it suffices to prove that the set $S_{\text{non}}^{(r)} := [0,1]^r \setminus S_{\text{rep}}^{(r)}$ is convex, where $S_{\text{rep}}^{(r)}$ is the set of all representable tuples, which are tuples that can be generated by some generator, as defined below.

Definition 6.3 (generator)**.** *We call a vector $(a_{ij})_{i\neq j\in[r]}$ with $r(r-1)$ coordinates a* generator *if for each $i \neq j$ we have $0 \leq a_{ij} \leq 1$ and $a_{ij} + a_{ji} \leq 1$. The generator $(a_{ij})_{i\neq j\in[r]}$ generates the r-dimensional tuple $(a_1, \ldots, a_r)$ with $a_i = \prod_{j\in[r]\setminus\{i\}} a_{ij}$ for $i \in [r]$. We call a generator* non-zero*, if none of its coordinates is 0. We use a shorthand notation and denote the generator $(a_{ij})_{i\neq j\in[r]}$ simply as (a_{ij}).*

Note that if (a_{ij}) is a non-zero generator, then $a_{ij} < 1$ for each $i \neq j \in [r]$.

Definition 6.4 (representable tuples)**.** *A tuple $(a_1, \ldots, a_r) \in \mathbb{R}^r_{\geq 0}$ is called representable if there exists a generator (a_{ij}) that generates it. Let $S_{\text{rep}}^{(r)} = \{(a_1, \ldots, a_r) \in \mathbb{R}^r_{\geq 0} \,|(a_1, \ldots, a_r)$ is representable $\}$ denote the set of all representable tuples.*

Note that $S_{\text{rep}}^{(3)} \neq S_{\text{rep}}$, as we require $a_{ij} + a_{ji} \leq 1$ instead of $a_{ij} + a_{ji} \leq 2$. We consider this scaled version, as this makes the proof cleaner later on: note that $[0,1]^3 \setminus S_{\text{rep}}^{(3)}$ being convex directly implies that $[0,2]^3 \setminus S_{\text{rep}}$ is convex as the latter is just a scaled variant of the former set. In the following, we drop the superscripts when clear from context and we denote with S_{rep} the set of representable tuples with respect to the scaled down version and S_{non} as the set of points in $[0,1]^r$ which are not representable. Our main contribution is the proof of the following theorem.

Theorem 6.5. *For every $r \geq 2$, $S_{non}^{(r)}$ is convex.*

This settles Conjecture 6.1 as described above.

6.2 Proving convexity

In this section we prove that set S_{non} is convex, omitting two longer proofs that are postponed to Section 6.3 and Section 6.4.

6.2.1 Notation

We work with the standard Euclidean space $\mathbb{R}^m$ where distances are measured with the Euclidean norm; $\mathbf{0}$ and $\mathbf{1}$ denote the vectors $(0, 0, \dots, 0)^T$ and $(1, 1, \dots, 1)^T$, respectively. We define $B(x, R) := \{y \in \mathbb{R}^m, \|x - y\| \leq R\}$ as the closed ball around x with radius R. A subset $S \subseteq \mathbb{R}^m$ is open if for any $x \in S$, there exists $R > 0$ such that $B(x, R) \subseteq S$. A subset $S \subseteq \mathbb{R}^m$ is closed if $\mathbb{R}^m \setminus S$ is open. A set $S \subseteq \mathbb{R}^m$ is bounded if there exists $R > 0$ such that $S \subseteq B(\mathbf{0}, R)$. A set S is compact if it is closed and bounded. Equivalently, S is compact if every sequence $x_1, x_2, \dots$ with each $x_i \in S$ has a subsequence $x_{s(i)}$ that converges to some $x \in S$. The subset $[0, 1]^m \subseteq \mathbb{R}^m$ is compact. The interior of a set S is an open subset of S and defined as $S^o = \{x \in S, \exists R > 0 : B(x, R) \subseteq S\}$. The boundary of a set S is defined as $\partial S = \{x \in \mathbb{R}^m, \forall R > 0 : B(x, R) \cap S \neq \emptyset \text{ and } B(x, R) \cap (\mathbb{R}^m \setminus S) \neq \emptyset\}$. A set S is path-connected if for any $x, y \in S$ there exists a continuous function $f : [0, 1] \to S$ such that $f(0) = x$ and $f(1) = y$.

A hyperplane $H \subset \mathbb{R}^m$ is an affine subspace of dimension $m - 1$. Equivalently, it is a set of points $H = \{x \in \mathbb{R}^m, h^T x = b\}$ for some vector $h \in \mathbb{R}^m \setminus \{\mathbf{0}\}$ and $b \in \mathbb{R}$. A weakly supporting hyperplane for S intersecting $y \in \partial S$ is a hyperplane $H = \{x \in \mathbb{R}^m, h^T x = b\}$ with $h^T y = b$ and $h^T z \geq b$ for any $z \in S$. Finally, a weakly locally supporting hyperplane for S intersecting $y \in \partial S$ is a hyperplane $H = \{x \in \mathbb{R}^m, h^T x = b\}$ with $h^T y = b$ satisfying the following property: there exists an $\varepsilon > 0$ such that for any $z \in S \cap B(y, \varepsilon)$ we have $h^T z \geq b$.

6.2.2 Proof

Convexity of a set can be verified in several equivalent ways. As we outlined in Section 6.1, we rely on the "supporting hyperplane formulation", i.e., a set is convex if for each boundary point we can find a hyperplane such that the whole set lies on one side of the hyperplane. Moreover, for connected sets, it is enough to prove that each such hyperplane is "locally" supporting as formalized in the following theorem, which is stated in a more general form in [295] (Theorem 4.10 there).

Theorem 6.6. *Let $S \subseteq \mathbb{R}^r$ be an open and path-connected set in $\mathbb{R}^r$. The set $S \subseteq \mathbb{R}^r$ is convex if for every point y contained in the boundary of S, there exists a weakly locally supporting hyperplane with respect to S going through y.*

Note that Theorem 6.6 can only be used to prove convexity of open sets and thus cannot directly applied to establish the convexity of S_{non}. Instead, we use Theorem 6.6 to first establish convexity of the interior of S_{non}, which is an open set and which we denote by S^o_{non}. Once we have established the convexity of S^o_{non}, we prove the convexity of S_{non} by induction on the dimension r. To prove convexity of S^o_{non}, we need to show that S^o_{non} is path-connected and that for every boundary point of S^o_{non}, there exists a weakly locally supporting hyperplane going through the boundary point. We now prove the former, using the following simple observation, which will be used in several other proofs.

Observation 6.7. *Let $a = (a_1, \dots, a_r)$ be a representable tuple. Then any tuple a' with $0 \le a'_i \le a_i$ for all $i \in [r]$ is also representable.*

Proof. Consider a generator (a_{ij}) of a. For any i, pick some $j \neq i$ and set $a'_{ij} = a_{ij} \cdot \frac{a'_i}{a_i} \le 1$. Set all other values in (a'_{ij}) equal to the corresponding value in (a_{ij}). (a'_{ij}) is a valid generator generating the tuple a'. □

Now, we are ready to prove that $S^{\mathrm{o}}_{\mathrm{non}}$ is path-connected.

Lemma 6.8. *The set S^{o}_{non} is path-connected.*

Proof. For any $u, u' \in S^{\mathrm{o}}_{\mathrm{non}}$, consider the vector $u'' \in \mathbb{R}^r$ with $u''_i = \max\{u_i, u'_i\} > 0$ for every $i \in [r]$. Note that the union of the two segments between u and u'' and between u'' and u' is a path. Moreover, any tuple on this path is contained in $(0,1)^r$ and either dominates u or u'. Hence, by Observation 6.7, each tuple on the path is in $S^{\mathrm{o}}_{\mathrm{non}}$. □

Next, we need to understand the boundary between S_{rep} and S_{non}. To do so, it will be helpful to prove that S_{rep} is closed. As $S_{\mathrm{rep}} \subseteq [0,1]^r$ is bounded, this is equivalent to show that S_{rep} is compact.

Lemma 6.9. *The set S_{rep} is compact.*

Proof. The set S_{rep} is defined as an image of a continuous function that maps each generator (Theorem 6.3) from the compact set of all generators to the corresponding representable tuple. Hence, it is compact as an image of a compact set under continuous function is always compact. □

Next, we set up the notion of maximal tuples.

Definition 6.10 (domination and maximal tuples)**.** *Let $a = (a_1, \dots, a_r)$ and $a' = (a'_1, \dots, a'_r)$ be two representable tuples. We say that a'* weakly dominates *a if $a'_i \ge a_i$ for all $i \in [r]$, and $a' \neq a$. Moreover, we say that a'* strongly dominates *a if $a'_i > a_i$ for all $i \in [r]$. We call a representable tuple a* maximal *if there is no representable tuple a' that weakly dominates a.*

Intuitively, maximal tuples are forming the boundary between S_{rep} and S_{non} and this is indeed what we prove.

Lemma 6.11. *Let $x \in \mathbb{R}^r$ be contained in ∂S_{non}. Then, there either exists $i \in [r]$ such that $x_i \in \{0, 1\}$ or x is a maximal representable tuple.*

We defer the easy, yet slightly technical proof, together with proofs of a few other technical lemmas, to Section 6.3. Our main technical contribution is a proof that a locally supporting hyperplane can be found for any maximal tuple a.

Lemma 6.12. *For each maximal representable tuple a, there exists a locally supporting hyperplane for S^{o}_{non} intersecting a.*

The non-trivial proof of the above lemma is deferred to Section 6.4. As a corollary, we infer that the whole set $S^{\mathrm{o}}_{\mathrm{non}}$ is convex.

Corollary 6.13. *The set S^{o}_{non} is convex.*

Proof. By Theorem 6.6 it suffices to provide a weakly locally supporting hyperplane for any $a \in \partial S_{\mathrm{non}}$. By Theorem 6.11, any $a \in \partial S_{\mathrm{non}}$ is either a maximal representable tuple and hence the existence of the supporting hyperplane follows from Theorem 6.12, or we have $a_i = 0$ or $a_i = 1$, respectively, for some i. But then the hyperplane $\{x \in \mathbb{R}^r : e_i^T x = 0\}$ or $\{x \in \mathbb{R}^r : -e_i^T x = -1\}$, respectively, is a weakly (locally) supporting hyperplane for $S^{\mathrm{o}}_{\mathrm{non}}$ intersecting a. □

The proof of Theorem 6.5 now easily follows.

Proof of Theorem 6.5. We prove the statement by induction on r. For $r = 2$, the statement trivially holds. Now, let $r \geq 3$ arbitrary and assume that $S^{(r-1)}_{\mathrm{non}}$ is convex. Let $x \neq y \in S^{(r)}_{\mathrm{non}}$ and $\alpha \in (0,1)$ be arbitrary. We need to show that for $z := \alpha x + (1-\alpha)y$ we have $z \in S^{(r)}_{\mathrm{non}}$. As $S^{(r)}_{\mathrm{rep}}$ is a closed set (Theorem 6.9), there exists some ε with $0 < \varepsilon < \min(\alpha, 1-\alpha)$ such that $x' = (1-\varepsilon)x + \varepsilon y \notin S^{(r)}_{\mathrm{rep}}$ and, hence, $x' \in S^{(r)}_{\mathrm{non}}$ since the whole segment $\{\beta x + (1-\beta)y, 0 < \beta < 1\}$ is contained in $[0,1]^r$, and $y' := (1-\varepsilon)y + \varepsilon x \in S^{(r)}_{\mathrm{non}}$. Furthermore, there exists an $\alpha' \in (0,1)$ such that $z = \alpha' x' + (1-\alpha')y'$.

If $x', y' \in {S^{\mathrm{o}}_{\mathrm{non}}}^{(r)}$, then, by Theorem 6.13, it follows that $z \in {S^{\mathrm{o}}_{\mathrm{non}}}^{(r)}$ and we are done. Otherwise, $x' \notin {S^{\mathrm{o}}_{\mathrm{non}}}^{(r)}$ or $y' \notin {S^{\mathrm{o}}_{\mathrm{non}}}^{(r)}$. Without loss of generality, assume that $x' \notin {S^{\mathrm{o}}_{\mathrm{non}}}^{(r)}$. Since $x' \notin S^{(r)}_{\mathrm{non}}$, Theorem 6.11 implies that there exists some $i \in [r]$ with $x'_i \in \{0,1\}$. Our choice of $\varepsilon > 0$ now implies that either $x_i = y_i = z_i = 1$ or $x_i = y_i = z_i = 0$.

In the first case, as x is not representable, there exists some $j \in [r] \setminus \{i\}$ with $x_j > 0$. Therefore, $z_j > 0$ and as $z_i = 1$, any generator of z would need to have $z_{ij} = 1$ and $z_{ji} > 0$, a contradiction with $z_{ij} + z_{ji} \leq 1$. Hence, $z \in S^{(r)}_{\mathrm{non}}$.

In the second case, assume without loss of generality that $i = r$. Let $\widetilde{x}$, $\widetilde{y}$, $\widetilde{z} \in [0,1]^{r-1}$ be equal to the vectors x, y and z restricted to the first $r-1$ coordinates. We have $\widetilde{x}, \widetilde{y} \in S^{(r-1)}_{\mathrm{non}}$, since otherwise taking their generator and augmenting it by zeros would generate x or y, respectively. As $\widetilde{z}$ is a convex combination of $\widetilde{x}$ and $\widetilde{y}$, the induction hypothesis implies that $\widetilde{z} \in S^{(r-1)}_{\mathrm{non}}$ and therefore $z \in S^{(r)}_{\mathrm{non}}$, which concludes the induction step. □

6.3 Technical preparation

In this section we prove several technical results that are needed for the proof. First, we prove the equivalence of the notions of weak and strong dominance. To this end, we first show a simple "continuity" statement that shows that for any representable tuple a, one can increase all but one of its coordinates a little bit at the expense of decreasing the remaining one.

Lemma 6.14. *Let $(a_1, \ldots, a_r)$ be a representable tuple with $a_i > 0$ for each $i \in [r]$. For each $k \in [r]$, there exist an $\varepsilon > 0$ and a $\xi > 0$ such that for all t with $0 < t < \varepsilon$, the tuple a' defined by $a'_k = a_k - t$ and $a'_i = a_i + \xi t$ for $i \neq k$ is also representable.*

Proof. Let (a_{ij}) be a generator of $(a_1, \ldots, a_r)$. As $a_i > 0$ for each $i \in [r]$, (a_{ij}) is a non-zero generator. Now, for some $\delta > 0$, consider (b_{ij}) with

$$b_{ij} = \begin{cases} a_{ij} - \delta \text{ if } i = k \\ a_{ij} + \delta \text{ if } j = k \\ a_{ij} \qquad \text{otherwise} \end{cases}$$

for each $i \neq j \in [r]$. We have $b_{ij} + b_{ji} = a_{ij} + a_{ji} \leq 1$ for each $i, j \in [r], i \neq j$. Furthermore, if we choose δ such that $0 < \delta < \varepsilon' := \min_{i \neq j \in [r]} \min(a_{ij}, 1 - a_{ij}) < 1$, we have $0 \leq b_{ij} \leq 1$ for each $i, j \in [r], i \neq j$. In that case, (b_{ij}) is a valid generator that generates a tuple $(b_1, \ldots, b_r)$ such that:

$$b_k = \prod_{j \neq k} b_{kj} = \prod_{j \neq k} (a_{kj} - \delta) \geq \left(\prod_{j \neq k} a_{kj} \right) - \delta f((a_{ij})) = a_k - \delta f((a_{ij}))$$

for some function f with $f((a_{ij})) > 0$. Note that such a function f exists, as $\delta < 1$ and therefore $\delta^e \leq \delta$ for each $e \geq 1$. For each $i \in [r] \setminus \{k\}$, we have:

$$\begin{aligned} b_i = \prod_{j \neq i} b_{ij} &= (a_{ik} + \delta) \cdot \prod_{j \notin \{i,k\}} a_{ij} = a_i + \delta \cdot \prod_{j \notin \{i,k\}} a_{ij} \\ &\geq a_i + \delta \cdot \prod_{j \neq i} a_{ij} = a_i + \delta a_i \end{aligned}$$

Set $t = \delta \cdot f((a_{ij}))$, $\xi = \frac{1}{f((a_{ij}))} \min_{i \in [r]} a_i > 0$ and $\varepsilon = \varepsilon' \cdot f((a_{ij})) > 0$. Now, consider some arbitrary t with $0 < t < \varepsilon$. The definition of ε implies that $0 < \delta = \frac{t}{f((a_{ij}))} < \frac{\varepsilon}{f((a_{ij}))} = \varepsilon'$. Thus, we can represent a tuple $(b_1, \ldots, b_r)$ with $b_k \geq a_k - \delta f((a_{ij})) \geq a_k - t = a'_k$ and $b_i \geq a_i + \delta \cdot a_i \geq a_i + \xi \cdot t = a'_i$ for $i \neq k$. This tuple dominates the tuple a'. As $a'_i \geq 0$ for each $i \in [r]$, Observation 6.7 implies that we can represent a'. □

Now we are ready to show that if a tuple is weakly dominated by some other tuple, it is also strongly dominated by (a potentially different) one.

Corollary 6.15 (strong vs weak domination)**.** *Let $a = (a_1, \dots, a_r)$ be a representable tuple such that for all $i \in [r]$ we have $0 < a_i < 1$. If there exists a representable tuple that weakly dominates a, then there also exists a representable tuple that strongly dominates a.*

Proof. Let $(a'_1, \dots, a'_r)$ be a representable tuple that weakly dominates a. Note that we have $a'_i > 0$ for each $i \in [r]$. Let $k \in [r]$ such that $a'_k > a_k$. According to *Theorem* 6.14, there exists $\varepsilon > 0$ and $\xi > 0$ such that for all $0 < t < \varepsilon$, the tuple $(a'_1 + \xi t, \dots, a'_{k-1} + \xi t, a'_k - t, a'_{k+1} + \xi t, \dots, a'_r + \xi t)$ is representable. For t small enough, this tuple strictly dominates the tuple $(a_1, \dots, a_r)$. $\square$

We are now ready to prove Theorem 6.11.

Proof of Theorem 6.11. We show the contrapositive. Let $x \in \mathbb{R}^r$ such that $x_i \notin \{0, 1\}$ for each $i \in [r]$ and x is not a maximal representable tuple. We show that this implies the existence of a ball $B(x, \varepsilon)$ around x with radius $\varepsilon > 0$ such that either $B(x, \varepsilon) \subseteq S_{\text{non}}$ or $B(x, \varepsilon) \cap S_{\text{non}} = \emptyset$, which in turn implies $x \notin \partial S_{\text{non}}$.

For $x \notin [0, 1]^r$ there is clearly such a ball. Otherwise, we have $x \in (0, 1)^r$, as we assume that for each $i \in [r]$, $x_i \notin \{0, 1\}$. If $x \in S_{\text{rep}}$, but x is not maximal representable, there is a representable tuple that weakly dominates x and as $x \in (0, 1)^r$, Theorem 6.15 provides a representable tuple that strongly dominates x. Hence, there exists some $\varepsilon > 0$ such that the tuple $(x_1 + \varepsilon, \dots, x_r + \varepsilon)$ is a representable tuple. This implies that for $\varepsilon' = \min(\varepsilon, \min_{i \in [r]} x_i)$ we have $B(x, \varepsilon') \subseteq S_{\text{rep}}$ due to Observation 6.7 and, hence, $B(x, \varepsilon') \cap S_{\text{non}} = \emptyset$ as needed. Finally, in the case $x \in (0, 1)^r \cap S_{\text{non}}$ Theorem 6.9 implies that the complement of S_{rep} is open which in turn implies the existence of an $\varepsilon > 0$ so that $B(x, \varepsilon) \cap S_{\text{rep}} = \emptyset$. For $\varepsilon' = \min(\varepsilon, \min_{i \in [r]} \min(x_i, 1 - x_i))$ we then have $B(x, \varepsilon') \subseteq [0, 1]^r \setminus S_{\text{rep}} = S_{\text{non}}$, as needed. $\square$

6.4 Construction of hyperplanes

In this section we prove our main technical contribution: Theorem 6.12 that states that for each maximal tuple we can find a locally weakly supporting hyperplane for the set S_{non}. First, we give an informal proof of this result for the case $r = 3$, which captures the intuition behind the general proof for all r that we give later.

6.4.1 Informal outline for r=3

Our main observation is that finding a locally supporting hyperplane comes down to proving that a certain set of tuples in the neighbourhood of a is representable. In this section, we denote the tuples, more intuitively, as triples. So, we now focus on how to generate triples similar to a.

Generating more triples: Given a representable triple $a \in (0,1)^3$ generated by the generator (a_{ij}), what other triples close to a are representable? Certainly, all the triples that a dominates. Besides, we can play with the generator itself. Adding α_{12} to a_{12} and subtracting it from a_{21} gives us again a valid generator that generates triples of the form

$$(a_{13}(a_{12} + \alpha_{12}), (a_{21} - \alpha_{12})a_{23}, a_{31}a_{32}) = a + \alpha_{12}\,(a_{13}, -a_{23}, 0)$$

I.e., it generates triples on the line

$$a + \alpha_{12}(a_{13}, -a_{23}, 0) = a + \alpha_{12}w_{12}, \;\; \alpha_{12} \in \mathbb{R}$$

for $|\alpha_{12}|$ small enough. Similarly, by adding α_{13} to a_{13} and subtracting it from a_{31}, we can generate triples on the line

$$a + \alpha_{13}(a_{12}, 0, -a_{32}) = a + \alpha_{13}w_{13}, \;\; \alpha_{13} \in \mathbb{R}$$

and by adding α_{23} to a_{23} and subtracting it from a_{32}, we can generate triples on the line

$$a + \alpha_{23}(0, a_{21}, -a_{31}) = a + \alpha_{23}w_{23}, \;\; \alpha_{23} \in \mathbb{R}$$

in some neighborhood around the triple a. We call the three lines ℓ_1, ℓ_2 and ℓ_3.

Since all components of the generator of a are nonzero, these three lines define an affine subspace of dimension at least two. Later we prove that if a is a maximal representable triple, then the three lines lie on a common plane. The plane spanned by ℓ_1, ℓ_2 and ℓ_3 then becomes an obvious suspect for the supporting hyperplane we wish to find!

In fact, we prove that not only triples on the lines ℓ_1, ℓ_2 and ℓ_3 are representable, given that they lie in some small neighborhood around the maximal representable triple a, but *any triple a' in the affine span of the three lines* is representable, provided that $a' \in B(a, \varepsilon)$ for some positive ε that depends on a. This finishes our proof, as we can now find a weakly locally supporting hyperplane for each maximal representable triple a.

We now prove that for maximal triples all three lines lie in a common plane and all triples in that plane are representable (if they are close enough to a).

Claim 6.16. *For a maximal triple a, the affine hull of ℓ_1, ℓ_2 and ℓ_3 is a plane.*

Assume the contrary. Then, there exist $\alpha_{12}, \alpha_{13}, \alpha_{23} \in \mathbb{R}$, such that $(1,1,1) = \alpha_{12}w_{12} + \alpha_{13}w_{13} + \alpha_{23}w_{23}$. Now, change the values of (a_{ij}) proportional to the values of α to obtain the generator (a'_{ij}) with

$$\begin{aligned} a'_{12} &= a_{12} + \xi\alpha_{12}, & a'_{21} &= a_{21} - \xi\alpha_{12}; \\ a'_{13} &= a_{13} + \xi\alpha_{13}, & a'_{31} &= a_{31} - \xi\alpha_{13}; \\ a'_{23} &= a_{23} + \xi\alpha_{23}, & a'_{32} &= a_{32} - \xi\alpha_{23}. \end{aligned}$$

Intuitively, we expect these changes to give us a generator of $a' = a+\xi\alpha_{12}w_{12}+\xi\alpha_{13}w_{13}+\xi\alpha_{23}w_{23} = a + \xi \cdot (1,1,1)$. This is almost the case:

$$\begin{aligned}
a' &= \left(a'_{12}a'_{13}, a'_{21}a'_{23}, a'_{31}a'_{32}\right)\\
&= ((a_{12}+\xi\alpha_{12})(a_{13}+\xi\alpha_{13}), (a_{21}-\xi\alpha_{12})(a_{23}+\xi\alpha_{23}),\\
&\quad (a_{31}-\xi\alpha_{13})(a_{32}-\xi\alpha_{23}))\\
&= (a_{12}a_{13}, a_{21}a_{23}, a_{31}a_{32}) + \xi\alpha_{12}(a_{13}, -a_{23}, 0)\\
&\quad + \xi\alpha_{13}(a_{12}, 0, -a_{32}) + \xi\alpha_{23}(0, a_{21}, -a_{31})\\
&\quad + \xi^2(\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23})\\
&= a + \xi\alpha_{12}w_{12} + \xi\alpha_{13}w_{13} + \xi\alpha_{23}w_{23}\\
&\quad + \xi^2(\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23})\\
&= a + \xi(1,1,1) + \xi^2(\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23})
\end{aligned}$$

Choosing $\xi > 0$ small enough, we conclude that the triple $a+\xi/2\cdot(1,1,1)$ is representable and therefore a is not maximal, a contradiction!

Generating triples on the plane: We are given a maximal triple a and some a' in the affine hull of ℓ_1, ℓ_2 and ℓ_3 that is sufficiently close to a. We need to prove that a' is representable. To do so, we first note that as a' is contained in the affine hull, there exist α_{12}, α_{13} and α_{23} such that $a' = a + \alpha_{12}w_{12} + \alpha_{13}w_{13} + \alpha_{23}w_{23}$. Now, we employ the same strategy as above and observe that we can change the generator of a as follows

$$\begin{aligned}
a'_{12} &= a_{12} + \alpha_{12}, & a'_{21} &= a_{21} - \alpha_{12};\\
a'_{13} &= a_{13} + \alpha_{13}, & a'_{31} &= a_{31} - \alpha_{13};\\
a'_{23} &= a_{23} + \alpha_{23}, & a'_{32} &= a_{32} - \alpha_{23},
\end{aligned}$$

so as to generate the triple

$$\begin{aligned}
&((a_{12}+\alpha_{12})(a_{13}+\alpha_{13}), (a_{21}-\alpha_{12})(a_{23}+\alpha_{23}),\\
&\ (a_{31}-\alpha_{13})(a_{32}-\alpha_{23}))\\
&= (a_{12}a_{13}, a_{21}a_{23}, a_{31}a_{32}) + \alpha_{12}(a_{13}, -a_{23}, 0) + \alpha_{13}(a_{12}, 0, -a_{32})\\
&\quad + \alpha_{23}(0, a_{21}, -a_{31}) + (\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23})\\
&= a + \alpha_{12}w_{12} + \alpha_{13}w_{13} + \alpha_{23}w_{23} + (\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23})\\
&= a' + (\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23})
\end{aligned}$$

The term $(\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23})$ is important now. We require $(\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23}) \geq \mathbf{0}$ to prove that a' is a representable triple. This property does not hold for every choice of the coefficients $\alpha_{12}, \alpha_{13}, \alpha_{23}$. However, as w_{12}, w_{13} and w_{23} are linearly dependent, we have a certain flexibility to choose the α's. In particular, one can choose the α's in such a way that at most 2 of them are non-zero.

It turns out that it is indeed always possible to choose the α's in such a way that $(\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23}) \geq \mathbf{0}$. To see why, observe that the first coordinate of the quadratic term is negative if and only if out of the two numbers a_{12} and a_{13} (recall that $a_1 = a_{12}a_{13}$), one is increased and one is decreased. This holds analogously also for the other coordinates. So, we show how to generate a' such that this does not happen.

First, one can observe that a' does not dominate a or vice versa: if this was the case, one could obtain a contradiction by showing that a is not a maximal representable triple similarly to the proof of Theorem 6.16. Thus, we can assume that there exist $i \neq j \in [3]$ such that $a_i < a'_i$ and $a_j > a'_j$. Assume (without loss of generality) that $a_1 < a'_1$, $a_2 > a'_2$ and $a_3 \geq a'_3$. In that case, we first fix $\alpha_{23} = 0$. As $a_2 \geq a'_2$ and $a_3 \geq a'_3$, one can show that a' lies in the span of ℓ_1 and ℓ_2, thus one can write

$$\begin{aligned} a' &= a + \alpha_{12}w_{12} + \alpha_{13}w_{13} \\ &= a + \alpha_{12}(a_{13}, -a_{23}, 0) + \alpha_{13}(a_{12}, 0, -a_{32}) \\ &= (a_1 + \alpha_{12}a_{13} + \alpha_{13}a_{12}, a_2 - \alpha_{12}a_{23}, a_3 - \alpha_{13}a_{32}). \end{aligned}$$

Since $a'_2 < a_2$, we have $\alpha_{12} > 0$. Similarly, since $a'_3 \leq a_3$, we have $\alpha_{13} \geq 0$. Together with $\alpha_{23} = 0$, we get $(\alpha_{12}\alpha_{13}, -\alpha_{12}\alpha_{23}, \alpha_{13}\alpha_{23}) \geq \mathbf{0}$, as needed.

This concludes the proof outline for $r = 3$. For general r, the last step is slightly more tricky: generally, we set $\alpha_{ij} = 0$ if both a_i and a_j needs to be increased or both needs to be increased. Additionally, if a_k needs to be increased, $\alpha_{ij}(w_{ij})_k$ is non-negative for all $i < j$ and if a_k needs to be decreased, $\alpha_{ij}(w_{ij})_k$ is non-positive for all $i < j$. Moreover, for general r, augmenting the generator according to the α-values might lead to negative higher order terms. However, these are always dominated by the quadratic increase in a neighborhood around a.

6.4.2 Construction of hyperplanes, in general

We start by defining "movement vectors", analogues to vectors w_{12}, w_{13}, w_{23} from Section 6.4.1, that correspond to "allowed movements" that we may make to construct representable tuples in the vicinity of a representable tuple a.

Definition 6.17 (movement vectors)**.** *Let $a \in (0,1)^r$ be a maximal tuple and $(a_{ij})_{i \neq j \in [r]}$ an arbitrary (nonzero) generator of a. For each $i \neq j \in [r]$, we define w_{ij} as the r-dimensional vector such that for each $k \in [r]$,*

$$(w_{ij})_k = \begin{cases} \frac{a_i}{a_{ij}} \text{ if } k = i \ , \\ \frac{-a_j}{a_{ji}} \text{ if } k = j \ , \\ 0 \text{ otherwise } . \end{cases}$$

Similarly to Section 6.4.1, we now define the span of the movement vectors H_a that we later prove to be a hyperplane for the case of maximal representable tuples.

Definition 6.18. *We define* $H_a := \{a + \sum_{i \neq j \in [r]} \alpha_{ij} w_{ij} \mid \alpha_{ij} \in \mathbb{R}$ *for all* $i \neq j \in [r]\}$.

Observation 6.19. H_a *is an affine subspace with a dimension at least* $r - 1$.

Proof. Consider the $r-1$ vectors $w_{12}, w_{13}, \dots, w_{1r}$. Among those $r-1$ vectors, w_{1j} is the only vector with a non-zero j-th coordinate. Hence, the $r-1$ vectors are linearly independent. □

The next lemma corresponds to Theorem 6.16 of the informal outline.

Lemma 6.20. *Let* $a \in (0,1)^r$ *be a maximal tuple and* $q \in \mathbb{R}_{\geq 0}$ *be a non-negative vector such that there exists some index* $k \in [r]$ *with* $q_k > 0$. *Then,* $a + q \notin H_a$.

Proof. We show that the existence of such a vector q would contradict the fact that a is a maximal tuple. Thus, for the sake of contradiction, assume that there exists a non-negative vector q and some $k \in [r]$ such that $q_k > 0$ and $a + q \in H$. Thus, there exist α_{ij}'s such that $q = \sum_{i \neq j \in [r]} \alpha_{ij} w_{ij}$. For $\delta > 0$, consider (a'_{ij}) with $a'_{ij} = a_{ij} + \delta \cdot (\alpha_{ij} - \alpha_{ji})$ for $i \neq j \in [r]$. Note that

$$a'_{ij} + a'_{ji} = (a_{ij} + \delta \cdot (\alpha_{ij} - \alpha_{ji})) + (a_{ji} + \delta \cdot (\alpha_{ji} - \alpha_{ij})) = a_{ij} + a_{ji} \leq 1.$$

Thus, for δ small enough, (a'_{ij}) is a valid generator. Let a' denote the tuple that (a'_{ij}) generates. Then, for each $i \in [r]$, we get:

$$\begin{aligned}
a'_i &= \prod_{j \neq i} (a_{ij} + \delta(\alpha_{ij} - \alpha_{ji})) \\
&= \prod_{j \neq i} a_{ij} + \left(\sum_{\ell \neq i} \delta(\alpha_{i\ell} - \alpha_{\ell i}) \prod_{j \notin \{i,\ell\}} a_{ij} \right) - O(\delta^2) \\
&= a_i + \left(\sum_{\ell \neq i} \delta(\alpha_{i\ell} - \alpha_{\ell i}) \frac{a_i}{a_{i\ell}} \right) - O(\delta^2) \\
&= a_i + \left(\sum_{\ell \neq i} \delta(\alpha_{i\ell}(w_{i\ell})_i + \alpha_{\ell i}(w_{\ell i})_i) \right) - O(\delta^2) \\
&= a_i + \delta \left(\sum_{\ell \neq j} \alpha_{\ell j}(w_{\ell j})_i \right) - O(\delta^2) \\
&= (a + \delta q)_i - O(\delta^2)
\end{aligned}$$

Thus, there exists some constant $c \geq 0$, such that for each sufficiently small $\delta > 0$, there is a non-negative representable tuple b with $b(\delta) := a + \delta q - c\delta^2 \cdot \mathbf{1}$. Theorem 6.14 implies the existence of some $\xi > 0$ such that for each sufficiently small $t > 0$, we can represent the tuple $b'(\delta, t)$ with $b'(\delta, t)_k = b(\delta) - t$ and $b'(\delta, t)_i = b(\delta) + \xi t$ for each $i \neq k$. In particular, we can choose $\delta > 0$ small enough such that for $t = (q_k/2)\delta$, the tuple $b'(\delta, t)$ is representable and furthermore:

$$b'(\delta, t)_k = b(\delta)_k - t = a_k + \delta q_k - c\delta^2 - t = a_k + \delta(q_k/2) - c\delta^2 > a_k$$

and for $i \neq k$,

$$b'(\delta, t)_i = b(\delta)_i + \xi t = a_i + \delta q_i - c \cdot \delta^2 + \xi t \geq a_i - c \cdot \delta^2 + \xi(q_k/2)\delta > a_i$$

which contradicts the maximality of a. □

Corollary 6.21. *For any maximal representable tuple a, the set H_a defines a hyperplane. That is, there exist $h \in \mathbb{R}^r \setminus \{\mathbf{0}\}$ and $b \in \mathbb{R}$ such that $H_a = \{x \in \mathbb{R}^r : h^T x = b\}$. Furthermore, one can choose h such that $h \geq \mathbf{0}$.*

Proof. The set H_a defines an affine subspace of dimension at least $r - 1$ (Theorem 6.19) and of dimension at most $r - 1$, because $a + \mathbf{1} \notin H_a$ according to Theorem 6.20. Thus, H_a is an affine subspace of dimension $r - 1$. Hence, there exist $h \in \mathbb{R}^r \setminus \{\mathbf{0}\}$ and $b \in \mathbb{R}$ such that $H_a = \{x \in \mathbb{R}^r : h^T x = b\}$. Assume that there exist two indices $i \neq j \in [r]$ such that $h_i > 0$ and $h_j < 0$. This would imply the existence of a non-zero vector $q \geq \mathbf{0}$ with $h^T q = 0$. As $a \in H_a$ and $h^T a = b$ we would get $h^T(a + q) = b$. However, as q is non-zero and $q \geq \mathbf{0}$, $a + q \notin H_a$ according to Theorem 6.20, a contradiction. Thus, either $h \geq 0$ or $h \leq 0$. As $\{x \in \mathbb{R}^r : h^T x = b\} = \{x \in \mathbb{R}^r : (-h)^T x = -b\}$, this proves the lemma. □

The next lemma lies at the heart of our argument.

Lemma 6.22. *For any maximal representable tuple a and any $a' \in H_a$, there exist values $\alpha'_{ij} \in \mathbb{R}$ for $i \neq j \in [r]$ with $a' = a + \sum_{i \neq j \in [r]} \alpha'_{ij} w_{ij}$ such that for each $k \in [r]$, either $\alpha'_{ij}(w_{ij})_k \leq 0$ for each $i \neq j \in [r]$ or $\alpha'_{ij}(w_{ij})_k \geq 0$ for each $i \neq j \in [r]$.*

Proof. Let $a' \in H_a$ be arbitrary. Let $b \in H_a$ a vector such that for each $i \in [r]$, either $a_i \leq b_i \leq a'_i$ or $a'_i \leq b_i \leq a_i$ and there exist values $\beta_{ij} \in \mathbb{R}$ for $i \neq j \in [r]$ such that $b = a + \sum_{i \neq j \in [r]} \beta_{ij} w_{ij}$. Furthermore, for each $k \in [r]$, either $\beta_{ij}(w_{ij})_k \leq 0$ for each $i \neq j \in [r]$ or $\beta_{ij}(w_{ij})_k \geq 0$ for each $i \neq j \in [r]$. Note that such a vector b always exists, as setting $b = a$ and all the β_{ij}'s to 0 would fulfill all the criteria. We choose b in such a way that the number of coordinates that b and a' disagree with is minimal. Note that showing $b = a'$ is equivalent to the statement of the lemma. For the sake of contradiction, assume that this is not the case.

As $a', b \in H_a$, we also have $a + (a' - b) \in H_a$ and $a + (b - a') \in H_a$. If $a'_i \geq b_i$ for all $i \in [r]$, then $a' - b$ is a non-zero vector with $a' - b \geq \mathbf{0}$. Hence, according to Theorem 6.20, $a + (a' - b) \notin H_a$, a contradiction. Similarly, $a'_i \leq b_i$ for all $i \in [r]$ would also lead to a contradiction. Thus, we can conclude that there exist two indices $k, \ell \in [r]$ such that $b_k < a'_k$ and $b_\ell > a'_\ell$. Now, consider the vector $c = a + \sum_{i \neq j \in [r]} \gamma_{ij} w_{ij}$ where for $i \neq j \in [r]$

we define

$$\gamma_{ij} = \begin{cases} \beta_{k\ell} + \min\left(\frac{(a'_k - b_k)}{(w_{k\ell})_k}, \frac{(b_\ell - a'_\ell)}{(w_{\ell k})_\ell}\right) & \text{if } i = k \text{ and } j = \ell \\ \beta_{ij} \text{ otherwise} \end{cases}$$

We show that c contradicts the assumption that b is a vector that agrees with a' on the maximum number of coordinates, among the vectors satisfying the properties stated in the beginning. We start by showing that for each coordinate $m \in [r]$, we either have $\gamma_{ij}(w_{ij})_m \leq 0$ for each $i \neq j \in [r]$ or $\gamma_{ij}(w_{ij})_m \geq 0$ for each $i \neq j \in [r]$. As we assume that this property holds for the vector b, we only need to show it for the coordinates k and ℓ.

Recall that we have either $a_i \leq b_i \leq a'_i$ or $a'_i \leq b_i \leq a_i$ and that $b_k < a'_k$. This implies that $a_k \leq b_k < a'_k$. In particular, this implies that $\beta_{ij}(w_{ij})_k \geq 0$ for each $i \neq j \in [r]$. Thus, it remains to show that $\gamma_{k\ell}(w_{k\ell})_k \geq 0$, which is the case as $(w_{k\ell})_k > 0$ implies $\beta_{k\ell} \geq 0$ and therefore also $\gamma_{k\ell} \geq 0$, as $\min\left(\frac{(a'_k - b_k)}{(w_{k\ell})_k}, \frac{(b_\ell - a'_\ell)}{(w_{\ell k})_\ell}\right) \geq 0$. Proceeding in the same manner, we get that $b_\ell > a'_\ell$ implies that $a'_\ell < b_\ell \leq a_\ell$. In particular, this implies that $\beta_{ij}(w_{ij})_\ell \leq 0$ for each $i \neq j \in [r]$. Thus, it remains to show that $\gamma_{kl}(w_{k\ell})_\ell \leq 0$, which is the case as $(w_{k\ell})_\ell \leq 0$ and $\gamma_{k\ell} \geq 0$.

Next, we show that for each $i \in [r]$, we either have $a_i \leq c_i \leq a'_i$ or $a_i \geq c_i \geq a'_i$. As $b_i = c_i$ for each $i \in [r] \setminus \{k, \ell\}$, we only need to show it for the coordinates k and ℓ. We have:

$$\begin{aligned} a_k \leq b_k &\leq b_k + \min\left(\frac{(a'_k - b_k)}{(w_{k\ell})_k}, \frac{(b_\ell - a'_\ell)}{(w_{\ell k})_\ell}\right)(w_{k\ell})_k \\ &\leq b_k + \frac{(a'_k - b_k)}{(w_{k\ell})_k}(w_{k\ell})_k = a'_k \end{aligned}$$

and

$$\begin{aligned} a_\ell \geq b_\ell &\geq b_\ell + \min\left(\frac{(a'_k - b_k)}{(w_{k\ell})_k}, \frac{(b_\ell - a'_\ell)}{(w_{\ell k})_\ell}\right)(w_{k\ell})_\ell \\ &\geq b_\ell + \frac{(b_\ell - a'_\ell)}{(w_{\ell k})_\ell}(w_{k\ell})_\ell = a'_\ell \end{aligned}$$

and therefore $a_k \leq c_k \leq a'_k$ and $a_\ell \geq c_\ell \geq a'_\ell$, as desired. In the second line, we used the fact that $(w_{k\ell})_\ell = -(w_{\ell k})_\ell \leq 0$. Furthermore, note that if $\min\left(\frac{(a'_k - b_k)}{(w_{k\ell})_k}, \frac{(b_\ell - a'_\ell)}{(w_{\ell k})_\ell}\right) = \frac{(a'_k - b_k)}{(w_{k\ell})_k}$, then $c_k = a'_k$, and otherwise $c_\ell = a'_\ell$. Therefore, c and a' differ in a smaller number of coordinates than b and a', which is a contradiction. □

We will use the following corollary of the above statement.

Corollary 6.23. *Let $c = \max_{i\in[r]} 1/a_i$. For each unit vector $v \in \mathbb{R}^r$ with $a + v \in H_a$, there exist α_{ij}'s with $v = \sum_{i\neq j\in[r]} \alpha_{ij}w_{ij}$ such that for each $k \in [r]$, either $\alpha_{ij}(w_{ij})_k \leq 0$ for each $i \neq j \in [r]$ or $\alpha_{ij}(w_{ij})_k \geq 0$ for each $i \neq j \in [r]$ and furthermore, $|\alpha_{ij}| \leq c$, for each $i \neq j \in [r]$.*

Proof. Let $v \in \mathbb{R}^r$ be a unit vector with $a + v \in H_a$. According to Theorem 6.22, there exist α_{ij}'s such that $v = \sum_{i\neq j\in[r]} \alpha_{ij}w_{ij}$ and for each $k \in [r]$, either $\alpha_{ij}(w_{ij})_k \leq 0$ for each $i \neq j \in [r]$ or $\alpha_{ij}(w_{ij})_k \geq 0$ for each $i \neq j \in [r]$. We show that $|\alpha_{ij}| \leq c$ for each $i \neq j \in [r]$. For the sake of contradiction, assume there exist $k \neq \ell \in [r]$ such that $|\alpha_{k\ell}| > c$. This implies

$$|\alpha_{k\ell}(w_{k\ell})_k| = \left|\alpha_{k\ell}\frac{a_k}{a_{k\ell}}\right| > \frac{1}{a_k}\cdot\frac{a_k}{a_{k\ell}} > 1$$

and therefore:

$$\left|v_k\right| = \Big|\sum_{i\neq j\in[r]} \alpha_{ij}(w_{ij})_k\Big| = \sum_{i\neq j\in[r]} \left|\alpha_{ij}(w_{ij})_k\right| \geq \left|\alpha_{k\ell}(w_{k\ell})_k\right| > 1$$

The second inequality follows as for each $i \neq j \in [r]$, $\alpha_{ij}(w_{ij})_k$ has the same sign. This is a contradiction as v is a unit vector and therefore $|v_k| \leq 1$. □

The main theorem now follows by carefully checking that the quadratic terms appearing when we generate a' are always positive.

Theorem 6.24. *For any maximal representable tuple a, there exists an $\varepsilon > 0$ such that for any $a' \in H_a \cap B(a, \varepsilon)$, a' is representable.*

Proof. Note that it is sufficient to prove the existence of an $\varepsilon > 0$, such that for any unit vector $v \in \mathbb{R}^r$ with $a + v \in H_a$ and every $0 \leq \delta < \varepsilon$, the tuple $a^\delta := a + \delta v$ is representable. As v is a unit vector with $a + v \in H_a$, according to Theorem 6.23, there exist α_{ij}'s such that $v = a + \sum_{i\neq j\in[r]} \alpha_{ij}w_{ij}$ and, moreover, for each $k \in [r]$, we either have $\alpha_{ij}(w_{ij})_k \leq 0$ for each $i \neq j \in [r]$ or $\alpha_{ij}(w_{ij})_k \geq 0$ for each $i \neq j \in [r]$, and $|\alpha_{ij}| \leq c := max_{i\in[r]}1/a_i$. We have $a^\delta := a + \delta v = a + \sum_{i\neq j\in[r]}(\delta\alpha_{ij})w_{ij}$. Consider now (a^δ_{ij}) with $a^\delta_{ij} = a_{ij} + \delta(\alpha_{ij} - \alpha_{ji})$. Note that $a^\delta_{ij} + a^\delta_{ji} = a_{ij} + a_{ji} \leq 1$ for each δ. Furthermore, for each $\delta \geq 0$ and $i \neq j \in [r]$, $|a^\delta_{ij} - a_{ij}| \leq \delta(|\alpha_{ij}| + |\alpha_{ji}|) \leq 2\delta c$. As c only depends on a, there exists some $\varepsilon' > 0$, independent of v, such that for each $0 \leq \delta \leq \varepsilon'$, (a^δ_{ij}) is a valid generator.

Next, we show that there exists some $0 < \varepsilon < \varepsilon'$, again independent of v, such that for each $0 \leq \delta \leq \varepsilon$, (a^δ_{ij}) generates a tuple with each coordinate being at least as large as the corresponding coordinate in a^δ. This implies that a^δ is representable, hence proving the claim. To that end, note that for an arbitrary $k \in [r]$ we have

$$\begin{aligned}
a_k^\delta &= \prod_{j\neq k} a_{kj}^\delta = \prod_{j\neq k}(a_{kj} + \delta\cdot(\alpha_{kj} - \alpha_{jk}))\\
&= a_k + \sum_{\ell\neq k}\delta\cdot(\alpha_{k\ell} - \alpha_{\ell k}) \prod_{j\notin\{k,\ell\}} a_{kj}\\
&\quad + \delta^2 \sum_{\ell'\neq k}\sum_{\ell''\notin\{k,\ell'\}} (\alpha_{k\ell'} - \alpha_{\ell' k})\cdot(\alpha_{k\ell''} - \alpha_{\ell'' k}) \prod_{j\notin\{k,\ell',\ell''\}} a_{kj}\\
&\quad + \delta^3 \sum_{\ell'\neq k}\sum_{\ell''\notin\{k,\ell'\}}\sum_{\ell'''\notin\{k,\ell',\ell''\}} (\alpha_{k\ell'} - \alpha_{\ell' k})\cdot(\alpha_{k\ell''} - \alpha_{\ell'' k})\\
&\qquad \cdot(\alpha_{k\ell'''} - \alpha_{\ell''' k}) \prod_{j\notin\{k,\ell',\ell'',\ell'''\}} a_{kj} + \dots
\end{aligned}$$

First, we take a look at the term linear in δ. We get:

$$\begin{aligned}
&\sum_{\ell\neq k}\delta\cdot(\alpha_{k\ell} - \alpha_{\ell k}) \prod_{j\notin\{k,\ell\}} a_{kj} = \sum_{\ell\neq k}\delta\cdot\left(\alpha_{k\ell}\frac{a_k}{a_{k\ell}} + \alpha_{\ell k}\frac{-a_k}{a_{k\ell}}\right)\\
&= \sum_{\ell\neq k}\delta\alpha_{k\ell}(w_{k\ell})_k + \delta\alpha_{\ell k}(w_{\ell k})_k = \sum_{i\neq j}\delta\alpha_{ij}(w_{ij})_k = \delta v_k
\end{aligned}$$

Next, we find a lower bound for the quadratic term. Note that for each $\ell' \neq \ell'' \in [r]\backslash\{k\}$, we have:

$$\begin{aligned}
&(\alpha_{k\ell'} - \alpha_{\ell' k})\cdot(\alpha_{k\ell''} - \alpha_{\ell'' k})\\
&= \left(\alpha_{k\ell'}(w_{k\ell'})_k\frac{a_{k\ell'}}{a_k} + \alpha_{\ell' k}(w_{\ell' k})_k\frac{a_{k\ell'}}{a_k}\right)\\
&\quad\cdot\left(\alpha_{k\ell''}(w_{k\ell''})_k\frac{a_{k\ell''}}{a_k} + \alpha_{\ell'' k}(w_{\ell'' k})_k\frac{a_{k\ell''}}{a_k}\right) \geq 0
\end{aligned}$$

as for all $i \neq j \in [r]$ $\alpha_{ij}(w_{ij})_k \geq 0$, or for all $i \neq j \in [r]$ $\alpha_{ij}(w_{ij})_k \leq 0$. Thus, each summand in the quadratic term is non-negative. Let us define

$$u := \max_{\ell'\neq\ell''\in[r]\backslash\{k\}} (\alpha_{k\ell'} - \alpha_{\ell' k})\cdot(\alpha_{k\ell''} - \alpha_{\ell'' k}) \geq 0$$

We can lower bound the quadratic term by:

$$\delta^2 \sum_{\ell'\neq k}\sum_{\ell''\notin\{\ell',k\}} (\alpha_{k\ell'} - \alpha_{\ell' k})\cdot(\alpha_{k\ell''} - \alpha_{\ell'' k}) \prod_{j\notin\{k,\ell',\ell''\}} a_{kj} \geq \delta^2\cdot u\cdot a_k$$

Next, we find a lower bound for each higher order term. Each such higher order term is the sum of expressions with the following form:

$$\delta^t \prod_{s=1}^{t}(\alpha_{k\ell_s} - \alpha_{\ell_s k}) \prod_{j\notin\{k,\ell_1,\dots,\ell_t\}} a_{kj}$$

$$\geq -\big|\delta^t \prod_{s=1}^{t}(\alpha_{k\ell_s} - \alpha_{\ell_s k}) \prod_{j \notin \{k,\ell_1,\ldots,\ell_t\}} a_{kj}\big|$$
$$\geq -\delta^t \cdot u \cdot (2c)^{t-2} \geq -u \cdot \delta^3 \cdot (2c)^r$$

for some distinct $\ell_1, \ldots, \ell_t \in [r] \setminus \{k\}$ and some $t \in \mathbb{N}$ with $t \geq 3$. As there are at most 2^r such terms, we can conclude that:

$$a_k^\delta = \prod_{j \neq k} a_{kj}^\delta \geq a_k + \delta v_k + \delta^2 \cdot u \cdot a_k - 2^r(u \cdot \delta^3 \cdot (2c)^r)$$
$$= a'(\delta)_k + u\delta^2(a_k - \delta \cdot 2^r \cdot (2c)^r) \geq a'(\delta)_k$$

for $\delta \leq \frac{a_k}{2^r \cdot (2c)^r} := \varepsilon''$ with ε'' only depending on the tuple $(a_1, \ldots, a_n)$ and not the vector v. Setting $\varepsilon = \min(\varepsilon', \varepsilon'')$, we can conclude that for each δ with $0 \leq \delta \leq \varepsilon$, $a'(\delta)$ is a representable tuple. This concludes the proof. □

Lemma 6.25. *For each maximal representable tuple a, there exists a weakly locally supportive hyperplane for S^o_{non} containing a.*

Proof. According to Theorem 6.21, there exist $h \in \mathbb{R}^r$ with $h \geq \mathbf{0}$ and $b \in \mathbb{R}$ such that $H_a = \{x \in \mathbf{R}^r \colon h^T x = b\}$. As $a \in H_a$, we have $h^T a = b$. Let $\varepsilon' > 0$ such that for each $a' \in H_a \cap B(a, \varepsilon')$, a' is a representable tuple. According to Theorem 6.24, such an ε' exists. We set $\varepsilon = \min(\varepsilon', \min_{i \in [r]} a_i) > 0$. Let $a'' \in B(a, \varepsilon)$ with $h^T a'' \leq b$. As $h^T a'' \leq b$, there exists some $\delta \geq 0$ such that $h^T a' = b$ with $a' := a'' + \delta h$. As $h^T(a - a') = b - b = 0$ and $a' - a'' = \delta h$, we can write $a - a'' = (a - a') + (a' - a'')$ with $(a - a')^T(a' - a'') = 0$. Thus, we can conclude that $||a - a'|| \leq ||a - a''|| \leq \varepsilon$. Hence, $a' \in H_a \cap B(a, \varepsilon)$ and therefore a' is a representable tuple. As $h \geq 0$, $a'' = a' - \delta h \geq \mathbf{0}$ is also a representable tuple and therefore $a'' \in S_{\text{rep}}$. Thus, $a'' \notin S^{\text{o}}_{\text{non}}$, as desired. □

CHAPTER 7

The Landscape of Distributed Complexities on Trees and Beyond

7.1 Introduction

This chapter proves the following theorem.

Theorem 7.1 (Informal version of Theorem 7.19)**.** *Let Δ be any fixed positive integer. Any locally checkable labeling problem on trees with maximum degree at most Δ with* LOCAL *complexity $o(\log^* n)$ has, in fact,* LOCAL *complexity $O(1)$.*

Our method, in a nutshell: Our approach is based on the round elimination technique [74], a highly successful technique for proving local lower bounds. In essence, round elimination is an explicit process that takes an LCL Π on trees as input and returns an LCL Π' with complexity exactly one round less. More precisely:

(1) If there is a T-round randomized algorithm $\mathcal{A}$ for Π, then there is also a $(T-1)$-round randomized algorithm $\mathcal{A}'$ for Π' such that the (local) failure probability of $\mathcal{A}'$ is bounded by a reasonable function of the (local) failure probability of $\mathcal{A}$.

(2) If we have a $(T-1)$-round algorithm $\mathcal{A}'$ for Π', we can use it to construct a T-round algorithm $\mathcal{A}$ for the original problem Π; if $\mathcal{A}'$ is deterministic, then $\mathcal{A}$ is deterministic as well.

So far, in the literature, the standard use case for applying round elimination has been to prove lower bounds for some *concrete, fixed* problem such as maximal matching or Lovász local lemma [73, 74, 22, 26]. We provide a novel application of round elimination by showing that, perhaps surprisingly, it can also be used to prove gap results, which are results that reason about *all* LCLs on a given graph class. More precisely, we show that with the tool of round elimination at hand, there is an elegant way to prove Theorem 7.1, which roughly proceeds as follows.

We start with any problem Π for which there exists a randomized algorithm $\mathcal{A}$ that solves Π in $T(n) = o(\log^* n)$ rounds, with probability $1 - 1/\operatorname{poly}(n)$. We fix some sufficiently large number n_0 of nodes, and apply bullet point (1) $T = T(n_0)$ times to get a 0-round algorithm $\mathcal{A}^{(T)}$ for a certain problem $\Pi^{(T)}$. By analyzing the development of the (local) failure probabilites of the algorithms appearing during the T applications of (1), we can show that algorithm $\mathcal{A}^{(T)}$ still has a large probability of success, and the fact that $\mathcal{A}^{(T)}$ is a 0-round algorithm enables us to infer that $\Pi^{(T)}$ is in fact so easy that it can be solved with a *deterministic* 0-round algorithm. Finally, we apply bullet point (2) T times to obtain a deterministic T-round algorithm for the original problem Π. Due to fixing the number of nodes to n_0, the obtained T-round algorithm is only guaranteed to produce a correct output on n_0-node trees; however, due to the nature of 0-round algorithms and the precise definition of the round elimination process (which are both independent of the number of nodes of the input graph), the obtained algorithm can be shown to also work on trees with an arbitrary number of nodes, with precisely the same, constant runtime $T(n_0) = O(1)$.

Unfortunately, the known approach for analyzing the change of (local) failure probability in bullet point (1) considers only the restricted setting of regular graphs and LCLs without inputs[1] (which usually suffices when proving a lower bound for a concrete LCL). One of our technical contributions is to develop an extension that also works in the general setting of irregular graphs and LCLs with inputs, which might be of independent interest.

We also prove the following theorem.

Theorem 7.2. *[Informal version of Theorem 7.20] Let d be a fixed positive constant. Any LCL on a d-dimensional oriented grid with local complexity $o(\log^* n)$ has, in fact, local complexity $O(1)$.*

7.2 Preliminaries

We use classical graph-theoretical notation, e.g. we write $G = (V, E)$ for an unoriented graph. A *half-edge* is a pair $h = (v, e)$, where $v \in V$, and $e \in E$ is an edge incident to v. We denote the set of half-edges of G by $H = H(G)$, i.e., $H = \{(v, e) \mid v \in e, v \in V, e \in E\}$. Furthermore, for every vertex v', we denote the set of half-edges $(v, e) \in H$ where $v = v'$ by $H[v']$, and for every edge e', we denote the set of half-edges $(v, e) \in H$ where $e = e'$ by $H[e']$. Often we assume that G additionally carries a labeling of vertices or half-edges. We use $B_G(u, r)$ to denote the ball of radius r around a node u in G and we call it the *r-hop neighborhood of u*. When talking about half-edges in $B_G(u, r)$, we mean all half-edges (v, e) such that $v \in B_G(u, r)$. For example, $B_G(u, 0)$ contains all half-edges incident to u.

The reader should keep in mind that our setting is graphs of maximum degree bounded by some constant Δ. This is sometimes explicitly stated (or it is implied by the constraints)

[1]We say that an LCL is an *LCL without inputs* if the correctness of a solution does not depend on input labels in the graph (such as lists in a list coloring problem). In the general setting, an LCL allows the correctness to depend on input labels (though we might emphasize this by using the term *LCL with inputs*).

but most of the time it is tacitly assumed. Of special interest to us will be the class of all trees with maximum degree at most Δ, which we denote by $\mathcal{T}$. Similarly, we denote the class of all forests with maximum degree at most Δ by $\mathcal{F}$. Moreover, for any positive integer n, any set N of positive integers, and any $\mathcal{G} \in \{\mathcal{F}, \mathcal{T}\}$, we will use $\mathcal{G}_n$, resp. $\mathcal{G}_N$, to denote the class of members of $\mathcal{G}$ with n nodes, resp. with a number of nodes that is contained in N.

Local Model and LCL Problems

In this section, we define our main model of computation and discuss the problem class considered in this work. Our main model of computation is the LOCAL model [236] extensively discussed in previous chapters. Yet, we will need to discuss some aspects again, as our following proofs rely on definitions being very precies.

The class of problems considered in this work are *LCL problems* (or *LCLs*, for short), which were introduced by Naor and Stockmeyer [258]. In their seminal paper, Naor and Stockmeyer provided a definition for LCL problems where input and output labels were assigned to nodes, and remarked that a similar definition can be given for edge-labeling problems. A modern definition that captures both kinds of LCL problems (and their combinations) assigns labels to *half-edges* (instead of vertices or edges). Before we can provide this definition, we need to define some required notions.

A *half-edge labeling* of a graph G (with labels from a set Σ) is a function $f\colon H(G) \to \Sigma$. A Σ_{in}-Σ_{out}-*labeled graph* is a triple $(G, f_{\text{in}}, f_{\text{out}})$ consisting of a graph G and two half-edge labelings $f_{\text{in}}\colon H(G) \to \Sigma_{\text{in}}$ and $f_{\text{out}}\colon H(G) \to \Sigma_{\text{out}}$ of G. We analogously define a Σ_{in}-*labeled graph* by omitting f_{out}.

We can now define an LCL problem as follows.

Definition 7.3 (LCL problem)**.** *An LCL problem Π is a quadruple $(\Sigma_{\text{in}}, \Sigma_{\text{out}}, r, \mathcal{P})$ where Σ_{in} and Σ_{out} are finite sets, r is a positive integer, and $\mathcal{P}$ is a finite collection of Σ_{in}-Σ_{out}-labeled graphs. A correct solution for an LCL problem Π on a Σ_{in}-labeled graph (G, f_{in}) is given by a half-edge labeling $f_{\text{out}}\colon H(G) \to \Sigma_{\text{out}}$ such that, for every node $v \in V(G)$, the triple $(B_G(v,r), f'_{\text{in}}, f'_{\text{out}})$ is isomorphic to a member of $\mathcal{P}$, where f'_{in} and f'_{out} are the restriction of f_{in} and f_{out}, respectively, to $B_G(v,r)$.*

Intuitively, the collection $\mathcal{P}$ provides the constraints of the problem by specifying how a correct output looks *locally*, depending on the respective local input. From the definition of a correct solution for an LCL problem it follows that members of $\mathcal{P}$ that have radius $> r$ can be ignored. Also the finiteness of $\mathcal{P}$ automatically implies that we are restricting ourselves to graphs of degree at most Δ for some constant Δ.

The main tool in our proof of Theorem 7.1, the round elimination technique, applies (directly) only to a subclass of LCL problems: roughly speaking, it is required that the local correctness constraints specified by $\mathcal{P}$ can be translated into node and edge constraints, i.e., correctness constraints that can be verified by looking at the label configurations on each edge and around each node. The definition of this subclass of LCL problems is given in the following. Note that, despite the complicated appearance, the definition is actually quite intuitive: essentially, in order to define the LCL problem, we simply specify a set of label configurations that are allowed on an edge, a set of label configurations that are allowed around a node, and an input-output label relation that specifies for each input label which output label is allowed at the same half-edge.

Definition 7.4 (Node-edge-checkable LCL problem)**.** *A* node-edge-checkable LCL Π *is a quintuple* $(\Sigma_{\text{in}}^{\Pi}, \Sigma_{\text{out}}^{\Pi}, \mathcal{N}_{\Pi}, \mathcal{E}_{\Pi}, g_{\Pi})$*, where* Σ_{in}^{Π} *and* $\Sigma_{\text{out}}^{\Pi}$ *are finite sets,* $\mathcal{E}_{\Pi}$ *is a collection of cardinality-*2 *multisets* $\{\mathsf{B}_1, \mathsf{B}_2\}$ *with* $\mathsf{B}_1, \mathsf{B}_2 \in \Sigma_{\text{out}}^{\Pi}$*,* $\mathcal{N}_{\Pi} = (\mathcal{N}_{\Pi}^{1}, \mathcal{N}_{\Pi}^{2}, \dots)$ *consists of collections* $\mathcal{N}_{\Pi}^{i}$ *of cardinality-*i *multisets* $\{\mathsf{A}_1, \dots, \mathsf{A}_i\}$ *with* $\mathsf{A}_1, \dots, \mathsf{A}_i \in \Sigma_{\text{out}}^{\Pi}$*, and* $g_{\Pi} \colon \Sigma_{\text{in}}^{\Pi} \to 2^{\Sigma_{\text{out}}^{\Pi}}$ *is a function that assigns to each label from* Σ_{in}^{Π} *a subset of the labels of* $\Sigma_{\text{out}}^{\Pi}$*. A correct solution for a node-edge-checkable LCL* Π *on a* Σ_{in}^{Π}*-labeled graph* (G, f_{in}) *is given by a half-edge labeling* $f_{\text{out}} \colon H(G) \to \Sigma_{\text{out}}^{\Pi}$ *such that*

1. *for every node* $v \in V(G)$*, the multiset consisting of the labels assigned by* f_{out} *to the half-edges in* $H[v]$ *is contained in* $\mathcal{N}_{\Pi}^{\deg(v)}$*,*
2. *for every edge* $e \in E(G)$*, the multiset consisting of the labels assigned by* f_{out} *to the half-edges in* $H[e]$ *is contained in* $\mathcal{E}_{\Pi}$*, and*
3. *for every half-edge* $h \in H(G)$*, the label* $f_{\text{out}}(h)$ *is contained in the label set* $g_{\Pi}(f_{\text{in}}(h))$*.*

We call $\mathcal{N}_{\Pi}$ *the* node constraint *of* Π *and* $\mathcal{E}_{\Pi}$ *the* edge constraint *of* Π*. Moreover, we call the elements* $\{\mathsf{A}_1, \dots, \mathsf{A}_i\}$ *of* $\mathcal{N}_{\Pi}^{i}$ node configurations *and the elements* $\{\mathsf{B}_1, \mathsf{B}_2\}$ *of* $\mathcal{E}_{\Pi}$ edge configurations *(of* Π*). In a* LOCAL *algorithm solving a node-edge-checkable problem* Π*, each node is supposed to output a label for each incident half-edge such that the induced global half-edge labeling is a correct solution for* Π*.*

Even though the round elimination technique can be applied directly only to node-edge-checkable LCL problems, the results we obtain apply to all LCL problems. The reason for this is that for each LCL problem Π there exists a node-edge-checkable LCL problem Π' such that the time complexities of Π and Π' differ only by an additive constant, as we show in Lemma 7.7. This fact suffices to lift our results for node-edge-checkable LCL problems to general LCL problems: in particular, the existence of an LCL problem with time complexity in $\omega(1)$ and $o(\log^* n)$ would imply the existence of a node-edge-checkable LCL problem with the same complexity constraints, leading to a contradiction.

Before stating and proving Lemma 7.7, we formally define the local failure probability of an algorithm solving a node-edge-checkable LCL, and the complexity of an LCL problem.

Definition 7.5 (Local failure probability)**.** *Let* $\Pi = (\Sigma_{\text{in}}^{\Pi}, \Sigma_{\text{out}}^{\Pi}, \mathcal{N}_{\Pi}, \mathcal{E}_{\Pi}, g_{\Pi})$ *be some node-edge-checkable LCL problem. We say that a half-edge labeling* $f_{\text{out}} \colon H(G) \to \Sigma_{\text{out}}$ *is* incorrect on some edge $e = \{u, v\}$ *of graph* (G, f_{in}) *if*

1. $\{f_{\text{out}}((u,e)), f_{\text{out}}((v,e))\} \notin \mathcal{E}_\Pi$*, or*

2. $f_{\text{out}}((u,e)) \notin g_\Pi(f_{\text{in}}((u,e)))$ *or* $f_{\text{out}}((v,e)) \notin g_\Pi(f_{\text{in}}((v,e)))$*.*

Similarly, we say that f_{out} *is* incorrect at some node v *if*

1. $\{f_{\text{out}}(h)\}_{h \in H[v]} \notin \mathcal{N}_\Pi^{\deg(v)}$*, or*

2. $f_{\text{out}}(h) \notin g_\Pi(f_{\text{in}}(h))$ *for some* $h \in H[v]$*.*

We say that an algorithm $\mathcal{A}$ fails on some edge e, resp. at some node v, *if the output produced by* $\mathcal{A}$ *is incorrect on* e*, resp. at* v*. Furthermore, we say that a (randomized) algorithm* $\mathcal{A}$ *has* local failure probability p *on some graph* G *if* p *is the smallest (real) number such that, for each edge* e *and node* v *in* G*, the probability that* $\mathcal{A}$ *fails on* e*, resp. at* v*, is upper bounded by* p*. Moreover, for each* n*, the local failure probability of* $\mathcal{A}$ *on some class of* n*-node graphs is the maximum of the local failure probabilities of* $\mathcal{A}$ *on the graphs in the class. (In contrast, the definition of* (global) failure probability *is as commonly used, i.e., we say that* $\mathcal{A}$ *has (global) failure probability* $p = p(n)$ *if the (worst-case) probability that* $\mathcal{A}$ *does not produce a correct solution for* Π *is upper bounded by* p *and* p *is minimal under this constraint.)*

The LOCAL complexity of a (node-edge-checkable or common) LCL is simply the minimum complexity of an algorithm $\mathcal{A}$ that solves it on all graphs.

Definition 7.6 (Complexity of an LCL problem)**.** *The* determinstic (round) complexity *of an LCL* Π *is the function* $T\colon \mathbb{N} \to \mathbb{N} \cup \{0\}$ *satisfying that for each* $n \in \mathbb{N}$*, there exists a deterministic algorithm* $\mathcal{A}_n$ *solving* Π *in* $T(n)$ *rounds on all* n*-node graphs* G *with each half-edge labeled with a label from* Σ_{in}*, but no deterministic algorithm solving* Π *in* $T(n) - 1$ *rounds on this class of graphs. The* randomized (round) complexity *of an LCL* Π *is defined analogously, where deterministic algorithms are replaced by randomized algorithms with a (global) failure probability of at most* $1/n$*.*

When we talk about the complexity of an LCL on trees, we further restrict the above definition to graphs that are trees (and similarly for other graph classes).

Now we are ready to state and prove the following lemma, which ensures that we can restrict attention to node-edge-checkable LCLs.

Lemma 7.7. *For any LCL problem* Π*, there exists a node-edge-checkable LCL problem* Π' *such that (in both the randomized and deterministic* LOCAL *model) the complexities of* Π *and* Π' *on trees (and on forests) are asymptotically the same.*

Proof. Suppose $\Pi = (\Sigma_{\text{in}}, \Sigma_{\text{out}}, r, \mathcal{P})$ is an LCL. We create a node-edge-checkable LCL $\Pi' = (\Sigma_{\text{in}}^{\Pi'}, \Sigma_{\text{out}}^{\Pi'}, \mathcal{N}_{\Pi'}, \mathcal{E}_{\Pi'}, g_{\Pi'})$ as follows:

- $\Sigma_{\text{in}}^{\Pi'} = \Sigma_{\text{in}}$.
- $\Sigma_{\text{out}}^{\Pi'}$ contains all possible labelings of r-hop neighborhoods of a node, each neighborhood has marked a special half-edge, each vertex and each edge has an order on

incident half-edges and each half-edge is labeled with a label from Σ_{out}, moreover, the labeling by Σ_{out} has to be accepted by $\mathcal{P}$.

- $\mathcal{N}_{\Pi'}$ contains such sets $S = \{\sigma_1, \ldots, \sigma_d\}$ with $\sigma_i \in \Sigma_{\text{out}}^{\Pi'}$ such that there exists an r-hop neighborhood N of a node u of degree d together with each node and each edge having an order on incident half edges and such that half-edges have labels from Σ_{out} such that we can assign labels from S to half-edges around u in such a way that each σ_i assigned to (u, e_i) describes N with the special half-edge of σ_i being e_i.
- $\mathcal{E}_{\Pi'}$ is defined analogously to $\mathcal{N}_{\Pi'}$, but we require an existence of a neighborhood of an edge e that is consistent from the perspective of labels from $\Sigma_{\text{out}}^{\Pi'}$ assigned to (u, e) and (v, e).
- $g_{\Pi'}$ maps each label $\tau \in \Sigma_{\text{in}}$ to the set of labels $\sigma \in \Sigma_{\text{out}}^{\Pi'}$ such that the special half-edge of σ is labeled by τ.

Suppose we have a valid solution of Π. Then, in r rounds each half-edge can decide on its Π'-label by encoding its r-hop neighborhood, including port numbers of each vertex and each edge that give ordering on its half-edges, into a label from $\Sigma_{\text{out}}^{\Pi'}$. The constraints $\mathcal{N}_{\Pi'}, \mathcal{E}_{\Pi'}, g_{\Pi'}$ will be satisfied.

On the other hand, suppose we have a valid solution for Π'. In 0-rounds, each half-edge (v, e) can label itself with the label on the special half-edge in its Π'-label $f'_{\text{out}}((v, e))$. We claim that the Π-labeling we get this way around a half-edge (u, e) is isomorphic to the Σ_{in}-labeling described by the label $f_{\text{out}}((v, e))$, hence the new labeling is a solution to Π'. To see this, consider running a BFS from (v, e). The node constraints $\mathcal{N}_{\Pi'}$ and the edge constraints $\mathcal{E}_{\Pi'}$ are ensuring that the labels from $\Sigma_{\text{out}}^{\Pi'}$ of visited half-edges are describing compatible neighborhoods, while the function g ensures that the description of Σ_{in} labels in the r-hop neighborhood of u by the labels from $\Sigma_{\text{out}}^{\Pi'}$ agrees with the actual Σ_{in} labeling of the r-hop neighborhood of u. As the label $f'_{\text{out}}((v, e))$ needs to be accepted by $\mathcal{P}$, we get that $\mathcal{P}$ accepts the r-hop neighborhood of u, as needed. □

It is crucial that the way in which we define the node-edge-checkable LCL problem Π' in Lemma 7.7 guarantees that the considered (input-labeled) graph class remains the same as for Π (and does not turn into a graph class with a promise on the distribution of the input labels, which would be the result of the straightforward approach of defining Π' by encoding the input labels contained in a constant-sized ball in Π in a single input label in Π', and doing the same for output labels). If this property was not guaranteed, it would be completely unclear (and perhaps impossible) how to extend the round elimination framework of [74] to our setting with input labels.

7.2.1 Order-Invariant Algorithms

In this section, we formally define the notion of an order-invariant algorithm and introduce further computational models of interest. We also show that oftentimes order-invariant algorithms can be sped up to improve the round/probe complexity, both in the LOCAL and the VOLUME model.

Definition 7.8 (Order-invariant LOCAL algorithm [258])**.** *A deterministic $T(n)$-round LOCAL algorithm $\mathcal{A}$ is called order-invariant if the following holds: Consider two assignments of distinct identifiers to nodes in $B_G(v, T(n))$ denoted by μ and μ'. Assume that for all $u, w \in B_G(v, T(n))$ it holds that $\mu(u) > \mu(w)$ if and only if $\mu'(u) > \mu'(w)$. Then, the output of $\mathcal{A}$ on the set of half-edges $H[v]$ will be the same in both cases.*

7.3 The Local Model Gap on Trees

In this section we prove the $\omega(1) - o(\log^* n)$ gap for LCLs on trees in the LOCAL model. We do so by proving Theorem 7.19 (which is the slightly more formal version of Theorem 7.1) by explicitly designing, for any given (node-edge-checkable) LCL problem Π with complexity $o(\log^* n)$, a constant-round algorithm. A very rough outline of our approach is to generate from Π a sequence of node-edge-checkable LCL problems of decreasing randomized local complexities (where we allow the local failure probability to grow along the problems in the sequence), find a problem in the sequence that can be solved in 0 rounds with a reasonably low local failure probability, show that there exists a 0-round deterministic algorithm for that problem, and turn this algorithm into a constant-round algorithm for Π by going back up the sequence of problems and arguing that the deterministic complexities increase slowly along the sequence in this direction. While the round elimination framework [74, 22] provides a blueprint how to generate a suitable sequence, it unfortunately only does so for LCLs on regular trees without inputs. We provide an extension of the framework that also works for LCLs on irregular trees (or forests) with inputs.

We will start in Section 7.3.1 by extending the definition of the round elimination problem sequence to the setting with inputs (and taking care of a technical issue). In Section 7.3.2, we will carefully bound the evolution of failure probabilities along a sequence of algorithms with decreasing runtimes that solve the problems in the defined problem sequence. Section 7.3.3 takes care of the reverse step, i.e., showing that the deterministic complexities of the problems in the problem sequence do not increase fast when traversed towards Π. Finally, in Section 7.3.4, we will put everything together and prove Theorem 7.19.

7.3.1 The Problem Sequence

Similarly to the approach in [74], we define, for any node-edge-checkable LCL problem Π, two node-edge-checkable problems $\mathcal{R}(\Pi)$ and $\overline{\mathcal{R}}(\Pi)$. The problems in the aforementioned sequence are then obtained by iteratively applying $\overline{\mathcal{R}}(\mathcal{R}(\cdot))$, starting with Π.

Definition 7.9 ($\mathcal{R}(\Pi)$)**.** *Let $\Pi = (\Sigma_{\text{in}}^{\Pi}, \Sigma_{\text{out}}^{\Pi}, \mathcal{N}_{\Pi}, \mathcal{E}_{\Pi}, g_{\Pi})$ be a node-edge-checkable LCL problem. We define a new node-edge-checkable LCL problem $\mathcal{R}(\Pi) = (\Sigma_{\text{in}}^{\mathcal{R}(\Pi)}, \Sigma_{\text{out}}^{\mathcal{R}(\Pi)},$*

$\mathcal{N}_{\mathcal{R}(\Pi)}, \mathcal{E}_{\mathcal{R}(\Pi)}, g_{\mathcal{R}(\Pi)})$ by specifying the five components. We start by setting $\Sigma_{\text{in}}^{\mathcal{R}(\Pi)} := \Sigma_{\text{in}}^{\Pi}$ and $\Sigma_{\text{out}}^{\mathcal{R}(\Pi)} := 2^{\Sigma_{\text{out}}^{\Pi}}$, i.e., the input label set of $\mathcal{R}(\Pi)$ is simply the input label set of Π, and the output label set of $\mathcal{R}(\Pi)$ is the power set of the output label set of Π. Next, we define $g_{\mathcal{R}(\Pi)}$ by setting $g_{\mathcal{R}(\Pi)}(\ell) := 2^{g_{\Pi}(\ell)}$ for any label $\ell \in \Sigma_{\text{in}}^{\Pi}$, i.e., intuitively speaking, in problem $\mathcal{R}(\Pi)$ an input label ℓ on some half-edge requires that the output label on the same half-edge is a subset of the set of output labels that were allowed in Π on a half-edge with input label ℓ.

We define the edge constraint $\mathcal{E}_{\mathcal{R}(\Pi)}$ of $\mathcal{R}(\Pi)$ as the set of all cardinality-2 multisets $\{\mathsf{B}_1, \mathsf{B}_2\}$ such that $\mathsf{B}_1, \mathsf{B}_2 \in \Sigma_{\text{out}}^{\mathcal{R}(\Pi)}$ and, for all $\mathsf{b}_1 \in \mathsf{B}_1$, $\mathsf{b}_2 \in \mathsf{B}_2$, we have $\{\mathsf{b}_1, \mathsf{b}_2\} \in \mathcal{E}_{\Pi}$. Finally, we define the node constraint $\mathcal{N}_{\mathcal{R}(\Pi)}$ of $\mathcal{R}(\Pi)$ as follows. For each integer $i \geq 1$, define $\mathcal{N}_{\mathcal{R}(\Pi)}^{i}$ as the set of all cardinality-i multisets $\{\mathsf{A}_1, \dots, \mathsf{A}_i\}$ such that $\mathsf{A}_1, \dots, \mathsf{A}_i \in \Sigma_{\text{out}}^{\mathcal{R}(\Pi)}$ and there exists some selection $(\mathsf{a}_1, \dots, \mathsf{a}_i)$ of labels from $\mathsf{A}_1 \times \dots \times \mathsf{A}_i$ such that $\{\mathsf{a}_1, \dots, \mathsf{a}_i\} \in \mathcal{N}_{\Pi}^{i}$.

Note that our definition of $\mathcal{R}(\Pi)$ differs slightly from the usual definition of $\mathcal{R}(\Pi)$ as given in, e.g., [26] (beyond the obvious differences due to the fact that we consider LCL problems with input labels): our definition does not remove so-called "non-maximal" configurations. Removing such configurations can be beneficial when trying to determine the complexity of specific problems, but is not required (or helpful) in our setting, where we want to argue about the complexity of many problems at once.

Definition 7.10 ($\overline{\mathcal{R}}(\Pi)$)**.** *The problem $\overline{\mathcal{R}}(\Pi)$ differs from $\mathcal{R}(\Pi)$ only in the node and edge constraints; for the remaining three parameters we set $\Sigma_{\text{in}}^{\overline{\mathcal{R}}(\Pi)} := \Sigma_{\text{in}}^{\mathcal{R}(\Pi)}$, $\Sigma_{\text{out}}^{\overline{\mathcal{R}}(\Pi)} := \Sigma_{\text{out}}^{\mathcal{R}(\Pi)}$, and $g_{\overline{\mathcal{R}}(\Pi)} := g_{\mathcal{R}(\Pi)}$.*

We define the node constraint $\mathcal{N}_{\overline{\mathcal{R}}(\Pi)}$ of $\overline{\mathcal{R}}(\Pi)$ as follows. For each integer $i \geq 1$, define $\mathcal{N}_{\overline{\mathcal{R}}(\Pi)}^{i}$ as the set of all cardinality-i multisets $\{\mathsf{A}_1, \dots, \mathsf{A}_i\}$ such that $\mathsf{A}_1, \dots, \mathsf{A}_i \in \Sigma_{\text{out}}^{\overline{\mathcal{R}}(\Pi)}$ and, for all $(\mathsf{a}_1, \dots, \mathsf{a}_i) \in \mathsf{A}_1 \times \dots \times \mathsf{A}_i$, we have $\{\mathsf{a}_1, \dots, \mathsf{a}_i\} \in \mathcal{N}_{\Pi}^{i}$. Moreover, we define the edge constraint $\mathcal{E}_{\overline{\mathcal{R}}(\Pi)}$ of $\overline{\mathcal{R}}(\Pi)$ as the set of all cardinality-2 multisets $\{\mathsf{B}_1, \mathsf{B}_2\}$ such that $\mathsf{B}_1, \mathsf{B}_2 \in \Sigma_{\text{out}}^{\overline{\mathcal{R}}(\Pi)}$ and there exists some selection $(\mathsf{b}_1, \mathsf{b}_2)$ of labels from $\mathsf{B}_1 \times \mathsf{B}_2$ such that $\{\mathsf{b}_1, \mathsf{b}_2\} \in \mathcal{E}_{\Pi}$.

Note that, although the function $\overline{\mathcal{R}}(\cdot)$ can take any arbitrary node-edge-checkable LCL problem as argument, we will use as arguments only problems that are of the form $\mathcal{R}(\Pi)$ for some node-edge-checkable LCL problem Π.

Recall that $\mathcal{T}$, resp. $\mathcal{F}$, denotes the class of all trees, resp. forests, of maximum degree at most Δ. Before turning to the analysis of the evolution of the aforementioned failure probabilities, there is a technical issue we have to discuss. A crucial argument in said analysis is, roughly speaking, that if you consider some (sufficiently small) neighborhood of a node (or edge), and a set of extensions of this neighborhood via different edges leaving the neighborhood[2], then there must also be a graph in the considered graph class that (simultaneously) contains all extensions of the set (together with the neighborhood). Here the term "considered graph class" describes the input graph class restricted to the members that are consistent with the knowledge of the nodes about the number n of nodes, i.e., in particular if all nodes are aware of the exact value of n (as they are in our definition of the LOCAL model), then the considered graph class contains only n-node graphs. Now, if the input graph class is $\mathcal{T}$, it follows that the aforementioned crucial argument does not hold in general: if all extensions in the considered set "conclude" the tree (i.e., do not have leaving edges except those connecting them to the considered initial neighborhood) and the combined number of nodes in the initial neighborhood and the considered extensions does not happen to be precisely n, then there is no n-node tree that contains the neighborhood together with all considered extensions.

We solve this issue by proving our main theorem first for the class $\mathcal{F}$ of forests (which do not have the aforementioned issue as the number of nodes[3] in some maximal connected component is not known to the nodes) in Theorem 7.18 and then lifting it to $\mathcal{T}$ in Theorem 7.19 by showing that, for node-edge-checkable LCL problems, a complexity of $o(\log^* n)$ on trees implies a complexity of $o(\log^* n)$ on forests. Lemma 7.11 provides this relation between trees and forests; in Sections 7.3.2 and 7.3.3 we will then exclusively work with forests. Note that the complexity of any LCL problem Π on $\mathcal{F}$ is trivially at least as large as its complexity on $\mathcal{T}$ as $\mathcal{T}$ is a subclass of $\mathcal{F}$.

Lemma 7.11. *Let Π be some node-edge-checkable LCL problem that has deterministic, resp. randomized, complexity $o(\log^* n)$ on $\mathcal{T}$. Then the deterministic, resp. randomized, complexity of Π on $\mathcal{F}$ is in $o(\log^* n)$.*

Proof. Let $\mathcal{A}$ be a (deterministic or randomized) algorithm solving Π on $\mathcal{T}$ in $T(n) \in o(\log^* n)$ rounds (and observe that the existence of $\mathcal{A}$ (even if it is randomized) implies that a correct global solution exists). We design a new algorithm $\mathcal{A}'$ solving Π on $\mathcal{F}$ in $o(\log^* n)$ rounds. Algorithm $\mathcal{A}'$ proceeds as follows on any n-node input forest $G' \in \mathcal{F}$, where, for any node u, we denote the (maximal) connected component containing u by C_u and the number of nodes in C_u by $|C_u|$.

First, each node collects its $(2T(n^2) + 2)$-hop neighborhood in G'. Then, based on the collected information, each node u determines whether there exists a node v in C_u such that the $(T(n^2) + 1)$-hop neighborhood of v contains all of C_u.

If such a node v exists, then each node in C_u is aware of the whole component C_u and can simply choose the same solution for Π on C_u (by mapping component C_u (including unique identifiers or random bits) in some arbitrary, but fixed, deterministic fashion to

[2]An extension via some leaving edge is simply a possibility of how the graph could continue for the next hop beyond the respectively chosen leaving edge that is consistent with (some graph in) the considered graph class.

[3]We remark that the issue does not occur in the variant of the LOCAL model in which nodes are only aware of some upper bound on the number of nodes, but we believe that it is important to ensure that the correctness of (our) results does not depend on such minor details in the model specification.

some correct solution), and then output the part of the solution it is responsible for. Note that this requires all nodes in C_u to be distinguishable from each other (so that each node knows which part of the solution on C_u it is responsible for); for deterministic algorithms this is guaranteed by the unique identifiers, for randomized algorithms it is guaranteed with probability at least $1 - 1/n^2$ by having each node interpret its first $\lceil 4 \log n \rceil$ random bits as an identifier (which guarantees uniqueness of the created identifiers with probability at least $1 - 1/n^2$).

If no such node v exists, then u simply executes $\mathcal{A}$ with input parameter[4] n^2 (and each node in C_u will do likewise). In this case, due to the fact that no $(T(n^2) + 1)$-hop node neighborhood fully contains C_u, it holds for each node v in C_u that the $(T(n^2) + 1)$-hop neighborhood of v in G' is isomorphic to the $(T(n^2)+1)$-hop neighborhood of some node w in some n^2-node tree $G \in \mathcal{T}$. Hence, if $\mathcal{A}'$ fails on some node v in C_u or on some edge incident to v, then $\mathcal{A}$ fails on some node w in some n^2-node tree, or on some edge incident to w. Since the failure probability of $\mathcal{A}$ on n^2-node trees is at most $1/n^2$, it holds for any node w in any n^2-node tree that the probability that $\mathcal{A}$ fails on w or an edge incident to w is at most $1/n^2$. It follows for each node v in C_u that the probability that $\mathcal{A}'$ fails on v or an edge incident to v is at most $1/n^2$.

Now, a union bound over all components C_u of the first kind ("such a node v exists") and all nodes in components C_u of the second kind ("no such node v exists") yields that $\mathcal{A}'$ fails with probability at most $1/n$. Note that if $\mathcal{A}$ is deterministic, then all of the above failure probabilities are 0, and $\mathcal{A}'$ is deterministic as well.

For the runtime of $\mathcal{A}'$, observe that in either of the two considered cases, the initial collection of u's $(2T(n^2) + 2)$-hop neighborhood suffices to compute u's output in $\mathcal{A}'$. Since $T(n) \in o(\log^* n)$ implies $2T(n^2) + 2 \in o(\log^* n)$, it follows that the runtime of $\mathcal{A}'$, and therefore also the complexity of Π on $\mathcal{F}$, is in $o(\log^* n)$. □

7.3.2 From Harder to Easier Problems

Recall that, for any set N of positive integers, $\mathcal{F}_N$ denotes the class of forests with a number of nodes that is contained in N. The goal of this section is to prove the following theorem.

Theorem 7.12. *Let Π be a node-edge-checkable LCL problem and $\mathcal{A}$ a randomized algorithm solving Π on $\mathcal{F}$ with runtime $T(n)$ and local failure probability at most $p \leq 1$.*[5] *Let N be the set of all positive integers n satisfying $T(n) + 2 \leq \log_\Delta n$. Then, there exists a randomized algorithm $\mathcal{A}'$ solving $\overline{\mathcal{R}}(\mathcal{R}(\Pi))$ on $\mathcal{F}_N$ with runtime* $\max\{0, T(n) - 1\}$ *and*

[4]Recall that each node receives as input a parameter representing the number of nodes. Note that nothing prevents us from executing an algorithm using an input parameter that does not represent the correct number of nodes.

[5]Note that (bounds on) local failure probabilities such as p also (possibly) depend on the number n of nodes; however, for better readability we will omit this dependency.

local failure probability at most $Sp^{1/(3\Delta+3)}$*, where*

$$S = (10\Delta(|\Sigma_{\text{in}}^{\Pi}| + \max\{|\Sigma_{\text{out}}^{\Pi}|, |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|\}))^{4\Delta^{T(n)+1}}.$$

In other words, we want to show, roughly speaking, that we can solve $\overline{\mathcal{R}}(\mathcal{R}(\Pi))$ at least one round faster than Π if we allow the indicated increase in the local failure probability of the randomized algorithm solving the problem.

For the proof of Theorem 7.12, we will make use of an approach that is an extension of the approach known for the setting of LCLs on regular trees without inputs (see, e.g., [22, 26, 36]). More specifically , we will explicitly define an algorithm $\mathcal{A}'$ (depending on $\mathcal{A}$) that satisfies the properties stated in Theorem 7.12. While in the regular setting without inputs, the possibilities how the input graph could continue beyond the view of a node v differ only in the random bits of the nodes beyond v's view, the irregularity and inputs in our setting require us to first perform a simulation step in the definition of $\mathcal{A}'$ that simulates all possible topologies and inputs that a node could encounter (not too far) beyond its view before considering the randomness contained in each such extension defined by the combination of topology and inputs. As we show, the resulting more complex definition of $\mathcal{A}'$ still allows us to give an upper bound on the increase of the local failure probability from $\mathcal{A}$ to $\mathcal{A}'$ that suffices for our purposes.

It should be noted that, due to the fact that we consider a large class of LCL problems at once (and not a single fixed LCL problem, in which case certain simplification techniques might be applicable), the increase in the number of output labels from Π to $\overline{\mathcal{R}}(\mathcal{R}(\Pi))$ is doubly exponential. Since the bound on the local failure probability of $\mathcal{A}'$ depends (superlinearly) on the number of output labels of Π, and, ultimately, we want to apply Theorem 7.12 iteratively (starting with Π), we cannot apply Theorem 7.12 more than $\Theta(\log^* n)$ times before the local failure probability grows too large. This provides a (different) explanation why we cannot extend the $\omega(1) - o(\log^* n)$ gap further with our approach (which also follows from the fact that there are problems with complexity $\Theta(\log^* n)$).

Note that we will define $\mathcal{A}'$ for all forests from $\mathcal{F}$, but will prove the guarantee on the local failure probability of $\mathcal{A}'$ only for the forests in $\mathcal{F}_N$. Also recall that, for any node u in a graph G, we denote the r-hop neighborhood of u in G by $B_G(u, r)$. By abuse of notation, we will use $B_G(u, r)$ both for all the information contained in the r-hop neighborhood of u (i.e., the topology, inputs, and random bits) and for the respective subgraph of G (including input information, but no random bits).

Deriving $\mathcal{A}'$: Let $\mathcal{A}$ be a randomized algorithm for some node-edge-checkable LCL problem Π with runtime $T = T(n)$. If $T = 0$, we can simply let $\mathcal{A}'$ simulate $\mathcal{A}$, and then, for each half-edge h, transform the intermediate output $\ell \in \Sigma_{\text{out}}^{\Pi}$ returned by $\mathcal{A}$ on h into the final output $\{\{\ell\}\} \in \Sigma_{\text{out}}^{\overline{\mathcal{R}}(\mathcal{R}(\Pi))}$ on h. By construction, $\mathcal{A}'$ fails on some edge, resp. node, if and only if $\mathcal{A}$ fails on the same edge, resp. node; since $p \leq Sp^{1/(3\Delta+3)}$ for the S specified in Theorem 7.12, it follows that $\mathcal{A}'$ satisfies the properties required in Theorem 7.12. Hence, we will assume in the following that $T \geq 1$.

In order to derive $\mathcal{A}'$ from $\mathcal{A}$, we first derive an "intermediate" algorithm $\mathcal{A}_{1/2}$ for $\mathcal{R}(\Pi)$ from $\mathcal{A}$, and then we derive $\mathcal{A}'$ from $\mathcal{A}_{1/2}$. As we will show later, $\mathcal{A}_{1/2}$ solves $\mathcal{R}(\Pi)$ (on $\mathcal{F}_N$) with a moderately increased local failure probability (and, intuitively speaking, very slightly reduced runtime) compared to $\mathcal{A}$; a similar moderate increase in local failure probability (and slight decrease in runtime) is incurred when going from $\mathcal{A}_{1/2}$ to $\mathcal{A}'$.

Deviating from the usual convention that nodes are the entities performing the computation, we will assume for $\mathcal{A}_{1/2}$ that the edges of the input forest perform the computation. This is not in contradiction with the definition of the LOCAL model as $\mathcal{A}_{1/2}$ is only a construct defined for the design of $\mathcal{A}'$; the actual computation in $\mathcal{A}'$ is performed by the nodes. Algorithm $\mathcal{A}_{1/2}$ proceeds as follows.

Each edge $e = \{u, v\}$ in the input forest $G \in \mathcal{F}$ first collects all information (i.e., the topology, inputs, and random bits) contained in the union $B_G(e, T - 1/2) := B_G(u, T - 1) \cup B_G(v, T - 1)$ of the $(T - 1)$-hop neighborhoods of u and v. Then, e determines the output label ℓ' it outputs on half-edge (u, e) as follows, depending on some parameter $0 < K \leq 1$ that we will choose later. Label ℓ' is simply the set of all labels ℓ such that there exists an input forest $G' \in \mathcal{F}$ (including input labels) and an edge e' in G' such that $B_{G'}(e', T - 1/2) \cong B_G(e, T - 1/2)$ and the probability that the node u' corresponding to u in the isomorphism outputs ℓ on (u', e') according to $\mathcal{A}$ is at least K, conditioned on the assumption that the random bits in $B_{G'}(e', T - 1/2)$ are the same as in $B_G(e, T - 1/2)$. Here, the isomorphism is w.r.t. the topology and the input labels. In other words, in $\mathcal{A}_{1/2}$, edge e outputs on (u, e) the set of all labels ℓ for which the probability that, in $\mathcal{A}$, node u outputs ℓ on (u, e), conditioned on the random bits that e has collected, is at least K for at least one possible extension of (the topology and input labels of) the graph beyond the $(T - 1/2)$-hop view of e. Edge e computes the output label on half-edge (v, e) analogously. This concludes the description of $\mathcal{A}_{1/2}$; in the following we derive $\mathcal{A}'$ from $\mathcal{A}_{1/2}$ in a fashion dual to how we derived $\mathcal{A}_{1/2}$ from $\mathcal{A}$.

In $\mathcal{A}'$, each node u first collects all information contained in $B_G(u, T - 1)$. Then, for each incident edge e, node u determines the output label ℓ'' it outputs on half-edge (u, e) as follows, depending on some parameter $0 < L \leq 1$ that we will choose later. Label ℓ'' is simply the set of all labels ℓ' such that there exists an input forest $G'' \in \mathcal{F}$ and a node u'' in G'' such that $B_{G''}(u'', T - 1) \cong B_G(u, T - 1)$ and the probability that the edge e'' corresponding to e in the isomorphism outputs ℓ' on (u'', e'') according to $\mathcal{A}_{1/2}$ is at least L, conditioned on the assumption that the random bits in $B_{G''}(u'', T - 1)$ are the same as in $B_G(u, T - 1)$. In other words, in $\mathcal{A}'$, node u outputs on (u, e) the set of all labels ℓ' for which the probability that, in $\mathcal{A}_{1/2}$, edge e outputs ℓ' on (u, e), conditioned on the random bits that u has collected, is at least L for at least one possible extension of (the topology and input labels of) the graph beyond the $(T - 1)$-hop view of u. This concludes the description of $\mathcal{A}'$.

In the following, for all forests in $\mathcal{F}_N$, we bound the local failure probability of $\mathcal{A}_{1/2}$, depending on (the bound on) the local failure probability of $\mathcal{A}$. We start by proving two helper lemmas. For any edge e, resp. node u, let p^*_e, resp. p^*_u, denote the probability that $\mathcal{A}_{1/2}$ fails at edge e, resp. node u. Moreover, for graphs G, G', let f_{in}, resp. f'_{in}, be the functions that, for each half-edge in G, resp. G', return the input label of the half-edge. Finally, we will use $\mathcal{A}(h)$ and $\mathcal{A}_{1/2}(h)$ to denote the labels that $\mathcal{A}$ and $\mathcal{A}_{1/2}$, respectively, output on some half-edge h.

Lemma 7.13. *Let $e = \{u, v\}$ be an arbitrary edge in an arbitrary forest $G \in \mathcal{F}_N$. It holds that $p^*_e \leq ps/(K^2)$, where $s = (3|\Sigma^{\Pi}_{\text{in}}|)^{2\Delta^{T+1}}$.*

Proof. Consider an arbitrary assignment of random bits in $B_G(e, T - 1/2)$ for which $\mathcal{A}_{1/2}$ fails on e, i.e., for which

1. the cardinality-2 multiset of labels that $\mathcal{A}_{1/2}$ outputs on (u, e) and (v, e) is not contained in $\mathcal{E}_{\mathcal{R}(\Pi)}$
   , i.e., $\{\mathcal{A}_{1/2}((u, e)), \mathcal{A}_{1/2}((v, e))\} \notin \mathcal{E}_{\mathcal{R}(\Pi)}$, or
2. we have $\mathcal{A}_{1/2}((u, e)) \notin g_{\mathcal{R}(\Pi)}(f'_{\text{in}}((u, e)))$ or $\mathcal{A}_{1/2}((v, e)) \notin g_{\mathcal{R}(\Pi)}(f'_{\text{in}}((v, e)))$.

First, consider the case that Condition 1 is satisfied. Then, by the definition of $\mathcal{R}(\Pi)$, there are two labels $\ell_u \in \mathcal{A}_{1/2}((u, e))$, $\ell_v \in \mathcal{A}_{1/2}((v, e))$ such that $\{\ell_u, \ell_v\} \notin \mathcal{E}_{\Pi}$. Moreover, by the definition of $\mathcal{A}_{1/2}$, there is a forest $G' \in \mathcal{F}_N$ containing[6] $B_G(e, T - 1/2)$ such that $|V(G')| = |V(G)|$ and, conditioned on the already fixed random bits in $B_{G'}(e, T - 1/2) = B_G(e, T - 1/2)$, the probability that $\mathcal{A}$, when executed on G', returns ℓ_u on (u, e) and ℓ_v on (v, e) is at least K^2. Note that we use here that the input graph is a forest: in order to be able to simply multiply the two probabilities of ℓ_u being returned on (u, e) and ℓ_v being returned on (v, e) (which are both lower bounded by K), we require that those two probabilities are independent (which is guaranteed if the input graph is a forest, as then $B_{G'}(u, T) \setminus B_{G'}(e, T - 1/2)$ and $B_{G'}(v, T) \setminus B_{G'}(e, T - 1/2)$ are disjoint). Note further that we use that $G \in \mathcal{F}_N$ (which implies that the number of nodes in $B_G(e, T - 1/2)$ is sufficiently small compared to $|V(G)|$) to guarantee the property $|V(G')| = |V(G)|$, and that this property is needed because the lemma statement relating the probabilities p^*_e and p (which technically speaking are functions of the number n of nodes) is supposed to hold for any fixed n. Finally, note that this is a place where it is crucial that we consider forests, not trees, as on trees is might be the case that no graph G' as described exists: if the two extensions beyond $B_G(e, T - 1/2)$ that are responsible for the containment of ℓ_u in $\mathcal{A}_{1/2}((u, e))$ and of ℓ_v in $\mathcal{A}_{1/2}((v, e))$ both have no edges leaving the respective extension except those connecting them to $B_G(e, T - 1/2)$, and the total number of nodes in the union of $B_G(e, T - 1/2)$ and those two extensions does not happen to be precisely n, then those two extensions cannot appear simultaneously if we assume the input graph to be an n-node tree.

Now consider the case that Condition 2 is satisfied. Then, by the definition of $\mathcal{R}(\Pi)$,

[6]For better readability, we refrain from using the mathematically precise term "isomorphism" in the following, and instead identify isomorphic objects with each other, e.g., we consider $B_G(e, T - 1/2)$ to be a subgraph of G' if it is isomorphic to some subgraph of G'.

there is some label $\ell_u \in \mathcal{A}_{1/2}((u,e))$ satisfying $\ell_u \notin g_\Pi(f'_{\text{in}}((u,e)))$ or some label $\ell_v \in \mathcal{A}_{1/2}((v,e))$ satisfying $\ell_v \notin g_\Pi(f'_{\text{in}}((v,e)))$. With an analogous argumentation to the one used in the previous case, we obtain that there is a forest $G' \in \mathcal{F}_N$ containing $B_G(e, T-1/2)$ such that $|V(G')| = |V(G)|$ and the probability that $\mathcal{A}$, when executed on G', fails on e is at least K.

Since $0 < K \leq 1$, we can conclude that in either case the probability (conditioned on the already fixed random bits in $B_{G'}(e, T-1/2)$) that, in G', $\mathcal{A}$ fails on e is at least K^2. Observe that the output of $\mathcal{A}$ on the two half-edges belonging to e depends only on $B_{G'}(e, T+1/2) = B_{G'}(u,T) \cup B_{G'}(u,T)$. Given the fixed topology and input in $B_{G'}(e, T-1/2)$, there are at most $s := (3|\Sigma^\Pi_{\text{in}}|)^{2\Delta^{T+1}}$ different possibilities for the topology and input in $B_{G'}(e, T+1/2)$: there are at most $2\Delta^T$ nodes in $B_{G'}(e, T+1/2) \setminus B_{G'}(e, T-1/2)$, and for each of the Δ possible ports of such a node, there are at most 3 possibilities regarding topology (being connected to a node in $B_{G'}(e, T-1/2)$, being connected to a node outside of $B_{G'}(e, T+1/2)$, or non-existing due to the degree of the node being too small) and at most $|\Sigma^\Pi_{\text{in}}|$ possibilities regarding the input label on the half-edge corresponding to that port. Let $\mathcal{B}$ denote the set of different balls of the form $B_{G'}(e, T+1/2)$ (for some G') that contain $B_G(e, T-1/2)$. As shown above, $|\mathcal{B}| \leq s$.

Now, forget the fixing of the random bits in $B_G(e, T-1/2)$. By the above discussion, it follows that there is some ball $B \in \mathcal{B}$ (and therefore also some forest G') containing $B_G(e, T-1/2)$ such that the probability that $\mathcal{A}$, when executed on B (or G'), fails on e is at least $(p^*_e/s) \cdot K^2$. Since this probability is upper bounded by p, we obtain $p^*_e \leq ps/(K^2)$, as desired. □

Lemma 7.14. *Let u be an arbitrary node in an arbitrary forest $G \in \mathcal{F}_N$. It holds that $p^*_u \leq p + |\Sigma^\Pi_{\text{out}}|\Delta K + \frac{ps\Delta}{K}$, where $s = (3|\Sigma^\Pi_{\text{in}}|)^{2\Delta^{T+1}}$.*

Proof. Let $e_1, \dots, e_{\deg(u)}$ denote the edges incident to u. Moreover, let $p^{(1)}_u$ denote the probability that
$\{\mathcal{A}_{1/2}((u,e_i))\}_{1 \leq i \leq \deg(u)} \notin \mathcal{N}^{\deg(u)}_{\mathcal{R}(\Pi)}$ and $p^{(2)}_u$ the probability that there exists some $1 \leq i \leq \deg(u)$ satisfying $\mathcal{A}_{1/2}((u,e_i))\} \notin g_{\mathcal{R}(\Pi)}(f_{\text{in}}((u,e_i)))$. Since those two conditions together cover all cases in which $\mathcal{A}_{1/2}$ fails at u, we have $p^*_u \leq p^{(1)}_u + p^{(2)}_u$. We start by bounding $p^{(1)}_u$.

Observe that correctness at u, for both $\mathcal{A}$ and $\mathcal{A}_{1/2}$, depends only on $B_G(u,T)$. Given the topology and input in $B_G(u,T)$ (which is already fixed since we are considering some fixed graph G), we call, for each label $\ell \in \Sigma^\Pi_{\text{out}}$ and each $1 \leq i \leq \deg(u)$, an assigment of random bits in $B_G(u,T)$ *bad for the pair* (ℓ, i) if $\mathcal{A}((u,e_i)) = \ell$ and $\ell \notin \mathcal{A}_{1/2}((u,e_i))$ (under this assignment). Observe that the definition of $\mathcal{A}_{1/2}$ ensures, for each fixed assignment of random bits in $B_G(e_i, T-1/2)$, that if $\ell \notin \mathcal{A}_{1/2}((u,e_i))$ under this assignment[7], then the probability that $\mathcal{A}((u,e_i)) = \ell$ (conditioned on the fixed assignment in $B_G(e_i, T-1/2)$) is smaller than K. Hence, (prior to fixing any random

[7]Note that $\mathcal{A}_{1/2}((u,e_i))$ is uniquely determined after fixing the random bits in $B_G(e_i, T-1/2)$.

bits) for each pair $(\ell, i) \in \Sigma_{\text{out}}^{\Pi} \times \{1, \ldots, \deg(u)\}$, the probability that an assignment of random bits in $B_G(u, T)$ is bad for (ℓ, i) is smaller than K. It follows by a union bound that the probability that an assignment of random bits is bad for *some* pair $(\ell, i) \in \Sigma_{\text{out}}^{\Pi} \times \{1, \ldots, \deg(u)\}$ is upper bounded by $|\Sigma_{\text{out}}^{\Pi}| \cdot \Delta \cdot K$.

Now, consider an arbitrary assignment of random bits in $B_G(u, T)$ such that $\{\mathcal{A}_{1/2}((u, e_i))\}_{1 \leq i \leq \deg(u)} \notin \mathcal{N}_{\mathcal{R}(\Pi)}^{\deg(u)}$ under this assignment. We argue that then $\mathcal{A}$ fails at u or there is some pair $(\ell, i) \in \Sigma_{\text{out}}^{\Pi} \times \{1, \ldots, \deg(u)\}$ such that the assignment is bad for (ℓ, i): indeed, if there is no such pair and $\mathcal{A}$ does not fail at u, then, for each $1 \leq i \leq \deg(u)$, we have $\mathcal{A}((u, e_i)) \in \mathcal{A}_{1/2}((u, e_i))$, which (combined again with the correctness of $\mathcal{A}$ at u) would imply, by the definition of $\mathcal{N}_{\mathcal{R}(\Pi)}^{\deg(u)}$, that $\{\mathcal{A}_{1/2}((u, e_i))\}_{1 \leq i \leq \deg(u)} \in \mathcal{N}_{\mathcal{R}(\Pi)}^{\deg(u)}$, contradicting the stated property of the considered assignment. We conclude that $p_u^{(1)} \leq p + |\Sigma_{\text{out}}^{\Pi}| \cdot \Delta \cdot K$.

Next, we bound $p_u^{(2)}$. By a union bound, it follows from the definition of $p_u^{(2)}$ that there is some $1 \leq i \leq \deg(u)$ such that the probability that $\mathcal{A}_{1/2}((u, e_i)) \notin g_{\mathcal{R}(\Pi)}(f_{\text{in}}((u, e_i)))$ is at least $p_u^{(2)} / \deg(u) \geq p_u^{(2)} / \Delta$.

Consider an arbitrary assignment of random bits in $B_G(e_i, T - 1/2)$ such that $\mathcal{A}_{1/2}((u, e_i)) \notin g_{\mathcal{R}(\Pi)}(f_{\text{in}}((u, e_i)))$. Then, by the definition of $\mathcal{R}(\Pi)$, there is some label $\ell \in \mathcal{A}_{1/2}((u, e_i))$ satisfying $\ell \notin g_{\Pi}(f_{\text{in}}((u, e_i)))$. Moreover, by the definition of $\mathcal{A}_{1/2}$, there is some ball $B_{G'}(u, T)$ (in some forest $G' \in \mathcal{F}_N$ satisfying $|V(G')| = |V(G)|$) containing $B_G(e_i, T - 1/2)$ such that, conditioned on the already fixed random bits in $B_{G'}(e_i, T - 1/2) = B_G(e_i, T - 1/2)$, the probability that $\mathcal{A}$, when executed on $B_{G'}(u, T)$ (or G'), returns ℓ on (u, e_i) is at least K. Note that since $\ell \notin g_{\Pi}(f_{\text{in}}((u, e_i)))$, returning ℓ on (u, e_i) implies that $\mathcal{A}$ fails at u.

As already established in the proof of Lemma 7.13, there are at most $s := (3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}}$ different possibilities for the topology and input of a ball $B_{G'}(e_i, T + 1/2)$ containing $B_G(e_i, T - 1/2)$, which implies the same bound on the number of different balls $B_{G'}(u, T)$ containing $B_G(e_i, T - 1/2)$. Analogously to before (and forgetting the fixing of the random bits in $B_G(e_i, T - 1/2)$), we obtain that there is some ball $B_{G'}(u, T)$ (in some forest G' satisfying $|V(G')| = |V(G)|$) containing $B_G(e_i, T - 1/2)$ such that the probability that $\mathcal{A}$, when executed on $B_{G'}(u, T)$ (or G'), fails at u is at least $((p_u^{(2)} / \Delta) / s) \cdot K$. Since this probability is upper bounded by p, we obtain $p_u^{(2)} \leq ps\Delta / K$, which implies

$$p_u^* \leq p + |\Sigma_{\text{out}}^{\Pi}| \Delta K + \frac{ps\Delta}{K} \ , \tag{7.1}$$

as desired. □

By using Lemmas 7.13 and 7.14 and choosing K suitably, we now give an upper bound on the local failure probability of $\mathcal{A}_{1/2}$ on $\mathcal{F}_N$.

Lemma 7.15. *Algorithm $\mathcal{A}_{1/2}$ has local failure probability at most $2\Delta(s + |\Sigma_{\text{out}}^{\Pi}|) p^{1/3}$ on $\mathcal{F}_N$, where $s = (3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}}$.*

Proof. Choose the parameter in the definition of $\mathcal{A}_{1/2}$ as $K := p^{1/3}$. Then, by Lemma 7.14, for each node u in the input forest G, we obtain

$$p_u^* \le p + |\Sigma_{\text{out}}^{\Pi}|\Delta p^{1/3} + s\Delta p^{2/3} \le 2\Delta(s + |\Sigma_{\text{out}}^{\Pi}|)p^{1/3} \ ,$$

and, by Lemma 7.13, for each edge e in G, we obtain

$$p_e^* \le sp^{1/3} \le 2\Delta(s + |\Sigma_{\text{out}}^{\Pi}|)p^{1/3} \ .$$

The lemma statement follows by the definition of local failure probability. □

Next, we prove an analogous statement to Lemma 7.15 relating the local failure probabilities of $\mathcal{A}_{1/2}$ and $\mathcal{A}'$. For any edge e, resp. node u, let p'_e, resp. p'_u, denote the probability that $\mathcal{A}'$ fails at edge e, resp. node u.

Lemma 7.16. *If $\mathcal{A}_{1/2}$ has local failure probability at most $p^* \le 1$ on $\mathcal{F}_N$, then $\mathcal{A}'$ has local failure probability at most $3(s+|\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|)(p^*)^{1/(\Delta+1)}$ on $\mathcal{F}_N$, where $s = (3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}}$.*

Proof. We start by obtaining analogous statements to Lemmas 7.13 and 7.14. By simply exchanging the roles of nodes and edges[8] and reducing the radii of the considered balls by 1/2 (as well as using parameter L instead of K, and using $\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}$ instead of $\Sigma_{\text{out}}^{\Pi}$), we directly obtain analogous proofs resulting in the following two statements, (assuming the stated upper bound p^* on the local failure probability of $\mathcal{A}_{1/2}$):

1. For any node u in the input forest G, we have $p'_u \le p^* s/(L^{\Delta})$.
2. For any edge e in G, we have $p'_e \le p^* + |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}| \cdot 2L + p^* \cdot 2s/L$.

Note that, compared to Lemmas 7.13 and 7.14, each 2 has been replaced by Δ and vice versa, simply because the roles of edges (that end in 2 nodes) and nodes (that "end" in at most Δ edges) are reversed in the analogous proofs. Also observe that, technically, the expression for s that we obtain in the new proofs is $(3|\Sigma_{\text{in}}^{\Pi}|)^{\Delta\cdot\Delta^{(T-1)+1}}$; however, since this is upper bounded by the original expression for s, the obtained statements also hold for the original $s = (3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}}$ (which we will continue to use).

Now, analogously to the choice of $K = p^{1/(2+1)}$ in the proof of Lemma 7.15, set $L := (p^*)^{1/(\Delta+1)}$. We obtain, for each edge e in G,

$$\begin{aligned} p'_e &\le p^* + |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}| \cdot 2(p^*)^{1/(\Delta+1)} + 2s(p^*)^{\Delta/(\Delta+1)} \\ &\le 3(s + |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|)(p^*)^{1/(\Delta+1)} \ , \end{aligned}$$

and, for each node u in G,

$$p'_u \le s(p^*)^{1/(\Delta+1)} \le 3(s + |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|)(p^*)^{1/(\Delta+1)} \ .$$

Again, the lemma statement follows by the definition of local failure probability. □

[8] Note that, as usual for round elimination, all proofs also directly extend to hypergraphs. Hence, exchanging the roles of nodes and edges is very natural: a node v of degree $\deg(v)$ simply becomes a hyperedge containing $\deg(v)$ endpoints, while an edge becomes a node of degree 2.

Combining Lemmas 7.15 and 7.16, we are finally ready to prove Theorem 7.12.

Proof of Theorem 7.12. Let S be as specified in the theorem, and let $\mathcal{A}'$ be as derived before. As already argued during the definition of $\mathcal{A}'$, if the runtime $T = T(n)$ of $\mathcal{A}$ is 0, then $\mathcal{A}'$ satisfies the stated properties regarding runtime and local failure probability. Hence, assume in the following that $T \geq 1$.

From the definition of $\mathcal{A}'$, it follows directly that the runtime of $\mathcal{A}'$ is $T-1$. It remains to show that $\mathcal{A}'$ has local failure probability at most $Sp^{1/(3\Delta+3)}$ on $\mathcal{F}_N$. We start by showing that $\mathcal{A}'$ has local failure probability at most $S'p^{1/(3\Delta+3)}$ on $\mathcal{F}_N$, where

$$\begin{aligned} S' := 3 \cdot ((3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}} + |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|) \\ \cdot (2\Delta((3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}} + |\Sigma_{\text{out}}^{\Pi}|))^{1/(\Delta+1)} \ . \end{aligned}$$

Indeed, if p satisfies $2\Delta((3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}} + |\Sigma_{\text{out}}^{\Pi}|)p^{1/3} \leq 1$, then this is a direct consequence of Lemmas 7.15 and 7.16. If p does not satisfy the given condition, then the straightforward combination of Lemmas 7.15 and 7.16 does not work, as the condition $p^* \leq 1$ in Lemma 7.16 is not satisfied. However, in this case, i.e., if $2\Delta((3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}} + |\Sigma_{\text{out}}^{\Pi}|)p^{1/3} > 1$, we obtain $S'p^{1/(3\Delta+3)} > 1$, which trivially implies that $\mathcal{A}'$ has local failure probability at most $S'p^{1/(3\Delta+3)}$.

Now, the theorem statement follows from the fact that $S' \leq S$, which in turn follows from

$$\begin{aligned} S' &= 3 \cdot ((3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}} + |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|) \\ &\quad \cdot (2\Delta((3|\Sigma_{\text{in}}^{\Pi}|)^{2\Delta^{T+1}} + |\Sigma_{\text{out}}^{\Pi}|))^{1/(\Delta+1)} \\ &\leq (9(|\Sigma_{\text{in}}^{\Pi}| + |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|))^{2\Delta^{T+1}} \cdot (6\Delta(|\Sigma_{\text{in}}^{\Pi}| + |\Sigma_{\text{out}}^{\Pi}|))^{2\Delta^{T+1}} \\ &\leq (10\Delta(|\Sigma_{\text{in}}^{\Pi}| + \max\{|\Sigma_{\text{out}}^{\Pi}|, |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|\}))^{2\Delta^{T+1}} \\ &\quad \cdot (10\Delta(|\Sigma_{\text{in}}^{\Pi}| + \max\{|\Sigma_{\text{out}}^{\Pi}|, |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|\}))^{2\Delta^{T+1}} \\ &= (10\Delta(|\Sigma_{\text{in}}^{\Pi}| + \max\{|\Sigma_{\text{out}}^{\Pi}|, |\Sigma_{\text{out}}^{\mathcal{R}(\Pi)}|\}))^{4\Delta^{T+1}} \\ &= S \ . \end{aligned}$$

□

7.3.3 From Easier to Harder Problems

In this section, we prove the following lemma, which, in a sense, provides a counterpart to Theorem 7.12: it states that the time needed to solve some problem Π is not much larger than the time needed to solve $\overline{\mathcal{R}}(\mathcal{R}(\Pi))$. It is a simple extension of one direction of [74, Theorem 4.1] to the case of graphs with inputs. We note that the lemma also holds for randomized algorithms (with essentially the same proof), but we will only need the deterministic version.

Lemma 7.17. *Let* Π *be a node-edge-checkable LCL problem and let* $\mathcal{A}$ *be a deterministic algorithm solving* $\overline{\mathcal{R}}(\mathcal{R}(\Pi))$ *in* $T(n)$ *rounds, for some function* T *(on some arbitrary class of graphs). Then there exists a deterministic algorithm* $\mathcal{A}'$ *solving* Π *in* $T(n)+1$ *rounds.*

Proof. We will define $\mathcal{A}'$ as follows, depending on $\mathcal{A}$.[9]

For each half-edge h, let $\mathcal{A}(h)$ denote the output label that $\mathcal{A}$ outputs at h. In $\mathcal{A}'$, each node v starts by computing $\mathcal{A}(h)$ for each half-edge $h = (w, e)$ such that w is a neighbor of or identical to v and e is an edge incident to v. As, in $\mathcal{A}$, each neighbor of v computes its output in $T(n)$ rounds, v can compute all mentioned $\mathcal{A}(h)$ in $T(n) + 1$ rounds, by simulating $\mathcal{A}$. Node v will decide on its output solely based on the information collected so far, which implies that the runtime of $\mathcal{A}'$ is $T(n) + 1$.

For choosing its output, v proceeds in two steps (computed without gathering any further information). In the first step, for each incident edge $e = \{v, w\}$, node v chooses, in some deterministic fashion, a label $L_{(v,e)} \in \mathcal{A}((v, e))$ and a label $L_{(w,e)} \in \mathcal{A}((w, e))$ such that $\{L_{(v,e)}, L_{(w,e)}\} \in \mathcal{E}_{\mathcal{R}(\Pi)}$. Such a pair of labels exists, by the definition of $\mathcal{E}_{\overline{\mathcal{R}}(\mathcal{R}(\Pi))}$ (and the fact that $\mathcal{A}$ correctly solves $\overline{\mathcal{R}}(\mathcal{R}(\Pi))$). Moreover, the definition of $\mathcal{N}^{\deg(v)}_{\overline{\mathcal{R}}(\mathcal{R}(\Pi))}$ implies that the multiset $\{L_{(v,e')}\}_{e' \ni v}$ is contained in $\mathcal{N}^{\deg(v)}_{\mathcal{R}(\Pi)}$, and the definition of $g_{\overline{\mathcal{R}}(\mathcal{R}(\Pi))}$ implies that $L_{(v,e)} \in g_{\mathcal{R}(\Pi)}(i_{(v,e)})$ and $L_{(w,e)} \in g_{\mathcal{R}(\Pi)}(i_{(w,e)})$ where $i_{(v,e)}, i_{(w,e)} \in \Sigma^{\Pi}_{\text{in}}$ denote the input labels assigned to (v, e) and (w, e), respectively. Note also that since the labels $L_{(v,e)}$ and $L_{(w,e)}$ are chosen in a deterministic fashion, depending only on $\mathcal{A}((v, e))$ and $\mathcal{A}((w, e))$, node v and node w will compute the same label $L_{(v,e)}$ and the same label $L_{(w,e)}$. We conclude that labeling each half-edge h with label L_h yields a correct solution for $\mathcal{R}(\Pi)$.

In the second step, v computes the final output for each half-edge incident to v, in a fashion analogous to the first step. More precisely, for each incident edge e, node v chooses a final output label $\ell_{(v,e)} \in L_{(v,e)}$ such that the multiset $\{\ell_{(v,e')}\}_{e' \ni v}$ is contained in $\mathcal{N}^{\deg(v)}_{\Pi}$. Such labels $\ell_{(v,e)}$ exist, by the definition of $\mathcal{N}^{\deg(v)}_{\mathcal{R}(\Pi)}$ (and the fact that the labeling assigning L_h to each half-edge h correctly solves $\mathcal{R}(\Pi)$). Moreover, the definition of $\mathcal{E}_{\mathcal{R}(\Pi)}$ implies that, for any edge $e = \{v, w\}$, the multiset $\{\ell_{(v,e)}, \ell_{(w,e)}\}$ is contained in $\mathcal{E}_{\Pi}$, and the definition of $g_{\mathcal{R}(\Pi)}$ implies that, for any half-edge (v, e), we have $\ell_{(v,e)} \in g_{\Pi}(i_{(v,e)})$.

We conclude that labeling each half-edge h with label ℓ_h yields a correct solution for Π. It follows that $\mathcal{A}'$ solves Π in $T(n) + 1$ rounds, as desired. □

7.3.4 Proving the Gap

In this section, we will finally prove our main result that on forests or trees, any LCL problem with complexity $o(\log^* n)$ can be solved in constant time. We will first take care of the case of forests.

[9]Note that the $\mathcal{A}$ and $\mathcal{A}'$ considered here are not the same as the ones considered in Section 7.3.2.

Theorem 7.18. *Let* Π *be an arbitrary LCL problem that has (deterministic or randomized) complexity* $o(\log^* n)$ *on* $\mathcal{F}$*. Then* Π *can be solved in constant time on* $\mathcal{F}$ *(both deterministically and randomized).*

Proof. Observe that, by Lemma 7.7, it suffices to prove the theorem for node-edge-checkable LCL problems. Hence, assume in the following that Π is node-edge-checkable (and has deterministic or randomized complexity $o(\log^* n)$ on $\mathcal{F}$).

Observe further that if each node independently chooses an identifier (of length $O(\log n)$ bits) uniformly at random from a set of size n^3, then the probability that there are two nodes that choose the same identifier is at most $1/n$, by a union bound over all pairs of nodes. Hence, a deterministic algorithm solving Π in $o(\log^* n)$ rounds can be transformed into a randomized algorithm with the same runtime and a failure probability of at most $1/n$ by having each node create its own identifier from its random bits before executing the deterministic algorithm. Thus, we can assume that the $o(\log^* n)$-round algorithm for Π guaranteed in the theorem is randomized and fails with probability at most $1/n$. Let $\mathcal{A}$ denote this algorithm, and observe that the bound of $1/n$ on the failure probability of $\mathcal{A}$ implies that also the local failure probability of $\mathcal{A}$ is bounded by $1/n$.

Let $T(n)$ denote the runtime of $\mathcal{A}$ on the class of n-node forests from $\mathcal{F}$. Let n_0 be a sufficiently large positive integer; in particular, we require that

$$T(n_0) + 2 \leq \log_\Delta n_0, \tag{7.2}$$

$$2T(n_0) + 5 \leq \log^* n_0, \tag{7.3}$$

and

$$\left((S^*)^2 \cdot (\log n_0)^{2\Delta}\right)^{(3\Delta+3)^{T(n_0)}} < n_0 \tag{7.4}$$

where $S^* := (10\Delta(|\Sigma_{\text{in}}^{\Pi}| + \log n_0))^{4\Delta^{T(n_0)+1}}$. Since $T(n) \in o(\log^* n)$ (and Δ and Σ_{in}^{Π} are fixed constants), such an n_0 exists. Moreover, for simplicity, define $f(\cdot) := \overline{\mathcal{R}}(\mathcal{R}(\cdot))$, and recall that $\mathcal{F}_{n_0}$ denotes the class of all forests from $\mathcal{F}$ with n_0 nodes. Repeatedly applying $f(\cdot)$, starting with Π, yields a sequence of problems $\Pi, f(\Pi), f^2(\Pi), \dots$. Our first goal is to show that there is a 0-round algorithm solving $f^{T(n_0)}(\Pi)$ on $\mathcal{F}_{n_0}$ with small local failure probability, by applying Theorem 7.12 repeatedly.

Recall that for any integer $i \geq 0$, we have $\Sigma_{\text{out}}^{\mathcal{R}(f^i(\Pi))} = 2^{\Sigma_{\text{out}}^{f^i(\Pi)}}$ and $\Sigma_{\text{out}}^{\overline{\mathcal{R}}(\mathcal{R}(f^i(\Pi)))} = 2^{\Sigma_{\text{out}}^{\mathcal{R}(f^i(\Pi))}}$, by definition. This implies that for any $0 \leq i \leq T(n_0)$, we have

$$\max\{|\Sigma_{\text{out}}^{f^i(\Pi)}|, |\Sigma_{\text{out}}^{\mathcal{R}(f^i(\Pi))}|\} \leq 2^{2^{\cdot^{\cdot^{\cdot^{2^{\Sigma_{\text{out}}^{\Pi}}}}}}}$$

where the power tower is of height $2T(n_0) + 3$. By (7.3), we obtain that

$$\max\{|\Sigma_{\text{out}}^{f^i(\Pi)}|, |\Sigma_{\text{out}}^{\mathcal{R}(f^i(\Pi))}|\} \leq \log n_0 \text{ for all } 0 \leq i \leq T(n_0). \tag{7.5}$$

Recall that $S^* = (10\Delta(|\Sigma_{\text{in}}^{\Pi}|+\log n_0))^{4\Delta^{T(n_0)+1}}$ and note that $\Sigma_{\text{in}}^{f^{T(n_0)}(\Pi)} = \Sigma_{\text{in}}^{f^{T(n_0)-1}(\Pi)} = \ldots = \Sigma_{\text{in}}^{\Pi}$. By the above discussion, we conclude that when applying Theorem 7.12 to problem $f^i(\Pi)$ for some $0 \le i \le T(n_0)$, then the parameter S in the obtained upper bound $Sp^{1/(3\Delta+3)}$ (in Theorem 7.12) is upper bounded by S^*. Hence, by applying Theorem 7.12 $T(n_0)$ times, each time using the upper bound $S^* p^{1/(3\Delta+3)}$ (instead of $Sp^{1/(3\Delta+3)}$ with the respective S from Theorem 7.12), we obtain that there exists a randomized algorithm $\mathcal{A}^*$ solving $f^{T(n_0)}(\Pi)$ on $\mathcal{F}_{n_0}$ in 0 rounds with local failure probability p^* at most

$$\begin{aligned}
&S^* \cdot (S^*)^{1/(3\Delta+3)} \cdot (S^*)^{1/(3\Delta+3)^2} \cdot \ldots \cdot (S^*)^{1/(3\Delta+3)^{T(n_0)-1}} \\
&\cdot 1/(n_0)^{1/(3\Delta+3)^{T(n_0)}} \\
&= (S^*)^{\sum_{i=0}^{T(n_0)-1}(1/(3\Delta+3)^i)} \cdot 1/(n_0)^{1/(3\Delta+3)^{T(n_0)}} \\
&\le (S^*)^2 \cdot 1/(n_0)^{1/(3\Delta+3)^{T(n_0)}} \\
&< 1/(\log n_0)^{2\Delta} \\
&\le 1/\left(\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}\right)^{2\Delta},
\end{aligned}$$

where the second-to-last inequality follows from (7.4), and the last inequality from (7.5). Note that (7.2) guarantees the applicability of Theorem 7.12 to the graph class $\mathcal{F}_{n_0}$ for our purposes. Moreover, we can assume that $\mathcal{A}^*$ outputs only labels from $\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}$ (even if it fails), as otherwise we can turn $\mathcal{A}^*$ into such an algorithm (with the same runtime and a smaller or equally small local failure probability) by simply replacing each label that $\mathcal{A}^*$ outputs at some node and is not contained in $\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}$ by some arbitrary label from $\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}$.

Our next step is to show that the obtained bound on the local failure probability p^* of $\mathcal{A}^*$ implies that there exists a *deterministic* algorithm solving $f^{T(n_0)}(\Pi)$ on $\mathcal{F}_{n_0}$ in 0 rounds. To this end, let $\mathcal{I}$ be the set of all tuples $I = (i_1, \ldots, i_k)$ consisting of $k \in \{1, \ldots, \Delta\}$ labels from Σ_{in}^{Π}, and $\mathcal{O}$ the set of all tuples $O = (o_1, \ldots, o_k)$ consisting of $k \in \{1, \ldots, \Delta\}$ labels from $\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}$. We say that $I = (i_1, \ldots, i_k)$, resp. $O = (o_1, \ldots, o_k)$, is the *input tuple*, resp. *output tuple*, of some node v if $\deg(v) = k$ and, for each $1 \le j \le k$, we have that i_j is the input label assigned to, resp. the output label returned by $\mathcal{A}^*$ at, the half-edge incident to v corresponding to port j at v.

Since the runtime of $\mathcal{A}^*$ is 0 rounds, any node v chooses its output tuple solely based on its random bit string and its input tuple. We now define a function $\mathcal{A}_{\text{det}} \colon \mathcal{I} \to \mathcal{O}$ (that will constitute the desired deterministic algorithm) as follows. Consider an arbitrary tuple $I = (i_1, \ldots, i_k) \in \mathcal{I}$ and let v be a node with input label I. Since $\mathcal{A}^*$ outputs only labels from $\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}$ and $k \le \Delta$, there are at most $\left(\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}\right)^{\Delta}$ different possibilities for the output tuple of v. Hence, there exists a tuple $O \in \mathcal{O}$ that v outputs with probability at least $1/\left(\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}\right)^{\Delta}$ (when executing $\mathcal{A}^*$). Fix such an O arbitrarily, and set $\mathcal{A}_{\text{det}}(I) = O$. This concludes the description of $\mathcal{A}_{\text{det}}$. Note that the definition of

$\mathcal{A}_{\text{det}}$ is independent of the choice of v (under the given restrictions) as the only relevant parameters are the input tuple at v (which is fixed) and the random bit string at v (which comes from the same distribution for each node).

We claim that for any two (not necessarily distinct) configurations $I, I' \in \mathcal{I}$, and any two (not necessarily distinct) labels $o \in \mathcal{A}_{\text{det}}(I)$ and $o' \in \mathcal{A}_{\text{det}}(I')$, we have $\{o, o'\} \in \mathcal{E}_{f^{T(n_0)}(\Pi)}$. For a contradiction, assume that there are two configurations $I = (i_1, \dots, i_k), I' = (i'_1, \dots, i'_{k'})$ from $\mathcal{I}$ and two labels $o \in \mathcal{A}_{\text{det}}(I)$ and $o' \in \mathcal{A}_{\text{det}}(I')$ satisfying $\{o, o'\} \notin \mathcal{E}_{f^{T(n_0)}(\Pi)}$. Let $O = (o_1, \dots, o_k)$ and $O' = (o'_1, \dots, o'_{k'})$ be the tuples that I and I', respectively, are mapped to by $\mathcal{A}_{\text{det}}$, i.e., $\mathcal{A}_{det}(I) = O$ and $\mathcal{A}_{\text{det}}(I') = O'$. Let $j \in \{1, \dots, k\}$ and $j' \in \{1, \dots, k'\}$ be two ports/indices such that $o_j = o$ and $o'_{j'} = o'$. Consider a forest (with n_0 nodes) containing two adjacent nodes v, v' such that I is the input tuple of v, I' is the input tuple of v', and the edge $e := \{v, v'\}$ corresponds to port j at v and to port j' at v'. By the definition of $\mathcal{A}_{\text{det}}$, we know that, when executing $\mathcal{A}^*$, the probability that v outputs o on half-edge (v, e) and the probability that v' outputs o' on half-edge (v', e) are each at least $1/\left(\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}\right)^{\Delta}$. Since these two events are independent (as one depends on the random bit string of v and the other on the random bit string of v'), it follows that the probability that the output on edge e is $\{o, o'\}$ is at least $1/\left(\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}\right)^{2\Delta}$. Now, $\{o, o'\} \notin \mathcal{E}_{f^{T(n_0)}(\Pi)}$ yields a contradiction to the fact the local failure probability of $\mathcal{A}^*$ is strictly smaller than $1/\left(\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}\right)^{2\Delta}$, proving the claim.

Observe that for any configuration $I = (i_1, \dots, i_k)$ from $\mathcal{I}$, we also have that

1. $\mathcal{A}_{\text{det}}(I)$ (or, more precisely, the unordered underlying multiset) is contained in $\mathcal{N}^k_{f^{T(n_0)}(\Pi)}$, and
2. for each port $1 \le j \le k$, the entry from $\mathcal{A}_{\text{det}}(I)$ corresponding to port j is contained in $g_{f^{T(n_0)}(\Pi)}(i_j)$,

as otherwise for each node v with input tuple I, algorithm $\mathcal{A}^*$ would fail at v with probability at least $1/\left(\Sigma_{\text{out}}^{f^{T(n_0)}(\Pi)}\right)^{\Delta}$, yielding again a contradiction to the aforementioned upper bound on the local failure probability of $\mathcal{A}^*$. By the above discussion, we conclude that the output returned by $\mathcal{A}_{\text{det}}$ is correct at each node and each edge, and therefore $\mathcal{A}_{\text{det}}$ constitutes a deterministic 0-round algorithm solving $f^{T(n_0)}(\Pi)$ on $\mathcal{F}_{n_0}$.

By definition, algorithm $\mathcal{A}_{\text{det}}$ is simply a function from $\mathcal{I}$ to $\mathcal{O}$. Hence, while technically $\mathcal{A}_{\text{det}}$ has been defined only for forests from $\mathcal{F}_{n_0}$, we can execute $\mathcal{A}_{\text{det}}$ also on forests with an arbitrary number of nodes and it will still yield a correct output. We conclude that $\mathcal{A}_{\text{det}}$ constitutes a deterministic 0-round algorithm solving $f^{T(n_0)}(\Pi)$ on $\mathcal{F}$.

Now, we can simply apply Lemma 7.17 $T(n_0)$ times, starting with $f^{T(n_0)}(\Pi)$, and obtain that there exists a deterministic algorithm solving Π on $\mathcal{F}$ in $T(n_0)$ rounds (which implies the existence of a randomized algorithm solving Π on $\mathcal{F}$ in $T(n_0)$ rounds, using the same argument as in the beginning of the proof). As n_0 is a fixed positive integer (depending only on Δ and Π), it follows that Π can be solved in constant time on $\mathcal{F}$ (both deterministically and randomized). $\square$

Now, by combining Theorem 7.18 with Lemma 7.11 and Lemma 7.7 (which guarantees that Lemma 7.11, which is stated for node-edge-checkable LCL problems, can be applied), we obtain our main result as a corollary.

Theorem 7.19 (Formal version of Theorem 7.1)**.** *Let Π be an arbitrary LCL problem that has (deterministic or randomized) complexity $o(\log^* n)$ on $\mathcal{T}$. Then Π can be solved in constant time on $\mathcal{T}$ (both deterministically and randomized).*

7.4 Speedup in Grids

In this section we prove that on d-dimensional oriented grids, any $o(\log^* n)$-round LOCAL algorithm for an LCL problem Π (with input labels) can be turned into a LOCAL algorithm for Π with a round complexity of $O(1)$. An oriented d-dimensional grid is a grid where each edge is labeled with a label from $[d]$ that says which dimension this edge corresponds to. Moreover, all edges corresponding to the same dimension are consistently oriented. The grid is assumed to be toroidal, that is, without boundaries. We believe, but do not prove, that the same result can be shown also for non-toroidal grids, that is, grids with boundaries. From now on fix a dimension d.

Theorem 7.20. *There is no LCL problem Π with a randomized/deterministic LOCAL complexity between $\omega(1)$ and $o(\log^* n)$ in d-dimensional oriented grids for $d = O(1)$.*

The proof of the speedup is a relatively simple extension of an argument of Suomela in the paper of Chang and Pettie [94]. There, the argument is proven for LCLs without inputs on unoriented grids. It shows that every $o(\log^* n)$-round algorithm can be even sped up to a 0-round algorithm. This is not the case anymore with inputs, which makes the proof slightly more involved.

The high-level idea of the proof is the following. First, we observe that a LOCAL algorithm on oriented grids implies a $\mathsf{PROD\text{-}LOCAL}$ algorithm with the same asymptotic complexity. A $\mathsf{PROD\text{-}LOCAL}$ algorithm is defined as follows.

Definition 7.21 ($\mathsf{PROD\text{-}LOCAL}$ model)**.** *An algorithm in the $\mathsf{PROD\text{-}LOCAL}$ model is defined as an algorithm in the LOCAL model, but it expects that each node $u \in V(G)$ for G an oriented grid gets an ordered sequence of d identifiers $id_1(u), \ldots, id_d(u) \in [n^{O(1)}]$, one for each dimension of the input grid G. We require that the i-th identifier $id_i(u), id_i(v)$ of two nodes u, v is the same if and only if the two nodes have the same i-th coordinate in G.*

An order-invariant $\mathsf{PROD\text{-}LOCAL}$ model is defined as follows. We say that two identifier assignments are order-indistinguishable in some neighborhood around a node u if for any *two nodes v, w in that neighborhood of u and any $i, j \in [d]$, the i-th identifier of one node is either smaller than the j-th identifier of the other node in both ID assignments or larger in both ID assignments. Then, an order-invariant $t(n)$-round* PROD-LOCAL *algorithm outputs the same value at the half edges of a node in case two ID assignments are order-indistinguishable in its $t(n)$-hop neighborhood.*

We have the following fact:

Proposition 7.22. *If an LCL Π allows has local deterministic/randomized complexity $t(n)$ in the* LOCAL *model, it has complexity $t(n)$ in the* PROD-LOCAL *model.*

Proof. By result of [94], a $t(n)$ round randomized algorithm in the local model implies a $t(2^{O(n^2)})$ round deterministic algorithm. This implies that we can turn an $O(\log^* n)$-rounds randomized algorithm into $O(\log^* n)$-rounds deterministic one.

Each of the d identifiers of a node are bounded by n^c for some fixed constant c in the PROD-LOCAL model. Assigning each node the identifier $I = \sum_{i=1}^{d} I_i \cdot n^{c(i-1)}$, where (I_i) is its i-th identifier results in globally unique identifiers from a polynomial range. This allows simulation of the deterministic LOCAL algorithm in the PROD-LOCAL model as needed. □

Next, we argue that an $o(\log^* n)$-round PROD-LOCAL algorithm can be turned into an order-invariant PROD-LOCAL algorithm. To show the existence of such an order-invariant algorithm, we use a Ramsey theory based argument very similar to the one in Section 8.6 and [94].

Proposition 7.23. *If there is a $t(n) = o(\log^* n)$ round* PROD-LOCAL *algorithm for an LCL Π, then there is also an order-invariant* PROD-LOCAL *algorithm with the same round complexity.*

Proof. The algorithm $\mathcal{A}$ of local complexity $t(n)$ can be seen as a function that maps $(2t(n)+1) \times (2t(n)+1)$ sized grids with edges consistently oriented and labeled with numbers from $[d]$. Moreover, each vertex has a d-tuple of its identifiers and each half-edge a label from Σ_{in}. The function $\mathcal{A}$ maps this input to an output label for each half-edge incident to a given node. Let $R(p, m, c)$ be defined as in the previous section and let H be a hypergraph on $n^{O(1)}$ nodes, each representing an identifier.

1. Assume we have a $t(n)$-hop neighborhood already labeled without assigned identifiers and we want to assign the identifiers. There are at most $p = d \cdot (2t(n)+1)$ numbers to assign.

2. Assume we have a $t(n) + r$-hop neighborhood already labeled without assigned identifiers and we want to assign the identifiers. There are at most $m = d \cdot (2(t(n) + r) + 1)$ numbers to assign. Here, r is the local checkability radius of the problem.

3. The number z is defined to be the number of possible input neighborhoods not

labeled with identifiers. We have $z \leq |\Sigma_{in}|^{2d(2t(n)+1)^d}$. Note that the extra $2d$-factor in the exponent comes from the fact that we label half-edges.

4. Finally, c is the number of colors such that each color encodes a distinct function which takes as input one of the z possible neighborhoods labeled with Σ_{in} inputs. Moreover, for each such neighborhood we fix a given permutation on p elements π. The function outputs a tuple of d output labels from Σ_{out}, one entry corresponding to each half-edge incident to a vertex. Hence, for the number of such colors c we have $c = |\Sigma_{out}|^{dp!z}$.

We color each p-edge T of H by the following color out of c colors: we consider all of the z possible ways of labeling the $t(n)$-hop neighborhood with labels from Σ_{in} and all $p!$ ways of how one can assign a set of p different identifiers of T to that neighborhood (we think of T as a sorted list on which we apply π and then in this order we fill in identifiers to the $t(n)$-hop neighborhood). For each such input (out of $p! \cdot z$ possible) we now apply $\mathcal{A}$ to that neighborhood and record the output from Σ_{out}^d. This gives one of c possible colors. We now use the bound $\log^* R(p, m, c) = p + \log^* m + \log^* c + O(1)$ as in the previous section. Hence, as we assume $t(n) = o(\log^* n)$, this implies $|V(H)| \geq R(p, m, c)$, and therefore there exists a set $S \subseteq V(H)$ of m distinct IDs such that all p-sized subsets $T \subseteq S$ have the same color.

We now define the new algorithm $\mathcal{A}'$ as follows. Let N be a neighborhood from the order-invariant PROD-LOCAL model; take an arbitrary p-sized subset $T \subseteq S$ and label N with identifiers from T by the permutation π given by the order on N that we got. Then, apply $\mathcal{A}$ on this input.

First, this algorithm is well defined as it does not matter which T we pick, the algorithm always answers the same because every $T \subseteq S$ is colored by the same color. Second, the algorithm is correct, since for any $(t(n)+r)$-hop neighborhood that has at most m vertices we can choose some way of assigning identifiers from S ($S = m$) to that neighbourhood and $\mathcal{A}'$ answers the same as what $\mathcal{A}$ would answer with this assignment of identifiers for all nodes within the local checkability radius. □

Once we have the order-invariant algorithm with a round complexity of $o(\log^* n)$, an easy adaptation of the proof for Theorem 8.27 implies that we can turn it into an order-invariant PROD-LOCAL algorithm with a round complexity of $O(1)$. However, because of the oriented-grid, we get a local order on the vertices for free, so we can turn the order-invariant PROD-LOCAL algorithm into an (order-invariant) LOCAL algorithm, thus finishing the proof.

Proposition 7.24. *If an LCL Π has complexity $o(n^{1/d})$ in the order-invariant* PROD-LOCAL *model, it has deterministic/randomized/deterministic order-invariant local complexity $O(1)$ in the* LOCAL *model.*

Proof. This is an argument very similar to [88]. Let $\mathcal{A}$ be the algorithm in the PROD-LOCAL model. We choose n_0 large enough and run $\mathcal{A}$ "fooled" into thinking that the size of the grid is n_0. As the order on the identifiers, choose the order where two identifiers $id_i(u), id_j(v)$ have $id_i(u) < id_j(v)$ if and only if either $i < j$ or $i = j$ and v is further than u in the i-th coordinate (the notion of "further than" is given by the orientation of the grid). A standard argument as in [88] shows that if n_0 is chosen large enough, the "fooled" algorithm needs to be correct, as otherwise $\mathcal{A}$ would be incorrect when being run on grids of size n_0. □

Theorem 7.20 follows by application of Theorems 7.22 to 7.24.

CHAPTER 8

The Randomized Local Computation Complexity of the Lovász Local Lemma

8.1 Introduction

As our main contribution, we show in this chapter that the randomized LCA complexity of the distributed Lovász local lemma (LLL) is $\Theta(\log n)$. See Section 8.6 for the informal discussions of the volume complexity (VOLUME) and the local computation algorithms (LCA). Formal definitions are given later in Section 8.2.

Theorem 8.1. *The randomized* LCA *complexity of the distributed LLL on constant degree graphs is $\Theta(\log n)$. The upper bound holds for the polynomial criterion $p \leq (e\Delta)^{-c}$ for some $c = O(1)$, while the lower bound holds even for the exponential criterion $p \leq 2^{-\Delta}$.*

In fact, for the minimally more restrictive criterion $p < 2^{-\Delta}$, the distributed LLL can already be solved in $O(\log^* n)$ rounds in the LOCAL model [78, 79], which implies also a probe complexity of $O(\log^* n)$ in the LCA model [127]. Second, we show the following general speedup theorem.

Theorem 8.2. *For any LCL Π, if there is a randomized* LCA *algorithm that solves Π and has a probe complexity of $o(\sqrt{\log n})$, then there is also a deterministic* LCA *algorithm for Π with a probe complexity of $O(\log^* n)$.*

We also show the following theorem in Section 8.6.

Theorem 8.3. *There does not exist an LCL with a deterministic* VOLUME *complexity between $\omega(1)$ and $o(\log^* n)$.*

Our Method in a Nutshell

Let us first informally present the high-level ideas of our proofs.

The Gap Result of Theorem 8.2:

We start to describe the high-level idea of the proof that there is no LCL with a randomized/deterministic LCA /VOLUME complexity between $\omega(\log^* n)$ and $o(\sqrt{\log n})$. This is the conceptually simplest result and it works by directly adapting ideas known from the LOCAL model. The main trick is to consider the deterministic VOLUME model where the identifiers can come from an exponential instead of a polynomial range (cf. Section 7.1 in [270]). The result then follows from two observations. First, a variant of the Chang-Pettie speedup [88] shows that any VOLUME algorithm with a probe complexity of $o(n)$ that works with exponential IDs can be sped up to have a probe complexity of $\Theta(\log^* n)$. Second, a variant of the Chang-Kopelowitz-Pettie derandomization [94] shows that any randomized algorithm with probe complexity $o(\sqrt{\log n})$ can be derandomized to give a deterministic $o(n)$-probe algorithm that works with exponential IDs. The root comes from the exponential IDs: during the argument we apply a union bound over all bounded degree n-node graphs equipped with unique IDs from an exponential range.

The LLL Complexity of Theorem 8.1:

The upper bound can directly be proven by adapting the LLL algorithm of [?] for the LOCAL model to the VOLUME model. To prove an $\Omega(\log n)$ lower bound, we prove a corresponding lower bound of $\Omega(\log n)$ for the Sinkless Orientation Problem, which can be seen as an instance of the distributed LLL. To prove this lower bound, we use the same high-level proof idea as described in the previous paragraph. However, to get a tight $\Omega(\log n)$ lower bound we need additional ideas. More concretely, the number $\sqrt{\log n}$ is an artifact of a union bound over $2^{O(n^2)}$ many non-isomorphic ID-labeled graphs. As we show the Sinkless Orientation lower bound on bounded-degree trees, it is actually sufficient to union bound over all non-isomorphic ID-labeled bounded-degree trees. As the number of non-isomorphic unlabeled trees is $2^{O(n)}$, we already made progress. However, this itself is still not enough as we need identifiers from an exponential range to speed the algorithm up to $\Theta(\log^* n)$ and there are $2^{O(n^2)}$ many ways to assign unique exponential IDs. Note that even if we would assign IDs from a polynomial range, there still would be $2^{O(n \log n)}$ ways to do it, which would only allow to hope for an $\Omega(\log n/ \log\log n)$ bound.

To circumvent this issue we borrow an idea from a parallel paper [83] where the technique of ID graphs is developed to overcome a different issue. The idea is as follows: we will work in the deterministic model with exponential IDs. However, we promise that the ID assignment satisfies certain additional properties. More concretely, we construct a so-called ID graph (which is not to be confused with the actual input graph). Each node in the ID graph corresponds to one of the exponentially many IDs. Moreover, the maximum degree of the ID graph is constant and two nodes in the input graph that are neighbors can only be assigned IDs that are neighbors in the ID graph. Having restricted the ID assignment in this way, it turns out that we only need to union bound over $2^{O(n)}$ different ID-labeled trees. Hence, a $o(\log n)$ randomized VOLUME algorithm would imply a $o(n)$ deterministic VOLUME algorithm that works on graphs with exponential IDs that satisfy the constraints imposed by the ID graph. Unfortunately, due to this additional restriction, we cannot simply speed-up the algorithm to the $\Theta(\log^* n)$ complexity. However, it turns out that the famous round elimination lower bound for Sinkless Orientation from [73] works even relative to an ID graph (the formal proof is in fact simpler as one does not pass through a randomized model). This finishes the lower bound proof for the VOLUME model. This directly leads to the same lower bound for the LCA model by a result of [179].

8.2 Preliminaries

We need to state the following formal definitions.

Definition 8.4 (LCA model [277] [8]). *In the* LCA *model, each node is assigned a unique ID from the set* $[n]$. *Moreover, each node is equipped with a port numbering of its edges and each node might have an additional input labeling. The* LCA *algorithm needs to answer queries. That is, given a vertex/edge, it needs to output the local solution of the vertex/edge in such a way that combining the answers of all vertices/edges constitutes a valid solution. To answer a query, the algorithm can probe the input graph. A probe consists of an integer* $i \in [n]$ *and a port number and the answer to the probe is the local information associated with the other endpoint of the edge corresponding to the specific port number of the vertex with ID* i. *The answer to a query is only allowed to depend on the input graph itself and possibly a shared random bit string in case of randomized* LCA *algorithms. The complexity of an* LCA *algorithm is defined as the maximum number of probes the algorithm needs to perform to answer a given query, where the maximum is taken over all input graphs and all query vertices/edges. A randomized* LCA *algorithm needs to produce a valid complete output (obtained by answering the query for each vertex) with probability* $1 - 1/n^c$ *for any desirably large constant* c.

Definition 8.5 (VOLUME model [270]). *The* VOLUME *model is very similar to the* LCA *model and hence we only discuss the differences. The IDs in the* VOLUME *model are from the set* $\{1, 2, \ldots, \mathrm{poly}(n)\}$, *as in the* LOCAL *model, instead of* $[n]$. *Moreover, a* VOLUME *model algorithm is confined to probe a connected region. In the case of randomized* VOLUME *algorithms, each node has a private source of random bits which is considered as part of the local information and is therefore returned together with the ID of a given vertex. We note that the shared randomness of the* LCA *model is strictly stronger than the private randomness of the* VOLUME *model and therefore the* LCA *model is strictly more powerful than the* VOLUME *model.*

The following is a well-known fact that follows from a straightforward simulation idea.

Lemma 8.6 (Parnas-Ron reduction, [263]). *Any* LOCAL *algorithm with a round complexity of* $t(n)$ *can be converted into an* LCA /VOLUME *algorithm with a probe complexity of* $\Delta^{O(t(n))}$, *where* Δ *denotes the maximum degree of the input graph.*

The following lemma is a restatement of Theorem 3 in [179].

Lemma 8.7. *Suppose that there is a randomized* LCA *algorithm with probe complexity* $t(n)$ *that solves an LCL* Π*. Then there is a randomized* LCA *algorithm with probe complexity* $t'(n) = t(\text{poly}(n))$ *that solves* Π*, does not perform any far probes, and works even if the unique identifiers come from a polynomial range (instead of from* $[n]$*). This also holds if we restrict ourselves to trees.*

We use this lemma to extend a lower bound that we prove for the VOLUME model to LCA algorithms (that are allowed to perform far probes).

We also need to use the following lemma which informally states that far probes are of no use for deterministic LCA algorithms with a small probe complexity.

Theorem 8.8 (Theorem 1 in [179])**.** *Any LCL problem that can be solved in the* LCA *model deterministically with probe complexity* $t(n)$ *can be solved with a deterministic round complexity of* $t'(n) = t(n^{\log n})$ *in the* LOCAL *model provided* $t(n) = o(\sqrt{\log n})$*.*

For our purposes, we actually need a slightly stronger version of Theorem 8.8 which is, in fact, what the proof of Theorem 8.8 in [179] gives.

Theorem 8.9. *Any LCL problem that can be solved in the* LCA *model deterministically with probe complexity* $t(n)$ *can be solved deterministically with a probe complexity of* $t'(n) = t(n^{\log n})$ *in the* VOLUME *model provided* $t(n) = o(\sqrt{\log n})$*.*

8.3 Warm-up: Speedup of Randomized Algorithms in LCA

In this section we prove Theorem 8.2 that we restate here for convenience.

Theorem 8.2. *For any LCL* Π*, if there is a randomized* LCA *algorithm that solves* Π *and has a probe complexity of* $o(\sqrt{\log n})$*, then there is also a deterministic* LCA *algorithm for* Π *with a probe complexity of* $O(\log^* n)$*.*

In fact, we show that the resulting deterministic LCA algorithm with a probe complexity of $O(\log^* n)$ can also be turned into a VOLUME algorithm with the same probe complexity. The following lemma is proven with a similar argument as Theorem 3 in [94].

Lemma 8.10 (Derandomization in LCA)**.** *If there exists a randomized* LCA *algorithm* $\mathcal{A}$ *with a probe complexity of* $t(n) = o(\sqrt{\log n})$ *for a given LCL* Π*, then there also exists a deterministic* VOLUME *algorithm* $\mathcal{A}'$ *for* Π *with a probe complexity of* $o(n)$ *and where the identifiers are from* $[2^{O(n)}]$*.*

Proof. Let r be the local checkability radius of Π. First, we apply Theorem 8.7 to convert the algorithm $\mathcal{A}$ into one having the same asymptotic complexity but which does not use any far probes and assumes only identifiers from a polynomial range. Without loss of generality, we can assume that $\mathcal{A}$ works in a setting where, instead of being assigned an identifier, each node has access to a private random bit string (in other words, $\mathcal{A}$

works in the VOLUME model with the addition of shared randomness). This is justified as, with access to private randomness in each node, the algorithm can generate unique identifiers with probability $1 - 1/\operatorname{poly}(n)$, as the first $O(\log n)$ random bits of each node are unique with probability $1 - 1/\operatorname{poly}(n)$.

Next, let $\mathcal{G}_n$ denote the set of all n-node graphs (up to isomorphism) of maximum degree Δ with each vertex labeled with a unique identifier from $[2^{O(n)}]$ and where each vertex has an input label from the finite set of input labels from the given LCL. The number of unlabeled graphs of maximum degree at most Δ is $2^{O(n \log n)}$ as it can be described by $O(n \cdot \Delta \cdot \log n)$ bits of information. Next, for a fixed n-node graph, the number of its labelings with identifiers from $[2^{O(n)}]$ is upper bounded by $(2^{O(n)})^n = 2^{O(n^2)}$ and the number of distinct input label assignments is upper bounded by $2^{O(n)}$. Hence, the number of (labeled) graphs in $\mathcal{G}_n$ is strictly smaller than some suitably chosen $N = 2^{O(n \log n)} \cdot 2^{O(n^2)} \cdot 2^{O(n)} = 2^{O(n^2)}$.

Now, let $\rho : [2^{O(n)}] \to \{0,1\}^{\mathbb{N}}$ be a function that maps each ID to a stream of bits, chosen uniformly at random from the space of all such functions. Similarly, let ρ^* be a bit string chosen uniformly at random from $\{0,1\}^{\mathbb{N}}$. Consider the algorithm $\mathcal{A}_{\rho,\rho^*}$ solving Π on all graphs $G \in \mathcal{G}_n$ that is defined as follows. First, $\mathcal{A}_{\rho,\rho^*}$ (internally) maps each identifier I_v it sees (at some node v) in G to a bit string by applying the function ρ, and then it simulates algorithm $\mathcal{A}$, where the private random bit string that $\mathcal{A}$ has access to in each node v is given by $\rho(I_v)$, the shared random bit string is given by ρ^*, and the input parameter given to $\mathcal{A}$ describing the number of nodes is set to N. Equivalently, this can be seen as running $\mathcal{A}$ (with randomness provided by ρ and ρ^*) on the graph H obtained from G by adding $N - n$ isolated nodes (where we are only interested in the output of $\mathcal{A}$ on the nodes of H that correspond to nodes in G).

Since, on N-node graphs, $\mathcal{A}$ provides a correct output with probability at least $1 - 1/N$, and ρ and ρ^* are chosen uniformly at random, we see that, for every $G \in \mathcal{G}_n$, the probability that $\mathcal{A}_{\rho,\rho^*}$ fails on G is at most $1/N$. Since the number of graphs in $\mathcal{G}_n$ is strictly smaller than N, it follows that there are a function $\rho_{\text{det}} : [2^{O(n)}] \to \{0,1\}^{\mathbb{N}}$ and a bit string $\rho^*_{\text{det}} \in \{0,1\}^{\mathbb{N}}$ such that $\mathcal{A}_{\rho_{\text{det}},\rho^*_{\text{det}}}$ does not fail on any graph in $\mathcal{G}_n$.

As the runtime of $\mathcal{A}_{\rho_{\text{det}},\rho^*_{\text{det}}}$ is equal to the runtime of $\mathcal{A}$ on N-node graphs, we can conlude that $\mathcal{A}_{\rho_{\text{det}},\rho^*_{\text{det}}}$ is a deterministic VOLUME algorithm with probe complexity $t(N) = o(\sqrt{\log N}) = o(\sqrt{\log 2^{O(n^2)}}) = o(n)$, as desired. $\square$

Similarly, the next lemma is a variant of Theorem 6 in [94] and the discussion in Section 7.1 in [270].

Lemma 8.11 (Speedup in VOLUME with exponential identifiers)**.** *If there exists a deterministic* VOLUME *algorithm $\mathcal{A}$ for an LCL $\Pi = (\Sigma_{in}, \Sigma_{out}, r, \mathcal{P})$ with a probe complexity of $o(n)$ that works with unique identifiers from $[2^{O(n)}]$, then there also is a deterministic* VOLUME *algorithm $\mathcal{A}'$ for Π with a probe complexity of $O(\log^* n)$.*

Proof. Let n_0 be a big enough constant. The algorithm $\mathcal{A}'$ does the following on a given input graph G with maximum degree Δ: first, it uses the algorithm of Even et al. [127] to construct a coloring of the power graph G^{n_0+r}—the graph with the vertex set of G and where two nodes are connected by an edge iff they have a distance of at most $n_0 + t$ in the graph G—with $\Delta^{n_0+r} + 1 = 2^{O(n_0)}$ colors with a probe complexity of $O(\log^* n)$. Then we interpret those colors as identifiers and run the algorithm $\mathcal{A}$ on G but we tell the algorithm that the input graph has n_0 nodes instead of n. The query complexity of $\mathcal{A}'$ is $O(\log^* n \cdot o(n_0)) = O(\log^* n)$.

To see that $\mathcal{A}'$ produces a valid output, note that if $\mathcal{A}'$ fails at a node u, we may consider all the nodes that were needed for the simulation of $\mathcal{A}$ in the r-hop neighborhood of u together with their neighbors—the number of such nodes is bounded by $\Delta{\cdot}\Delta^r{\cdot}o(n_0) < n_0$, by choosing n_0 large enough. But as all those nodes are labeled by unique identifiers from $[2^{O(n_0)}]$, we can get a graph of size n_0 on which the original algorithm $\mathcal{A}$ fails, a contradiction. □

Theorem 8.2 now follows from Theorems 8.10 and 8.11.

We remark that by using polynomial instead of exponential identifiers in Theorem 8.10, we would get that a randomized LCA /VOLUME algorithm of complexity $t(n) = o(\log n/ \log\log n)$ can be derandomized to a deterministic algorithm with complexity $t(2^{O(n \log n)}) = o(n)$. The term $2^{O(n \log n)}$ comes from a union bound over all n-node graphs of maximum degree Δ labeled with polynomial-sized unique identifiers. This is the reason for the segment between the complexity pairs $[\log n, \log\log n]$ and $[n, \log n/ \log\log n]$ (cf. Section 1.2 and Figure 2 in [270] that does not differentiate between $\Theta(\log n)$ and $\Theta(\log n/ \log\log n)$).

8.4 The Volume Lower Bound for Sinkless Orientation

In this section, we prove the lower bound of Theorem 8.1. We prove it by providing the respective lower bound for the Sinkless Orientation problem on trees.

Theorem 8.12. *There is no randomized LCA algorithm with probe complexity $o(\log n)$ for the problem of Sinkless Orientation or Δ-coloring, even if the input graph is a tree with a precomputed Δ-edge coloring. In particular, the* LCA *complexity of LLL is $\Omega(\log n)$, even in the regime $p \leq 2^{-\Delta}$.*

The general idea of the proof is the same as in Section 8.3: we want to derandomize the assumed $o(\log n)$-probe randomized algorithm for Sinkless Orientation to a deterministic $o(n)$-probe VOLUME algorithm, this time restricting what constitutes a valid ID assignment. We then reduce the problem of showing that such a VOLUME algorithm does not exist to the problem of showing that there does not exist a nontrivial deterministic LOCAL algorithm for Sinkless Orientation. We will explain later in more detail what we mean by nontrivial. Finally, we conclude the proof by showing that such a LOCAL algorithm does not exist.

As we saw in Section 8.3, a direct application of the derandomization by Chang-Kopelowitz-Pettie allows us only to argue that an $o(\sqrt{\log n})$-probe algorithm leads to an $o(n)$-probe deterministic algorithm, so we need to do better. There are two obstacles to obtaining a union bound over $2^{O(n)}$ graphs in Theorem 8.10:

1. the number of n-node graphs of maximum degree Δ is bounded only by $2^{O(n \log n)}$,
2. the number of ways of labeling n objects with labels from $[2^{O(n)}]$ is $2^{O(n^2)}$.

To get an $\Omega(\log n)$ lower bound, both terms need to be improved to $2^{O(n)}$.

This is easy with the first term – in fact, the whole lower bound works even if we restrict ourselves to trees. The number of trees with maximum degree Δ can easily be upper bounded by $\Delta^{O(n)}$ and an even stronger upper bound of 2.96^n is known.

The issue is with the second bullet point—the number of labelings of n objects with labels from range $2^{O(n)}$ clearly cannot be improved from $2^{O(n^2)}$.

To decrease the number of possibilities, we need to restrict our space of labelings of a tree with unique identifiers in such a way that the number of possibilities drops to $2^{O(n)}$, yet this restriction should not make it easier to solve Sinkless Orientation in the VOLUME model. This is done by the ID graph technique developed in a parallel paper [83] for a different purpose. An ID graph H is a graph that states which pairs of identifiers are allowed for a pair of neighboring nodes of the input tree. In our case, one should think about it as a high-girth high chromatic number graph on $2^{O(n)}$ nodes with each node representing an identifier. The girth of the graphs is at least $\Theta(n)$ and its chromatic number at least Δ (i.e., the maximum degree of the *input graph*). Our definition is a little subtler due to the fact that we work on edge-colored trees where arguments are usually easier.

ID graph: We now define the ID graph, prove that the number of labelings of n-node trees consistent with it is bounded by $2^{O(n)}$ in Theorem 8.18, and then prove Theorem 8.19. Each vertex of the ID graph can be considered as an identifier that will later be used to provide IDs to the considered input graph.

Definition 8.13 (ID graph)**.** *Let R and Δ be positive integers. An ID graph $H = H(R, \Delta)$ is a collection of graphs $H_1, H_2, \ldots, H_\Delta$ such that the following hold:*

1. *For all i, j satisfying $1 \leq i, j \leq \Delta : V(H_i) = V(H_j)$; we use $V(H)$ to denote the set of vertices in H, that is, $V(H) = V(H_1)$,*
2. *$|V(H)| = \Delta^{10R}$,*
3. *$\forall v \in V(H), \forall 1 \leq i \leq \Delta : 1 \leq \deg_{H_i}(v) \leq \Delta^{10}$,*
4. *$\mathrm{girth}(H) \geq 10R$,*

5. *Any independent set of H_i has less than $|V(H)|/\Delta$ vertices.*

Lemma 8.14 (ID graph existence)**.** *There exists an ID graph $H = H(R, \Delta)$ for all sufficiently large $R, \Delta > 0$.*

This lemma is proved in a parallel paper [83] developing the technique for a different purpose.

As there are possibly many ID graphs that satisfy the given conditions, from now on, whenever we write $H(R, \Delta)$, we mean the lexicographically smallest ID graph $H(R, \Delta)$.

Definition 8.15 (Proper H-labeling of a Δ-edge-colored tree)**.** *Let T be a tree having a maximum degree of at most Δ and whose edges are properly colored with colors from $[\Delta]$. A proper H-labeling of T with an ID graph H is a labeling $h : V(T) \to V(H)$ of each vertex $u \in V(T)$ with a vertex $h(u) \in V(H)$ such that whenever $u, v \in V(T)$ are incident to a common edge colored with color $c \in [\Delta]$, then $h(u)$ and $h(v)$ are neighboring in H_c.*

Definition 8.16 (Solving an LCL relative to H)**.** *When we say that a deterministic* VOLUME *or* LOCAL *algorithm for an LCL Π works* relative to an ID graph H*, we mean that the algorithm works if the unique node identifiers are replaced by a proper $H(n, \Delta)$-labeling for instances of size n and with maximum degree at most Δ.*

We also say that an algorithm works relative to ID graphs $\{H_n\}_{n\in\mathbb{N}}$ *if it works for every n-node input graph that is H-labeled by H_n.*

Observation 8.17. *If a deterministic local algorithm $\mathcal{A}$ solves a problem Π of checkability radius t in r rounds (in case of the LOCAL model) or with r probes (in case of the volume model) with $\Delta^{O(r)}$-sized identifiers, then it also solves Π relative to $H(t + r, \Delta)$.*

Proof. The validity of $\mathcal{A}$ depends on all possible ways of labeling an $(r + t)$-hop neighborhood of a vertex u with identifiers, as the correctness of the output at u depends only on the outputs given by $\mathcal{A}$ at each vertex v in u's t-hop neighborhood (by the definition of an LCL problem), and each such output depends only on the r-hop neighborhood of the respective node v. But the set of allowed labelings relative to $H(t+r, \Delta)$ is a subset of all labelings with unique identifiers. □

Lemma 8.18. *The number of non-isomorphic n-node trees with maximum degree $\Delta = O(1)$ that are labeled with a proper Δ-edge coloring and an H-labeling for $H = H(n, \Delta)$, is $2^{O(n)}$.*

Proof. There are $2^{O(n)}$ non-isomorphic unrooted trees on n vertices. Additionally, there are at most Δ^{n-1} ways of assigning edge colors from the set $[\Delta]$ once the tree is fixed. Hence, there are $2^{O(n)}$ non-isomorphic trees labeled with edge colors.

For any such tree T, pick an arbitrary vertex u in it. There are $|V(H)| = \Delta^{O(n)}$ ways of labeling u with a label from H (Property 2 in Theorem 8.13). Once u is labeled with a label $h(u) \in V(H)$, we can construct the labeling of the whole tree T by gradually labeling it, vertex by vertex, always labeling a node whose neighbor was labeled already. Every time we label a new node v with the label $h(v) \in V(H)$ such that v is adjacent to an already labeled node w via an edge of color c, we have poly(Δ) possible choices for the label of v, since this is the degree of $h(w)$ in H_c (Property 3 in Theorem 8.13). Hence, the total number of H-labelings of T is $2^{O(n)}$.

Putting everything together, the number of non-isomorphic trees labeled by edge colors from $[\Delta]$ and IDs from H is $2^{O(n)}$. □

The following lemma is analogous to Theorem 8.10, i.e., the derandomization from [94], but working relative to the ID graph and only on the set of trees allows us to derandomize a randomized VOLUME algorithm with probe complexity $t(n)$ to obtain a deterministic VOLUME algorithm with probe complexity $t(2^{O(n)})$, while the original construction only obtains a deterministic VOLUME algorithm with probe complexity $t(2^{O(n^2)})$.

Lemma 8.19 (From randomized LCA to deterministic VOLUME relative to an ID graph)**.** *If there exists a randomized LCA algorithm with probe complexity $t(n) = o(\log n)$ for Sinkless Orientation on trees with maximum degree $\Delta = O(1)$ that are properly Δ-edge colored, then there also exists a deterministic VOLUME algorithm for Sinkless Orientation on trees with maximum degree $\Delta = O(1)$ that are properly Δ-edge colored with probe complexity $o(n)$ relative to ID graphs $\{H(n, \Delta)\}_{n \geq n_0}$ where n_0 is a large enough constant.*

Proof. We omit the proof as it can be proven in the exact same way as Theorem 8.10, except that now we tell the randomized algorithm that the number of nodes is $N = 2^{O(n)}$ instead of $N = 2^{O(n^2)}$ as we need to union bound over a smaller number of labeled graphs. One also needs to make use of the fact that all the IDs in an n-node $H(n, \Delta)$-labeled graph are unique, which follows from the fact that the girth of $H(n, \Delta)$ is strictly larger than n. □

Hardness of Deterministic Sinkless Orientation relative to an ID graph: We now prove that there cannot be a deterministic local algorithm with volume complexity $o(n)$ that works with exponential IDs, even when those IDs satisfy constraints defined by an ID graph $H(n, \Delta)$.

To do so, we first show that an $o(n)$-probe VOLUME algorithm implies that there exists some constant $n^* \in \mathbb{N}$ and a deterministic LOCAL algorithm that solves Sinkless Orientation on an (infinite) properly $H(n^*, \Delta)$-labeled tree in strictly less than n^* rounds. This is an analogue of Theorem 8.11.

Lemma 8.20. *If there exists a deterministic VOLUME algorithm $\mathcal{A}$ that solves Sinkless Orientation on n-node trees with maximum degree $\Delta = O(1)$ that are properly Δ-edge colored relative to $H(n, \Delta)$ and with a probe complexity of $f(n) \leq n/(3\Delta)$ for all large enough n, then there exists a constant n^* and a deterministic LOCAL algorithm $\mathcal{A}'$ that* *solves Sinkless Orientation on all (possibly infinite) trees with maximum degree Δ that are properly Δ-edge colored and $H(n^*, \Delta)$-labeled in fewer than n^* rounds.*

Proof. Consider running the algorithm $\mathcal{A}$ for fixed n^* and Δ such that $f(n^*) \leq n^*/(3\Delta)$ on *any* tree T having a maximum degree of Δ that is properly Δ-edge colored and $H(n^*, \Delta)$-labeled. We now prove that $\mathcal{A}$ solves the Sinkless Orientation problem on T. If not, then there exists a node u such that either all the edges are oriented towards u or u has a neighbor v such that the outputs of u and v are inconsistent. Consider now the set S consisting of the at most $2 \cdot (n^*/3\Delta)$ vertices that $\mathcal{A}$ probes in order to compute the answer for both u and v. The set $S \cup N(S)$ has at most $\Delta \cdot |S| < n^*$ vertices. If we run $\mathcal{A}$ on $G[S \cup N(S)]$, it will fail, too, since the two runs of $\mathcal{A}$ on T and $T[S \cup N(S)]$ are identical. Appending a non-zero number of vertices to the vertices from $N(S)$ and labeling them so that the final labeling is still a $H(n^*, \Delta)$-labeling gives a graph T' on exactly n^* nodes that is properly $H(n^*, \Delta)$-labeled such that $\mathcal{A}$ fails on T'. This is a contradiction with the correctness of $\mathcal{A}$ on n^*-node trees.

Hence, we get a deterministic VOLUME algorithm for Sinkless Orientation that performs at most $n^*/(3\Delta)$ probes and that works for any $H(n^*, \Delta)$-labeled tree T. This directly implies that there exists a LOCAL algorithm with a round complexity of at most $n^*/(3\Delta)$ that solves Sinkless Orientation on any $H(n^*, \Delta)$-labeled tree T, as needed. □

Finally, the following theorem is a simple adaptation of the lower bound for Sinkless Orientation in the LOCAL model via round elimination in [73]).

Theorem 8.21. *The deterministic complexity of Sinkless Orientation in the* LOCAL *model relative to $H(k, \Delta)$ is at least k.*

This theorem is proved Section 2.2. Finally, we can put all the pieces together to prove Theorem 8.12.

Proof of Theorem 8.12. We first apply the derandomization of Theorem 8.19 to deduce the existence of a deterministic algorithm with $o(n)$ probes, relative to $\{H(n, \Delta)\}_{n \in \mathbb{N}}$. Afterwards, we use Theorem 8.20 to conclude that there is an $r > 0$ and a deterministic local algorithm that solves sinkless orientation in less than r rounds relative to an ID graph $H(r, \Delta)$, which is finally shown to be impossible by Theorem 8.21. □

Remark 8.22. *One can check that we actually prove the existence of some $\varepsilon = \varepsilon(\Delta) > 0$ such that any randomized LCA algorithm for sinkless orientation needs at least $\varepsilon \log n$ probes for all $n \geq n_0$.*

8.5 Upper Bound for LLL on Constant Degree Graphs

In this section, we complement the lower bound by proving the following matching upper bound result.

Theorem 8.23. *There exists a fixed constant c such that the randomized* LCA /VOLUME *complexity of the LLL on constant degree graphs under the polynomial criterion* $p(e\Delta)^c \leq 1$ *is* $O(\log n)$.

Proof. The result follows by a slight adaptation of the LOCAL algorithm of [?] to the VOLUME model. In particular, Fischer and Ghaffari show how to shatter a constant-degree graph in $O(\log^* n)$ rounds of the LOCAL model. By shattering, we mean that their algorithm fixes the values for a subset of the random variables such that the following two properties are satisfied.

1. The probability of each bad event conditioned on the previously fixed random variables is upper bounded by $\Delta^{-\Omega(c)}$.
2. Consider the graph induced by all the nodes whose corresponding bad event has a non-zero probability of occurring. The connected components in this graph all have a size of $O(\log n)$ with probability $1 - 1/\operatorname{poly}(n)$.

Note that the shattering procedure – denoted as the pre-shattering phase – directly implies a randomized VOLUME algorithm with probe complexity $O(\log n \cdot \Delta^{O(\log^* n)})$. To see why, note that we can determine the state of all random variables after the pre-shattering phase that a given bad event depends on with a probe complexity of $\Delta^{O(\log^* n)}$ by applying the Parnas-Ron reduction to the $O(\log^* n)$ round LOCAL algorithm. This in turn allows us to find the connected component of a given node with $O(\log n \cdot \Delta^{O(\log^* n)})$ probes, as long as the connected component has a size of $O(\log n)$, which happens with probability $1 - 1/\operatorname{poly}(n)$. Afterwards, one can find a valid assignment of all the random variables in the connected component in a brute-force centralized manner. Note that the standard LLL criterion $ep(\Delta + 1) \leq 1$ guarantees the existence of such an assignment.

To improve the probe complexity to $O(\log n)$, we show how to adapt the pre-shattering phase such that it runs in $O(1)$ LOCAL rounds while still retaining the two properties stated above. Once we have shown this, the aforementioned VOLUME simulation directly proves Theorem 8.23. The pre-shattering phase of [?] works by first computing a 2-hop-coloring with $\Delta^2 + 1$ colors, where a 2-hop coloring is a coloring that assigns any two nodes having a distance of at most 2 a different color. Once this coloring is computed, the algorithm iterates through the constantly many color classes and fixes in each iteration a subset of the random variables. Each random variable will be set with probability $1 - \Delta^{-\Omega(c)}$, even when fixing the randomness outside the c_1-hop neighborhood for some fixed constant $c_1 \geq 2$ not depending on c adversarially. The only step that takes more than constant time is the computation of the coloring. Instead of computing the coloring deterministically, we instead assign each node one out of $\Delta^{c'}$ colors for some fixed positive constant $c' \gg 1$, independently and uniformly at random. We say that a node fails if its chosen color is not unique in its 2-hop neighborhood. Note that a given node fails with probability at most $1/\Delta^{\Omega(c')}$. We then postpone the assignment of each random variable that affects any of the failed nodes. For all the other random variables, we run the same process as described in [?] by iterating through the $\operatorname{poly}(\Delta) = O(1)$ color classes in $O(1)$ rounds of the LOCAL model. The pre-shattering phase of [?] still deterministically guarantees that each bad event occurs with probability $1 - \Delta^{-\Omega(c)}$ conditioned on the random variables set in the pre-shattering phase. Moreover, each bad event that does not correspond to a failed node occurs with probability 0 after the pre-shattering partial assignment with probability at least $1 - \Delta^{-\Omega(c)}$, independent of the randomness outside the c_1-hop neighborhood. An application of Theorem 8.24 therefore guarantees that by choosing c and c' large enough, each connected component after the pre-shattering phase has a size of $O(\log n)$ with high probability in n, thus concluding the proof of Theorem 8.23.

Lemma 8.24 (The Shattering Lemma, cf. with Lemma 2.3 of [**?**]). *Let $G = (V, E)$ be a graph with maximum degree $\Delta = O(1)$. Consider a process which generates a random subset $B \subseteq V$ where $Pr[v \in B] \leq \Delta^{-c_1}$, for some constant $c_1 \geq 1$, independent on the randomness of nodes outside the c_2-hop neighborhood of v, for all $v \in V$, for some constant $c_2 \geq 1$. Then, with probability at least $1-n^{-c_3}$, for any constant $c_3 < c_1-4c_2-2$, we have that each connected component of $G[B]$ has size $O(\log n)$.*

□

8.6 The Volume Model Gap

In this section, we show that a deterministic or randomized VOLUME algorithm (LCA) with a probe complexity of $o(\log^* n)$ implies a deterministic VOLUME algorithm (LCA) with a probe complexity of $O(1)$. We first show the speed-up only for deterministic VOLUME algorithms and later discuss how to extend the result to the full generality.

Definition 8.25. *For an arbitrary $S \subseteq \mathbb{N}$, we define*

$$Tuples_S = \{(id, deg, in) \colon id \in S, deg \in [\Delta], in \colon [deg] \mapsto \Sigma_{in}\}$$

and for $i \in \mathbb{N}$, we define

$$Tuples_{i,S} = \{(t_1, t_2, \ldots, t_i) \colon \text{for every } j \in [i],\ t_j \in Tuples_S\}.$$

For a given $i \in \mathbb{N}$ and $\ell \in [2]$, let

$$t^{(\ell)} =$$
$$((id^\ell_1, deg_1, in_1), (id^\ell_2, deg_2, in_2), \ldots, (id^\ell_i, deg_i, in_i)) \in Tuples_{i,\mathbb{N}}$$

be arbitrary. We say that the tuples $t^{(1)}$ and $t^{(2)}$ are almost identical if for every $j_1, j_2 \in [i]$, $id^1_{j_1} < id^2_{j_2}$ implies $id^2_{j_1} < id^2_{j_2}$, $id^1_{j_1} > id^1_{j_2}$ implies $id^2_{j_1} > id^2_{j_2}$ and $id^1_{j_1} = id^1_{j_2}$ implies $id^2_{j_1} = id^2_{j_2}$.

A tuple in $Tuples_S$ can encode the local information of a node v, including its ID, its degree and the input assigned to each of its incident half-edges, i.e., $in(k)$ is the input assigned to the k-th half edge. We denote with t_v the tuple that encodes the local information of v. A tuple in $Tuples_{i,S}$ can be used to encode all the information (modulo the number of nodes of the input graph) that a node knows about the input graph after having performed $i-1$ probes. The notion of almost identical tuples will be helpful for defining the notion of order-invariance for the VOLUME model.

The notion of an order-invariant algorithm (cf. Chapter 7) naturally extends to algorithms in the VOLUME model.

Definition 8.26. *(Order-invariant VOLUME algorithm) We say that $\mathcal{A}$ is order invariant if for every $n \in \mathbb{N}$ and $i \in [T(n)+1]$, $f_{n,i}(t) = f_{n,i}(t')$ for every $t, t' \in Tuples_{i,S}$ with t and t' being almost identical.*

We will use the following basic speed-up result for order-invariant algorithms in both the LOCAL and the VOLUME model.

Theorem 8.27 (Speed-up of order-invariant algorithms (cf. [94]))**.** *Let $\mathcal{A}$ be an order-invariant algorithm solving a problem Π in $f(n) = o(\log n)$ rounds of the LOCAL model or with $f(n) = o(n)$ probes of the VOLUME model. Then, there is an order-invariant algorithm solving Π in $O(1)$ rounds of the LOCAL model or, equivalently, $O(1)$ probes in the VOLUME model.*

Proof. The result for the LOCAL model is proven in [94]. The proof for the VOLUME model follows in the exact same manner. We provide it here for completeness. Let n_0 be a fixed constant such that $\Delta^{r+1} \cdot (T(n_0)+1) \leq n_0/\Delta$. We now define a VOLUME algorithm $\mathcal{A}'$ with probe complexity $T'(n) = \min(n, T(n_0)) = O(1)$ as follows. For $n \in \mathbb{N}$ and $i \in [T'(n)+1]$, we define $f^{\mathcal{A}'}_{n,i} = f^{\mathcal{A}}_{\min(n,n_0),i}$. It remains to show that $\mathcal{A}'$ indeed solves Π. For the sake of contradiction, assume that this is not the case. This implies the existence of a Σ_{in}-labeled graph (G, f_{in}) on $n \geq n_0$ nodes (with IDs from a polynomial range, port assignments, and no isolated nodes) such that $\mathcal{A}'$ "fails" on (G, f_{in}). Put differently, there exists a node v such that $\mathcal{A}'$ produces a mistake in the r-hop neighborhood of v. The r-hop neighborhood of v consists of at most Δ^{r+1} vertices. To answer a given query, $\mathcal{A}'$ "sees" at most $T(n_0)+1$ nodes. Hence, to compute the output of all the nodes in the r-hop neighborhood of v, $\mathcal{A}'$ "sees" at most $\Delta^{r+1}(T(n_0)+1) \leq \frac{n_0}{\Delta}$ many nodes. We denote the set consisting of those nodes as $V^{visible}$. Now, let (G', f'_{in}) be a Σ_{in}-labeled graph on n' nodes (with IDs from a polynomial range, port assignments, and no isolated nodes) such that every $u \in V^{visible}$ is also contained in G', with its degree being the same in both graphs, as well as the input assigned to each of its incident half edges. The assigned ID can be different, however, the relative orders of the IDs assigned to nodes in $V^{visible}$ in G and G' are the same. As $\Delta^{r+1}(T(n_0)+1) \leq \frac{n_0}{\Delta}$, such a (G', f'_{in}) exists. As $\mathcal{A}$ is order invariant, so is $\mathcal{A}'$. Moreover, $f^{\mathcal{A}'}_{n,i} = f^{\mathcal{A}'}_{n',i}$ for any i. Hence, it follows that $\mathcal{A}'$ assigns the same output to all the half-edges in the r-hop neighborhood of v in G and G'. Therefore, $\mathcal{A}'$ also fails on the graph G'. From the way we defined $\mathcal{A}'$, this directly implies that $\mathcal{A}$ also fails on G, a contradiction with the assumption that $\mathcal{A}$ is a correct algorithm. This finishes the proof. □

Theorem 8.3. *There does not exist an LCL with a deterministic* VOLUME *complexity between* $\omega(1)$ *and* $o(\log^* n)$.

Proof. Consider some LCL $\Pi = (\Sigma_{\text{in}}, \Sigma_{\text{out}}, r, \mathcal{P})$. Let $\mathcal{A}$ be a VOLUME model algorithm with a probe complexity of $T(n) = o(\log^* n)$ that solves Π. We show that this implies the existence of a VOLUME model algorithm that solves Π and has a probe complexity of $O(1)$. To do so, we first show that we can turn $\mathcal{A}$ into an order-invariant algorithm with the same probe complexity. Once we have shown this, Theorem 8.27, which shows a speed-up result for order-invariant algorithms, implies that there also exists a VOLUME model algorithm for Π with a probe complexity of $O(1)$, as needed.

We start by proving the lemma below, which, informally speaking, states that there exists a sufficiently large set $S_n \subseteq [n]$ such that $\mathcal{A}$ is order-invariant as long as the IDs it "encounters" are from the set S_n. The proof adapts a Ramsey-theoretic argument first introduced in [258] and further refined in [94] in the context of the LOCAL model to the VOLUME model.

Lemma 8.28. *There exists a* $n_0 \in \mathbb{N}$ *such that the following holds for every* $n \geq n_0$. *There exists a set* $S_n \subseteq [n]$ *of size* $(T(n) + 1) \cdot \Delta^{r+1}$ *such that for every* $i \in [T(n) + 1]$ *and* $t, t' \in Tuples_{i,S_n}$ *it holds that* $f_{n,i}(t) = f_{n,i}(t')$ *if* t *and* t' *are almost identical.*

Proof. Consider an $n \in \mathbb{N}$. We denote with H a complete $(T(n)+1)$-uniform hypergraph on n nodes. Each node in H corresponds to a (unique) ID in the set $\{1, 2, \ldots, n\}$. For each hyperedge $X \subseteq \{1, 2, \ldots, n\}$ we define a function f_X.

The input to f_X is a tuple $t = (t_1, \ldots, t_{T(n)+1}) \in Tuples_{T(n)+1,[T(n)+1]}$. For $j \in [T(n)+1]$, let $t_j = (id_j, deg_j, in_j)$. We define a new tuple $t_j^X = (id_j^X, deg_j, in_j)$, where id_j^X is the id_j-th smallest element in X. We now define

$$f_X(t) = (f_{n,1}(t_1^X), f_{n,2}(t_1^X, t_2^X), \ldots, f_{n,T(n)+1}(t_1^X, t_2^X, \ldots, t_{T(n)+1}^X)).$$

We now prove two things. First, we show that for n being sufficiently large, there exists a set $S_n^{big} \subseteq [n]$ of size $(T(n) + 1) \cdot \Delta^{r+1} + T(n) + 1$ such that for any two hyperedges $X, Y \subseteq S_n^{big}$, $f_X = f_Y$. This will follow by Ramsey Theory and an upper bound on the number of possible different functions f_X. Second, let S_n be the set one obtains from S_n^{big} by discarding the $T(n) + 1$ largest elements of S_n^{big}. We show that S_n satisfies the conditions of Theorem 8.28.

For the Ramsey-theoretic argument, we start by upper bounding the total number of possible functions. Note that $|Tuples_{[T(n)+1]}| \leq (T(n) + 1) \cdot \Delta \cdot |\Sigma_{in}|^{\Delta}$ and therefore the possible number of inputs to the function f^X is $|Tuples_{T(n)+1,[T(n)+1]}| \leq \left((T(n) + 1) \cdot \Delta \cdot |\Sigma_{in}|^{\Delta}\right)^{T(n)+1} = T(n)^{O(T(n))}$. Note that the output of f_X is contained in the set $\left(\bigtimes_{i=1}^{T(n)}([i] \times [\Delta])\right) \times (\Sigma_{out})^{[\Delta]}$ and therefore there are at most $(T(n) \cdot \Delta)^{T(n)} \cdot |\Sigma_{out}|^{\Delta} = T(n)^{O(T(n))}$ possible outputs. Hence, there exist at most

$\left(T(n)^{O(T(n))}\right)^{T(n)^{O(T(n))}}$ different possible functions. Let $R(p,m,c)$ denote the smallest number such that any p-uniform complete hypergraph on $R(p,m,c)$ nodes with each hyperedge being assigned one of c colors contains a monochromatic clique of size m. It holds that $\log^*(R(p,m,c)) = p + \log^* m + \log^* c + O(1)$ [88].

Setting $p = T(n)+1$, $m = (T(n)+1)\cdot\Delta^{r+1}+T(n)+1$ and $c = \left(T(n)^{O(T(n))}\right)^{T(n)^{O(T(n))}}$, we can conclude that H contains a set $S_n^{big} \subseteq [n]$ of size m such that for any two hyperedges $X, Y \subseteq S_n^{big}$, $f_X = f_Y$ as long as $\log^*(n) \geq T(n)+1+\log^* m+\log^* c+O(1)$, which is the case for sufficiently large n as $T(n) = o(\log^* n)$, $\log^* m = o(T(n))$ and $\log^* c = o(T(n))$.

Now, let S_n be the set one obtains form S_n^{big} by discarding the $T(n)+1$ largest elements from S_n^{big}. Let $i \in [T(n)+1]$ and $t^{(\ell)} = ((id_1^\ell, deg_1, in_1),\ (id_2^\ell, deg_2, in_2),\ \ldots, (id_i^\ell, deg_i, in_i)) \in Tuples_{i,S_n}$ for $\ell \in [2]$ such that $t^{(1)}$ and $t^{(2)}$ are almost identical. It remains to show that $f_{n,i}(t^{(1)}) = f_{n,i}(t^{(2)})$. For $\ell \in [2]$, let $X^\ell \subseteq S_n^{big}$ such that $\{id_1^\ell, id_2^\ell, \ldots, id_i^\ell\}$ contains the $|\{id_1^\ell, id_2^\ell, \ldots, id_i^\ell\}|$-th lowest elements of X^ℓ. Now, let $t = (t_1, t_2, \ldots, t_{T(n)+1}) \in S_{T(n)+1,[T(n)+1]}$ such that for every $j \in [i]$, $t_j^{X^\ell} = (id_j^\ell, deg_j, in_j)$. Note that it follows from the way we defined X^1 and X^2 and the fact that $t^{(1)}$ and $t^{(2)}$ are almost identical that such a tuple t exists. As $X^\ell \subseteq S_n^{big}$, we have $f_{X^1}(t) = f_{X^2}(t)$. In particular, this implies that

$$\begin{aligned} f_{n,i}(t^{(1)}) &= f_{n,i}(t_1^{X^1}, t_2^{X^1}, \ldots, t_i^{X^1}) \\ &= f_{n,i}(t_1^{X^2}, t_2^{X^2}, \ldots, t_i^{X^2}) = f_{n,i}(t^{(2)}), \end{aligned}$$

which finishes the proof.

□

We now construct an order-invariant algorithm $\mathcal{A}'$ with probe complexity $T'(n) = \max(O(1), T(n)) = o(\log^* n)$. Note that it is easy to make $\mathcal{A}'$ order-invariant for every input graph having fewer than n_0 nodes. For $n \geq n_0$, we have $T'(n) = T(n)$ and for every $i \in \{1, 2, \ldots, T'(n)\}$ and tuple $t = ((id_1, deg_1, in_1),\ (id_2, deg_2, in_2)\ ,\ldots,\ (id_i, deg_i, in_i)) \in Tuples_{i,\mathbb{N}}$, we define

$$\begin{aligned} & f_{n,i}^{\mathcal{A}'}(t) = f_{n,i}^{\mathcal{A}}(t') = \\ & (id_1', deg_1, in_1), (id_2', deg_2, in_2), \ldots, (id_i', deg_i, in_i)) \end{aligned}$$

where $id_1', \ldots, id_i'$ is chosen in such a way that $\{id_1', \ldots, id_i'\}$ contains the $|\{id_1', \ldots, id_i'\}|$-th smallest elements of S_n and t and t' are almost identical. It is easy to verify that $\mathcal{A}'$ is indeed order-invariant.

It remains to show that $\mathcal{A}'$ indeed solves Π. For the sake of contradiction, assume that this is not the case. This implies the existence of a Σ_{in}-labeled graph (G, f_{in}) on n nodes (with each node having a unique ID from a polynomial range, a port assignment, and G does not have any isolated nodes) such that $\mathcal{A}'$ "fails" on (G, f_{in}). Put differently, there exists a node v such that $\mathcal{A}'$ produces a mistake in the r-hop neighborhood of v. The r-hop neighborhood of v consists of at most Δ^{r+1} vertices. To answer a given query, $\mathcal{A}'$ "sees" at most $T(n)+1$ nodes. Hence, to compute the output of all the half-edges in the r-hop neighborhood of v, $\mathcal{A}'$ "sees" at most $\Delta^{r+1}(T(n)+1) \leq |S_n|$ many nodes. We denote this set of nodes by $V^{visible}$. Even if the IDs of nodes outside of $V^{visible}$ are changed, $\mathcal{A}'$ still fails around v. Moreover, as $\mathcal{A}'$ is order-invariant, changing the IDs of the nodes in $V^{visible}$ in a manner that preserves the relative order does not change the fact that $\mathcal{A}'$ fails around v. Hence, we can find a new assignment of IDs such that each node in $V^{visible}$ is assigned an ID from the set S_n such that $\mathcal{A}'$ still fails around v. However, from the way we defined $\mathcal{A}'$ and the property that S_n satisfies, it follows that $\mathcal{A}'$ and $\mathcal{A}$ produce the same output in the r-hop neighborhood around v. This contradicts the fact that $\mathcal{A}$ is a correct algorithm.

□

We already argued in the preliminaries that Theorem 8.3 also implies that there does not exist an LCL with a deterministic LCA complexity between $\omega(1)$ and $o(\log^* n)$.

As noted in [270], the derandomization result by [94] can be used to show that randomness does not help (up to an additive constant in the round/probe complexity) in the LOCAL and VOLUME model for complexities in $O(\log^* n)$, and the same is true for the LCA model.

Hence, we obtain the following more general theorem.

Theorem 8.29. *There does not exist an LCL with a randomized/deterministic* LCA /VOLUME *complexity between* $\omega(1)$ *and* $o(\log^* n)$.

CHAPTER 9

Deterministic Network Decompositions for Weighted Graphs with Applications

9.1 Introduction

This chapter gives deterministic parallel & distributed algorithms for low-diameter clusterings in weighted graphs. These results generalize the algorithms for network decomposition that we have discussed in Section 1.5. An example is the following theorem.

Theorem 9.1. *[A corollary of Theorem 9.30] Let G be a weighted graph. We are given a set of* terminals *$Q \subseteq V(G)$ and a parameter $R > 0$ such that for every $v \in V(G)$ we have $d(Q, v) \le R$. Also, a precision parameter $0 < \varepsilon < 1$ is given. There is a deterministic distributed and parallel algorithm outputting a partition $\mathcal{C}$ of the vertices into clusters and a subset of terminals $Q' \subseteq Q$ with the following properties:*

1. *Each cluster $C \in \mathcal{C}$ contains exactly one terminal $q \in Q'$. Moreover, for any $v \in C$ we have $d_{G[C]}(q, v) \le (1 + \varepsilon)R$.*
2. *For the set E^{bad} of edges connecting different clusters of $\mathcal{C}$ we have*

$$|E^{bad}| = \widetilde{O}\left(\frac{1}{\varepsilon R}\right) \cdot \sum_{e \in E(G)} \ell(e).$$

The PRAM *variant of the algorithm has work $\widetilde{O}(m)$ and depth $\widetilde{O}(1)$. The* CONGEST *variant of the algorithm runs in $\widetilde{O}(\sqrt{n} + \mathrm{HopDiam}(G))$ rounds.*

Our result above is in fact quite model-independent as we essentially reduce the problem to $\mathrm{poly}(\log n)$ $(1 + 1/\mathrm{poly}(\log n))$-approximate distance computations. The final complexities then follow from the recent work of [274] where the authors give efficient deterministic parallel and distributed approximate shortest path algorithms in PRAM and CONGEST .

Low-Stretch Spanning Trees: As a straightforward corollary of the clustering result in Theorem 9.1, we obtain an efficient deterministic parallel and distributed algorithm for computing low-stretch spanning trees. Low-stretch spanning trees were introduced in a seminal paper by Alon et al. [7], where they were shown useful for the online k-server problem. The algorithm of [7] constructed spanning trees with average stretch $\exp(\sqrt{\log n \log\log n})$. In a subsequent work Bartal [50, 51] and Fakchraenphol et al. [128] showed that one can get logarithmic stretch if one allows the trees to use edges that are not present in the original graph. In [123] it was shown that the original problem of low-stretch spanning trees admits a solution with polylogarithmic stretch. That bound was later improved to a nearly-logarithmic bound in [1]. These constructions have important applications to the framework of spectral sparsification [285].

In the distributed setting the problem was studied in [55]. However, the latter algorithm relies on the computation of exact distances. Our approach, on the other hand, only relies on approximate distance computations that, unlike exact distances, can be computed with near-optimal parallel and distributed complexity [274]. Hence, we are able to present the first distributed and parallel algorithm for this problem that provides polylogarithmic stretch, polylogarithmic depth and near-linear work.

Theorem 9.2 (Deterministic Low-Stretch Spanning Tree)**.** *Let G be a weighted graph. Each edge e has moreover a nonnegative importance $\mu(e)$. There exists a deterministic parallel and distributed algorithm which outputs a spanning tree T of G such that*

$$\sum_{e=\{u,v\}\in E(G)} \mu(e) d_T(u,v) = \widetilde{O}\left(\sum_{e=\{u,v\}\in E(G)} \mu(e) d_G(u,v)\right). \tag{9.1}$$

The PRAM *variant of the algorithm has work $\widetilde{O}(m)$ and depth $\widetilde{O}(1)$. The* CONGEST *variant of the algorithm runs in $\widetilde{O}(\sqrt{n} + \mathrm{HopDiam}(G))$ rounds.*

Note that plugging in $\mu(e) := \mu'(e)/\ell(e)$ into (9.1) and using $d_G(u,v) \le \ell(e)$, we also get the following similar guarantee of

$$\sum_{e=\{u,v\}\in E(G)} \mu'(e) \cdot \frac{d_T(u,v)}{\ell(e)} = \widetilde{O}\left(\sum_{e=\{u,v\}\in E(G)} \mu'(e)\right)$$

The stretch is optimal up to polylogarithmic factors.

ℓ_1 **Embedding**: Embeddings of networks in low dimensional spaces like ℓ_1-space are a basic tool with a number of applications. For example, the parallel randomized approximate shortest path algorithm of [233] uses ℓ_1-embeddings as a crucial subroutine. By using our clustering results, we can use an approach similar to the one from [52] to obtain an efficient deterministic parallel and distributed algorithm for ℓ_1-embedding.

Theorem 9.3 (ℓ_1-Embedding)**.** *Let G be a weighted graph. There exists a deterministic parallel and distributed algorithm which computes an embedding in $\widetilde{O}(1)$-dimensional ℓ_1-space with distortion $\widetilde{O}(1)$. The* PRAM *variant of the algorithm has work $\widetilde{O}(m)$ and depth $\widetilde{O}(1)$. The* CONGEST *variant of the algorithm runs in $\widetilde{O}(\sqrt{n} + \text{HopDiam}(G))$ rounds.*

Other Applications: Since low-diameter clusterings are an important subroutine for numerous problems, there are many other more standard applications for problems like (h-hop) Steiner trees or Steiner forests, deterministic variants of tree embeddings, problems in network design, etc. [56, 193] We do not discuss these applications here due to space constraints. We also note that the distributed round complexities of our algorithms are almost-universally-optimal. We refer the interested reader to [171, 194] for more details regarding the notion of universal optimality.

9.1.1 Our Techniques and Contributions

We give a clean interface for various distributed clustering routines in weighted graphs that allows to give results in different models (distributed and parallel).

Simple Deterministic Strong-Diameter Network Decomposition in CONGEST :

In the previous section, we mentioned that the state-of-the-art strong-diameter network decomposition algorithm of [87] runs in $O(\log^{11} n)$ CONGEST rounds and produces clusters with diameter $D = O(\log^2 n)$.

Our first result improves upon their algorithm by giving an algorithm with the same guarantees running in $O(\log^5 n)$ CONGEST rounds.

Theorem 9.4. *There is a deterministic* CONGEST *algorithm computing a network decomposition with $C = O(\log n)$ clusterings such that each cluster has strong-diameter $O(\log^2 n)$. The algorithm runs in $O(\log^5 n)$* CONGEST *rounds.*

Note that the round complexity of our algorithm matches the complexity of the weak-diameter algorithm of [170]. This is because both our result and the result of [87] use the weak-diameter algorithm of [170] as a subroutine.

We prove Theorem 9.4 in Section 9.2 and the technical overview of our approach is deferred to Section 9.2.1. Here, we only note that on a high-level our algorithm can be seen as a derandomization of the randomized algorithm of [255]. That is, instead of clusters, the algorithm operates with nodes and assigns “head starts” to them in a careful manner.

Simple Blurry Ball Growing Procedure:

The blurry ball growing problem is defined as follows: given a set S and distance parameter D, we want to find a superset $S^{sup} \supseteq S$ such that the following holds. First, for any $v \in S^{sup}$ we have $d_{G[S^{sup}]}(S, v) \leq D$, that is, the set S “does not grow too much”. On the other hand, in the randomized variant of the problem we ask for each edge e to be cut by S^{sup} with probability $O(\ell(e)/D)$, while in the deterministic variant of the problem we ask for the total number of edges cut to be at most $O(\sum_{e \in E(G)} \ell(e)/D)$.

Here is a simple application of this problem: suppose that we want to solve the low-diameter clustering problem where each edge needs to be cut with probability $\ell(e)/D$ and clusters should have diameter $\widetilde{O}(D)$. Assume we can solve the separated clustering problem, that is, we can construct a clustering $\mathcal{C}$ such that the clusters are D-separated and their diameter is $\widetilde{O}(D)$. To solve the former problem, we can simply solve the blurry ball growing problem with $S = \bigcup_{C \in \mathcal{C}} C$ and $D_{blurry} = D/3$. This way, we "enlarge" the clusters of $\mathcal{C}$ only by a nonsignificant amount, while achieving the edge cutting guarantee.

The blurry ball growing problem was defined and its randomized variant was solved in [56, Theorem 3.1]. Since blurry ball growing is a useful subroutine in our main clustering result, we generalize their result by giving an efficient algorithm solving the deterministic variant. Furthermore, we believe that our approach to solving that problem is simpler: we require the approximate distance oracle to be $(1 + 1/\log n)$-approximate instead of $\left(1 + \left(\frac{\log\log n}{\log n}\right)^2\right)$-approximate.

Theorem 9.5. *Given a weighted graph G, a subset of its nodes S and a parameter $D > 0$, there is a deterministic algorithm computing a superset $S^{sup} \supseteq S$ such that $\max_{v \in S^{sup}} d_{G[S^{sup}]}(S, v) \leq D$, and moreover,*

$$\sum_{e \in E(G) \cap (S^{sup} \times (V(G) \setminus S^{sup}))} \ell(e) = O\left(\sum_{e \in E(G)} \ell(e)/D\right).$$

The algorithm uses $O(\log D)$ calls to an $(1 + 1/\log D)$-approximate distance oracle.

Our deterministic algorithm is a standard derandomization of the following simple randomized algorithm solving the randomized variant of the problem. The randomized algorithm is based on a simple binary search idea: in each step we flip a fair coin and decide whether or not we "enlarge" the current set S_i by adding to it all nodes of distance at most roughly $D/2^i$. We start with $S_0 = S$ and the final set $S_{\log_2 D} = S^{sup}$. Hence, we need $O(\log D)$ invocations of the approximate distance oracle. We prove a more general version of Theorem 9.5 in Section 9.3 and give more intuition about our approach in Section 9.3.1.

Main Contribution: A General Clustering Result:

We will now state a special case of our main clustering result. The clustering problem that we solve generalizes the already introduced low-diameter clustering problem that asks for a partition of the vertex set into clusters such that only a small amount of edges is cut. In our more general clustering problem we are also given a set of *terminals* $Q \subseteq V(G)$ as input. Moreover, we are given a parameter R such that Q is R-ruling. Each cluster of the final output clustering has to contain at least one terminal. Moreover, one of these terminals should $(1+\varepsilon)R$-rule its cluster.

We note that in order to get the classical low-diameter clustering with parameter D as an output of our general result, it suffices to set $Q = V(G), R = D$ and $\varepsilon = 1/2$.

A more general version of Theorem 9.1 is proven in Section 9.4. The intuition behind the algorithm is explained in Section 9.4.1. Here, we note that the algorithm combines the clustering idea of the algorithm from Theorem 9.4 and uses as a subroutine the blurry ball growing algorithm from Theorem 9.5.

Another corollary of our general clustering result is the following theorem.

Theorem 9.6. *[A corollary of Theorem 9.30] We are given an input weighted graph G, a distance parameter D and each node $v \in V(G)$ has a* preferred radius $r(v) > 0$.

There is a deterministic distributed algorithm constructing a partition $\mathcal{C}$ of G that splits $V(G)$ into two sets $V^{good} \sqcup V^{bad}$ such that

1. *Each cluster $C \in \mathcal{C}$ has diameter $\widetilde{O}(D)$.*
2. *For every node $v \in V^{good}$ such that v is in a cluster C we have $B_G(v, r(v)) \subseteq C$.*
3. *For the set V^{bad} of nodes we have*

$$\sum_{v \in V^{bad}} r(v) = \frac{1}{2D} \cdot \sum_{v \in V(G)} r(v).$$

The algorithm needs $\widetilde{O}(1)$ calls to an $(1 + 1/\operatorname{poly}\log n)$-approximate distance oracle.

One reason why we consider each node to have a preferred radius is that it allows us to deduce Theorem 9.1 from our general theorem by considering the subdivided graph where each edge is split by adding a node "in the middle of it", with a preferred radius of $\ell(e)$.

Let us now compare the clustering of Theorem 9.6 with the D-separated clustering that we already introduced. Recall that in the D-separated clustering problem, we ask for clusters with radius $\widetilde{O}(D)$ and require the clusters to be D-separated. Moreover, only half of the nodes should be unclustered.

In our clustering, we can choose $r(v) = D$ for all nodes $v \in V(G)$, we again get clusters of diameter $\widetilde{O}(D)$ and only half of the nodes are bad. The difference with the D-separated clustering is that we cluster all the nodes, but we require the good nodes to be "D-padded".

This is a slightly weaker guarantee then requiring the clusters to be D-separated: we can take any solution of the D-separated problem, and enlarge each cluster by adding all nodes that are at most $D/3$ away from it. We mark all original nodes of the clusters as good and all the new nodes as bad. Moreover, each remaining unclustered node forms its own cluster and is marked as bad. This way, we solve the special case of Theorem 9.6 with the padding parameter $D/3$. We do not know of an application of D-separated clustering where the slightly weaker D-padded clustering of Theorem 9.6 does not suffice. However, we also use a different technique to solve the D-separated problem.

Theorem 9.7. *We are given a weighted graph G and a separation parameter $D > 0$. There is a deterministic algorithm that outputs a clustering $\mathcal{C}$ of D-separated clusters of diameter $\widetilde{O}(D)$ such that at least $n/2$ nodes are clustered.*

The algorithm needs $\widetilde{O}(1)$ calls to an $(1 + 1/\operatorname{poly}\log(n))$-approximate distance oracle computing approximate shortest paths from a given set up to distance $\widetilde{O}(D)$.

The algorithm is based on the ideas of the weak-diameter network decomposition result of [276] and the strong-diameter network decomposition of [87]. Since shortest paths up to distance D can be computed in unweighted graphs by breadth first search, we get as a corollary that we can compute a separated strong-diameter network decomposition in unweighted graphs. No $\widetilde{O}(D)$-round deterministic CONGEST algorithm for separated strong-diameter network decomposition was known.

Corollary 9.8. *[D-separated strong-diameter network decomposition] We are given an unweighted graph G and a separation parameter $D > 0$. There is a deterministic algorithm that outputs $O(\log n)$ clusterings $\{\mathcal{C}_1, \ldots, \mathcal{C}_{O(\log n)}\}$ such that*

1. *Each node $u \in V(G)$ is contained in at least one clustering $\mathcal{C}_i$.*
2. *Each clustering $\mathcal{C}_i$ consists of D-separated clusters of diameter $\widetilde{O}(D)$.*

The algorithm needs $\widetilde{O}(D)$ CONGEST rounds.

9.1.2 Roadmap

The chapter is structured as follows. In Section 9.1.2 we define some basic notions and models that we work with in the chapter. Section 9.2 contains the proof of Theorem 9.4. We believe that an interested reader should understand Section 9.2 even after she skips Section 9.1.2. In Section 9.3, we prove a general version of Theorem 9.5 and the main clustering result that generalizes Theorem 9.1 is proven in Section 9.4.

Preliminaries

In this preliminary section, we first explain the terminology used in the chapter. Then, we review the notation we use to talk about clusterings and the distributed models we work with. Finally, we explain the language of distance oracles that we use throughout the chapter to make our result as independent on a particular choice of a distributed/parallel computational model as possible.

Basic Notation:

A weighted graph G is an unweighted graph together with a weight (or length) function ℓ. This function assigns each edge $e \in E(G)$ a polynomially bounded nonnegative weight $\ell(e) \geq 0$. We will assume that all lengths are polynomially bounded, i.e., $\ell(e) \leq n^C$ for some absolute constant C. This implies that each weight can be encoded by $O(\log n)$ bits. We denote by $d_G(u, v)$ the weight (i.e., length) of the shortest path between two nodes $u, v \in V(G)$. We sometimes drop the subscript when the graph is clear from context and write just $d(u, v)$. The distance function naturally extends to sets by $d(A, B) = \min_{a \in A, b \in B} d(a, b)$ and we also write $d(u, S)$ instead of $d(\{u\}, S)$.

We say that H is a subgraph of a weighted graph G and write $H \subseteq G$ if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and for every $e \in E(H)$, $\ell_H(e) = \ell_G(e)$. Given a weighted graph G, a weighted rooted (sub)forest in G is a forest F which is a subgraph of G. Moreover, each component of F contains a special node – a root – that defines a natural orientation of edges of F towards a unique root. By $d_F(u, v)$ we mean the distance in the unoriented graph F, i.e., d_F is a metric. For any $v \in V(F)$ we denote by $\mathrm{root}_F(v)$ the unique root node in $V(F)$ that lies in the same component of F as v. We also use the shorthand $d_F(v) = d_F(\mathrm{root}_F(v), v)$. A ball $B_G(u, r) \subseteq V(G)$ is a set of nodes consisting of those nodes $v \in V(G)$ with $d_G(u, v) \leq r$.

Weight, Radius and Delay Functions:

Sometimes we need nonnegative and polynomially bounded functions that assign each vertex or edge of a given graph such that their domain is the set $V(G)$, $E(G)$ or a subset. One should think of these functions as parameters of the nodes (edges) of the input graph in the sense that during the algorithms, each node u starts with an access to the value of these functions at u.

There are three functions that we need:

1. A function μ assigning each vertex v (edge e) of a given graph a *weight* $\mu(v)$ ($\mu(e)$); we use $\mu(U) := \sum_{u \in U} \mu(u)$.
2. A function r assigning each vertex v of a given graph a *preferred radius* $r(v)$.
3. A function del assigning a subset of nodes Q a *delay*; For a subset $Q' \subseteq Q$, we define $d_{\mathrm{del}}(Q', v) := \min_{q \in Q'} \mathrm{del}(q) + d(q, v)$.

Clustering Notation:

Next, we define the notation that is necessary for stating our clustering results.

Cluster A cluster C is simply a subset of nodes of $V(G)$. We use $\mathrm{diam}(C)$ to denote the diameter of a cluster C, i.e., the diameter of the graph $G[C]$. When we construct a cluster C, we are also often constructing a (small diameter) tree T_C with $V(T_C) = C$. In Section 9.2 we use a result from [170] that constructs so-called weak-diameter clusters. A weak-diameter cluster is a cluster C together with a (small diameter) tree T_C such that $V(T_C) \supseteq C$.

Padding A node $v \in C$ is r-padded in the cluster C of a graph G if it is the case that $B_G(v, r) \subseteq C$.

Separation Suppose we have two disjoint clusters C_1, C_2. We say that they are D-separated in G if $d_G(C_1, C_2) \geq D$.

Clustering and Partition A *clustering* $\mathcal{C}$ is a family of disjoint clusters. If the clustering covers all nodes of G, that is, if $\bigcup_{C \in \mathcal{C}} C = V(G)$, we refer to the clustering as a *partition*.

The *diameter* $\mathrm{diam}(\mathcal{C})$ of a clustering $\mathcal{C}$ is defined as $\mathrm{diam}(\mathcal{C}) = \max_{C \in \mathcal{C}} \mathrm{diam}(C)$. A clustering $\mathcal{C}$ is D-separated if every two clusters $C_1 \neq C_2 \in \mathcal{C}$ are D-separated.

Cover A *cover* $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_q\}$ is a collection of clusterings or partitions.

Computational Models

CONGEST Model [264]:

We are given an undirected graph G also called the "communication network". Its vertices are also called nodes and they are individual computational units, i.e., they have their own processor and private memory. Communication between the nodes occurs in synchronous rounds. In each round, each pair of nodes adjacent in G exchange an $b = O(\log n)$-bit message. Nodes perform arbitrary computation between rounds. Initially, nodes only know their unique $O(\log n)$-bit ID and the IDs of adjacent nodes in case of deterministic algorithms. In case of randomized algorithms, every node starts with a long enough random string containing independently sampled random bits. Each node also starts with a polynomial upper bound on the number of nodes, n.

Unless stated otherwise, we always think of G as a weighted graph, where the weights are provided in a distributed manner.

In all our results, in each round, each node v can run an algorithm whose PRAM work is at most $\widetilde{O}(\deg(v))$ and depth at most $\widetilde{O}(1)$. Note that this allows the model to compute e.g. simple aggregation operations of the messages received by the neighbors such as computing the minimum or the sum.

Oracle Definition:

Except of simple local communication and computation captured by the above CONGEST model, our algorithms can be stated in terms of simple primitives such as computing approximate shortest paths or aggregating some global information. To make our results more model-independent and more broadly applicable, we abstract these primitives away as calls to an oracle. We next define the oracles used in the chapter.

Definition 9.9 (Approximate Distance Oracle $\mathcal{O}^{Dist}_{\varepsilon,D}$). *This oracle is parameterized by a* distance *parameter $D > 0$ and a* precision *parameter $\varepsilon \geq 0$.*

The input to the oracle consists of three parts. First, a weighted graph $H \subseteq G$. Second, a subset $S \subseteq V(H)$. Third, for each node $s \in S$ a delay $del(s)$. If the third input is not specified, set $del(s) = 0$ for every $s \in S$.

The output is a weighted forest $F \subseteq H$ rooted at some subset $S' \subseteq S$. The output has to satisfy the following:

1. *For every $v \in V(F)$, $del(root_F(v)) + d_F(v) \leq (1+\varepsilon)d_{H,del}(S, v) \leq (1+\varepsilon)D$.*
2. *For every $v \in V(H)$, if $d_{H,del}(S, v) \leq D$, then $v \in V(F)$.*

Definition 9.10 (Forest Aggregation Oracle $\mathcal{O}^{Forest\text{-}Agg}_{D}$). *The input consists of two parts. First, a weighted and rooted forest $F \subseteq G$ with $d_F(v) \leq D$ for every $v \in V(F)$. Second, an integer value $x_v \in \{0, 1, \ldots, \mathrm{poly}(n)\}$ for every node $v \in V(F)$. The oracle can be used to compute a sum or a minimum. If we compute a sum, the oracle outputs for each node $v \in V(F)$ the two values $\sum_{v\in A(v)} x_v$ and $\sum_{v\in D(v)} x_v$, where $A(v)$ and $D(v)$ denote the set of ancestors and descendants of v in F, respectively. Computing the minimum is analogous.*

Definition 9.11 (Global Aggregation Oracle $\mathcal{O}^{Global\text{-}Agg}$). *The input consists of an integer value $x_v \in \{0, 1, \ldots, \mathrm{poly}(n)\}$ for every node $v \in V(G)$. The output of the oracle is $\sum_{v\in V(G)} x_v$.*

Whenever we say e.g. that "the algorithm runs in T steps, with each oracle call having distance parameter at most D and precision parameter ε", we mean that the algorithm runs in T CONGEST rounds, and in each CONGEST round the algorithm performs at most one oracle call. Moreover, when the oracle is parameterized by a distance parameter or/and a precision parameter, then the distance parameter is at most D and the precision parameter is at most ε .

Compilation to Distributed and Parallel Models:

The theorem below is a direct consequence of the deterministic approximate shortest path paper of [274]. They show that the approximate distance oracle with precision parameter $\varepsilon = 1/\mathrm{poly}\log(n)$ can be implemented in the bounds claimed in bullet points 1 to 4. We note that the results 2 to 4 follow from the theory of universal-optimality in the CONGEST model [299, 164]. The bullet point 5 follows from the fact that the distance oracle in unweighted graphs can be implemented by breadth first search.

Theorem 9.12. *Suppose that for a given problem there is an algorithm that runs in $T = \mathrm{poly}\log(n)$ steps, with each oracle call having precision parameter $\varepsilon = 1/\mathrm{poly}\log(n)$. Then, the problem can be solved in the following settings with the following bounds on the complexity.*

1. *In* PRAM *, there is a deterministic algorithm with $\widetilde{O}(m+n)$ work and* $\operatorname{poly}\log n$ *depth.*
2. *In* CONGEST *, there is a deterministic algorithm with $\widetilde{O}(\mathrm{HopDiam}(G)+\sqrt{n})$ rounds. [154]*
3. *In* CONGEST *, there is a deterministic algorithm for any minor-free graph family with $\widetilde{O}(\mathrm{HopDiam}(G))$ rounds (the hidden constants depend on the family). [155]*
4. *If* $\mathrm{ShortcutQuality}(G) \le n^{o(1)}$*, there is a randomized algorithm in the* CONGEST *model with $n^{o(1)}$ rounds. See [195] for the definition of* $\mathrm{ShortcutQuality}(G)$ *and the proof.*
5. *If only the distance oracle $\mathcal{O}^{Dist}_{\varepsilon,D}$ is used and the graph is unweighted, there is a deterministic* CONGEST *algorithm with $\widetilde{O}(D)$ rounds, even for $\varepsilon = 0$.*

9.2 Strong-Diameter Clustering in Polylogarithmic Number of Rounds

In this section, we present an algorithm clustering a constant fraction of the vertices into non-adjacent clusters of diameter $O(\log^2 n)$ in $O(\log^4 n)$ CONGEST rounds.

Theorem 9.13. *Consider an unweighted n-node graph G where each node has a unique $b = O(\log n)$-bit identifier. There is a deterministic algorithm computing a 2-separated $O(\log^2 n)$-diameter clustering $\mathcal{C}$ with $|\bigcup_{C\in\mathcal{C}} C| \ge n/3$ in $O(\log^4 n)$* CONGEST *rounds.*

Note that in the above theorem, 2-separated clustering is equivalent to positing that the clusters of $\mathcal{C}$ are not adjacent.

Our algorithm is quite simple and in some aspects similar to the deterministic distributed clustering algorithm of [276]. Let us explain the main difference. During their algorithm, one works with a set of clusters that expand or shrink and which progressively become more and more separated. In our algorithm, we instead focus on *potential cluster centers.* These centers preserve a “ruling property” that asserts that every node is close to some potential cluster center. This in turn implies that running a breadth first search from the set of potential cluster centers always results in a set of (not necessarily separated) small diameter clusters. This way, we make sure that the final clusters have small strong-diameter, whereas in the algorithm of [276] the final clusters have only small weak-diameter.

An important downside compared to their algorithm is that we rely on global coordination. To make everything work, we hence need to start by using the state-of-the-art algorithm for weak-diameter clustering. This allows us to use global coordination inside each weak-diameter cluster and run our algorithm in each such cluster in parallel. This is the reason why our algorithm needs $O(\log^4 n)$ CONGEST rounds. In fact, the main routine runs only in $O(\log^3 n)$ rounds, but first we need to run the fastest deterministic distributed algorithm for weak-diameter clustering from [170] that needs $O(\log^4 n)$ rounds which dominates the round complexity.

9.2.1 Intuition and Proof Sketch of the Main Algorithm

Our algorithm runs in b phases; one phase for each bit in the b-bit node identifiers. During each phase, up to $\frac{n}{3b}$ of the nodes are removed and declared as unclustered. Hence, at most $\frac{n}{3}$ nodes are declared as unclustered throughout the b phases, with all the remaining nodes being clustered.

We set $G_0 = G$ and define G_{i+1} as the graph one obtains from G_i by deleting all the nodes from G_i which are declared as unclustered during phase i. Besides removing nodes in each phase i, the algorithm works with a set of *potential cluster centers* Q_i. Initially, all the nodes are potential cluster centers, that is, $Q_0 = V(G)$. During each phase, some of the potential cluster centers stop being potential cluster centers. At the end, each potential cluster center will in fact be a cluster center. More precisely, each connected component of G_b contains exactly one potential cluster center and the diameter of each connected component of G_b is $O(\log^2 n)$.

For each $i \in \{0, 1, \dots, b\}$, the algorithm maintains two invariants. The *ruling invariant* states that each node in G_i has a distance of at most $6ib$ to the closest potential cluster center in Q_i. The *separation invariant* states that two potential cluster centers u and v can only be in the same connected component of G_i if the first i bits of their identifiers coincide. Note that this condition is trivially satisfied at the beginning for $i = 0$ and for $i = b$ it implies that each connected component contains at most one potential cluster center.

The goal of the i-th phase is to preserve the two invariants. To that end, we partition the potential cluster centers in Q_i based on the $(i+1)$-th bit of their identifiers into two sets Q_i^R and Q_i^B. In order to preserve the separation invariant, it suffices to separate the nodes in Q_i^R from the nodes in Q_i^B. One way to do so is as follows: Each node in G_i clusters itself to the closest potential cluster center in Q_i. In that way, each node is either part of a red cluster with a cluster center in Q_i^R or a blue cluster with a cluster center in Q_i^B. Now, removing all the nodes in blue clusters neighboring a red cluster would preserve both the separation invariant as well as the ruling invariant. However, the number of removed nodes might be too large.

In order to ensure that at most $\frac{n}{3b}$ nodes are deleted, we do the following: for each node v, let $\mathrm{diff}(v) = d_i(Q_i^R, v) - d_i(Q_i^B, v)$. We now define $K_j = \{v \in V_i \colon \mathrm{diff}(v) = j\}$ and let $j^* = \arg\min_{j \in \{0,2,4,\dots,6b-2\}} |K_j \cup K_{j+1}|$. Then, we declare all the nodes in $K_{j^*} \cup K_{j^*+1}$ as unclustered. Moreover, each blue potential cluster center v with $\mathrm{diff}(v) < j^* - 1$ stops being a potential cluster center.

We remark that the described algorithm cannot efficiently be implemented in the distributed CONGEST model. The reason is that deciding whether a given index j is good or not requires global coordination. In particular, the communication primitive we need can be described as follows: each node v is assigned $O(b)$ numbers $\{x_{v,j} \in \{0,1\} \colon j \in \{0,2,4,\ldots,6b-2\}\}$ with $x_{v,j} = 1$ if v would be removed if $j = j^*$ and 0 otherwise. Now, each node has to learn $\sum_{v \in V(G)} x_{v,j}$ for each $j \in \{0,2,4,\ldots,6b-2\}$. We denote by $\mathcal{O}^{count}$ the oracle for this communication primitive. By formalizing the high-level overview, we then obtain the following theorem.

Theorem 9.14. *Consider an unweighted n-node graph G where each node has a unique b-bit identifier. There is a deterministic algorithm computing a 2-separated $O(b^2)$-diameter clustering $\mathcal{C}$ with $|\bigcup_{C \in \mathcal{C}} C| \geq (2/3)n$ in $O(b^3)$ CONGEST rounds and performing $O(b)$ oracle calls to $\mathcal{O}^{count}$.*

We note that the theorem does not assume $b = O(\log n)$.

Before giving a formal proof of Theorem 9.14 in Section 9.2.3, we first show formally how one can use it to proof Theorem 9.13.

9.2.2 Anylysis of the Strong Clustering Algorithm

Proof of Theorem 9.13. The algorithm starts by computing a clustering $\mathcal{C}^{weak} = \{C_1^{weak}, C_2^{weak}, \ldots, C_N^{weak}\}$ with weak-diameter $O(\log^2 n)$ such that $\mathcal{C}^{weak}$ clusters at least half of the nodes. This can be computed in $O(\log^4 n)$ CONGEST rounds by invoking the following theorem from [170].

Theorem 9.15 (Restatement of Theorem 2.2 in [170])**.** *Consider an arbitrary n-node graph G where each node has a unique $b = O(\log n)$-bit identifier. There is a deterministic algorithm that in $O(\log^4 n)$ rounds of the CONGEST model computes a 2-separated clustering $\mathcal{C}$ such that $|\bigcup_{C \in \mathcal{C}} C| \geq n/2$. For each cluster C, the algorithm returns a tree T_C with diameter $O(\log^2 n)$ such that $C \subseteq V(T_C)$. Each vertex in G is in $O(\log n)$ such trees.*

Now, for each $i \in [N]$, let $\mathcal{C}_i^{strong}$ denote the clustering one obtains by invoking Theorem 9.14 with input graph $G[C_i^{weak}]$. Then, the algorithm returns the clustering $\mathcal{C}^{strong} = \bigcup_{i=1}^{n} \mathcal{C}_i^{strong}$.
We first show that $\mathcal{C}^{strong}$ is a 2-separated clustering with diameter $O(\log^2 n)$ clustering at least $(1/3)n$ nodes.

First, the clustering $\mathcal{C}^{strong}$ is 2-separated: This directly follows from the fact that $\mathcal{C}^{weak}$ is 2-separated and for $i \in [N]$, $\mathcal{C}_i^{strong}$ is 2-separated. Moreover, the clustering has diameter $O(\log^2 n)$: This follows from the fact that each cluster in $\mathcal{C}_i^{weak}$ has diameter $O(b^2)$ and $b = O(\log n)$.

It remains to show that $|\bigcup_{C \in \mathcal{C}^{strong}} C| \geq n/3$. We have

$$|\bigcup_{C \in \mathcal{C}^{strong}} C| = \sum_{i=1}^{N} |\bigcup_{C \in \mathcal{C}_i^{strong}} C| \geq \sum_{i=1}^{N} \frac{2|C_i^{weak}|}{3} = \frac{2}{3}|\bigcup_{C \in \mathcal{C}^{weak}} C|$$

$$\geq \frac{2}{3}\frac{1}{2}n = \frac{1}{3}n$$

where the first inequality follows from the guarantees of Theorem 9.14 and the second follows from the guarantees of Theorem 9.15.

Finally, we discuss an efficient $\mathsf{CONGEST}$ implementation of the algorithm: For every $i \in [N]$, we need to show that we can compute $\mathcal{C}_i^{strong}$ in $O(\log^3 n)$ $\mathsf{CONGEST}$ rounds in each cluster $C_i^{weak} \in \mathcal{C}^{weak}$. Moreover, the communication capacity in each round is limited: For the computation inside C_i^{weak}, we can use the full capacity of $O(\log n)$ bits along edges contained in $G[C_i^{weak}]$, but only a single bit for edges contained in the tree $T_{C_i^{weak}}$, and no communication for all other edges. The reason why we have the capacity of one bit per edge of $T_{C_i^{weak}}$ is that by Theorem 9.15, each node of G and hence each edge of G is contained in $O(\log n)$ different trees T_C, hence by assuming without loss of generality that b is large enough, each edge can allocate one bit per tree it is in.

According to Theorem 9.14, for $i \in [N]$, we can compute $\mathcal{C}_i^{strong}$ in $O(b^3) = O(\log^3 n)$ $\mathsf{CONGEST}$ rounds together with performing $O(b) = O(\log n)$ oracle calls to $\mathcal{O}^{count}$ in $G[C_i^{weak}]$. The $\mathsf{CONGEST}$ rounds use only edges of $G[C_i^{weak}]$, so we only need to discuss the implementation of the calls to $\mathcal{O}^{count}$. To that end, we will use the following variant of [170, Lemma 5.1]. [1]

Lemma 9.16 (A variant of the "pipelining" Lemma 5.1 from [170])**.** *Consider the following problem. Let T be a rooted tree of depth d. Each node u has k m-bit numbers $x_u^1, x_u^2, \dots, x_u^k$. In one round of communication, each node can send a b-bit message, $b \leq m$, to all its neighbors in T. There is a protocol such that in $O(d + km/b)$ message-passing rounds on T performs the following operations:*

1. *Broadcast: the root of T, r, sends $x_r^1, \dots, x_r^k$ to all nodes in T.*
2. *Sum: The root r computes the value of $\sum_{u\in T} x_u^i \mod 2^{O(m)}$ for every $1 \leq i \leq k$.*

In our case, to implement $\mathcal{O}^{count}$ we first use the sum operation and afterwards the broadcast operation from the statement of Theorem 9.16 on the tree $T_{C_i^{weak}}$. Note that we use the following parameters: $d_{9.16} = O(\log^2 n)$ since this is the diameter of $T_{C_i^{weak}}$ by Theorem 9.15; $k_{9.16} = O(\log n)$ since we need to aggregate $O(\log n)$ different sums; $m_{9.16} = O(\log n)$ as this is the size of the messages we are broadcasting; $b_{9.16} = 1$ as this is the capacity of the channel. Therefore, the oracle $\mathcal{O}^{count}$ is implemented in $O(d_{9.16} + k_{9.16}m_{9.16}/b_{9.16}) = O(\log^2 n)$ rounds. We need to call it $O(\log n)$ times, hence the overall round complexity is $O(\log^3 n)$, as desired.

$\square$

[1]The Lemma 5.1 in [170] proves this result only for $k = 1$, but the generalization for bigger k is straightforward.

9.2.3 Proof of the Clustering Theorem

In this section, we formalize the proof sketch given in Section 9.2.1.

Proof of Theorem 9.14. The algorithm computes two sequences $V_0 := V(G) \supseteq V_1 \supseteq \ldots \supseteq V_b$ and $Q_0 := V(G) \supseteq Q_1 \supseteq \ldots \supseteq Q_b$. For $i \in \{0, 1, \ldots, b\}$, we define $G_i = G[V_i]$ and $d_i = d_{G_i}$. Besides $Q_i \subseteq V_i$, the following three invariants will be satisfied:

1. Separation Invariant: Let $u, v \in Q_i$ be two nodes that are contained in the same connected component in G_i. Then, the first i bits of the identifiers of u and v agree.
2. Ruling Invariant: For every node $v \in V_i$, $d_i(Q_i, v) \leq 6ib$.
3. Deletion Invariant: We have $|V_i| \geq n - \frac{in}{3b}$.

It is easy to verify that setting $V_0 = Q_0 = V(G)$ results in the three invariants being satisfied for $i = 0$. For $i = b$, the separation invariant implies that every connected component in G_b contains at most one vertex in Q_b. Together with the ruling invariant, this implies that the diameter of every connected component in G_b is $O(b^2)$. Moreover, the deletion invariant states that $|V_b| \geq n - \frac{bn}{3b} = (2/3)n$. Hence, the connected components of G_b define a 2-separated clustering in G with diameter $O(b^2)$ that clusters at least $(2/3)n$ of the vertices, as desired.

Let $i \in \{0, 1, \ldots, b-1\}$. It remains to describe how to compute (V_{i+1}, Q_{i+1}) given (V_i, Q_i) while preserving the three invariants.

Our algorithm makes sure that the following three properties are satisfied. First, let $u, v \in Q_i$ be two arbitrary nodes that are contained in the same connected component in G_i and whose identifiers disagree on the $(i+1)$-th bit. Then, at least one of them is not contained in Q_{i+1} or u and v end up in different connected components in G_{i+1}. Second, for every node $v \in V_{i+1}$, $d_{i+1}(Q_{i+1}, v) \leq d_i(Q_i, v) + 6b$. Third, $|V_i \setminus V_{i+1}| \leq \frac{n}{3b}$, i.e., the algorithm 'deletes' at most $\frac{n}{3b}$ many nodes.

Note that satisfying these three properties indeed suffices to preserve the invariants. Let $Q_i = Q_i^B \sqcup Q_i^R$ with Q_i^B containing all the nodes in Q_i whose $(i+1)$-th bit in their identifier is 0. We compute (V_{i+1}, Q_{i+1}) from (V_i, Q_i) as follows:

1. For $j \in \{0, 1, \ldots, 6b-1\}$, let $K_i^j = \{u \in V_i \colon \mathrm{diff}_i(u) = j\}$ with $\mathrm{diff}_i(u) = d_i(Q_i^R, u) - d_i(Q_i^B, u)$
2. $j^* = \arg\min_{j \in \{0,2,4,\ldots,6b-2\}} |K_i^j \cup K_i^{j+1}|$
3. $V_{i+1} = V_i \setminus (K_i^{j^*} \cup K_i^{j^*+1})$
4. $Q_{i+1} = Q_i \setminus (\bigcup_{j=0}^{j^*+1} K_i^j)$

We first show that computing (Q_{i+1}, V_{i+1}) in this way indeed satisfies the three properties stated above. Afterwards, we show that we can compute (Q_{i+1}, V_{i+1}) given (Q_i, V_i) in $O(b^2)$ CONGEST rounds and using the oracle $\mathcal{O}^{count}$ once. It directly follows from the pigeonhole principle that $|K_i^{j^*} \cup K_i^{j^*+1}| \le \frac{n}{3b}$ and therefore $|V_i \setminus V_{i+1}| \le \frac{n}{3b}$. Hence, it remains to verify the other two properties.

Claim 9.17. *Let $u, v \in Q_i$ be two arbitrary nodes that are contained in the same connected component in G_i and whose identifiers disagree on the $(i+1)$-th bit. Then, either at least one of them is not contained in Q_{i+1} or u and v end up in different connected components in G_{i+1}.*

Proof. We assume without loss of generality that $u \in Q_i^R$ and $v \in Q_i^B$. Furthermore, assume that $u, v \in Q_{i+1}$. We need to show that u and v are in different connected components in G_{i+1}. To that end, consider an arbitrary u-v-path $\langle u = w_1, w_2, \ldots, v = w_k \rangle$ in G_i. From the definition of diff_i and the fact that $u \in Q_i^R$ and $v \in Q_i^B \cap Q_{i+1}$, it follows that $\mathrm{diff}_i(u) < 0$ and $\mathrm{diff}_i(v) > j^* + 1$. Together with the fact that $|\mathrm{diff}_i(w_\ell) - \mathrm{diff}_i(w_{\ell+1})| \le 2$ for every $\ell \in [k-1]$, we get that there exists an $\ell \in \{2, 3, \ldots, k-1\}$ with $\mathrm{diff}_i(w_\ell) \in \{j^*, j^*+1\}$. For this ℓ, $w_\ell \notin V_{i+1}$ and therefore the u-v-path is not fully contained in G_{i+1}. Since we considered an arbitrary u-v-path, this implies that u and v are in different connected components in G_{i+1}, as desired. □

Claim 9.18. *For every $u \in V_{i+1}$, $d_{i+1}(Q_{i+1}, u) \le d_i(Q_i, u) + 6b$.*

Proof. Consider any $u \in V_{i+1}$ and recall that $\mathrm{diff}_i(u) = d_i(Q_i^R, u) - d_i(Q_i^B, u)$. As $u \notin K_i^{j^*} \cup K_i^{j^*+1}$, either $\mathrm{diff}_i(u) < j^*$ or $\mathrm{diff}_i(u) > j^* + 1$.

1. $\mathrm{diff}_i(u) < j^*$: Consider a shortest path from Q_i^R to u. Note that any node v on this path also satisfies $\mathrm{diff}_i(v) < j^*$.

   Therefore, the path is fully contained in G_{i+1}. Moreover, $Q_i^R \subseteq Q_{i+1}$ and therefore

   $$\begin{aligned} &d_{i+1}(Q_{i+1}, u) \le d_i(Q_i^R, u) \\ &\le d_i(Q_i, u) + \max(0, \mathrm{diff}_i(u)) \le d_i(Q_i, u) + 6b, \end{aligned}$$

   as needed.

2. $\mathrm{diff}_i(u) > j^* + 1$: Consider a shortest path from Q_i^B to u. Note that any node v on this path also satisfies $\mathrm{diff}_i(v) > j^* + 1$. In particular, the path is fully contained in G_{i+1}. Also, the start vertex $v' \in Q_i^B$ of the path satisfies $\mathrm{diff}_i(v') > j^* + 1$ and thus it is contained in Q_{i+1}. Hence,

   $$d_{i+1}(Q_{i+1}, u) \le d_i(Q_i^B, u) = d_i(Q_i, u) \le d_i(Q_i, u) + 6b,$$

   as needed.

□

First, each node v computes the two values $\min(d_i(Q_i^{\mathcal{R}}, v), 6(i+1)b)$ and $\min(d_i(Q_i^{\mathcal{B}}, v), 6(i+1)b)$. This can be done in $O(b^2)$ CONGEST rounds by computing a BFS forest from both $Q_i^{\mathcal{R}}$ and $Q_i^{\mathcal{B}}$ up to distance $6(i+1)b$. As $d_i(Q_i, v) \le 6ib$, it holds for each

$j \in \{0,1,\dots,6b-1\}$ that $\mathrm{diff}_i(u) := d_i(Q_i^{\mathcal{R}}, v) - d_i(Q_i^{\mathcal{B}}, v) = j$ if and only if $\min(d_i(Q_i^{\mathcal{R}}, v), 6(i+1)b) - \min(d_i(Q_i^{\mathcal{B}}, v), 6(i+1)b) = j$. Thus, a node can decide with no further communication whether it is contained in K_i^j. Now, one can use the oracle $\mathcal{O}^{count}$ to compute j^*. Given j^*, each node can decide whether it is contained in V_{i+1} and Q_{i+1}, as needed.

Hence, the overall algorithm runs in $O(b^3)$ CONGEST rounds and invokes the oracle $\mathcal{O}^{count}$ $O(b)$ times.

□

9.3 Blurry Ball Growing

The blurry ball growing problem asks for the following: in its simplest variant (randomized, edge-cutting), we are given a set S and a distance parameter D. The goal is to construct a superset S^{sup} of S with $S^{sup} \subseteq B_G(S, D)$ such that every edge e of length $\ell(e)$ is "cut" by S^{sup} (that is, neither contained in S^{sup}, nor in $V(G) \setminus S^{sup}$) with probability $O(\ell(e)/D)$.

This section is dedicated to prove Theorem 9.19 that generalizes Theorem 9.5 that we restate here for convenience.

Theorem 9.5. *Given a weighted graph G, a subset of its nodes S and a parameter $D > 0$, there is a deterministic algorithm computing a superset $S^{sup} \supseteq S$ such that $\max_{v \in S^{sup}} d_{G[S^{sup}]}(S, v) \leq D$, and moreover,*

$$\sum_{e \in E(G) \cap (S^{sup} \times (V(G) \setminus S^{sup}))} \ell(e) = O\left(\sum_{e \in E(G)} \ell(e)/D\right).$$

The algorithm uses $O(\log D)$ calls to an $(1 + 1/\log D)$-approximate distance oracle.

First, in Section 9.3.1, we sketch a proof for the randomized edge-cutting variant of the problem. The main result, Theorem 9.19, is proven in Section 9.3.2. Finally, in Section 9.3.3 we derive simple corollaries of Theorem 9.19 used later in the paper.

9.3.1 Intuition and Proof Sketch

We will sketch a proof of the randomized variant of Theorem 9.5 (change the guarantee on the sum of the lengths of edges cut to the individual guarantee that each edge e is cut with probability $O(\ell(e)/D)$). First, note that it is easy to solve the blurry ball growing problem using an exact distance oracle: one can simply pick a number $\overline{D} \in [0, D)$ uniformly at random and define $S^{sup} := \{u : d(S, u) \leq \overline{D}\}$. From now on, let $e = \{u, v\}$ be an arbitrary edge with $d_u \leq d_v$ for $d_u := d(S, u)$ and $d_v := d(S, v)$. Choosing $\overline{D}$ as above, we indeed have

$$\mathrm{P}(e \text{ is cut by } S^{sup}) = \mathrm{P}(\overline{D} \in [d_u, d_v))$$

$$= \frac{|[0, D) \cap [d_u, d_v)|}{D} \leq \ell(e)/D,$$

as needed.

What happens if we only have access to a $(1+\varepsilon)$-approximate distance oracle, i.e., if we define $S^{sup} = \{u \colon \widetilde{d}(S, u) \leq \overline{D}\}$ with $\widetilde{d}$ being $(1+\varepsilon)$-approximate? The calculation above would only give

$$\begin{aligned} \mathrm{P}(e \text{ is cut by } S^{sup}) &= \mathrm{P}(\overline{D} \in [\widetilde{d}(S,u), \widetilde{d}(S,v))) \\ &\leq \mathrm{P}(\overline{D} \in [d_u, d_v + \varepsilon D)) \leq \ell(e)/D + \varepsilon. \end{aligned}$$

This bound is only sufficient for edges of length $\Omega(\varepsilon D)$.

To remedy this problem, let us consider the algorithm ExactBlur given below. ExactBlur only performs a *binary* decision in each of the $O(\log D)$ recursion levels. This allows us later to straightforwardly generalise it to the more complicated approximate and deterministic setting.

Algorithm 1 Simple Randomized Blurry Ball Growing with Exact Distances

```
Procedure ExactBlur($S, D$)
if $D \leq 1$ then
    return $S$
else
    $S^{\mathrm{big}} = \{u : d(S, u) \leq D/2\}$
    if (fair coin comes up heads) then
        return ExactBlur($S, D/2$)
    else
        return ExactBlur($S^{\mathrm{big}}, D/2$)
    end if
end if
```

If all of the edges of G had length 1 and D was a power of two, the algorithm ExactBlur would actually be the same as the simple uniformly sampling algorithm discussed above. In that case, it would correspond to sampling the value of $\overline{D}$ bit by bit, starting with the most significant bit. However, in general the two procedures are somewhat different. Assume that $u \in S$, u is the only neighbor of some $v \notin S$ and $\ell(\{u, v\}) = 2D/3$. With probability 1, $v \notin \mathrm{ExactBlur}(S, D)$. That is, the probability of $\{u, v\}$ being cut is $1 > \ell(\{u, v\})/D$.

However, we will now (informally) prove that Algorithm 1 nevertheless satisfies $P(e \text{ is cut}) = O(\ell(e)/D)$.

Proof (informal). Let p be the probability of e being cut, p_1 the probability of e being cut provided that the coin comes up heads (we decide not to grow) and p_2 the probability that e is cut if the coin comes up tails; we have $p = (p_1 + p_2)/2$.

Recall that we want to prove (by induction) that $p \leq C\ell(e)/D$ for some $C > 0$. In particular, we are going to show that

$$p \leq 10(1 + \mathbf{1}_{middle}(e))\ell(e)/D. \tag{9.2}$$

Here, $\mathbf{1}_{middle}$ is the indicator of whether $0 < d_u \leq d_v < D - 1$, i.e., u is not in S and v is sufficiently close to S.

To prove the bound (9.2), first consider the case $d_u, d_v < D/2$. We have, by induction, that $p_1 \leq 10(1 + \mathbf{1}_{middle}(e))\ell(e)/(D/2)$, while $p_2 = 0$. Here we are using the fact that if $\mathbf{1}_{middle}(e) = 1$ in the recursive call, it is also certainly equal to one now. We get

$$p = p_1/2 + p_2/2 = p_1/2 \leq 10(1 + \mathbf{1}_{middle}(e))\ell(e)/D,$$

as needed. An analogous argument works if $d_u, d_v \geq D/2$.

It remains to analyze the case $d_u < D/2 \leq d_v$. First, note that we can assume that $\ell(e) \leq D/10$ and therefore $\mathbf{1}_{middle}(e) = 1$. Moreover, in all of the subsequent recursive calls it will be the case that either $u \in S_{rec}$, or, on the other hand, $d(S_{rec}, v) \geq D_{rec}$. Thus, $\mathbf{1}_{middle}(e) = 0$ during all of the subsequent recursive calls.

The fact that currently $\mathbf{1}_{middle}(e) = 1$ but $\mathbf{1}_{middle}(e) = 0$ in the recursive call allows us to conclude that

$$p = \frac{p_1 + p_2}{2} \leq \frac{10 \cdot 1 \cdot \ell(e)/(D/2) + 10 \cdot 1 \cdot \ell(e)/(D/2)}{2} \tag{9.3}$$

$$= 10(1+1)\ell(e)/D = 10(1 + \mathbf{1}_{middle})\ell(e)/D, \tag{9.4}$$

as needed. □

Our main result Theorem 9.19 is a generalization of Algorithm 1 and the above analysis. First, the analysis can also be made to work with approximate distances. One difference is that we multiply D by $(1-\varepsilon)/2$ and not by $1/2$ in the recursive call, to account for the errors we make when computing the set S^{big}. By setting $\varepsilon = O(1/\log(D))$, the errors accumulated over the $O(\log D)$ iterations do not explode.

Second, we solve a deterministic variant of the problem where the objective is to minimize the (weighted) sum of edges that are cut. We achieve this by derandomizing the random choices in Algorithm 1. For that, it comes in handy that the algorithm samples just one bit in every iteration: in essence, the basic idea of the deterministic variant of the algorithm is that it computes in every step which choice makes the expected number of edges being cut smaller.

9.3.2 General Result

The main result of this section is Theorem 9.19. It solves the general blurry ball growing problem discussed in Section 9.1. We now define this general version of the problem. In particular, we generalize the guarantee for edges to guarantees for input balls: every node v wants the ball $B(v, r(v))$ to end up fully in S^{sup} or $V(G) \setminus S^{sup}$. Our algorithm outputs a set V^{bad} which contains all the nodes for which this condition fails (and potentially even nodes for which the condition is satisfied). In the randomized version, we show that each node v is contained in V^{bad} with probability $O(r(v)/D)$. In the deterministic version, we show that the (weighted) number of nodes in V^{bad} is sufficiently small. We note that explicitly outputting a set of "bad" nodes is needed for the applications later on and we anyways have to track certain quantities (e.g. whether a node can potentially become bad) to derandomize the algorithm.

Theorem 9.19 (Deterministic And Randomized Blurry Ball Growing Problem)**.** *Consider the following problem on a weighted input graph G. The input consists of:*

1. *A set $S \subseteq V(G)$.*
2. *Each node $v \in V(G)$ has a preferred radius $r(v)$.*
3. *In the deterministic version, each node $v \in V(G)$ additionally has a weight $\mu(v)$.*
4. *A distance parameter $D > 0$.*

The output is a set S^{sup} with $S \subseteq S^{sup} \subseteq V(G)$ together with a set $V^{bad} \subseteq V(G)$ such that

1. *for every $v \in S^{sup}$, $d_{G[S^{sup}]}(S, v) \leq D$,*
2. *for every $v \in V^{good} := V(G) \setminus V^{bad}$, $B(v, r(v)) \subseteq S^{sup}$ or $B(v, r(v)) \subseteq V(G) \setminus S^{sup}$,*
3. *in the deterministic version, $\mu(V^{bad}) = O(\sum_{v \in V(G)} \mu(v)\ r(v)/D)$*
4. *and in the randomized version, $Pr[v \in V^{bad}] = O(r(v)/D)$ for every $v \in V(G)$.*

There is an algorithm which returns a pair (S^{sup}, V^{bad}) satisfying the above properties in $O(\log(D) + 1)$ steps. The algorithm performs all oracle calls with precision parameter $\varepsilon = \frac{1}{\log(n)}$ and distance parameter no larger than D.

Proof of Theorem 9.19: Let us first give some intuition about Algorithm 2. The set S^{big} corresponds to the set of the same name in Algorithm 1. The trees $T_{S^{big}}, T_{V \setminus S^{big}}$ give us, informally speaking, the approximate distance from the cut $S^{big} \times (V(G) \setminus S^{big})$. The set V^{unsafe} contains all the nodes which can potentially be cut by the set S^{sup} returned at the end. At the beginning we set $V^{\text{unsafe}} = V(G)$, while in the leaf of the recursion we return $V^{\text{bad}} = V^{\text{unsafe}}$. We postpone the intuitive discussion about the set V^{middle}. However, note that the randomized version "ignores" the set V^{middle}. It is only necessary as an input to the deterministic algorithm. However, we still use the set V^{middle} to analyze the randomized version. Finally, the potential Φ_1 (Φ_2) in the deterministic version of Algorithm 2 can be seen as a pessimistic estimator for the expected number of nodes (according to their weight) which will be labeled as bad at the end of the algorithm, i.e., which are contained in V^{bad}, if the coin comes up heads (tails).

Algorithm 2 The Blurry Ball Growing Algorithm

```
**Procedure** Blur($S, D, V^{\text{unsafe}}, V^{\text{middle}}$)
Works with an arbitrary precision parameter $\varepsilon \in [0, 0.1]$.
**if** $D \leq 1$ **then**
    **return** $(S, V^{\text{unsafe}})$
**else**
    $T_S \leftarrow \mathcal{O}^{Dist}_{\varepsilon, D/2}(S)$
    $S^{\text{big}} = V(T_S)$
    $T_{S^{\text{big}}} \leftarrow \mathcal{O}^{Dist}_{\varepsilon, D}(S^{\text{big}})$
    $T_{V \setminus S^{\text{big}}} \leftarrow \mathcal{O}^{Dist}_{\varepsilon, D}(V(G) \setminus S^{\text{big}})$
    $V_1^{\text{unsafe}} = V^{\text{unsafe}} \cap \left( \{v \in V(T_{S^{\text{big}}}) \colon d_{T_{S^{\text{big}}}}(v) \leq (1+\varepsilon) r(v)\} \cup \{v \in V(G) \colon r(v) > D/10\} \right)$
    $V_2^{\text{unsafe}} = V^{\text{unsafe}} \cap \left( \{v \in V(T_{V \setminus S^{\text{big}}}) \colon d_{T_{V \setminus S^{\text{big}}}}(v) \leq (1+\varepsilon) r(v)\} \cup \{v \in V(G) \colon r(v) > D/10\} \right)$
    $V'^{\text{middle}} = V^{\text{middle}} \setminus (V_1^{\text{unsafe}} \cap V_2^{\text{unsafe}} \cap \{v \in V(G) \colon r(v) \leq D/10\})$
    $\forall i \in \{1, 2\} \colon \Phi_i = \sum_{v \in V_i^{\text{unsafe}} \colon r(v) \leq D/10} (1 + \mathbf{1}_{V'^{\text{middle}}}(v)) \mu(v) r(v)$
    **if** (*randomized* and fair coin comes up heads) or (*deterministic* and $\Phi_1 \leq \Phi_2$) **then**
        **return** Blur($S, (1-\varepsilon)D/2, V_1^{\text{unsafe}}, V'^{\text{middle}}$)
    **else**
        **return** Blur($S^{\text{big}}, (1-\varepsilon)D/2, V_2^{\text{unsafe}}, V'^{\text{middle}}$)
    **end if**
**end if**
```

Proof. We invoke the deterministic/randomized recursive procedure of Algorithm 2 with precision parameter $\varepsilon = \frac{1}{\log(n)}$. We let $(S^{sup}, V^{bad}) = Blur(S, D, \{v \in V(G) \colon r(v) > 0\}, V(G))$.

We need to prove that the four properties in the theorem statement are satisfied. The proof is structured as follows. Theorem 9.20 implies that the first property is satisfied. Theorem 9.21 implies that the second property is satisfied. In the randomized version, Theorem 9.25 gives that $Pr[v \in V^{bad}] \leq \frac{20r(v)}{D(1-\varepsilon)^{\max(0,\log(2D))}}$ for every $v \in V(G)$. For $x \in [0, 0.5]$, it holds that $1 - x \geq e^{-2x}$. Hence, for n being larger than a fixed constant, we have

$$Pr[v \in V^{bad}] \leq \frac{20r(v)}{D(1-\varepsilon)^{\max(0,\log(2D))}}$$
$$\leq \frac{20r(v)}{De^{-2\frac{\max(0,\log(2D))}{\log(n)}}} = O(r(v)/D).$$

In the deterministic version, Theorem 9.26 gives that

$$\mu(V^{\mathrm{bad}}) \leq \frac{10}{D(1-\varepsilon)^{\max(0,\log_2(2D))}} \sum_{v \in V(G)} \mu(v)r(v)$$
$$= O\left(\sum_{v \in V(G)} \mu(v)r(v)/D\right),$$

where we again assume that n is a large enough constant. The recursion depth of the Blur-procedure is $O(\log D)$. Hence, it is easy to see that running the procedure takes $O(\log(D) + 1)$ steps and all oracle calls are performed with precision parameter $\varepsilon = \frac{1}{\log(n)}$ and distance parameter no larger than D. This finishes the proof, modulo proving Theorems 9.20, 9.21, 9.25 and 9.26, which we will do next. □

Claim 9.20. *Let $(S^{sup}, .) = Blur(S, D, ., .)$. For every $v \in S^{sup}$, we have $d_{G[S^{sup}]}(S, v) \leq D$.*

Proof. We prove the statement by induction on the recursion depth. For the base case $D \leq 1$, we have $S^{sup} = S$ and therefore the statement trivially holds. Next, consider the case $D > 1$. We either have $(S^{sup}, .) = \mathrm{Blur}(S, (1 - \varepsilon)D/2, ., .)$ or $(S^{sup}, .) = \mathrm{Blur}(S^{\mathrm{big}}, (1 - \varepsilon)D/2, ., .)$. In the first case, the induction hypothesis gives that for any $v \in S^{sup}$ we have $d_{G[S^{sup}]}(S, v) \leq (1 - \varepsilon)D/2 \leq D$, as desired. In the second case, the induction hypothesis states that there exists a vertex $u \in S^{\mathrm{big}}$ with $d_{G[S^{sup}]}(u, v) \leq (1 - \varepsilon)D/2$. However, from the way S^{big} is defined, properties of $\mathcal{O}^{Dist}$, and the fact that $S^{\text{big}} \subseteq S^{sup}$, it follows that $d_{G[S^{sup}]}(S,u) \leq (1+\varepsilon)D/2$. Hence, by using the triangle inequality, we obtain

$$\begin{aligned} d_{G[S^{sup}]}(S,v) &\leq d_{G[S^{sup}]}(S,u) + d_{G[S^{sup}]}(u,v) \\ &\leq (1+\varepsilon)D/2 + (1-\varepsilon)D/2 \leq D, \end{aligned}$$

which finishes the proof. □

Claim 9.21. *Let $(S^{sup}, V^{bad}) = Blur(S, D, V^{unsafe}, .)$ for some $V^{unsafe} \subseteq V(G)$. For every $v \in V^{unsafe}$, if $B(v, r(v)) \cap S^{sup} \neq \emptyset$ and $B(v,r(v)) \setminus S^{sup} \neq \emptyset$, then $v \in V^{bad}$.*

Proof. We prove the statement by induction on the recursion depth. For the base case $D \leq 1$, we have $V^{\text{bad}} = V^{\text{unsafe}}$ and therefore the statement trivially holds. Next, consider the case $D > 1$. Let $v \in V^{\text{unsafe}}$ and assume that $B(v,r(v)) \cap S^{sup} \neq \emptyset$ and $B(v,r(v)) \setminus S^{sup} \neq \emptyset$. We have to show that this implies $v \in V^{\text{bad}}$.

We either have $(S^{sup}, .) = \text{Blur}(S, (1-\varepsilon)D/2, V_1^{\text{unsafe}}, .)$ or $(S^{sup}, .) = \text{Blur}(S^{\text{big}}, (1-\varepsilon)D/2, V_2^{\text{unsafe}}, .)$.

We first consider the case $(S^{sup}, .) = \text{Blur}(S, (1-\varepsilon)D/2, V_1^{\text{unsafe}}, .)$. By assumption, $B(v,r(v)) \cap S^{sup} \neq \emptyset$. Let $u \in S^{sup}$. By Theorem 9.20, $d_G(S,u) \leq d_{G[S^{sup}]}(S,u) \leq (1-\varepsilon)D/2 \leq D/2$. Thus, $u \in S^{\text{big}}$ according to the second property of the distance oracle $\mathcal{O}^{Dist}$. Hence, $S^{sup} \subseteq S^{\text{big}}$ and therefore there exists a vertex $u \in S^{\text{big}}$ with $d_G(v,u) \leq r(v)$. If $r(v) > D/10$, then $v \in V_1^{\text{unsafe}}$ and it follows by induction that $v \in V^{bad}$. If $r(v) \leq D/10$, then $d_G(S^{\text{big}}, v) \leq r(v) \leq D/10 \leq D$. Hence, the second property of $\mathcal{O}^{Dist}$ implies that $v \in V(T_{S^{\text{big}}})$ and the first property of $\mathcal{O}^{Dist}$ implies that $d_{T_{S^{\text{big}}}}(v) \leq (1+\varepsilon)d_G(S^{\text{big}}, v) \leq (1+\varepsilon)r(v)$ which together with $v \in V^{\text{unsafe}}$ implies that $v \in V_1^{\text{unsafe}}$. It follows by induction that $v \in V^{\text{bad}}$.

It remains to consider the case $(S^{sup}, .) = \text{Blur}(S^{\text{big}}, (1-\varepsilon)D/2, V_2^{\text{unsafe}}, .)$. By assumption, $B(v,r(v)) \setminus S^{sup} \neq \emptyset$ and as $S^{\text{big}} \subseteq S^{sup}$ therefore also $B(v,r(v)) \setminus S^{\text{big}} \neq \emptyset$. Hence, $d_G(V(G) \setminus S^{\text{big}}, v) \leq r(v)$. If $r(v) > D/10$, then $v \in V_2^{\text{unsafe}}$ and it follows by induction that $v \in V^{bad}$. If $r(v) \leq D/10$, then $d_G(V(G) \setminus S^{\text{big}}, v) \leq r(v) \leq D/10 \leq D$. Hence, the second property of $\mathcal{O}^{Dist}$ implies that $v \in V(T_{V \setminus S^{\text{big}}})$ and the first property of $\mathcal{O}^{Dist}$ implies that $d_{T_{V \setminus S^{\text{big}}}}(v) \leq (1+\varepsilon)d_G(V(G) \setminus S^{\text{big}}, v) \leq (1+\varepsilon)r(v)$ which together with $v \in V^{\text{unsafe}}$ implies that $v \in V_2^{\text{unsafe}}$. It follows by induction that $v \in V^{\text{bad}}$. □

Definition 9.22. *We refer to the tuple (S, D, V^{middle}) as a valid input if every $v \in V(G) \setminus V^{middle}$ is either very close or very far (or both), defined as follows.*

1. *very close:* $d_G(S,v) \leq (1+\varepsilon)r(v)$
2. *very far:* $\max_{u \in B_G(v,(1+\varepsilon)r(v))} d_G(S,u) \geq D$

Claim 9.23. *Assume (S, D, V^{middle}) is a valid input. Then, both $(S, (1-\varepsilon)D/2, V'^{middle})$ and $(S^{big}, (1-\varepsilon)D/2, V'^{middle})$ are valid inputs.*

Proof. Let $v \in V(G) \setminus V'^{\text{middle}}$. To show that $(S, (1-\varepsilon)D/2, V'^{\text{middle}})$ is a valid input, it suffices to show that one of the following holds:

1. $d_G(S, v) \leq (1+\varepsilon)r(v)$
2. $\max_{u \in B_G(v,(1+\varepsilon)r(v))} d_G(S, u) \geq (1-\varepsilon)D/2$

To show that $(S^{\text{big}}, (1-\varepsilon)D/2, V'^{\text{middle}})$ is a valid input, it suffices to show that one of the following holds:

1. $d_G(S^{\text{big}}, v) \leq (1+\varepsilon)r(v)$
2. $\max_{u \in B_G(v,(1+\varepsilon)r(v))} d_G(S^{\text{big}}, u) \geq (1-\varepsilon)D/2$

First, consider the case that $v \in V(G) \setminus V^{\text{middle}}$. As $(S, D, V^{\text{middle}})$ is a valid input , $d_G(S, v) \leq (1+\varepsilon)r(v)$ or $\max_{u \in B_G(v,(1+\varepsilon)r(v))} d_G(S, u) \geq D$. If $d_G(S, v) \leq (1+\varepsilon)r(v)$, then also $d_G(S^{\text{big}}, v) \leq (1+\varepsilon)r(v)$ as $S \subseteq S^{\text{big}}$. Now, assume $\max_{u \in B_G(v,(1+\varepsilon)r(v))} d_G(S, u) \geq D \geq (1-\varepsilon)D/2$. Hence, there exists $w \in B_G(v, (1+\varepsilon)r(v))$ with $d_G(S, w) \geq D$ and thus

$$\begin{aligned}
\max_{u \in B_G(v,(1+\varepsilon)r(v))} d_G(S^{\text{big}}, u) &\geq d_G(S^{\text{big}}, w) \\
&= \min_{s^{big} \in S^{\text{big}}} d_G(s^{big}, w) \\
&\geq \min_{s^{big} \in S^{\text{big}}} d_G(S, w) - d_G(S, s^{big}) \\
&= d_G(S, w) - \max_{s^{big} \in S^{\text{big}}} d_G(S, s^{big}) \\
&\geq D - (1+\varepsilon)D/2 \\
&= (1-\varepsilon)D/2,
\end{aligned}$$

as desired. It remains to consider the case $v \in V^{\text{middle}}$. Hence, $v \in V^{\text{middle}} \setminus V'^{\text{middle}}$ and therefore $v \in V_1^{\text{unsafe}} \cap V_2^{\text{unsafe}}$ and $r(v) \leq D/10$. As $v \in V_1^{\text{unsafe}}$ and $r(v) \leq D/10$, $d_{T_{S^{\text{big}}}}(v) \leq (1+\varepsilon)r(v)$ and therefore $d_G(S^{\text{big}}, v) \leq (1+\varepsilon)r(v)$, which already finishes the proof that $(S^{\text{big}}, (1-\varepsilon)D/2, V'^{\text{middle}})$ is a valid input. As $v \in V_2^{\text{unsafe}}$ and $r(v) \leq D/10$, $d_{T_{V \setminus S^{\text{big}}}}(v) \leq (1+\varepsilon)r(v)$. Hence, $d_G(V \setminus S^{\text{big}}, v) \leq (1+\varepsilon)r(v)$ and therefore $B_G(v, (1+\varepsilon)r(v)) \cap V \setminus S^{\text{big}} \neq \emptyset$. As every $w \in V \setminus S^{\text{big}}$ satisfies $d_G(S, w) \geq D/2 \geq (1-\varepsilon)D/2$, we have $\max_{u \in B_G(v,(1+\varepsilon)r(v))} d_G(S, u) \geq (1-\varepsilon)D/2$, which finishes the proof that $(S, (1-\varepsilon)D/2, V'^{\text{middle}})$ is a valid input. □

Claim 9.24. *Assume (S, D, V^{middle}) is a valid input. For every $v \in V_1^{unsafe} \cap V_2^{unsafe}$ with $r(v) \leq D/10$, we have $v \in V^{middle}$.*

Proof. Let $v \in V_1^{\text{unsafe}} \cap V_2^{\text{unsafe}}$ with $r(v) \leq D/10$. We have to show that $v \in V^{\text{middle}}$. As $(S, D, V^{\text{middle}})$ is a valid input, it suffices by Theorem 9.22 to show that $d_G(S, v) > (1+\varepsilon)r(v)$ and $\max_{u \in B_G(v,(1+\varepsilon)r(v))} d_G(S, u) < D$. As $v \in V_1^{\text{unsafe}}$ and $r(v) \leq D/10$, $d_G(S^{\text{big}}, v) \leq d_{T_{S^{\text{big}}}}(v) \leq (1+\varepsilon)r(v)$. Therefore,

$$\begin{aligned}
&\max_{u \in B_G(v,(1+\varepsilon)r(v))} d_G(S, u) \\
&\leq d_G(S, v) + (1+\varepsilon)r(v) \\
&\leq \max_{u \in S^{\text{big}}} d_G(S, u) + d_G(S^{\text{big}}, v) + (1+\varepsilon)r(v) \\
&\leq (1+\varepsilon)(D/2) + 2(1+\varepsilon)r(v) \\
&< D,
\end{aligned}$$

as needed. As $v \in V_2^{\text{unsafe}}$ and $r(v) \leq D/10$, $d_G(V(G) \setminus S^{\text{big}}, v) \leq d_{T_{V \setminus S^{\text{big}}}}(v) \leq (1+\varepsilon)r(v)$. Hence,

$$\begin{aligned}
&d_G(S, v) \geq d_G(S, V \setminus S^{\text{big}}) - d_G(V \setminus S^{\text{big}}, v) \\
&\geq D/2 - (1+\varepsilon)r(v) > (1+\varepsilon)r(v),
\end{aligned}$$

which finishes the proof. $\square$

Claim 9.25 (Randomized Lemma)**.** *Let $(., V^{bad}) = Blur_{rand}(S,\ D, \{v \in V(G)\colon r(v) > 0\}, V(G))$ for $D > 0$. For every $v \in V(G)$, $Pr[v \in V^{bad}] \leq \frac{20r(v)}{D(1-\varepsilon)^{\max(0,\log_2(2D))}}$.*

Proof. Consider the following more general claim: Let $(., V^{\text{bad}}) = Blur_{rand}(S, D, V^{\text{unsafe}}, V^{\text{middle}})$ with $(S, D, V^{\text{middle}})$ being a valid input and $\{v \in V^{\text{unsafe}}\colon r(v) = 0\} = \emptyset$. For a vertex $v \in V(G)$, we define $p_{v,S,D,V^{\text{unsafe}},V^{\text{middle}}} = Pr[v \in V^{\text{bad}}]$. Then,

$$p_{v,S,D,V^{\text{unsafe}},V^{\text{middle}}} \leq \begin{cases} \frac{(1+\mathbf{1}_{V^{\text{middle}}}(v))10r(v)}{D(1-\varepsilon)^{\max(0,\log_2(2D))}} & \text{if } v \in V^{\text{unsafe}} \\ 0 & \text{if } v \notin V^{\text{unsafe}} \end{cases}$$

We prove the more general claim by induction on the recursion depth. The base case $D \in (0, 1]$ directly follows as $\frac{(1+\mathbf{1}_{V^{\text{middle}}}(v))10r(v)}{D(1-\varepsilon)^{\max(0,\log_2(2D))}} \geq 10r(v)/D \geq 1$ as long as $r(v) \geq 1$. Next, consider the case $D > 1$. Let $p := p_{v,S,D,V^{\text{unsafe}},V^{\text{middle}}}$, $p_1 := p_{v,S,(1-\varepsilon)D/2,V_1^{\text{unsafe}},V'^{\text{middle}}}$ and $p_2 := p_{v,S^{\text{big}},(1-\varepsilon)D/2,V_2^{\text{unsafe}},V'^{\text{middle}}}$. From the algorithm definition, we have

$$p = p_1/2 + p_2/2.$$

By Theorem 9.23, both $(S, (1-\varepsilon)D/2, V'^{\text{middle}})$ and $(S^{\text{big}}, (1-\varepsilon)D/2, V'^{\text{middle}})$ are valid inputs. Hence, by induction we obtain for $i \in \{1, 2\}$ that

$$p_i \le \begin{cases} \dfrac{2\cdot(1+\mathbf{1}_{V'^{\text{middle}}}(v))10r(v)}{D(1-\varepsilon)(1-\varepsilon)^{\max(0,\log_2((1-\varepsilon)D))}} & \\ \le \dfrac{2\cdot(1+\mathbf{1}_{V'^{\text{middle}}}(v))10r(v)}{D(1-\varepsilon)^{\max(0,\log_2(2D))}} & \text{if } v \in V_i^{\text{unsafe}} \\ 0 & \text{if } v \notin V_i^{\text{unsafe}} \end{cases}$$

First, if $v \notin V^{\text{unsafe}}$, then $v \notin V_1^{\text{unsafe}} \cup V_2^{\text{unsafe}}$ and therefore $p = 0.5p_1 + 0.5p_2 = 0.5\cdot 0 + 0.5\cdot 0 = 0$, as desired.

From now on, assume that $v \in V^{\text{unsafe}}$. Note that we can furthermore assume that $r(v) \le D/10$ as otherwise we claim $p \le 1$ which trivially holds. First, consider the case that $v \in V_1^{\text{unsafe}} \cap V_2^{\text{unsafe}}$. As $r(v) \le D/10$, Theorem 9.24 implies that $v \in V^{\text{middle}}$ and together with the algorithm description it follows that $v \in V^{\text{middle}} \setminus V'^{\text{middle}}$.

Hence,

$$p \le 0.5p_1 + 0.5p_2 \le 2\cdot 0.5 \cdot \frac{2\cdot(1+\mathbf{1}_{V'^{\text{middle}}}(v))10r(v)}{D(1-\varepsilon)^{\max(0,\log_2(2D))}}$$
$$= \frac{(1+\mathbf{1}_{V^{\text{middle}}}(v))10r(v)}{D(1-\varepsilon)^{\max(0,\log_2(2D))}},$$

as desired.

It remains to consider the case $v \notin V_1^{\text{unsafe}} \cap V_2^{\text{unsafe}}$. By induction, this implies $p_1 = 0$ or $p_2 = 0$ and therefore

$$p \le 0.5p_1 + 0.5p_2 \le 0.5\frac{2\cdot(1+\mathbf{1}_{V'^{\text{middle}}}(v))10r(v)}{D(1-\varepsilon)^{\max(0,\log_2(2D))}}$$
$$= \frac{(1+\mathbf{1}_{V^{\text{middle}}}(v))10r(v)}{D(1-\varepsilon)^{\max(0,\log_2(2D))}},$$

which finishes the proof. □

Claim 9.26 (Deterministic Lemma)**.** *Let* $(., V^{bad}) = Blur_{det}(S,\ D, \{v \in V(G) \colon r(v) > 0\}, V(G))$ *for* $D > 0$. *Then,* $\mu(V^{bad}) \le \frac{10}{D(1-\varepsilon)^{\max(0,\log_2(2D))}} \sum_{v\in V(G)} \mu(v)r(v)$.

Proof. Consider the following more general claim: Let $(., V^{\text{bad}}) = Blur_{det}(S, D, V^{\text{unsafe}}, V^{\text{middle}})$ with $(S, D, V^{\text{middle}})$ being a valid input and $\{v \in V^{\text{unsafe}} \colon r(v) = 0\} = \emptyset$. Then,

$$\mu(V^{\text{bad}}) \le \frac{1}{(1-\varepsilon)^{\max(0,\log_2(2D))}}\left(\sum_{v\in V^{\text{unsafe}}, r(v)\le D/10} (1+\right.$$

$$\mathbf{1}_{V^{\text{middle}}}(v))\frac{\mu(v)r(v)}{D} + \sum_{v \in V^{\text{unsafe}}\colon r(v) > D/10} \mu(v)\Bigg).$$

We prove the more general claim by induction on the recursion depth. The base case $D \in (0, 1]$ trivially holds as for every $v \in V(G)$, $r(v) = 0$ or $r(v) \geq 1 > D/10$. Next, consider the case $D > 1$. Assume that $\Phi_1 \leq \Phi_2$. In particular, $2\Phi_1 \leq \Phi_1 + \Phi_2$ and therefore

$$\begin{aligned}
&2 \sum_{v \in V_1^{\text{unsafe}}\colon r(v) \leq D/10} (1 + \mathbf{1}_{V'^{\text{middle}}}(v))\mu(v)r(v) \\
&\leq \sum_{i=1}^{2} \sum_{v \in V_i^{\text{unsafe}}\colon r(v) \leq D/10} (1 + \mathbf{1}_{V'^{\text{middle}}}(v))\mu(v)r(v) \\
&= \sum_{v \in V_1^{\text{unsafe}} \cup V_2^{\text{unsafe}}\colon r(v) \leq D/10} (1 + \mathbf{1}_{V'^{\text{middle}}}(v))\mu(v)r(v) + \\
&\quad \sum_{v \in V_1^{\text{unsafe}} \cap V_2^{\text{unsafe}}\colon r(v) \leq D/10} (1 + \mathbf{1}_{V'^{\text{middle}}}(v))\mu(v)r(v) \\
&\leq \sum_{v \in V^{\text{unsafe}}\colon r(v) \leq D/10} (1 + \mathbf{1}_{V'^{\text{middle}}}(v))\mu(v)r(v) + \\
&\quad \sum_{v \in V_1^{\text{unsafe}} \cap V_2^{\text{unsafe}}\colon r(v) \leq D/10} \mathbf{1}_{V^{\text{middle}} \setminus V'^{\text{middle}}}(v)\mu(v)r(v) \\
&= \sum_{v \in V^{\text{unsafe}}\colon r(v) \leq D/10} (1 + \mathbf{1}_{V^{\text{middle}}}(v))\mu(v)r(v),
\end{aligned}$$

where the second inequality follows from the following three facts: First, $V_1^{\text{unsafe}} \cup V_2^{\text{unsafe}} \subseteq V^{\text{unsafe}}$. Second, as $(S, D, V^{\text{middle}})$ is a valid input, Theorem 9.24 states that for every $v \in V_1^{\text{unsafe}} \cap V_2^{\text{unsafe}}$ with $r(v) \leq D/10$, we have $v \in V^{\text{middle}}$. Third, it directly follows from the algorithm description that there exists no $v \in (V_1^{\text{unsafe}} \cap V_2^{\text{unsafe}}) \cap V'^{\text{middle}} = \emptyset$ with $r(v) \leq D/10$.

Now, let $D' := (1 - \varepsilon)D/2$. From the induction hypothesis, it follows that

$$\begin{aligned}
\mu(V^{\text{bad}}) &\leq \frac{1}{(1-\varepsilon)^{\max(0, \log_2(2D'))}} \Bigg(\sum_{v \in V_1^{\text{unsafe}}, r(v) \leq D'/10} (1 + \\
&\quad \mathbf{1}_{V'^{\text{middle}}}(v))\frac{\mu(v)r(v)}{D'} + \sum_{v \in V_1^{\text{unsafe}}\colon r(v) > D'/10} \mu(v)\Bigg) \\
&\leq \frac{1}{(1-\varepsilon)^{\max(0, \log_2(2D'))}} \Bigg(\sum_{v \in V_1^{\text{unsafe}}, r(v) \leq D/10} (1 +
\end{aligned}$$

$$
\begin{aligned}
&\mathbf{1}_{V'^{\text{middle}}}(v))\frac{\mu(v)r(v)}{D'} + \sum_{v \in V_1^{\text{unsafe}}\colon r(v) > D/10} \mu(v)\Big) \\
&\leq \frac{1}{(1-\varepsilon)^{\max(0,\log_2(2D))}}\Big(2 \sum_{v \in V^{\text{unsafe}}, r(v) \leq D/10} (1+ \\
&\mathbf{1}_{V'^{\text{middle}}}(v))\frac{\mu(v)r(v)}{D} + \sum_{v \in V_1^{\text{unsafe}}\colon r(v) > D/10} \mu(v)\Big) \\
&\leq \frac{1}{(1-\varepsilon)^{\max(0,\log_2(2D))}}\Big(\sum_{v \in V^{\text{unsafe}}, r(v) \leq D/10} (1+ \\
&\mathbf{1}_{V^{\text{middle}}}(v))\frac{\mu(v)r(v)}{D} + \sum_{v \in V^{\text{unsafe}}\colon r(v) > D/10} \mu(v)\Big),
\end{aligned}
$$

as desired. The case $\Phi_2 < \Phi_1$ follows in the exact same manner and is therefore omitted.

□

9.3.3 Corollaries

In this section we show how the rather general Theorem 9.19 implies the solution to the edge variant of the blurry growing problem from Theorem 9.5. In particular, we prove here Theorem 9.29 that solves both the randomized and deterministic version of the problem.

Definition 9.27 (Subdivided Graph)**.** *Let G be a weighted graph. The subdivided graph G_{sub} of G is defined as the weighted graph that one obtains from G by replacing each edge $e = \{u, v\} \in E(G)$ with one new vertex v_e and two new edges $\{u, v_e\}$ and $\{v_e, v\}$. Moreover, we define $\ell_{G_{sub}}(u, v_e) = 0$ and $\ell_{G_{sub}}(v_e, v) = \ell_G(u, v)$ where we assume that u has a smaller ID than v.*

Lemma 9.28 (Simulation of the Subdivided Graph)**.** *Assume that some problem $\mathcal{P}$ defined on a weighted graph H can be solved in T steps and with performing all oracle calls with precision parameter ε and distance parameter no larger than D for arbitrary $T > 0$, ε and D. Now, assume that the weighted input (and communication) graph is G. Then, we can solve the problem $\mathcal{P}$ on the weighted graph G_{sub} in $O(T)$ steps performing all oracle calls with precision parameter ε and distance parameter no larger than D.*

Proof. For each new vertex w that subdivides the edge $\{u, v\} \in E(G)$, the node w is simulated by the node u (where we assume that u has a smaller ID than v). That is, if w wants to send a message to v, then u sends that message to v. Similarly, if v wants to send a message to w, then v sends the message to u instead. The node u also performs all the local computation that w would do. As w has exactly two neighbors, our computational model only allows it to perform $\widetilde{O}(1)$ (P)RAM operations in each step and therefore each node in the original graph only needs to do additional work proportional to its degree and which can be performed with depth $\widetilde{O}(1)$. Hence, each node can efficiently simulate all the new nodes that it has to simulate. It remains to discuss how to simulate the oracles in the graph G_{sub} with the oracles for the original graph G. The global aggregation oracle in G_{sub} can be simulated in G as follows: first, each node computes the sum of the values of all the nodes it simulates (including its own value), which can efficiently be done in PRAM. Then, one can use the aggregation oracle in G to sum up all those sums, which is equal to the total sum of all the node values in G_{sub}. For the forest aggregation oracle, let F_{sub} be a rooted forest in G_{sub}. Now, let F be the rooted forest with $V(F) = V(F_{sub}) \cap V(G)$ and which contains each edge $\{u, v\} \in E(G)$ if both $\{u, w\}$ and $\{w, v\}$ are contained in the forest F_{sub}. Moreover, the set of roots of F is given by all the roots in F_{sub} that are vertices in G together with all vertices in G whose parent in F_{sub} is a root. Note that for every node $v \in V(F)$, $d_F(v) \leq d_{F_{sub}}(v) \leq D$. Moreover, it is easy to see that the aggregation on F_{sub} can be performed in $O(1)$ steps using only aggregations on F as an oracle. Finally, one can also simulate the distance oracle $\mathcal{O}^{Dist}_{\varepsilon,D}$ in G_{sub} in $O(1)$ steps and only performing one oracle call to $\mathcal{O}^{Dist}_{\varepsilon,D}$ in G. □

It is easy to deduce the specific version for edges from the above Theorem 9.19 and the fact that we can efficiently simulate an algorithm on the subdivided graph.

Corollary 9.29. *Consider the following problem on a weighted input graph G. The input consists of the following.*

1. *A set $S \subseteq V(G)$.*

2. *In the deterministic version, each edge $e \in E(G)$ has a weight $\mu(e)$.*

3. *A distance parameter $D > 0$.*

The output is a set S^{sup} with $S \subseteq S^{sup} \subseteq V(G)$. Let E^{bad} denote the set consisting of those edges having exactly one endpoint in S^{sup}, then S^{sup} satisfies

1. *for every $v \in S^{sup}, d_{G[S^{sup}]}(S, v) \leq D$,*

2. *in the deterministic version, $\mu(E^{bad}) = O(\sum_{e \in E(G)} \mu(e)\ \ell(e)/D)$*

3. *and in the randomized version, $Pr[e \in E^{bad}] = O(\ell(e)/D)$ for every $e \in E(G)$.*

There is an algorithm that solves the problem above in $O(\log(D) + 1)$ steps, performing all oracle calls with precision parameter $\varepsilon = \frac{1}{\log(n)}$ and distance parameter no larger than D.

Proof. Run the algorithm of Theorem 9.19 on the subdivided graph with input $r(v_e) = \ell(e)$ and $\mu(v_e) = \mu(e)$ for every edge $e \in E$. For all other vertices in the subdivided graph set $r(v) = 0$ and $\mu(v) = 0$. □

9.4 A General Clustering Result

In this section we prove our main clustering result Theorem 9.30. For the application to low stretch spanning trees in Theorem 9.42, it is important that the result works by essentially only having access to an approximate distance oracles and that it works even if we start with an input set of *terminals* and require that each final cluster contains at least one such terminal. We note that the algorithm of our main result Theorem 9.30 uses the blurry ball growing algorithm of Section 9.3 as a subroutine. Since our main result Theorem 9.30 is rather general, we start by sketching a simpler version of our result in Section 9.4.1. Afterwards, we prove Theorem 9.30 in Section 9.4.2. Finally, we derive useful corollaries of Theorem 9.30 in Section 9.4.3.

9.4.1 Intuition and Proof Sketch

We now sketch the proof of Theorem 9.1 – a corollary of the general result Theorem 9.30. Theorem 9.1 was discussed in Section 9.1, we restate it here for convenience.

Theorem 9.1. *[A corollary of Theorem 9.30] Let G be a weighted graph. We are given a set of* terminals *$Q \subseteq V(G)$ and a parameter $R > 0$ such that for every $v \in V(G)$ we have $d(Q, v) \leq R$. Also, a precision parameter $0 < \varepsilon < 1$ is given. There is a deterministic distributed and parallel algorithm outputting a partition $\mathcal{C}$ of the vertices into clusters and a subset of terminals $Q' \subseteq Q$ with the following properties:*

1. *Each cluster $C \in \mathcal{C}$ contains exactly one terminal $q \in Q'$. Moreover, for any $v \in C$ we have $d_{G[C]}(q, v) \leq (1 + \varepsilon)R$.*
2. *For the set E^{bad} of edges connecting different clusters of $\mathcal{C}$ we have*

$$|E^{bad}| = \widetilde{O}\left(\frac{1}{\varepsilon R}\right) \cdot \sum_{e \in E(G)} \ell(e).$$

The PRAM *variant of the algorithm has work $\widetilde{O}(m)$ and depth $\widetilde{O}(1)$. The* CONGEST *variant of the algorithm runs in $\widetilde{O}(\sqrt{n} + \mathrm{HopDiam}(G))$ rounds.*

Our approach to prove Theorem 9.1 is somewhat similar to the one taken in Section 9.2 to derive our strong-diameter clustering result. As in the proof of Theorem 9.13, we solve it by repeatedly solving the following problem $O(\log n)$ times: in the i-th iteration, we split the still active terminals $Q_i \subseteq Q$ based on the i-th bit in their identifier into a set of blue terminals $Q_i^{\mathcal{B}}$ and a set of red terminals $Q_i^{\mathcal{R}}$. Then, we solve the following "separation" problem:

In that problem, we are given a set Q_i that is R_i-ruling in G_i for $R_i = (1+\varepsilon/\operatorname{poly}\log n)^i R$. We want to select a subset $Q_{i+1} = Q_{i+1}^{\mathcal{R}} \sqcup Q_{i+1}^{\mathcal{B}}$ of $Q_i = Q_i^{\mathcal{R}} \sqcup Q_i^{\mathcal{B}}$ and cut a small fraction of edges in G_i to get a new graph G_{i+1} such that the following three properties hold:

1. Separation Property: The sets $Q_{i+1}^{\mathcal{R}}$ and $Q_{i+1}^{\mathcal{B}}$ are disconnected in G_{i+1}.

2. Ruling Property: The set Q_{i+1} is $(1+\varepsilon/\operatorname{poly}\log n)R_i$-ruling in G_{i+1}.
3. Cut Property: The number of edges cut is at most $\widetilde{O}\left(\frac{1}{\varepsilon R}\right) \cdot \sum_{e \in E(G)} \ell(e)$.

If we can solve this partial problem, then we simply repeat it $O(\log n)$ times going bit by bit and obtain an algorithm proving Theorem 9.1 (cf. the reduction of Theorem 9.30 to Theorem 9.32 in the general proof in Section 9.4.2).

Our solution that achieves the three properties above is more complicated than the proof of Theorem 9.13 in Section 9.2: we need to be more careful because we only have access to approximate distances and also because we want to cluster all of the vertices.

We start by computing an approximate shortest path forest F with the active terminals Q_i being the set of roots. We define the set of blue nodes $U^{\mathcal{B}}$ and red nodes $U^{\mathcal{R}}$ as the set of the nodes such that the root of their tree in F is in $Q_i^{\mathcal{B}}$ and $Q_i^{\mathcal{R}}$, respectively.

What happens if we cut all edges between $U^{\mathcal{B}}$ and $U^{\mathcal{R}}$ and define $Q_{i+1}^{\mathcal{R}} = Q_i^{\mathcal{R}}$ and $Q_{i+1}^{\mathcal{B}} = Q_i^{\mathcal{B}}$? The separation and the ruling property will be clearly satisfied. However, we do not have any guarantees on the number of edges cut, hence the cut property is not necessarily satisfied.

To remedy this problem, we use the tool of blurry ball growing developed in Section 9.3. In particular, we choose one of the two colors (which one we discuss later). Let us name it $\mathcal{A}$ and define $W^{\mathcal{A}}$ as the set of nodes returned by the blurry ball growing procedure from Theorem 9.29 starting from $U^{\mathcal{A}}$ with distance parameter $D = \varepsilon/\operatorname{poly}\log n \cdot R$.

Let us note that by the properties of Theorem 9.29, if we now delete all the edges between $W^{\mathcal{A}}$ and $U^{\overline{\mathcal{A}}} \setminus W^{\mathcal{A}}$ (here, $\overline{\mathcal{A}} \in \{\mathcal{R}, \mathcal{B}\} \setminus \mathcal{A}$), the cut property would be satisfied. For $Q_{i+1}^{\mathcal{A}} = Q_i^{\mathcal{A}}$ and $Q_{i+1}^{\overline{\mathcal{A}}} = Q_i^{\overline{\mathcal{A}}} \setminus W^{\mathcal{A}}$, we also get the separation property. The problem is the ruling property. Hence, the set $Q_{i+1}^{\overline{\mathcal{A}}}$ may fail to be $(1+\varepsilon/\operatorname{poly}\log n)R$-ruling in the respective component of G_{i+1}.

The final trick that we need is to realize that, although we are not done yet, we still made some progress, which allows us to set up a recursion: if we choose $\mathcal{A}$ to be the color class such that $|U^{\mathcal{A}}| \geq |V(G)|/2$, for at least half of the nodes, in particular those in $W^{\mathcal{A}}$, we can now safely say that they will belong to a connected component containing a node from $Q^{\mathcal{A}}$ in the final partition. For the nodes in $U^{\overline{\mathcal{A}}} \setminus W^{\mathcal{A}}$ we do not know yet, however, we can simply solve the problem there recursively.

This recursion works as follows. We will recurse on the graph $G_{rec} = G[U^{\overline{\mathcal{A}}} \setminus W^{\mathcal{A}}]$. The set of terminals $Q_{rec}^{\overline{\mathcal{A}}}$ in the recursive problem is simply $Q_i^{\overline{\mathcal{A}}} \cap V(G_{rec})$. The set of terminals $Q_{rec}^{\mathcal{A}}$ contains every node $u \in V(G_{rec})$ such that the parent of u in the forest F is contained in $W^{\mathcal{A}}$. To reflect the fact that u is not a terminal in the original problem, we introduce the notion of delays – see Section 9.1.2 for their definition. The delay of u is (roughly) set to the computed approximate distance to $Q^{\mathcal{A}}$. This means that in the recursive call, the shortest path forest starting from $Q_{rec}^{\mathcal{A}}$ behaves as if it started from $Q^{\mathcal{A}}$, modulo small errors in distances of order $\varepsilon R/\operatorname{poly}\log n$ that we inflicted by using approximate distances and by using blurry ball growing to obtain the set $W^{\mathcal{A}}$.

When we return from the recursion, the nodes of $\overline{G_{rec}}$ are split into those belonging to terminals in $Q'^{\mathcal{R}}_{rec}$ and $Q'^{\mathcal{B}}_{rec}$. We define $Q'^{\overline{\mathcal{A}}} = Q'^{\overline{\mathcal{A}}}_{rec}$ and $Q'^{\mathcal{A}} = Q^{\mathcal{A}}$. We also mark the nodes of $W^{\mathcal{A}}$ as belonging to terminals in $\mathcal{A}$.

To finish the i-th iteration, we define $Q^{\mathcal{R}}_{i+1} = Q'^{\mathcal{R}}$ and $Q^{\mathcal{B}}_{i+1} = Q'^{\mathcal{B}}$. We cut all edges between the nodes belonging to terminals in $Q'^{\mathcal{R}}$ and $Q'^{\mathcal{B}}$. This definition of Q_{i+1} preserves the separation property.

Moreover, one can check that in every recursive step we distort the distances multiplicatively by $1 + \varepsilon/\operatorname{poly}\log(n)$ and additively by $\varepsilon/\operatorname{poly}\log(n) \cdot R$. This implies that the ruling property is satisfied. Similarly, the cut property is satisfied since each recursive step contributes only $\widetilde{O}\left(\frac{1}{\varepsilon R}\right) \cdot \sum_{e \in E(G)} \ell(e)$ to the final number of edges cut.

9.4.2 Main Proof

We are now ready to state and prove our main clustering result. As before in Section 9.3, we first consider a version where the goal is to minimize the number of vertices v whose ball of radius $r(v)$ is not fully contained in one of the clusters. Later, the edge cutting version follows as a simple corollary.

Moreover, as written above, the theorem allows each terminal to be assigned a delay. Allowing these delays helps us with solving the clustering problem recursively and the delays are also convenient when we apply our clustering result to efficiently compute low-stretch spanning trees.

The final algorithm invokes the blurry ball growing procedure a total of $O(\log^2 n)$ times, each time with parameter D. That's the reason why the ruling guarantee at the end contains an additive $O(\log^2 n)D$ term.

In the simplified version presented above, we set D equal to $\varepsilon R/\operatorname{poly}(\log n)$.

Theorem 9.30. *Consider the following problem on a weighted input graph G. The input consists of the following.*

1. *A weighted subgraph $H \subseteq G$.*
2. *Each node $v \in V(H)$ has a preferred radius $r(v)$.*
3. *In the deterministic version, each node $v \in V(H)$ additionally has a weight $\mu(v)$.*
4. *There is a set of center nodes $Q \subseteq V(H)$, with each center node $q \in Q$ having a delay $del(q) \geq 0$.*
5. *There is a parameter R such that for every $v \in V(H)$ we have $d_{H,del}(Q, v) \leq R$.*
6. *There are two global variables $D > 0$ and $\varepsilon \in \left[0, \frac{1}{\log^2(n)}\right]$.*

The output consists of a partition $\mathcal{C}$ of H, a set $Q' \subseteq Q$ and two sets $V^{good} \sqcup V^{bad} = V(H)$ such that

1. *each cluster $C \in \mathcal{C}$ contains exactly one node q'_C in Q',*
2. *for each $C \in \mathcal{C}$ and $v \in C$, we have $d_{H[C],del}(q'_C, v) \leq (1+\varepsilon)^{O(\log^2 n)} d_{H,del}(Q, v) + O(\log^2 n)D$,*
3. *for every $v \in V^{good}$, $B_H(v, r(v)) \subseteq C$ for some $C \in \mathcal{C}$ and*
4. *in the deterministic version, $\mu(V^{bad}) = O(\log(n) \cdot \sum_{v \in V(H)} \mu(v) r(v)/D)$ and in the randomized version, for every $v \in V(H)$: $P[v \in V^{bad}] = O(\log(n) r(v)/D)$.*

There is an algorithm that solves the problem above in $O(\log^3 n)$ steps, performing all oracle calls with precision parameter ε and distance parameter no larger than $(1+\varepsilon)^{O(\log^2 n)} R + O(\log^2 n)D$.

Proof. Recall that b denotes the number of bits in the node-IDs of the input graph. The algorithm computes a sequence of weighted graphs $H = H_0 \supseteq H_1 \supseteq \ldots \supseteq H_b$ with $V(H_i) = V(H)$ for $i \in \{0, 1, \ldots, b\}$, a sequence of centers $Q = Q_0 \supseteq Q_1 \supseteq Q_2 \supseteq \ldots \supseteq Q_b$ and a sequence of good nodes $V = V_0^{good} \supseteq V_1^{good} \supseteq V_2^{good} \supseteq \ldots \supseteq V_b^{good}$. Moreover, $V^{good} := V_b^{good}$, $V^{bad} := V(G) \setminus V^{good}$ and $V_i^{bad} := V \setminus V_i^{good}$. The connected components of H_b will be the clusters of the output partition $\mathcal{C}$ and there will be exactly one node in $Q' := Q_b$ contained in each cluster of $\mathcal{C}$.

The following invariants will be satisfied for every $i \in \{0, 1, \ldots, b\}$:

1. Separation Invariant: Let $u, v \in Q_i$ be two nodes contained in the same connected component of H_i. Then, the first i bits of the IDs of u and v coincide.
2. Ruling Invariant: For every $v \in V(H)$, we have

$$d_{H_i,\mathrm{del}}(Q_i, v) \leq (1+\varepsilon)^{i \cdot O(\log n)} d_{H,\mathrm{del}}(Q, v) + \left(\sum_{j=1}^{i \cdot O(\log n)} (1+\varepsilon)^j \right) 3D.$$

3. Good Invariant: For every $v \in V_i^{good}$, $B_{H_i}(v, r(v)) = B_H(v, r(v))$.
4. Bad Invariant (Deterministic): $\mu(V_i^{bad}) = i \cdot O(\sum_{v \in V(H)} \mu(v) r(v)/D)$.
(Randomized): For every $v \in V(H)$: $\mathrm{P}[v \in V_i^{bad}] = i \cdot O(r(v)/D)$.

It is easy to verify that setting $H_0 = H$, $Q_0 = Q$ and $V_0^{good} = V(H)$ results in all of the invariants being satisfied for $i = 0$. For $i = b$, the separation invariant implies that each cluster of $\mathcal{C}$ (that is, a connected component of H_b) contains at most one node in $Q' = Q_b$. Together with the ruling invariant, this implies that every cluster $C \in \mathcal{C}$ contains exactly one node q'_C in Q' such that for each node $v \in C$, it holds that

$$\begin{aligned} d_{H[C],\mathrm{del}}(q'_C, v) &= d_{H_b,\mathrm{del}}(Q_b, v) \\ &\leq (1+\varepsilon)^{b \cdot O(\log n)} d_{H,\mathrm{del}}(Q, v) + \left(\sum_{j=1}^{b \cdot O(\log n)} (1+\varepsilon)^j \right) 3D \\ &= (1+\varepsilon)^{O(\log^2 n)} d_{H,\mathrm{del}}(Q, v) + \left(\sum_{j=1}^{O(\log^2 n)} O(1) \right) 3D \\ &= (1+\varepsilon)^{O(\log^2 n)} d_{H,\mathrm{del}}(Q, v) + O(\log^2 n) D, \end{aligned}$$

where we used that $\varepsilon \leq \frac{1}{\log^2(n)}$.

Furthermore, it follows from the good invariant that for every $v \in V^{good} = V_b^{good}$, $B_H(v, r(v)) = B_{H_b}(v, r(v))$. In particular, all vertices in $B_H(v, r(v))$ are contained in the same connected component in H_b and therefore $B_H(v, r(v)) \subseteq C$ for some cluster $C \in \mathcal{C}$. For the deterministic version, the bad invariant implies

$$\begin{aligned} \mu(V^{bad}) = \mu(V_b^{bad}) &= b \cdot O(\sum_{v \in V(H)} \mu(v) r(v)/D) \\ &= O\left(\log(n) \sum_{v \in V(H)} \mu(v) r(v)/D \right). \end{aligned}$$

For the randomized version, the bad invariant implies that for every $v \in V(H)$,

$$\mathrm{P}[v \in V^{bad}] = \mathrm{P}[v \in v_b^{bad}] = b \cdot O(r(v)/D) = O(\log(n) r(v)/D).$$

Hence, we output a solution satisfying all the criteria.

Let $i \in \{0, 1, \ldots, b-1\}$. It remains to describe how to compute $(H_{i+1}, Q_{i+1}, V_{i+1}^{good})$ given (H_i, Q_i, V_i^{good}) while preserving the invariants.

In each phase, we split $Q_i = Q_i^{\mathcal{R}} \sqcup Q_i^{\mathcal{B}}$ according to the i-th bit in the unique identifier of each node. Then, we apply Theorem 9.32 with $H_{\mathrm{L}9.32} = H_i, Q^{\mathcal{R}}_{\mathrm{L}9.32} = Q_i^{\mathcal{R}}, Q^{\mathcal{B}}_{\mathrm{L}9.32} = Q_i^{\mathcal{B}}$ and $R_{\mathrm{L}9.32} = (1+\varepsilon)^{O(\log^2 n)} R + O(\log^2 n) D$. In the randomized version we set the recursion depth parameter $i_{\mathrm{L}9.32} = \lceil \log_2(D) \rceil$.

Note that the input is valid as for every $v \in V(H_{\mathrm{L}9.32})$ we have

$$d_{H_{\mathrm{L}9.32},\mathrm{del}}(Q_{\mathrm{L}9.32}, v) = d_{H_i,\mathrm{del}}(Q_i, v) \leq (1+\varepsilon)^{i \cdot O(\log n)} d_{H,\mathrm{del}}(Q, v)$$

$$+ \left(\sum_{j=1}^{i \cdot O(\log n)} (1+\varepsilon)^j \right) 3D \leq R_{\text{L}9.32},$$

where the first inequality follows from the ruling invariant. Finally, we set $H_{i+1} = H'_{\text{L}9.32}, Q_{i+1} = Q'^{\mathcal{R}}_{\text{L}9.32} \sqcup Q'^{\mathcal{B}}_{\text{L}9.32}$ and $V^{good}_{i+1} = V^{good}_i \cap (V^{good})_{\text{L}9.32}$.

Claim 9.31. *Computing $(H_{i+1}, Q_{i+1}, V_{good,i+1})$ from $(H_i, Q_i, V_{good,i})$ as written above preserves the four invariants.*

Proof. We start with the separation invariant. Let $u, v \in Q_{i+1}$. Assume that u and v are in the same connected component of H_{i+1}. In particular, this implies that u and v are also in the same connected component of H_i and therefore the separation invariant for i implies that the first i bits of the IDs of u and v coincide. Moreover, the separation property of Theorem 9.32 together with our assumption that u and v are in the same connected component of $H_{i+1} := H'_{\text{L}9.32}$ implies that the first $i+1$ bits of the IDs of u and v coincide, as desired. Next, we check that the ruling invariant is satisfied. The ruling property of Theorem 9.32 implies that for every $v \in V(H_{i+1})$ we have

$$\begin{aligned}
& d_{H_{i+1},\text{del}}(Q_{i+1}, v) = d_{H'_{\text{L}9.32},\text{del}}(Q'^{\mathcal{R}}_{\text{L}9.32} \cup Q'^{\mathcal{B}}_{\text{L}9.32}, v) \\
& \leq (1+\varepsilon)^{2(i_{\text{L}9.32}+1)} d_{H_{\text{L}9.32},\text{del}}(Q^{\mathcal{R}}_{\text{L}9.32} \cup Q^{\mathcal{B}}_{\text{L}9.32}, v) \\
& + \left(\sum_{j=1}^{i_{\text{L}9.32}} (1+\varepsilon)^{2j} \right) 3D \\
& \leq (1+\varepsilon)^{O(\log n)} d_{H_i,\text{del}}(Q_i, v) + \left(\sum_{j=1}^{O(\log n)} (1+\varepsilon)^j \right) 3D \\
& \leq (1+\varepsilon)^{O(\log n)} \Big((1+\varepsilon)^{i \cdot O(\log n)} d_{H,\text{del}}(Q, v) \\
& + \left(\sum_{j=1}^{i \cdot O(\log n)} (1+\varepsilon)^j \right) 3D \Big) + \left(\sum_{j=1}^{O(\log n)} (1+\varepsilon)^j \right) 3D \\
& = (1+\varepsilon)^{(i+1) \cdot O(\log n)} d_{H,\text{del}}(Q, v) + \left(\sum_{j=1}^{(i+1) \cdot O(\log n)} (1+\varepsilon)^j \right) 3D.
\end{aligned}$$

Hence, the ruling property is preserved. To check that the good invariant is preserved, consider an arbitrary node $v \in V^{good}_{i+1}$. We have to show that $B_H(v, r(v))$ is fully contained in one of the connected components of H_{i+1}.

First, $v \in V^{good}_{i+1}$ directly implies $v \in V^{good}_i$ and therefore the good invariant implies that $B_{H_i}(v, r(v)) = B_H(v, r(v))$. Second, $v \in (V^{good})_{\text{L}9.32}$ together with the good property of

Theorem 9.32 implies that

$$B_{H_{i+1}}(v, r(v)) = B_{H'_{\text{L9.32}}}(v, r(v)) = B_{H_{\text{L9.32}}}(v, r(v)) = B_{H_i}(v, r(v)).$$

Hence, $B_{H_{i+1}}(v, r(v)) = B_H(v, r(v))$, as needed.

It remains to check the bad property. For the deterministic version, we have

$$\begin{aligned}
&\mu(V_{i+1}^{bad}) \leq \mu(V_i^{bad}) + \mu(V_{\text{L9.32}}^{bad}) \\
&\leq i \cdot O\left(\sum_{v \in V(H)} \mu(v)r(v)/D\right) + O\left(\sum_{v \in V(H)} \mu(v)r(v)/D\right) \\
&\leq (i+1) \cdot O\left(\sum_{v \in V(H)} \mu(v)r(v)/D\right).
\end{aligned}$$

For the randomized version, we have for every $v \in V(H)$

$$\begin{aligned}
\mathrm{P}[v \in V_{i+1}^{bad}] &\leq \mathrm{P}[v \in V_i^{bad}] + \mathrm{P}[v \in V_{\text{L9.32}}^{bad}] \\
&\leq i \cdot O(r(v)/D) + \left(\frac{1}{2^{\lceil \log(D) \rceil}} + \frac{1}{D}\right) O(r(v)) \\
&\leq (i+1) \cdot O(r(v)/D),
\end{aligned}$$

as needed.

□

Each of the $O(\log n)$ invocations of the algorithm of Theorem 9.32 takes $O(\log^2 n)$ steps, and all oracle calls are performed with precision parameter ε and distance parameter no larger than $(1+\varepsilon)^{O(\log n)} R_{\text{L9.32}} + \left(\sum_{j=1}^{O(\log n)} (1+\varepsilon)^{2j}\right) 3D = (1+\varepsilon)^{O(\log^2 n)} R + O(\log^2 n) D$. This finishes the proof of Theorem 9.30. □

Lemma 9.32. *Consider the following problem on a weighted input graph G. The input consists of the following.*

1. *A weighted subgraph $H \subseteq G$.*
2. *Each node $v \in V(H)$ has a preferred radius $r(v)$.*
3. *In the deterministic version, each node $v \in V(H)$ additionally has a weight $\mu(v)$.*
4. *There is a recursion depth parameter $i \in \mathbb{N}_0$. In the randomized version i is part of the input. In the deterministic version we define $i = 1 + \lfloor \log(\sum_{v \in V(H)} \mu(v) r(v)) \rfloor$ if $\sum_{v \in V(H)} \mu(v) r(v) > 0$ and $i = 0$ otherwise.*

5. *There are sets $Q = Q^{\mathcal{R}} \sqcup Q^{\mathcal{B}} \subseteq V(H)$ with each center $q \in Q$ having a delay $del(q) \geq 0$.*
6. *There is a parameter R such that for every $v \in V(H)$ we have $d_{H,del}(Q, v) \leq R$.*
7. *There are two global variables $D > 0$ and $\varepsilon \in \left[0, \frac{1}{\log(n)}\right]$.*

The output consists of two sets $Q'^{\mathcal{R}} \subseteq Q^{\mathcal{R}}$ and $Q'^{\mathcal{B}} \subseteq Q^{\mathcal{B}}$, a weighted graph $H' \subseteq H$ with $V(H') = V(H)$ together with a partition $V(H) = V^{good} \sqcup V^{bad}$ such that

1. *Separation Property: For each connected component C of H', $Q'^{\mathcal{R}} \cap C = \emptyset$ or $Q'^{\mathcal{B}} \cap C = \emptyset$.*
2. *Ruling Property: For every $v \in V(H')$, $d_{H',del}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, v) \leq (1+\varepsilon)^{2(i+1)} d_{H,del}(Q, v) + \left(\sum_{j=1}^{i} (1+\varepsilon)^{2j}\right) 3D$.*
3. *Good Property: For every $v \in V^{good}$, $B_{H'}(v, r(v)) = B_H(v, r(v))$.*
4. *Bad Property, Deterministic Version: $\mu(V^{bad}) \leq \left(1 - \frac{1}{2^i}\right) O(\sum_{v \in V(H)} \mu(v) r(v)/D)$.*
Randomized Version: For every $v \in V(H)$, $P[v \in V^{bad}] = (\frac{1}{2^i} + \frac{1-2^{-i}}{D}) O(r(v))$.

There is an algorithm that solves the problem above in $O((i+1)(\log(D)+1))$ steps, performing all oracle calls with precision parameter ε and distance parameter no larger than $(1+\varepsilon)^{2i} R + \left(\sum_{j=1}^{i} (1+\varepsilon)^{2j}\right) 3D$.

Proof. We will first consider the special, base case with $i = 0$. Then, we analyse the general case.

Base Case: : We start with the base case $i = 0$.

Let F be the weighted and rooted forest returned by $\mathcal{O}^{Dist}_{\varepsilon,R}(H, Q, \text{del})$. Note that we are allowed to perform this oracle call as the distance parameter R satisfies $R \leq (1+\varepsilon)^0 R + \left(\sum_{j=1}^{0} (1+\varepsilon)^{2j}\right) 3D$. Moreover, as for every $v \in V(H)$, $d_{H,\text{del}}(Q, v) \leq R$, the second property of the distance oracle ensures that $V(F) = V(H)$.

For $\mathcal{A} \in \{\mathcal{R}, \mathcal{B}\}$, we define

$$U^{\mathcal{A}} = \{v \in V(F) = V(H) \colon \text{root}_F(v) \in Q^{\mathcal{A}}\}.$$

Note that $V(H) = U^{\mathcal{R}} \sqcup U^{\mathcal{B}}$ as for every $v \in V(F)$, $\text{root}_F(v) \in Q = Q^{\mathcal{R}} \sqcup Q^{\mathcal{B}}$. The output now looks as follows: We set $Q'^{\mathcal{R}} = Q^{\mathcal{R}} \cap U^{\mathcal{R}}$ and $Q'^{\mathcal{B}} = Q^{\mathcal{B}} \cap U^{\mathcal{B}}$. We obtain the graph H' from H by deleting every edge with one endpoint in $U^{\mathcal{B}}$ and the other endpoint in $U^{\mathcal{R}}$. We set $V^{bad} = \{v \in V(H) \colon r(v) > 0\}$ and $V^{good} = V(H) \setminus V^{bad}$.

We now verify that all the four properties from the theorem statement are satisfied.

Separation property: Let C be a connected component of H'. As we obtained H' from H by deleting every edge with one endpoint in $U^{\mathcal{B}}$ and one endpoint in $U^{\mathcal{R}}$, we directly get that $C \subseteq U^{\mathcal{B}}$ or $C \subseteq U^{\mathcal{R}}$. If $C \subseteq U^{\mathcal{B}}$, then $C \cap Q'^{\mathcal{R}} = \emptyset$ and if $C \subseteq U^{\mathcal{R}}$, then $C \cap Q'^{\mathcal{B}} = \emptyset$.

Ruling property: Let $v \in V(H')$. From the way we defined H', it directly follows that every edge in the forest F is also contained in H' i.e., $E(F) \subseteq E(H')$. As $\mathrm{root}_F(v) \in Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}$, it therefore follows that

$$d_{H',\mathrm{del}}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, v) \leq d_{H',\mathrm{del}}(\mathrm{root}_F(v), v) \leq \mathrm{del}(\mathrm{root}_F(v)) + d_F(v).$$

The first property of the distance oracle directly states that $\mathrm{del}(\mathrm{root}_F(v)) + d_F(v) \leq (1+\varepsilon)d_{H,\mathrm{del}}(v)$ and therefore combining the inequalities implies

$$\begin{aligned} &d_{H',\mathrm{del}}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, v) \leq (1+\varepsilon)d_{H,\mathrm{del}}(v) \\ &\leq (1+\varepsilon)^{2(0+1)}d_{H,\mathrm{del}}(Q,v) + \left(\sum_{j=1}^{0}(1+\varepsilon)^{2j}\right)3D, \end{aligned}$$

as needed.

Good property: We have $V^{good} = \{v \in V(H) \colon r(v) = 0\}$ and for every node v with $r(v) = 0$ it trivially holds that $B_{H'}(v, r(v)) = B_H(v, r(v))$.

Bad property: For the deterministic case, note that by definition $i = 0$ implies $\sum_{v \in V(H)} \mu(v)r(v) = 0$. Hence, for every $v \in V(H)$ with $r(v) > 0$, we have $\mu(v) = 0$. As $V^{bad} = \{v \in V(H) \colon r(v) > 0\}$, we therefore have $\mu(V^{bad}) = 0 = (1 - \frac{1}{2^0})O(\sum_{v \in V(H)} \mu(v)r(v)/D)$. For the randomized case, $v \in V^{bad}$ implies $r(v) > 0$, but then $P[v \in V^{bad}] = 1 \leq (\frac{1}{2^0} + \frac{1-2^{-0}}{D})O(r(v))$.

Recursive Step: Now, assume $i > 0$. We compute the sets $U^{\mathcal{R}}$ and $U^{\mathcal{B}}$ in the same way as in the base case. The recursive step is either a *red step* or a *blue step*. In the randomized version, we flip a fair coin to decide whether the recursive step is a red step or a blue step. In the deterministic version, the recursive step is a red step if $\sum_{u \in U^{\mathcal{R}}} \mu(u)r(u) \geq \sum_{u \in U^{\mathcal{B}}} \mu(u)r(u)$ and otherwise it is a blue step. Set $\mathcal{A} = \mathcal{R}$ and $\bar{\mathcal{A}} = \mathcal{B}$ if the step is a red step and otherwise set $\mathcal{A} = \mathcal{B}$ and $\bar{\mathcal{A}} = \mathcal{R}$.

We invoke the deterministic/randomized version of Theorem 9.19 with input $G_{\mathrm{T9.19}} = H, S_{\mathrm{T9.19}} = U^{\mathcal{A}}$ and $D_{\mathrm{T9.19}} = D$. After the invocation, we set $W^{\mathcal{A}} = S^{sup}_{\mathrm{T9.19}}$ and $V_i^{\mathrm{bad}} = (V^{bad})_{\mathrm{T9.19}}$.

We next perform a recursive call with inputs $H_{rec}, i_{rec}, Q_{rec} = Q^{\mathcal{R}}_{rec} \sqcup Q^{\mathcal{B}}_{rec}$, del_{rec} and R_{rec}, which are defined below. (The preferred radii, weights, and parameters D and ε will be the same)

We set $H_{rec} = H[V(H) \setminus W^{\mathcal{A}}]$. For the randomized version, we set $i_{rec} = i - 1$. For the deterministic version, it follows from the way we decide whether it is a red/blue step that

$$\begin{aligned}\sum_{v \in V(H_{rec})} \mu(v)r(v) &= \sum_{v \in V(H)} \mu(v)r(v) - \sum_{v \in W^{\mathcal{A}}} \mu(v)r(v) \\ &\leq \sum_{v \in V(H)} \mu(v)r(v) - \sum_{v \in U^{\mathcal{A}}} \mu(v)r(v) \leq \frac{1}{2} \sum_{v \in V(H)} \mu(v)r(v).\end{aligned}$$

As $i_{rec} = 1 + \lceil \log(\sum_{v \in V(H_{rec})} \mu(v)r(v)) \rceil$ if $\sum_{v \in V(H_{rec})} \mu(v)r(v) > 0$ and $i_{rec} = 0$ otherwise, it therefore follows by a simple case distinction that $i_{rec} = i - 1$ in the deterministic version as well. For $\mathcal{Z} \in \{\mathcal{R}, \mathcal{B}\}$, we define

$$Q^{\mathcal{A}}_{rec} = \{v \in V(H_{rec})\colon v \text{ has a parent } p(v) \text{ in } F, \text{ and } p(v) \in W^{\mathcal{A}}\}$$

and

$$Q^{\bar{\mathcal{A}}}_{rec} = \{v \in V(H_{rec})\colon v \text{ is a root in } F\}.$$

For every $v \in Q_{rec} := Q^{\mathcal{R}}_{rec} \sqcup Q^{\mathcal{B}}_{rec}$, we define

$$\mathrm{del}_{rec}(v) = (1+\varepsilon)(\mathrm{del}(\mathrm{root}_F(v)) + d_F(v)) + 3D.$$

Finally, we set

$$R_{rec} = (1+\varepsilon)^2 R + 3D.$$

We have to verify that for every $v \in V(H_{rec})$, $d_{H_{rec},\mathrm{del}_{rec}}(Q_{rec}, v) \leq R_{rec}$, as otherwise the input is invalid. We actually show a stronger property, namely that for every $v \in V(H_{rec})$,

$$d_{H_{rec},\mathrm{del}_{rec}}(Q_{rec}, v) \leq (1+\varepsilon)^2 d_{H,\mathrm{del}}(Q, v) + 3D \leq (1+\varepsilon)^2 R + 3D.$$

To that end, we consider a simple case distinction based on whether the entire path from v to $\mathrm{root}_F(v)$ in F is contained in H_{rec} or not. If yes, then $d_{H_{rec}}(\mathrm{root}_F(v), v) \leq d_F(v)$ and as $\mathrm{del}_{rec}(\mathrm{root}_F(v)) = (1+\varepsilon)\mathrm{del}(\mathrm{root}_F(v)) + 3D$, we get

$$\begin{aligned}d_{H_{rec},\mathrm{del}_{rec}}(Q_{rec}, v) &\leq \mathrm{del}_{rec}(\mathrm{root}_F(v)) + d_F(v) && (9.5)\\ &= (1+\varepsilon)\mathrm{del}(\mathrm{root}_F(v)) + 3D + d_F(v) && (9.6)\\ &\leq (1+\varepsilon)(\mathrm{del}(\mathrm{root}_F(v)) + d_F(v)) + 3D && (9.7)\\ &\leq (1+\varepsilon)^2 d_{H,\mathrm{del}}(Q, v) + 3D. && (9.8)\end{aligned}$$

It remains to consider the case that the path from v to $\mathrm{root}_F(v)$ in F is not entirely contained in H_{rec}. Starting from v, let y be the first node such that y is contained in H_{rec} but y's parent in F is not. By definition, $y \in Q^{\mathcal{A}}_{rec}$ and $\mathrm{del}_{rec}(y) = (1+\varepsilon)(\mathrm{del}(\mathrm{root}_F(y)) + d_F(y)) + 3D$. We get

$$\begin{aligned}
&d_{H_{rec},\mathrm{del}_{rec}}(Q_{rec}, v) \le d_{H_{rec}}(v, y) + \mathrm{del}_{rec}(y)\\
&= d_{H_{rec}}(v, y) + (1+\varepsilon)(\mathrm{del}(\mathrm{root}_F(y)) + d_F(y)) + 3D\\
&\le (1+\varepsilon)(\mathrm{del}(\mathrm{root}_F(v)) + d_F(y) + d_{H_{rec}}(v, y)) + 3D\\
&\le (1+\varepsilon)(\mathrm{del}(\mathrm{root}_F(v)) + d_F(v)) + 3D\\
&\le (1+\varepsilon)^2 d_{H,\mathrm{del}}(Q, v) + 3D. \qquad (9.9)
\end{aligned}$$

□

We verified that the provided input is correct. We denote with $Q'^{\mathcal{R}}_{rec}, Q'^{\mathcal{B}}_{rec}, H'_{rec}, V^{good}_{rec}$ and V^{bad}_{rec} the output produced by the recursive call.

We now describe the final output. We set

$$Q'^{\mathcal{A}} := Q^{\mathcal{A}} \cap U^{\mathcal{A}} \subseteq Q^{\mathcal{A}}$$

and

$$Q'^{\bar{\mathcal{A}}} := Q'^{\bar{\mathcal{A}}}_{rec} \subseteq Q^{\bar{\mathcal{A}}}_{rec} \subseteq Q^{\bar{\mathcal{A}}}.$$

Next, we define the output graph H'. To that end, we first define the edge set

$$E_{bridge} = \{\{v, p(v)\} \in E(H) \colon v \in Q'^{\mathcal{A}}_{rec},\ p(v) \text{ is the parent of } v \text{ in } F\}.$$

Note that each edge in E_{bridge} has one endpoint in $V(H_{rec}) = V(H) \setminus W^{\mathcal{A}}$ and one endpoint in $W^{\mathcal{A}}$.

We now define

$$E(H') = E(H'_{rec}) \sqcup E(H[W^{\mathcal{A}}]) \sqcup E_{bridge}.$$

Finally, we set $V^{bad} = V^{bad}_{rec} \cup V^{bad}_i$ and $V^{good} = V(H) \setminus V^{bad}$.

We next show that the output satisfies all the required properties.

Separation property: Let C be a connected component of H'. We have to show that $Q'^{\mathcal{R}} \cap C = \emptyset$ or $Q'^{\mathcal{B}} \cap C = \emptyset$. For the sake of contradiction, assume there exists $q'^{\mathcal{A}} \in Q'^{\mathcal{A}} \cap C$ and $q'^{\bar{\mathcal{A}}} \in Q'^{\bar{\mathcal{A}}} \cap C$. Consider an arbitrary path P from $q'^{\mathcal{A}}$ to $q'^{\bar{\mathcal{A}}}$ in C.

As $q'^{\mathcal{A}} \in W^{\mathcal{A}}$ and $q'^{\bar{\mathcal{A}}} \in V(H_{rec}) = V(H) \setminus W^{\mathcal{A}}$, the path P contains at least one edge in E_{bridge}. Let $e = \{v, p(v)\}$ be the first edge in E_{bridge} that one encounters on the path P starting from $q'^{\bar{\mathcal{A}}}$.

We have, $q'^{\bar{\mathcal{A}}} \in Q'^{\bar{\mathcal{A}}} = Q'^{\bar{\mathcal{A}}}_{rec}$ and $v \in Q'^{\mathcal{A}}_{rec}$. Moreover, $q'^{\bar{\mathcal{A}}}$ and v are in the same connected component in H'_{rec}, a contradiction with the separation property of the recursive call.

Ruling property: Let $v \in V(H)$. We have to show that

$$\begin{aligned}
&d_{H',\mathrm{del}}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, v)\\
&\leq (1+\varepsilon)^{2(i+1)} d_{H,\mathrm{del}}(Q, v) + \left(\sum_{j=1}^{i} (1+\varepsilon)^{2j}\right) 3D.
\end{aligned}$$

First, we consider the case $v \in W^{\mathcal{A}}$. The guarantees of Theorem 9.19 implies the existence of a vertex $u \in U^{\mathcal{A}}$ with $d_{H[W^{\mathcal{A}}]}(v, u) \leq D$. We have

$$\begin{aligned}
&d_{H',\mathrm{del}}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, v) \leq d_{H',\mathrm{del}}(\mathrm{root}_F(u), v)\\
&\leq d_{H',\mathrm{del}}(\mathrm{root}_F(u), u) + D\\
&\leq \mathrm{del}(\mathrm{root}_F(u)) + d_F(u) + D\\
&\leq (1+\varepsilon) d_{H,\mathrm{del}}(Q, u) + D\\
&\leq (1+\varepsilon)(d_{H,\mathrm{del}}(Q, v) + D) + D\\
&\leq (1+\varepsilon) d_{H,\mathrm{del}}(Q, v) + 3D\\
&\leq (1+\varepsilon)^{2(i+1)} d_{H,\mathrm{del}}(Q, v) + \left(\sum_{j=1}^{i} (1+\varepsilon)^{2j}\right) 3D.
\end{aligned}$$

It remains to consider the case $v \in V(H_{rec})$. Recall that we already have shown in (9.9) that

$$d_{H_{rec},\mathrm{del}_{rec}}(Q_{rec}, v) \leq (1+\varepsilon)^2 d_{H,\mathrm{del}}(Q, v) + 3D.$$

By the ruling property of the recursive call, we therefore get

$$\begin{aligned}
&d_{H'_{rec},\mathrm{del}_{rec}}(Q'^{\mathcal{R}}_{rec} \cup Q'^{\mathcal{B}}_{rec}, v)\\
&\leq (1+\varepsilon)^{2(i_{rec}+1)} d_{H_{rec},\mathrm{del}_{rec}}(Q_{rec}, v) + \left(\sum_{j=1}^{i_{rec}} (1+\varepsilon)^{2j}\right) 3D\\
&\leq (1+\varepsilon)^{2(i_{rec}+1)} ((1+\varepsilon)^2 d_{H,\mathrm{del}}(Q, v) + 3D)\\
&+ \left(\sum_{j=1}^{i_{rec}+1} (1+\varepsilon)^{2j}\right) 3D
\end{aligned}$$

$$\leq (1+\varepsilon)^{2(i+1)} d_{H,\mathrm{del}}(Q,v) + \left(\sum_{j=1}^{i} (1+\varepsilon)^{2j}\right) 3D.$$

Let $q'_{rec} \in Q'^{\mathcal{R}}_{rec} \cup Q'^{\mathcal{B}}_{rec}$ with $d_{H'_{rec},\mathrm{del}_{rec}}(q'_{rec}, v) = d_{H'_{rec},\mathrm{del}_{rec}}(Q'^{\mathcal{R}}_{rec} \cup Q'^{\mathcal{B}}_{rec}, v)$. First, consider the case that $q'_{rec} \in Q'^{\bar{\mathcal{A}}}_{rec}$ and therefore also $q' \in Q'^{\bar{\mathcal{A}}}$. We have

$$\begin{aligned}
&d_{H',\mathrm{del}}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, v) \leq d_{H',\mathrm{del}}(q'_{rec}, v)\\
&\leq d_{H'_{rec},\mathrm{del}_{rec}}(q'_{rec}, v)\\
&\leq (1+\varepsilon)^{2(i+1)} d_{H,\mathrm{del}}(Q,v) + \left(\sum_{j=1}^{i} (1+\varepsilon)^{2j}\right) 3D,
\end{aligned}$$

as needed. It remains to consider the case that $q'_{rec} \in Q'^{\mathcal{A}}_{rec}$. In particular, this implies that q'_{rec} has a parent p in F which is contained in $W^{\mathcal{A}}$. We have

$$\begin{aligned}
&\mathrm{del}_{rec}(q'_{rec}) = (1+\varepsilon)(\mathrm{del}(\mathrm{root}_F(q'_{rec})) + d_F(q'_{rec})) + 3D\\
&= (1+\varepsilon)(\mathrm{del}(\mathrm{root}_F(p)) + d_F(p) + \ell(q'_{rec}, p)) + 3D\\
&\geq (1+\varepsilon)(\mathrm{del}(\mathrm{root}_F(p)) + d_F(p)) + 3D + \ell(q'_{rec}, p)\\
&\geq (1+\varepsilon) d_{H,\mathrm{del}}(Q,p) + 3D + \ell(q'_{rec}, p)\\
&\geq d_{H',\mathrm{del}}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, p) + \ell(q'_{rec}, p)\\
&\geq d_{H',\mathrm{del}}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, q'_{rec})
\end{aligned}$$

and therefore

$$\begin{aligned}
&d_{H',\mathrm{del}}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, v) \leq d_{H',\mathrm{del}}(Q'^{\mathcal{R}} \cup Q'^{\mathcal{B}}, q'_{rec}) + d_{H'}(q'_{rec}, v)\\
&\leq \mathrm{del}_{rec}(q'_{rec}) + d_{H'_{rec}}(q'_{rec}, v)\\
&= d_{H'_{rec},\mathrm{del}_{rec}}(Q'^{\mathcal{R}}_{rec} \cup Q'^{\mathcal{B}}_{rec}, v)\\
&\leq (1+\varepsilon)^{2(i+1)} d_{H,\mathrm{del}}(Q,v) + \left(\sum_{j=1}^{i} (1+\varepsilon)^{2j}\right) 3D,
\end{aligned}$$

as needed.

Good property: Let $v \in V^{good}$. We have to show that $B_{H'}(v, r(v)) = B_H(v, r(v))$. As $v \in V^{good}$, it holds that $v \notin V_i^{\mathrm{bad}} \cup V_{rec}^{bad}$. As $v \notin V_i^{\mathrm{bad}}$, we either have $B_H(v, r(v)) \subseteq W^{\mathcal{A}}$

or $B_H(v, r(v)) \subseteq V(H) \setminus W^{\mathcal{A}} = V(H_{rec})$. If $B_H(v, r(v)) \subseteq W^{\mathcal{A}}$, then it follows from $E(H[W^{\mathcal{A}}]) \subseteq E(H')$ that $B_{H'}(v, r(v)) = B_H(v, r(v))$. If $B_H(v, r(v)) \subseteq V(H_{rec})$, then $B_H(v, r(v)) = B_{H_{rec}}(v, r(v))$. As $v \notin V_{rec}^{bad}$, it follows that $B_{H_{rec}}(v, r(v)) = B_{H'_{rec}}(v, r(v))$ from the good property of the recursive call. As $E(H'_{rec}) \subseteq E(H')$, it therefore follows that $B_{H'}(v, r(v)) = B_H(v, r(v))$, as needed.

Bad property: We start with the deterministic version. We have

$$\begin{aligned}
&\mu(V^{bad}) \leq \mu(V_i^{bad}) + \mu(V_{rec}^{bad}) \\
&\leq \frac{1}{2} O\left(\sum_{v \in V(H)} \mu(v) r(v)/D\right) \\
&+ \left(1 - \frac{1}{2^{i_{rec}}}\right) O\left(\sum_{v \in V(H_{rec})} \mu(v) r(v)/D\right) \\
&\leq \frac{1}{2} O\left(\sum_{v \in V(H)} \mu(v) r(v)/D\right) \\
&+ \frac{1}{2}\left(1 - \frac{1}{2^{i_{rec}}}\right) O\left(\sum_{v \in V(H)} \mu(v) r(v)/D\right) \\
&= \left(1 - \frac{1}{2^i}\right) O\left(\sum_{v \in V(H)} \mu(v) r(v)/D\right),
\end{aligned}$$

as needed. Now, we analyze the randomized version. Let $v \in (H)$. We have

$$\begin{aligned}
&\mathrm{P}[v \in V^{bad}] \leq \mathrm{P}[v \in V_i^{\mathrm{bad}}] + \mathrm{P}[v \in V_{rec}^{bad}] \\
&\leq \frac{1}{2} O(r(v)/D) + \mathrm{P}[v \in V_{rec}^{bad} | v \in V(H_{rec})] \cdot \mathrm{P}[v \in V(H_{rec})] \\
&\leq \frac{1}{2} O(r(v)/D) + \left(\frac{1}{2^{i_{rec}}} + \frac{1 - 2^{-i_{rec}}}{D}\right) O(r(v)) \cdot \frac{1}{2} \\
&= \frac{1}{2} O(r(v)/D) + \left(\frac{1}{2^i} + \frac{\frac{1}{2} - 2^{-i}}{D}\right) O(r(v)) \\
&= \left(\frac{1}{2^i} + \frac{1 - 2^{-i}}{D}\right) O(r(v)),
\end{aligned}$$

as desired.

9.4.3 Corollaries

Next, we present three simple corollaries of the main clustering result Theorem 9.30. First, Theorem 9.33 informally states that one can efficiently compute a so-called padded low-diameter partition.

Corollary 9.33. *Consider the following problem on a weighted input graph G. The input consists of a global parameter D and in the deterministic version each node $v \in V(G)$ additionally has a weight $\mu(v)$.*

The output consists of a partition $\mathcal{C}$ of G together with two sets $V^{good} \sqcup V^{bad} = V(G)$ such that

1. *the diameter of $\mathcal{C}$ is $O(D\log^3(n))$,*
2. *for every $v \in V^{good}$, $B_G(v, D) \subseteq C$ for some $C \in \mathcal{C}$,*
3. *in the deterministic version, $\mu(V^{bad}) \leq 0.1 \cdot \mu(V(G))$,*
4. *and in the randomized version, for every $v \in V(G)$: $P[v \in V^{bad}] \leq 0.1$.*

There is an algorithm that solves the problem above in $O(\log^3(n))$ steps, performing all oracle calls with precision parameter $\varepsilon = \frac{1}{\log^2(n)}$ and distance parameter no larger than $O(\log^3(n)D)$.

Proof. We invoke Theorem 9.30 with input $H_{\text{T9.30}} = G$, $r(v)_{\text{T9.30}} = D$ and $\mu_{\text{T9.30}}(v) = \mu(v)$ for every $v \in V(G)$, $Q_{\text{T9.30}} = V(G)$ and $\text{del}_{\text{T9.30}}(v) = 0$ for every $v \in V(G)$, $R_{\text{T9.30}} = 0$, $D_{\text{T9.30}} = c \cdot \log(n) \cdot D$ for a sufficiently large constant c and $\varepsilon_{\text{T9.30}} = \frac{1}{\log^2(n)}$.

The input is clearly valid. In particular, $d_{G,\text{del}_{\text{T9.30}}}(Q_{\text{T9.30}}, v) = d_G(V(G), v) = 0$.

Let $\mathcal{C}$ denote the output partition of G and $V^{\text{good}} \sqcup V^{\text{bad}} = V(G)$ the two sets returned by the algorithm. According to the first two properties of Theorem 9.30, for every cluster $C \in \mathcal{C}$ there exists a node $v_C \in \mathcal{C}$ such that for every $v \in C$

$$d_{G[C]}(v_C, v) \leq O(\log^2 n)D_{\text{T9.30}} = O(\log^3 n)D$$

and therefore the diameter of $\mathcal{C}$ is $O(D\log^3 n)$. Moreover, the third property of Theorem 9.30 together with $r(v)_{\text{T9.30}} = D$ implies that $B_G(v, D) \subseteq C$ for some $C \in \mathcal{C}$. In the deterministic version, the fourth property of Theorem 9.30 implies

$$\begin{aligned}\mu(V^{\text{bad}}) &= O(\log(n) \cdot \sum_{v\in V(G)} \mu_{\text{T9.30}}(v)r(v)_{\text{T9.30}}/D_{\text{T9.30}}) \\ &= O(1/c)\mu(V) \leq 0.1\mu(V)\end{aligned}$$

for c being sufficiently large.

In the randomized version, the fourth property of Theorem 9.30 implies for every $v \in V(G)$ that

$$P[v \in V^{\text{bad}}] = O(\log(n) r_{\text{T9.30}}(v)/D_{\text{T9.30}}) = O(1/c) \leq 0.1$$

for c being sufficiently large.

The runtime bound directly follows from the runtime bound of Theorem 9.30.

□

Next, Theorem 9.34 asserts that we can compute a so-called sparse cover. The result follows from Theorem 9.33 together with the well-known multiplicative weights update method. We use Theorem 9.34 to efficiently compute an ℓ_1-embedding in Section 9.5.

Theorem 9.34. *Consider the following problem on a weighted input graph G. The input consists of a global parameter D.*

The output consists of a cover $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_t\}$ for some $t = O(\log n)$, with the hidden constant independent of D, such that each $\mathcal{C}_i$ is a partition of G with diameter $O(D \log^3 n)$ and for every node $v \in V(G)$, $|\{i \in [t] \colon B_G(v, D) \subseteq C$ for some $C \in \mathcal{C}_i\}| \geq 2t/3$.

There is an algorithm that solves the problem above in $O(\log^4(n))$ steps using the oracle $\mathcal{O}^{Dist}$, performing all oracle calls with precision parameter $\varepsilon = \frac{1}{\log^2(n)}$ and distance parameter no larger than $O(\log^3(n)D)$.

Proof. We set $t = \lceil c \cdot \log(n) \rceil$ for c being a sufficiently large constant. At the beginning, we set $\mu_1(v) = 1$ for every $v \in V(G)$.

In the i-th step, for $1 \leq i \leq t$, we apply Theorem 9.33 with input $D_{9.33} = D$ and $\mu_{9.33}(v) = \mu_i(v)$ for every node $v \in V(G)$. Let $\mathcal{C}_i$ denote the partition and V_i^{good} the set returned by the i-th invocation. For every $v \in V(G)$ and $i \in [t]$, we set $\mu_{i+1}(v) = \mu_i(v)$ if $v \in V_i^{\text{good}}$ and otherwise we set $\mu_{i+1}(v) = 2\mu_i(v)$. In the end, the algorithm returns the cover $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_t\}$. This finishes the description of the algorithm.

It remains to argue its correctness. All the properties from the statement directly follow from Theorem 9.33, except for the property that each node $v \in V(G)$ satisfies $|\{i \in [t] \colon B_G(v, D) \subseteq C$ for some $C \in \mathcal{C}_i\}| \geq t/2$. To show this, it suffices to show that $|\{i \in [t] \colon v \in V_i^{\text{good}}\}| \geq t/2$. Note that $\mu_1(V(G)) = n$ and $\mu_{t+1}(V(G)) = \mu_i(V(G)) + \mu_i(V(G) \setminus V_i^{\text{good}}) \leq 1.1\mu_i(V(G))$. Hence, $\mu_{t+1}(V(G)) \leq n \cdot 1.1^t$. On the other hand, for every $v \in V(G)$,

$$\mu_{t+1}(V(G)) \geq \mu_{t+1}(v) \geq 2^{t - |\{i \in [t] \colon v \in V_i^{\text{good}}\}|}.$$

Therefore,

$$2^{t-|\{i\in[t]\colon v\in V_i^{\text{good}}\}|} \le n \cdot 1.1^t,$$

which directly implies $|\{i \in [t]\colon v \in V_i^{\text{good}}\}| \ge 2t/3$ for c being a sufficiently large constant. □

The next corollary is the main building block for efficiently computing low-stretch spanning trees in Section 9.6.

Corollary 9.35. *Consider the following problem on a weighted input graph G. The input consists of the following.*

1. *A weighted subgraph $H \subseteq G$.*
2. *In the deterministic version, each edge $e \in E(H)$ has a weight $\mu(e)$.*
3. *There is a set of center nodes $Q \subseteq V(H)$, with each center node $q \in Q$ having a delay $del(q) \ge 0$.*
4. *There is a parameter R such that for every $v \in V(H)$ we have $d_{H,del}(Q, v) \le R$.*
5. *There is a precision parameter $\varepsilon \in [0, 1]$.*

The output consists of a partition $\mathcal{C}$ of H and a set $Q' \subseteq Q$. Let E^{bad} denote the set consisting of those edges in $E(H)$ whose two endpoints are in different clusters in $\mathcal{C}$. The output satisfies

1. *each cluster $C \in \mathcal{C}$ contains exactly one node $q'_C \in Q'$,*
2. *for each $C \in \mathcal{C}$ and $v \in C$, we have $d_{H[C],del}(q'_C, v) \le d_{H,del}(Q, v) + \varepsilon R$,*
3. *in the deterministic version, $\mu(E^{bad}) = O\left(\frac{\log^3(n)}{\varepsilon R}\right) \cdot \sum_{e\in E(H)} \mu(e)\ell(e)$,*
4. *and in the randomized version, for every $e \in E(H)$: $Pr[e \in E^{bad}] = O\left(\frac{\log^3(n)}{\varepsilon R}\right) \cdot \ell(e)$.*

There is an algorithm that solves the problem above in $O(\log^3 n)$ steps, performing all oracle calls with precision parameter $\varepsilon' = \Omega\left(\frac{\varepsilon}{\log^2(n)}\right)$ and distance parameter no larger than $2R$.

Proof. We pretend for a moment that the actual weighted input graph is not G but instead the subdivided graph G_{sub}. We now invoke Theorem 9.30 with the following input:

1. $H_{\text{T9.30}} = H_{sub}$
2. For every $e \in E(H)$ and the respective $v_e \in V(H_{sub})$, we set $r_{\text{T9.30}}(v_e) = \ell(e)$ and in the deterministic version $\mu_{\text{T9.30}}(v_e) = \ell(e)$.

3. For every $v \in V(H)$, we set $r_{\text{T9.30}}(v) = 0$ and in the deterministic version $\mu_{\text{T9.30}}(v) = 0$.
4. $R_{\text{T9.30}} = R, Q_{\text{T9.30}} = Q$ and $\text{del}_{\text{T9.30}} = \text{del}$
5. $D_{\text{T9.30}} = \frac{c_1}{\log^2(n)}\varepsilon R$ and $\varepsilon_{\text{T9.30}} = \frac{c_2}{\log^2(n)}\varepsilon$ for some small enough constants $c_1, c_2 > 0$.

We have to verify that the input is valid, namely that for every $v \in V(H_{sub}), d_{H_{sub},\text{del}}(Q, v) \leq R$. This directly follows from the way H_{sub} is constructed from H together with the fact that for every $v \in V(H)$, $d_{H,\text{del}}(Q, v) \leq R$. Let $\mathcal{C}$ be the partition of H that one obtains from the partition $(\mathcal{C})_{\text{T9.30}}$ of H_{sub} by removing from each cluster $C \in \mathcal{C}$ all the nodes that are not contained in $V(H)$. The final output is $\mathcal{C}$ and $Q' = (Q')_{\text{T9.30}}$.

We have to verify that the output satisfies all the properties. The property that each cluster $C \in \mathcal{C}$ contains exactly one node $q'_C \in Q'$ follows from the way we obtained $\mathcal{C}$ from $(\mathcal{C})_{\text{T9.30}}$, the fact that $Q' \subseteq V(H)$ and the fact that each cluster in $(\mathcal{C})_{\text{T9.30}}$ contains exactly one node in Q'. Next, consider any $C \in \mathcal{C}$ and $v \in C$. Let $C_{sub} \in (\mathcal{C})_{\text{T9.30}}$ with $v \in C_{sub}$. We have

$$\begin{aligned}
&d_{H[C],\text{del}}(q'_C, v) \leq d_{H_{sub}[C_{sub}],\text{del}}(q'_C, v) \\
&\leq (1 + \varepsilon_{\text{T9.30}})^{O(\log^2 n)} d_{H_{sub},\text{del}}(Q, v) + O(\log^2 n) D_{\text{T9.30}} \\
&\leq \mathrm{e}^{c_2 \varepsilon O(1)} d_{H_{sub},\text{del}}(Q, v) + O(1) c_1 \varepsilon R \\
&\leq d_{H_{sub},\text{del}}(Q, v) + \varepsilon R,
\end{aligned}$$

where the last inequality follows from the fact that c_1 and c_2 are small enough. Next, we verify the third property. It follows from the third property of Theorem 9.30 that for every $e \in E^{bad}$, $v_e \in V^{bad}_{\text{T9.30}}$. Hence, we obtain from the fourth property of Theorem 9.30 that

$$\begin{aligned}
\mu(E^{bad}) &\leq \mu_{\text{T9.30}}(V^{bad}_{\text{T9.30}}) \\
&= O(\log(n) \cdot \sum_{v \in V(H_{sub})} \mu_{\text{T9.30}}(v) \cdot (r(v))_{\text{T9.30}} / D_{\text{T9.30}}) \\
&= O\left(\frac{\log^3(n)}{\varepsilon R}\right) \cdot \sum_{e \in E(H)} \mu(e)\ell(e).
\end{aligned}$$

For the randomized version, we have for every $e \in E(H)$ that

$$\begin{aligned}
&\text{P}[e \in E^{bad}] \leq \text{P}[v_e \in V^{bad}_{\text{T9.30}}] \\
&= O(\log(n) \cdot r_{\text{T9.30}}(v_e)/D_{\text{T9.30}}) = O\left(\frac{\log^3(n)}{\varepsilon R}\right) \cdot \ell(e).
\end{aligned}$$

Running the algorithm of Theorem 9.30 takes $O(\log^3 n)$ steps, with all oracle calls having precision parameter $\varepsilon_{\text{T9.30}} = \Omega\left(\frac{\varepsilon}{\log^2(n)}\right)$ and distance parameter no larger than $(1+\varepsilon_{\text{T9.30}})^{O(\log^2 n)}R + O(\log^2 n)D_{\text{T9.30}} \leq 2R$, assuming that the actual weighted input graph is G_{sub} instead of G. However, due to Theorem 9.28, the same holds true, up to a constant factor in the number of steps, with the actual weighted input graph being G. This finishes the proof.

□

9.5 Deterministic Metric Embedding

In this section, we show how to deterministically compute an ℓ_1-embedding of a weighted input graph. Note that in this section we assume that the input graph is connected. The embedding problem is defined as follows.

Definition 9.36 (ℓ_1 embedding)**.** *Let G be a weighted and connected graph. An α-approximate ℓ_1-embedding of G into k dimensions is a function that maps each node v to a k-dimensional vector x_v such that for any two nodes $u, v \in V(G)$ we have*

$$||x_u - x_v||_1 \leq d_G(u,v) \leq \alpha||x_u - x_v||_1.$$

For our result, we also need an oracle for computing potentials; the dual problem of transshipment which is a generalization of shortest path. Fortunately, the work of [274] that we use to implement the distance oracle for our parallel and distributed results also constructs the following potential oracle as a byproduct with the same parallel/distributed complexity.

Definition 9.37 (Potential Oracle $\mathcal{O}_\varepsilon^{Pot}$)**.** *The input is a non-empty set $S \subseteq V(G)$. The output is a function φ assigning each node $v \in V(G)$ a value $\varphi(v) \geq 0$ such that*

1. *$\varphi(s) = 0$ for every $s \in S$,*
2. *$|\varphi(u) - \varphi(v)| \leq d_G(u,v)$ for every $u, v \in V(G)$, and*
3. *$(1+\varepsilon)\varphi(v) \geq d_G(S,v)$ for every $v \in V(G)$.*

As an application of our strong-clustering results for weighted graphs, we can now show a distributed deterministic algorithm that embeds a connected weighted graph in ℓ_1 space with polylogarithmic stretch. The arguments in this section are similar to those in [52, Section 7]. We use our clustering result Theorem 9.33 with $O(\log n)$ different distance scales $D_i = 2^i$ to get $O(\log n)$ partitions of G for all different scales. Next, we use these partitions to define the final embedding.

Theorem 9.38. *Let G be a weighted and connected graph. There is an algorithm that computes an $O(\log^4 n)$-approximate ℓ_1-embedding of G into $O(\log^3 n)$-dimensional ℓ_1*

space. The algorithm takes $O(\log^5 n)$ steps with access to the distance oracle $\mathcal{O}^{Dist}$, performing all oracle calls with precision parameter $\varepsilon = 1/\log^2 n$.

The proof below uses error-correcting codes. We note that the proof works also without them, the only difference then is that the ℓ_1 embedding is only $O(\log^5 n)$-approximate.

Proof. The algorithm computes $O(\log^2 n)$ partitions. For each partition, an embedding into $O(\log n)$ dimensions is computed. Combining these $O(\log^2 n)$ embeddings then results in an embedding into $O(\log^3 n)$ dimensions. The final embedding is then obtained from this embedding by scaling each entry by a factor determined later.

We first describe how the $O(\log^2 n)$ partitions are computed.

For every D in the set $\{2^i : i \in \mathbb{Z}, |i| \leq \lceil \log(n \cdot \max_{e \in E(G)} \ell(e)) \rceil\}$ the algorithm invokes Theorem 9.34 with input $D_{T9.34} = D$. Note that there are $O(\log n)$ such choices for D. The output is a cover $\{\mathcal{C}_1^D, \mathcal{C}_2^D, \ldots, \mathcal{C}_t^D\}$ with $t = O(\log n)$. Hence, we have $O(\log n)$ covers with each cover consisting of $O(\log n)$ partitions for a total of $O(\log^2 n)$ partitions.

For each partition $\mathcal{C}$, we compute an embedding of $V(G)$ into $O(\log n)$ dimensional ℓ_1-space as follows.

First, each cluster $C \in \mathcal{C}$ computes a bit string $s^C \in \{0,1\}^{10b}$ for $b = O(\log n)$ such that for every other cluster $C' \neq C$ in $\mathcal{C}$, $|\{i \in [10b] : s_i^C = 1, s_i^{C'} = 0\}| \geq b$. Each cluster C can compute such a string internally by applying an error-correcting function f to a b-bit identifier of an arbitrary node of C. It is well-known that such an efficiently computable function f exists.

Theorem 9.39 (cf. [242]). *There exists a function f from the set of b-bit strings to a set of $10b$-bit strings with the following properties. For every two b-bit strings s_1, s_2, the strings $f(s_1), f(s_2)$ differ on at least b positions. Moreover, the value of $f(\cdot)$ can be computed in* $\text{poly}(b)$ *time.*

For every $i \in [10b]$, we now define

$$S_i = \{v \in V(G) : v \text{ is contained in a cluster } C \in \mathcal{C} \text{ with } s_i^C = 0\}.$$

If $S_i \neq \emptyset$, then let $\varphi_i \leftarrow \mathcal{O}_1^{Pot}(S_i)$. The i-th coordinate of the embedding is set equal to φ_i, i.e., for every node v the i-th coordinate is set to $\varphi_i(v)$. If $S_i = \emptyset$, then the i-th coordinate of each node is set to 0.

This finishes the description of the embedding into $O(\log^3 n)$ dimensions. For each node $v \in V$, we denote by x_v the vector assigned to node v.

We finish by proving that there are constants A_1, A_2 such that for any two nodes u, v and n large we have

$$\frac{d_G(u,v)}{A_1 \log(n)} \leq ||x_u - x_v||_1 \leq A_2 \log^3 n \cdot d_G(u,v)$$

in the following two claims.

Claim 9.40.
$$||x_u - x_v||_1 \le O(\log^3 n) \cdot d_G(u,v)$$

Proof. If Φ stands for the set of all $O(\log^3 n)$ potential functions used in the definition of the embedding, we have

$$||x_u - x_v||_1 \le O(\log^3 n) \cdot \max_{\varphi \in \Phi} |\varphi(u) - \varphi(v)| = O(\log^3 n) \cdot d_G(u,v)$$

where the second bound follows from the second property in Theorem 9.37.

□

Claim 9.41.
$$||x_u - x_v||_1 \ge d_G(u,v) \cdot \Omega(1/\log n)$$

Proof. Consider any $u, v \in V(G)$. Let i be such that $D_i < d(u,v)/q \le D_{i+1}$ where $q = O(\log^3 n)$ is such that Theorem 9.34 outputs clusters of diameter at most qD. Such an i has to exist.

Consider the computed cover $\{\mathcal{C}_1^{D_i}, \mathcal{C}_2^{D_i}, \dots, \mathcal{C}_t^{D_i}\}$ for the distance scale D_i. Note that for at least $2t/3$ indices j we have that $B_G(u, D_i) \subseteq C$ for some $C \in \mathcal{C}_j$. The same holds for v. Hence, for at least $t/3$ indices j we have that both u and v have this property.

Fix any such partition $\mathcal{C}_j$ with this property. Recall that for at least b out of $10b$ potentials φ we defined using the partition $\mathcal{C}_j$ we have that exactly one of the nodes u, v is in the set S that defined φ via $\varphi \leftarrow \mathcal{O}_1^{Pot}(S)$. Without loss of generality, assume $u \in S$ and $v \notin S$. We have $\varphi(u) = 0$ by the first property in Theorem 9.37. On the other hand, we have $\varphi(v) \ge d_G(S, v)/2 \ge D_i/2$ by the second property in Theorem 9.37 and the fact that $B_G(v, D_i) \cap S = \emptyset$. We are getting

$$|\varphi(u) - \varphi(v)| \ge D_i/2. \tag{9.10}$$

Note that out of all potential functions we defined, (9.10) holds for at least $2t/3 \cdot b = \Omega(\log^2 n)$ of them by above discussion. This implies

$$\begin{aligned} ||x_u - x_v||_1 &\ge \Omega(\log^2 n) \cdot D_i/2 \\ &= \Omega(\log^2 n) \cdot \frac{d_G(u,v)}{4q} = \Omega(1/\log n) \cdot d_G(u,v) \end{aligned}$$

as needed. □

□

9.6 Low Stretch Spanning Trees

This section is dedicated to prove the following theorem.

Theorem 9.42 (Main Theorem)**.** *We can compute a deterministic/randomized $O(\log^5(n))$-stretch spanning tree of a weighted and connected graph G (with additional edge importance μ) in $\operatorname{poly}(\log n)$ steps, with each oracle call using distance parameter at most $O(diam(G))$ and precision parameter $\varepsilon = \Omega\left(1/\log^3(n)\right)$.*

Essentially, it states that we can deterministically compute a $\operatorname{poly}(\log n)$-stretch spanning tree with $\operatorname{poly}(\log n)$ calls to an approximate distance oracle. We start by recalling the notion of a low-stretch spanning tree.

Definition 9.43 (Deterministic Low Stretch Spanning Tree)**.** *A deterministic α-stretch spanning tree T of a weighted and connected graph G with additional edge-importance $\mu : E(G) \to \mathbb{R}_{\geq 0}$ is a spanning tree of G such that*

$$\sum_{e=\{u,v\}\in E(G)} \mu(e) d_T(u,v) \leq \alpha \sum_{e=\{u,v\}\in E(G)} \mu(e)\ell(u,v).$$

Definition 9.44 (Randomized Low Stretch Spanning Tree)**.** *A randomized α-stretch spanning tree of a weighted and connected graph G is a spanning tree T coming from a distribution $\mathcal{T}$ such that for every edge $e = \{u,v\} \in E(G)$ we have*

$$E_{T\sim\mathcal{T}}[d_T(u,v)] \leq \alpha\ell(u,v).$$

Equivalently, we may require

$$E_{T\sim\mathcal{T}}[d_T(u,v)] \leq \alpha d_G(u,v)$$

for any $u, v \in V(G)$.

The section is structured into two subsections. In Section 9.6.1 we show how to construct a so-called star decomposition that is used in Section 9.6.2 to derive Theorem 9.42. We note that the algorithm here is essentially the same as in [123, 55]. They start by computing a star decomposition, followed by recursively computing a low-stretch spanning tree in each of the clusters of the star decomposition. As we make use of oracles, it is not clear up-front that one can simultaneously recurse on vertex-disjoint subgraphs. The emphasis on this section is to formally prove that we can nevertheless implement the recursion efficiently. We refer the interested reader to the paper of [123] for a more intuitive and readable presentation of the low stretch spanning tree algorithm.

9.6.1 Star Decomposition

In this section we prove Theorem 9.46. It asserts that we can build a so-called star decomposition with $\operatorname{poly}(\log n)$ calls to an approximate distance oracle. We start with the definition of star decomposition.

Definition 9.45 ((ε, r_0, R)-Star Decomposition of G)**.** *Let G be a weighted graph, $\varepsilon > 0$, $r_0 \in V(G)$ and $R \geq 0$ such that $\max_{v \in V(G)} d_G(r_0, v) \leq R$.*
The output consists of a partition $V(G) = V_0 \sqcup V_1 \sqcup \ldots \sqcup V_k$ for some $k \geq 0$ with $r_0 \in V_0$ together with a set of edges $E^{bridge} = \{\{y_j, r_j\} \colon j \in [k]\} \subseteq E(G)$ such that for every $j \in [k]$, $y_j \in V_0$ and $r_j \in V_j$. For $j \in \{0, 1, \ldots, k\}$, we refer to r_j as the root of cluster V_j. The output has to satisfy the following conditions.

1. $\forall j \in \{0, 1, \ldots, k\} \colon \max_{v \in V_j} d_{G[V_j]}(r_j, v) \leq \frac{3}{4}R$.
2. $\forall v \in V_0 \colon d_{G[V_0]}(r_0, v) \leq (1 + \varepsilon) d_G(r_0, v)$.
3. $\forall j \in [k], v \in V_j \colon d_{G[V_0]}(r_0, y_j) + \ell(y_j, r_j) + d_{G[V_j]}(r_j, v) \leq (1 + \varepsilon) d_G(r_0, v)$.

Theorem 9.46 (Deterministic and Randomized Generalized Star Decomposition)**.** *Consider the following problem on a weighted (and connected) graph G. The input consists of the following.*

1. *A partition $V(G) = V_1 \sqcup V_2 \sqcup \ldots \sqcup V_k$ for some k.*
2. *A node $r_i \in V_i$ for every $i \in [k]$.*
3. *A number $R \geq 0$ such that $\max_{i \in [k], v \in V_i} d_{G[V_i]}(r_i, v) \leq R$.*
4. *In the deterministic version a priority $\mu(e)$ for every edge $e \in E(G)$.*
5. *A precision parameter $\varepsilon \in [0, 0.1]$.*

The output consists of the following for each $i \in [k]$: a partition $V_i = V_{i,0} \sqcup V_{i,1} \sqcup \ldots \sqcup V_{i,k_i}$ together with a set of edges E_i^{bridge}. We denote by E^{in} the set consisting of those edges in E that have both endpoints in V_i for some $i \in [k]$. Moreover, we denote by E^{out} the set consisting of those edges in E that have both endpoints in $V_{i,j}$ for some $i \in [k], j \in \{0, 1, \ldots, k_i\}$. The output satisfies

1. *for every $i \in [k]$, $(V_{i,0} \sqcup V_{i,1} \sqcup \ldots \sqcup V_{i,k_i}, E_i^{bridge})$ is an (ε, r_i, R)-star decomposition of $G[V_i]$,*
2. *in the deterministic version, $\mu(E^{in} \setminus E^{out}) = O\left(\frac{\log^3(n)}{\varepsilon R}\right) \sum_{e \in E^{in}} \mu(e) \ell(e)$,*
3. *and in the randomized version, for every $e \in E^{in}$, $Pr[e \notin E^{out}] = O\left(\frac{\log^3(n)}{\varepsilon R}\right) \ell(e)$.*

There is an algorithm that solves the problem above in $O(\log^3 n)$ steps, performing all oracle calls with precision parameter $\varepsilon' = \Omega\left(\frac{\varepsilon}{\log^2(n)}\right)$ and distance parameter no larger than R.

Proof. We define $H \subseteq G$ as the graph with $V(H) = V(G)$ and $E(H) = E^{in}$.
Let $F \leftarrow \mathcal{O}^{Dist}_{\varepsilon/100, R}(H, \{r_i \colon i \in [k]\})$. We now invoke Theorem 9.29 on the graph H with input $S_{\text{T9.29}} = \{v \in V(H) \colon d_F(v) \leq (2/3)R\}$, $\mu_{\text{T9.29}}(e) = \mu(e)$ for every edge $e \in E(H)$ and $D_{\text{T9.29}} = \frac{\varepsilon R}{100}$ and obtain as an output a set $S^{sup}_{\text{T9.29}}$.

For $i \in [k]$, we set $V_{i,0} = V_i \cap S^{sup}_{T9.29}$. We now invoke Theorem 9.35 with the following input, where we denote with $p(v)$ the parent of v in F.

1. $H_{9.35} = H[V(H) \setminus S^{sup}_{T9.29}]$
2. For every $e \in E(H_{9.35})$, $\mu_{9.35}(e) = \mu(e)$
3. $Q_{9.35} = \{v \in V(H_{9.35})\colon p(v) \in S^{sup}_{T9.29}\}$ and for every $q \in Q_{9.35}$, $\mathrm{del}(q) = d_F(q) - (2/3)R \geq 0$.
4. $R_{9.35} = R/2$
5. $\varepsilon_{9.35} = \varepsilon/10$

We have to verify that the input is valid, namely that for every $v \in V(H_{9.35})$, $d_{H_{9.35},\mathrm{del}_{9.35}}(Q_{9.35}, v) \leq R_{9.35}$. Consider an arbitrary $v \in V(H_{9.35})$ and let i such that $v \in V_i$. Then,

$$\begin{aligned}&d_{H_{9.35},\mathrm{del}_{9.35}}(Q_{9.35}, v) \leq d_F(v) - (2/3)R\\&\leq (1+\varepsilon/100)d_{G[V_i]}(r_i, v) - (2/3)R \leq R/2 = R_{9.35},\end{aligned}$$

as needed.

The output is a partition $\mathcal{C}_{9.35}$ of $H_{9.35}$ and a set $Q'_{9.35} \subseteq Q_{9.35}$.

For every $i \in [k]$, we now output $V_i = V_{i,0} \sqcup V_{i,1} \sqcup \ldots V_{i,k_i}$ such that for every $j \in [k_i]$, $V_{i,j}$ is one of the clusters in $\mathcal{C}_{9.35}$. Moreover, we define

$$E_i^{bridge} = \{\{v, p(v)\}\colon v \in V_i \cap Q'_{9.35}\}.$$

We start with verifying that $(V_{i,0} \sqcup V_{i,1} \sqcup \ldots \sqcup V_{i,k_i}, E_i^{bridge})$ is an (ε, r_i, R)-star decomposition of $G[V_i]$. First, we have to check that $E_i^{bridge} = \{\{y_{ij}, r_{ij}\}\colon j \in [k_i]\}$ for some y_{ij} and r_{ij} with $y_{ij} \in V_{i,0}$ and $r_{ij} \in V_{i,j}$.

Note that each cluster in $\mathcal{C}_{9.35}$ (and therefore each $V_{i,j}$ for $j \in [k_i]$) contains exactly one node in $Q'_{9.35}$, which we denote by r_{ij}. We have $r_{ij} \in V_{ij}$ and as $r_{ij} \in Q'_{9.35} \subseteq Q_{9.35}$, we directly get that $y_{ij} := p(r_{ij}) \in V_i \cap S^{sup}_{T9.29} =: V_{i,0}$, as desired. Next, we verify that for every $j \in \{0, 1, \ldots, k_i\}$, $\max_{v \in V_{i,j}} d_{G[V_{i,j}]}(r_{ij}, v) \leq \frac{3}{4}R$ where we set $r_{i0} = r_i$. We have

$$\begin{aligned}&\max_{v \in V_{i,0}} d_{G[V_{i,0}]}(r_i, v) = \max_{v \in V_i \cap S^{sup}_{T9.29}} d_{G[V_i \cap S^{sup}_{T9.29}]}(r_i, v)\\&\leq \max_{v \in V_i \cap S_{T9.29}} d_{G[V_i \cap S_{T9.29}]}(r_i, v) + \max_{v \in V_i \cap S^{sup}_{T9.29}} d_{G[V_i \cap S^{sup}_{T9.29}]}(S_{T9.29}, v)\\&\leq \max_{v \in S_{T9.29}} d_F(v) + \max_{v \in S^{sup}_{T9.29}} d_{H[S^{sup}_{T9.29}]}(S_{T9.29}, v)\\&\leq (2/3)R + D_{T9.29}\end{aligned}$$

$$\leq (3/4)R.$$

For $j \in [k_i]$ and $v \in V_{i,j}$, we have

$$\begin{aligned}
&d_{G[V_{i,j}]}(r_{ij}, v) \leq d_{G[V_{i,j}],\mathrm{del}_{9.35}}(r_{ij}, v) \\
&\leq (1+\varepsilon/10) d_{H_{9.35},\mathrm{del}_{9.35}}(Q_{9.35}, v) \leq (1+\varepsilon/10) R_{9.35} \leq (3/4)R.
\end{aligned}$$

Next, we verify the second property. Let $v \in V_{i,0}$. We have shown above that

$$d_{G[V_{i,0}]}(r_{i0}, v) \leq (2/3)R + D_{\mathrm{T}9.29} = (2/3)R + \frac{\varepsilon R}{100}$$

and therefore $d_{G[V_{i,0}]}(r_{i0}, v) \leq (1+\varepsilon/10) d_{G[V_i]}(r_{i0}, v)$ as long as

$$d_{G[V_i]}(r_{i0}, v) \geq \frac{(2/3)R + \frac{\varepsilon R}{100}}{1+\varepsilon/10}.$$

Therefore, it remains to consider the case

$$d_{G[V_i]}(r_{i0}, v) < \frac{(2/3)R + \frac{\varepsilon R}{100}}{1+\varepsilon/10} \leq \frac{(2/3)R}{1+\varepsilon/100},$$

which in particular implies

$$d_F(v) \leq (1+\varepsilon/100) d_{G[V_i]}(r_{i0}, v) \leq (2/3)R.$$

Thus, the entire path from r_{i0} to v in F is contained in $V_{i,0}$ and therefore

$$\begin{aligned}
&d_{G[V_{i,0}]}(r_{i0}, v) \leq d_F(v) \leq (1+\varepsilon/100) d_{G[V_i]}(r_{i0}, v) \\
&\leq (1+\varepsilon) d_{G[V_i]}(r_{i0}, v).
\end{aligned}$$

It remains to verify the third property. Consider an arbitrary $j \in [k_i]$ and $v \in V_{i,j}$. We have

$$\begin{aligned}
&d_{G[V_{i,j}]}(r_{ij}, v) = d_{G[V_{i,j}],\mathrm{del}_{9.35}}(r_{ij}, v) - \mathrm{del}_{9.35}(r_{ij}) \\
&\leq (1+\varepsilon) d_{H_{9.35},\mathrm{del}_{9.35}}(Q_{9.35}, v) - \mathrm{del}_{9.35}(r_{ij}) \\
&\leq (1+\varepsilon/10)(d_F(v) - (2/3)R) - (d_F(r_{ij}) - (2/3)R)
\end{aligned}$$

$$\begin{aligned}
&\leq d_F(v) - d_F(r_{ij}) + \frac{\varepsilon R}{10} \\
&= d_F(v) - d_F(y_{ij}) - \ell(y_{ij}, r_{ij}) + \frac{\varepsilon R}{10} \\
&\leq (1 + \varepsilon/100) d_{G[V_i]}(r_{i0}, v) - d_{G[V_i]}(r_{i0}, y_{ij}) - \ell(y_{ij}, r_{ij}) + \frac{\varepsilon R}{10} \\
&\leq d_{G[V_i]}(r_{i0}, v) - d_{G[V_i]}(r_{i0}, y_{ij}) - \ell(y_{ij}, r_{ij}) + \frac{\varepsilon R}{5}.
\end{aligned}$$

In particular,

$$\ell(y_{ij}, r_{ij}) + d_{G[V_{i,j}]}(r_{ij}, v) \leq d_{G[V_i]}(r_{i0}, v) - d_{G[V_i]}(r_{i0}, y_{ij}) + \frac{\varepsilon R}{5}.$$

Therefore,

$$\begin{aligned}
&d_{G[V_{i,0}]}(r_{i0}, y_{ij}) + \ell(y_{ij}, r_{ij}) + d_{G[V_{i,j}]}(r_{ij}, v) \\
&\leq (1 + \varepsilon/10) d_{G[V_i]}(r_{i0}, y_{ij}) + d_{G[V_i]}(r_{i0}, v) - d_{G[V_i]}(r_{i0}, y_{ij}) + \frac{\varepsilon R}{5} \\
&\leq d_{G[V_i]}(r_{i0}, v) + \frac{\varepsilon R}{2} \\
&\leq (1 + \varepsilon) d_{G[V_i]}(r_{i0}, v)
\end{aligned}$$

where the last inequality follows from $d_{G[V_i]}(r_{i0}, v) \geq \frac{d_F(v)}{1+\varepsilon/100} \geq \frac{(2/3)R}{1+\varepsilon/100} \geq R/2$.

Hence, $(V_{i,0} \sqcup V_{i,1} \sqcup \ldots \sqcup V_{i,k_i}, E_i^{bridge})$ is indeed an (ε, r_i, R)-star decomposition of $G[V_i]$.

Each edge in $E^{in} \setminus E^{out}$ either has exactly one endpoint in $S^{sup}_{\mathrm{T}9.29}$ or the two endpoints are contained in different clusters in $\mathcal{C}_{9.35}$. Therefore, by the guarantees of Theorem 9.29 and Theorem 9.35, we get in the deterministic version that

$$\begin{aligned}
\mu(E^{in} \setminus E^{out}) &= O\left(\sum_{e \in E(H)} \mu(e)\ell(e)/D_{\mathrm{T}9.29}\right) \\
&+ O\left(\frac{\log^3(n)}{\varepsilon R}\right) \cdot \sum_{e \in E(H_{9.35})} \mu(e)\ell(e) \\
&= O\left(\frac{\log^2(n)}{\varepsilon R}\right) \sum_{e \in E^{in}} \mu(e)\ell(e)
\end{aligned}$$

and in the randomized version for every $e \in E^{in}$ that

$$Pr[e \notin E^{out}] = O\left(\frac{\ell(e)}{D_{\text{T9.29}}}\right)$$
$$+ O\left(\frac{\log^3(n)}{\varepsilon R}\ell(e)\right) = O\left(\frac{\log^3(n)}{\varepsilon R}\right)\ell(e),$$

as desired.

□

9.6.2 Analysis of Low-Stretch Spanning Tree Algorithm

We are now ready to prove the main theorem of this section, Theorem 9.47. Theorem 9.42 follows as a simple corollary of it by setting $(V_1)_{\text{T9.47}} = V(G)$, letting $(r_1)_{\text{T9.47}}$ be an arbitrary node, $j_{\text{T9.47}} = O(\log \operatorname{diam}(G))$, i.e., such that $(4/3)^{j_{\text{T9.47}}} = O(\operatorname{diam}(G))$, and $\varepsilon_{\text{T9.47}} = \frac{1}{\log(n)}$.

Theorem 9.47. *Consider the following problem on a weighted (and connected) input graph G.*

1. *A partition $V(G) = V_1 \sqcup V_2 \sqcup \ldots \sqcup V_k$ for some k.*
2. *A node $r_i \in V_i$ for every $i \in [k]$.*
3. *A natural number $j \in \mathbb{N}$ such that $\max_{i\in[k],v\in V_i} d_{G[V_i]}(r_i, v) \leq (4/3)^{j-2}$*
4. *In the deterministic version a priority $\mu(e)$ for every edge $e \in E(G)$.*
5. *A precision parameter $\varepsilon \in [0, 0.1]$.*

The output is a weighted forest $F \subseteq G$ (in the randomized version coming from a distribution $\mathcal{F}$). We denote by E^{in} the set consisting of those edge in E that have both endpoints in V_i for some $i \in [k]$. The output satisfies the following.

1. *$V_1, V_2, \ldots, V_k$ are the connected components of F.*
2. *$\forall i \in [k], v \in V_i$: $d_F(r_i, v) \leq (1 + \varepsilon)^j d_{G[V_i]}(r_i, v)$*
3. *Deterministic Version: $\sum_{e=\{u,v\}\in E^{in}} \mu(e)d_F(u, v) \leq j\cdot(1+\varepsilon)^j O\left(\frac{\log^2(n)}{\varepsilon}\right) \sum_{e=\{u,v\}\in E^{in}} \mu(e)\ell(u, v)$.*
4. *Randomized Version: $\forall e = \{u, v\} \in E^{in}$: $E_{F\sim\mathcal{F}}[d_F(u, v)] \leq j \cdot (1 + \varepsilon)^j \cdot O\left(\frac{\log^2(n)}{\varepsilon}\right) \ell(u, v)$.*

We can compute the output in $(j + 1) \cdot \operatorname{poly}(\log n)$ steps, with each oracle call using precision parameter $\varepsilon' = \Omega\left(\frac{\varepsilon}{\log^2(n)}\right)$.

Proof. We prove the statement by induction on j. For $j = 1$, the third property of the input together with all edge weights being non-negative integer implies that the diameter of $G[V_i]$ is 0 for each $i \in [k]$. Hence, it is easy to verify that the forest F one obtains by calling $\mathcal{O}^{Dist}_{\varepsilon,(4/3)^{j-2}}(\{r_1, r_2, \dots, r_k\})$ satisfies all the conditions.

Now, consider an arbitrary $j > 1$ and assume that the statement holds for $j - 1$.

We first invoke Theorem 9.46 with the same input that we received (Setting $R = (4/3)^{j-2}$).

As an output, we obtain for every $i \in [k]$ a partition $V_{i,0} \sqcup V_{i,1} \sqcup \dots \sqcup V_{i,k_i}$ and a set of edges E_i^{bridge} such that $(V_{i,0} \sqcup V_{i,1} \sqcup \dots \sqcup V_{i,k_i}, E_i^{bridge})$ is a $(G[V_i], \varepsilon, r_i, R)$-star decomposition of $G[V_i]$. For $j \in \{0, 1, \dots, k_i\}$, we denote with r_{ij} the root of the cluster $V_{i,j}$.

Now, we perform a recursive call with input partition $V(G) = (V_{1,0} \sqcup V_{1,1} \sqcup \dots \sqcup V_{1,k_1}) \sqcup \dots \sqcup (V_{k,0} \sqcup V_{k,1} \sqcup \dots \sqcup V_{1,k_k})$, nodes $r_{ij} \in V_{i,j}$ for every $i \in [k], j \in \{0, 1, \dots, k_i\}$, setting $j_{rec} = j - 1$ and with the same priorities and precision parameter ε.

First, we have to verify that the input to the recursive call is valid. This requires us to show that $\max_{i \in [k], j \in [k_i], v \in V_{i,j}} d_{G[V_{i,j}]}\ (r_{ij}, v) \leq (4/3)^{j_{rec}-2} = (4/3)^{j-3}$. Consider an arbitrary $i \in [k]$. As $(V_{i,0} \sqcup V_{i,1} \sqcup \dots \sqcup V_{i,k_i}, E_i^{bridge})$ is a $(G[V_i], \varepsilon, r_i, R)$-star decomposition of $G[V_i]$, the first guarantee of Theorem 9.45 states that for every $j \in \{0, 1, \dots, k_i\}$,

$$\max_{v \in V_{i,j}} d_{G[V_{i,j}]}(r_{ij}, v) \leq \frac{3}{4}(4/3)^{j-2} = (4/3)^{j-3},$$

as desired.

Now, let F_{rec} denote the forest obtained from the recursive call. We now return the forest F that one obtains from F_{rec} by adding all the edges in $\bigcup_{i \in [k]} E_i^{bridge}$ to it. We now have to verify that F satisfies all the conditions.

We start by verifying the second condition. Consider an arbitrary $i \in [k]$ and $v \in V_i$. We have to show that $d_F(r_i, v) \leq (1+\varepsilon)^j d_{G[V_i]}(r_i, v)$. First, consider the cases that $v \in V_{i,0}$. As $r_i = r_{i0} \in V_{i,0}$, it follows from the guarantee of the recursive call that

$$\begin{aligned} d_F(r_i, v) &\leq d_{F_{rec}}(r_{i0}, v) \leq (1+\varepsilon)^{j_{rec}} d_{G[V_{i,0}]}(r_{i0}, v) \\ &\leq (1+\varepsilon)(1+\varepsilon)^{j_{rec}} d_{G[V_i]}(r_{i0}, v) = (1+\varepsilon)^j d_{G[V_i]}(r_i, v), \end{aligned}$$

where the last inequality follows from the second guarantee of Theorem 9.45. It remains to consider the case that $v \in V_{i,j}$ for some $j \in [k_i]$. According to the guarantees of the recursive call and of *Theorem* 9.45, there exists an edge $\{y_{ij}, r_{ij}\} \in E_i^{bridge}$ with $y_{ij} \in V_{i,0}$ and r_{ij} being the root of $V_{i,j}$ such that

$$d_F(r_i, v) \leq d_{F_{rec}}(r_{i0}, y_{i0}) + \ell(y_{ij}, r_{ij}) + d_{F_{rec}}(r_{ij}, v)$$

$$
\begin{aligned}
&\le (1+\varepsilon)^{j_{rec}} d_{G[V_{i,0}]}(r_{i0}, y_{ij}) \\
&+ \ell(y_{ij}, r_{ij}) + (1+\varepsilon)^{j_{rec}} d_{G[V_{i,j}]}(r_{ij}, v)) \\
&\le (1+\varepsilon)^{j_{rec}} \left(d_{G[V_{i,0}]}(r_{i0}, y_{ij}) + \ell(y_{ij}, r_{ij}) + d_{G[V_{i,j}]}(r_{ij}, v) \right) \\
&\le (1+\varepsilon)^{j_{rec}} \left((1+\varepsilon) d_{G[V_i]}(r_{i0}, v) \right) \\
&= (1+\varepsilon)^{j} d_{G[V_i]}(r_i, v),
\end{aligned}
$$

which finishes the proof of second property.

Next, we verify that F is indeed a forest and that $V_1, V_2, \ldots, V_k$ are the connected components of F. Note that the second property ensures that any two nodes in V_i are in the same connected component of F for every $i \in [k]$. Hence, to verify that F is a forest and $V_1, V_2, \ldots, V_k$ are the connected components of F, it suffices to show that F has at most $|V(G)| - k$ edges.

We have

$$
\begin{aligned}
|E(F)| &\le |E(F_{rec})| + \sum_{i=1}^{k} |E_i^{bridge}| \\
&\le |V(G)| - \sum_{i=1}^{k} (k_i + 1) + \sum_{i=1}^{k} k_i = |V(G)| - k,
\end{aligned}
$$

as desired.

We now verify the third property, which only applies to the deterministic version. we denote by E^{out} the set consisting of those edge in E that have both endpoints in $V_{i,j}$ for some $i \in [k], j \in \{0, 1, \ldots, k_i\}$. We have

$$
\begin{aligned}
&\sum_{e=\{u,v\} \in E^{in}} \mu(e) d_F(u,v) = \sum_{e=\{u,v\} \in E^{in} \setminus E^{out}} \mu(e) d_F(u,v) \\
&+ \sum_{e=\{u,v\} \in E^{out}} \mu(e) d_F(u,v) \\
&\le 2(1+\varepsilon)^j (4/3)^{j-2} \sum_{e=\{u,v\} \in E^{in} \setminus E^{out}} \mu(e) \\
&+ \sum_{e=\{u,v\} \in E^{out}} \mu(e) d_{F_{rec}}(u,v) \\
&\le 2(1+\varepsilon)^j (4/3)^{j-2} O\left(\frac{\log^2(n)}{\varepsilon (4/3)^{j-2}} \right) \sum_{e=\{u,v\} \in E^{in}} \mu(e) \ell(u,v)
\end{aligned}
$$

$$
\begin{aligned}
&+ \sum_{e=\{u,v\}\in E^{out}} \mu(e) d_{F_{rec}}(u,v) \\
&\leq O\left(\frac{\log^2(n)}{\varepsilon}\right)(1+\varepsilon)^j \sum_{e=\{u,v\}\in E^{in}} \mu(e)\ell(u,v) \\
&+ j_{rec}(1+\varepsilon)^{j_{rec}} O\left(\frac{\log^2(n)}{\varepsilon}\right) \sum_{e=\{u,v\}\in E^{out}} \mu(e)\ell(u,v) \\
&\leq j(1+\varepsilon)^j O\left(\frac{\log^2(n)}{\varepsilon}\right) \sum_{e=\{u,v\}\in E^{in}} \mu(e)\ell(u,v),
\end{aligned}
$$

as needed.

Finally we verify the fourth property, which only applies to the randomized version. Let $e = \{u, v\} \in E^{in}$ be an arbitrary edge.

$$
\begin{aligned}
&\mathrm{E}_{F\sim\mathcal{F}}[d_F(u,v)] \\
&\leq \mathrm{E}_{F\sim\mathcal{F}}[d_F(u,v)|e \notin E^{out}]Pr[e \notin E^{out}] \\
&+ \mathrm{E}_{F\sim\mathcal{F}}[d_F(u,v)|e \in E^{out}] \\
&\leq 2(1+\varepsilon)^j (4/3)^{j-2} O\left(\frac{\log^2(n)}{\varepsilon R}\right)\ell(u,v) \\
&+ j_{rec}(1+\varepsilon)^{j_{rec}}\left(\frac{\log^2(n)}{\varepsilon R}\right)\ell(u,v) \\
&\leq j \cdot (1+\varepsilon)^j \cdot O\left(\frac{\log^2(n)}{\varepsilon}\right)\ell(u,v),
\end{aligned}
$$

as desired.

It remains to analyze the running time. Running the algorithm of Theorem 9.46 takes $\mathrm{poly}(\log n)$ steps and all oracle calls use precision parameter $\varepsilon' = \Omega\left(\frac{\varepsilon}{\log^2(n)}\right)$. Performing the recursive call takes $(j_{rec} + 1)\,\mathrm{poly}(\log n)$ steps and all oracle calls use precision parameter $\varepsilon' = \Omega\left(\frac{\varepsilon}{\log^2(n)}\right)$. Hence, the algorithm overall performs $(j + 1)\,\mathrm{poly}(\log n)$ steps and uses precision parameter $\varepsilon' = \Omega\left(\frac{\varepsilon}{\log^2(n)}\right)$, as desired. □

CHAPTER 10

Local Problems on Trees: Distributed Algorithms and Descriptive Combinatorics

10.1 Introduction

This chapter studies local problems on regular trees from three different perspectives.

First, we consider the perspective of distributed algorithms. In distributed computing, the studied setup is a network of computers where each computer can only communicate with its neighbors. Roughly speaking, the question of interest in this area is which problems can be solved with only a few rounds of communication in the underlying network.

Second, we consider the perspective of (finitary) factors of iid processes. In probability, random processes model systems that appear to vary in a random manner. These include Bernoulli processes, Random walks etc. A particular, well-studied, example is the Ising model.

Third, we investigate the perspective of descriptive combinatorics. The goal of this area is to understand which constructions on infinite graphs can be performed without using the so-called axiom of choice.

Although many of the questions of interest asked in these three areas are quite similar to each other, no systematic connections were known until an insightful paper of Bernshteyn [60] who showed that results from distributed computing can automatically imply results in descriptive combinatorics. In this work, we show that the connections between the three areas run much deeper than previously known, both in terms of techniques and in terms of complexity classes. In fact, our work suggests that it is quite useful to consider all three perspectives as part of a common theory, and we will attempt to present our results accordingly.

In this chapter, we focus on the case where the graph under consideration is a regular tree. Despite its simplistic appearance, regular trees play an important role in each of the three areas, as we will substantiate at the end of this section. To already provide an example, in the area of distributed algorithms, almost all locality lower bounds are achieved on regular trees. Moreover, when regarding lower bounds, the property that they already apply on regular trees actually *strengthens* the result—a fact that is quite relevant for our work as our main contribution regarding the transfer of techniques between the areas is a new lower bound technique in the area of distributed computation that is an adaptation and generalization of a technique from descriptive combinatorics.

In the remainder of this section, we give a high-level overview of the three areas that we study. The purpose of these overviews is to provide the reader with a comprehensive picture of the studied settings that can also serve as a starting point for delving deeper into selected topics in those areas—in order to follow our chapter, it is not necessary to obtain a detailed understanding of the results and connections presented in the overviews. Necessary technical details will be provided in Section 10.3. Moreover, in Section 10.2, we present our contributions in detail.

(Finitary) Factors of iid Processes and Uniform Algorithms:

In recent years, *factors of iid (fiid) processes* on trees attracted a lot of attention in combinatorics, probability, ergodic theory and statistical physics [10, 58, 14, 15, 17, 16, 18, 19, 71, 72, 100, 110, 141, 143, 191, 199, 206, 211, 210, 209, 213, 228, 267, 268, 240, 253, 280, 292, 293]. Intuitively, factors of iid processes are randomized algorithms on, e.g., infinite Δ-regular trees, where each vertex outputs a solution to a problem after it explores random strings on vertices of the whole tree. As an example, consider the *perfect matching* problem. An easy parity argument shows that perfect matching cannot be solved by any local randomized algorithm on finite trees. However, if we allow a small fraction of vertices not to be matched, then, by a result of Nguyen and Onak [259] (see also [121]), there is a constant-round randomized algorithm that produces such a matching on high-girth graphs (where the constant depends on the fraction of unmatched vertices that we allow). This result can also be deduced from a result of Lyons and Nazarov [241], who showed that perfect matching can be described as a factor of iid process on an infinite Δ-regular tree. The high-level idea behind this connection is that high-girth graphs approximate the infinite Δ-regular tree and constant-round local algorithms approximate factors of iid processes. This correspondence is formalized in the notion of *Benjamini-Schramm* or *local-global* convergence [59, 202]. We note that getting only "approximate" solutions, that is, solutions where a small fraction of vertices does not have to satisfy the constraints of a given problem, is intrinsic in this correspondence. Regardless, there are many techniques, such as entropy inequality [17] or correlation decay [16], and particular results such as the aforementioned perfect matching problem [241] that provide lower and upper bounds, respectively, in our setting as well. We refer the reader to [240, 18] for a comprehensive summary of the field.

In this chapter, we mostly consider a stronger condition than fiid, namely so-called *finitary* factors of iid (ffiid) processes that are studied in the same context as fiid [207, 212, 286]. Perhaps surprisingly, the notion of ffiid is identical to the notion of so-called uniform distributed randomized algorithms [223, 185] that we now describe. We define an uniform local algorithm as a randomized local algorithm that does not know the size of the graph n – this enables us to run such an algorithm on infinite graphs, where there is no n. More precisely, we require that each vertex eventually outputs a solution that is compatible with the output in its neighborhood, but the time until the vertex finishes is a potentially unbounded random variable. As in the case of classical randomized algorithms, we can now measure the *uniform complexity* of an uniform local algorithm (known as the tail decay of ffiid [212]). The uniform complexity of an algorithm is defined as the function $t(\varepsilon)$ such that the probability that the algorithm run on a specific vertex needs to see outside its $t(\varepsilon)$-hop neighborhood is at most ε. As in the case of classical local complexity, there is a whole hierarchy of possible uniform complexities.

We remark that uniform distributed local algorithms can be regarded as Las Vegas algorithms. The output will always be correct; there is however no fixed guarantee at what point all vertices have computed their final output. On the other hand, a randomized distributed local algorithm can be viewed as a Monte Carlo algorithm as it needs to produce an output after a fixed number of rounds, though the produced output might be incorrect.

Descriptive Combinatorics:

The Banach-Tarski paradox states that a three-dimensional ball of unit volume can be decomposed into finitely many pieces that can be moved by isometries (distance preserving transformations such as rotations and translations) to form two three-dimensional balls each of them with unit volume(!). The graph theoretic problem lurking behind this paradox is the following: fix finitely many isometries of $\mathbb{R}^3$ and then consider a graph where x and y are connected if there is an isometry that sends x to y. Then our task becomes to find a perfect matching in the appropriate subgraph of this graph – namely, the bipartite subgraph where one partition contains points of the first ball and the other contains points of the other two balls. Banach and Tarski have shown that, with a suitably chosen set of isometries, the axiom of choice implies the existence of such a matching. In contrast, since isometries preserve the Lebesgue measure, the pieces in the decomposition cannot be Lebesgue measurable. Surprisingly, Dougherty and Foreman [119] proved that the pieces in the Banach-Tarski paradox can have the *Baire property*. The Baire property is a topological analogue of being Lebesgue measurable; a subset of $\mathbb{R}^3$ is said to have the Baire property if its difference from some open set is topologically negligible.

Recently, results similar to the Banach-Tarski paradox that lie on the border of combinatorics, logic, group theory, and ergodic theory led to an emergence of a new field often called *descriptive* or *measurable combinatorics*. The field focuses on the connection between the discrete and continuous and is largely concerned with the investigation of graph-theoretic concepts. The usual setup in descriptive combinatorics is that we have a graph with uncountably many connected components, each being a countable graph of bounded degree. For example, in case of the Banach-Tarski paradox, the vertices of the underlying graph are the points of the three balls, edges correspond to isometries, and the degree of each vertex is bounded by the number of chosen isometries. Some of the most beautiful results related to the field include [230, 245, 109, 119, 243, 140, 220, 244, 106, 111, 60], see [219, 265] for recent surveys.

Importantly, in many of these results, including the Banach-Tarski paradox, graphs where each component is an infinite Δ-regular tree appear naturally. Oftentimes, questions considered in descriptive combinatorics lead to constructing a solution to a local problem in the underlying uncountable graph (in the case of Banach-Tarski, the local problem is perfect matching). The construction needs to be such that the solution of the problem has some additional regularity properties. For example in the case of Banach-Tarski, a solution is possible when the regularity condition is the Baire property, but not if it is Lebesgue measurability. In fact, together with Borel measurability these are the most prominent regularity conditions studied in descriptive combinatorics. The corresponding complexity classes of local problems that always admit a solution with the respective regularity property are $\mathsf{BOREL}, \mathsf{MEASURE}, \mathsf{BAIRE}$. In this chapter, we moreover consider the setting where each connected component of the underlying graph is a Δ-regular tree.

The connection between distributed computing and descriptive combinatorics arises from the fact that in descriptive combinatorics we care about constructions that do not use the axiom of choice. In the distributed language, the axiom of choice corresponds to leader election, that is, the constructions in descriptive combinatorics do not allow picking exactly one point in every component. To get some intuition about the power of the complexity class BOREL, we note that Borel constructions allow us to alternate countably many times the following two operations. First, any local algorithm with constant local complexity can be run. Second, we have an oracle that provides a maximal independent set (MIS) on any graph that can be constructed locally from the information computed so far [220]. Note that from the speedup result of [94] we get that every local problem with local complexity $O(\log^* n)$ can be solved by constant local constructions and *one* call to such an MIS oracle. This implies the inclusion $\mathsf{LOCAL}(O(\log^* n)) \subseteq \mathsf{BOREL}$ proven in the insightful paper of Bernshteyn [60]. The relationship of the class $\mathsf{MEASURE}$ (and of the class fiid from the discussion of factors) to the class BOREL is analogous to the relationship of randomized distributed algorithms to deterministic distributed algorithms.

Local Problems on Regular Trees:

The motivation for studying regular trees in this work stems from different sources: (a) infinite Δ-regular trees are studied in the area of ergodic theory [71, 72], random processes [17, 16, 241] and descriptive combinatorics [244, 107], (b) many lower bounds in distributed computing are proven in regular trees [25, 22, 73, 75], and (c) connecting and comparing the techniques of the three areas in this simple setting reveals already deep connections, see Section 10.2.

10.2 Our Contributions

We believe that our main contribution is presenting all three perspectives as part of a common theory. Our technical contribution is split into three main parts.

10.2.1 Generalization of Marks' Technique

In Section 10.4 we extend the Borel determinacy technique of Marks [244], which was used to prove the nonexistence of Borel Δ-colorings and perfect matchings, to a broader class of problems, and adapt the extended technique to the distributed setting, thereby obtaining a simple method for proving distributed lower bounds. This method is the first lower bound technique for distributed computing using ideas coming from descriptive combinatorics (see [64] for a distributed computing upper bound motivated by descriptive combinatorics). Moreover, we show how to use the developed techniques to obtain both BOREL and LOCAL lower bounds for local problems from a natural class, called homomorphism problems. Our key technical ingredient for obtaining the mentioned techniques and results is a novel technique based on the notion of an *ID graph*. We note that a very similar concept to the ID graph was independently discovered by [279].

Marks' technique: In the following we give an introduction to Marks' technique by going through a variant of his proof [244] that shows that Δ-coloring has deterministic local complexity $\Omega(\log n)$. The proof already works in the case where the considered regular tree comes with an input Δ-edge coloring. In this case, the output color of a given vertex u can be interpreted as that u "grabs" the incident edge of that color. The problem is hence equivalent to the *edge grabbing* problem where every vertex is required to grab an incident edge such that no two vertices grab the same edge.

We first show the lower bound in the case that vertices do not have unique identifiers but instead are properly colored with $L > \Delta$ colors. Suppose there is an algorithm $\mathcal{A}$ solving the edge grabbing problem with local complexity $t(n) = o(\log n)$, and consider a tree rooted at vertex u of depth $t(n)$; such a tree has less than n vertices, for large enough n. Assume that u has input color $\sigma \in [L]$, and consider, for some fixed edge color α, the edge e that is incident to u and has color α. Two players, Alice and Bob, are playing the following game. In the i-th round, Alice colors the vertices at distance i from u in the subtree reachable via edge e with colors from $[L]$. Then, Bob colors all other vertices at distance i from u with colors from $[L]$. Consider the output of u when executing $\mathcal{A}$ on the obtained colored tree. Bob wins the game if u grabs the edge e, and Alice wins otherwise.

Note that either Alice or Bob has a winning strategy. Given the color σ of u, if, for each edge color α, Alice has a winning strategy in the game corresponding to the pair (σ, α), then we can create Δ copies of Alice and let them play their strategy on each subtree of u, telling them that the colors chosen by the other Alices are what Bob played. The result is a coloring of the input tree such that u, by definition, does not pick any edge, contradicting the fact that $\mathcal{A}$ provides a valid solution! So for every σ there is at least one α such that Bob has a winning strategy for the game corresponding to (σ, α). By the pigeonhole principle, there are two colors σ_1, σ_2, such that Bob has a winning strategy for both pairs (σ_1, α) and (σ_2, α). But now we can imagine a tree rooted in an edge between vertices u_1, u_2 that are colored with colors σ_1, σ_2. We can now take two copies of Bob, one playing at u_1 and the other playing at u_2 and let them battle it out, telling each copy that the other color from $\{\sigma_1, \sigma_2\}$ and whatever the other copy plays are the moves of Alice. The resulting coloring has the property that both u_1 and u_2, when executing $\mathcal{A}$ on the obtained colored tree, grab the edge between them, a contradiction that finishes the proof!

The ID graph: The downside of the proof is that it does not work in the model with unique identifiers (where the players' moves consist in assigning identifiers instead of colors), since gluing copies of the same player could result in an identifier assignment where the identifiers are not unique. One possible remedy is to conduct the whole proof in the context of Borel graphs as was done by Marks. This proves an even stronger statement, namely that Δ-coloring is not in the class BOREL, but requires additional ad-hoc tricks and a pretty heavy set theoretic tool—Martin's celebrated Borel determinacy theorem [246] stating that even for infinite two-player games one of the players has to have a winning strategy if the payoff set is Borel. The ID graph enables us to adapt the proof (and its generalization that we develop in Section 10.4) to the distributed setting, where the fact that one of the players has a winning strategy is obvious. Moreover, we use an infinite version of the ID graph to generalize Marks' technique also in the Borel setting.

Here is how it works: The ID graph is a specific graph whose vertices are the possible unique input identifiers (for input graphs of size n), that is, numbers from $[n^{O(1)}]$. Its edges are colored with colors from $[\Delta]$ and its girth is $\Omega(\log n)$. When we define the game between Alice and Bob, we require them to label vertices with identifiers in such a way that whenever a new vertex is labeled with identifier i, and its already labeled neighbor has identifier j, then ij is an edge in the ID graph. Moreover, the color of edge ij in the ID graph is required to be the same as the color of the edge between the vertices labeled i and j in the tree where the game is played. It is straightforward to check that these conditions enforce that even if we let several copies of the same player play, the resulting tree is labeled with unique identifiers. Hence, the same argument as above now finally proves that the deterministic local complexity of Δ-coloring is $\Omega(\log n)$.

We note that our ID graph technique is of independent interest and may have applications in many different contexts. To give an example from distributed computing, consider the proof of the deterministic $\Omega(\log n)$-round lower bound for Δ-coloring developed by the distributed community [73, 94], which is based on the celebrated round elimination technique. Even though the result is deterministic, the proof is quite technical due to the fact that it relies on examining randomized algorithms, for reasons similar to the reasons why Marks' proof does not apply directly to the setting with unique identifiers. Fortunately, it can be again streamlined with the use of the ID graph technique. Moreover, in a follow-up work, the ID graph technique has already led to a new lower bound for the Lovász local lemma [81] in the area of Local Computation Algorithms (which is part of the realm of sequential computation), thereby giving further evidence for the versatility of the technique.

Marks vs. Round Elimination: It is quite insightful to compare Marks' technique (and our generalization of it) with the powerful round elimination technique [75], which has been responsible for all locality lower bounds of the last years [73, 22, 75]. While, on the surface, Marks' approach developed for the Borel world may seem quite different from the round elimination technique, there are actually striking similarities between the two methods. On a high level, in the round elimination technique, the following argument is used to prove lower bounds in the LOCAL model: If a T-round algorithm exists for a problem Π_0 of interest, then there exists a $(T-1)$-round algorithm for some problem Π_1 that can be obtained from Π_0 in a mechanical manner. By applying this step iteratively, we obtain a problem Π_t that can be solved in 0 rounds; by showing that there is no 0-algorithm for Π_t (which is easy to do if Π_t is known), a $(T+1)$-round lower bound for Π_0 is obtained.

The interesting part regarding the relation to Marks' technique is how the $(T-i-1)$-round algorithms $\mathcal{A}'$ are obtained from the $(T-i)$-round algorithms $\mathcal{A}$ in the round elimination framework: in order to execute $\mathcal{A}'$, each vertex v, being aware of its $(T-i-1)$-hop neighborhood, essentially asks whether, for all possible extensions of its view by one hop along a chosen incident edge, there exists some extension of its view by one hop along all other incident edges such that $\mathcal{A}$, executed on the obtained $(T-i)$-hop neighborhood, returns a certain output at v, and then bases its output on the obtained answer. It turns out that the vertex sets corresponding to these two extensions correspond precisely to two moves of the two players in the game(s) played in Marks' approach: more precisely, in round $T-i$ of a game corresponding to the considered vertex v and the chosen incident edge, the move of Alice consists in labeling the vertices corresponding to the first extension, and the move of Bob consists in labeling the vertices corresponding to the second extension.

However, despite the similarities, the two techniques (at least in their current forms) have their own strengths and weaknesses and are interestingly different in that there are local problems that we know how to obtain good lower bounds for with one technique but not the other, and vice versa. Finding provable connections between the two techniques is an exciting research direction that we expect to lead to a better understanding of the possibilities and limitations of both techniques.

In Section 10.4 we use our generalized and adapted version of Marks' technique to prove new lower bounds for so-called homomorphism problems. Homomorphism problems are a class of local problems that generalizes coloring problems—each vertex is to be colored with some color and there are constraints on which colors are allowed to be adjacent. The constraints can be viewed as a graph—in the case of coloring this graph is a clique. In general, whenever the underlying graph of the homomorphism problem is Δ-colorable, its deterministic local complexity is $\Omega(\log n)$, because solving the problem would imply that we can solve Δ-coloring too (in the same runtime). It seems plausible that homomorphism problems of this kind are the only hard, i.e., $\Omega(\log n)$, homomorphism problems. However, our generalization of Marks' technique asserts that this is not true.

Theorem 10.1. *There are homomorphism problems whose deterministic local complexity on trees is $\Omega(\log n)$ such that the chromatic number of the underlying graph is $2\Delta - 2$.*

It is not known how to prove the same lower bounds using round elimination[1]; in fact, as far as we know, these problems are the only known examples of problems on Δ-regular trees for which a lower bound is known to hold but currently not achievable by round elimination. Proving the same lower bounds via round elimination is an exciting open problem.

10.2.2 Separation of Various Complexity Classes

Uniform Complexity Landscape: We investigate the connection between randomized and uniform distributed local algorithms, where uniform algorithms are equivalent to the studied notion of finitary factors of iid. First, it is simple to observe that local problems with uniform complexity $t(\varepsilon)$ have randomized complexity $t(1/n^{O(1)})$ – by definition, every vertex knows its local output after that many rounds with probability $1 - 1/n^{O(1)}$. The result thus follows by a union bound over the n vertices of the input graph.

On the other hand, we observe that on Δ-regular trees the implication also goes in the opposite direction in the following sense. Every problem that has a randomized complexity of $t(n) = o(\log n)$ has an uniform complexity of $O(t(1/\varepsilon))$.

One could naively assume that this equivalence also holds for higher complexities, but this is not the case. Consider for example the 3-coloring problem. It is well-known in the distributed community that 3-coloring a tree can be solved deterministically in $O(\log n)$ rounds using the rake-and-compress decomposition [88, 254]. On the other hand, there is no uniform algorithm for 3-coloring a tree. If there were such an uniform algorithm, we could run it on any graph with large enough girth and color 99% of its vertices with three colors. This in turn would imply that the high-girth graph has an independent set of size at least $0.99 \cdot n/3$. This is a contradiction with the fact that there exists high-girth graphs with a much smaller independence number [70].

Interestingly, the characterization of Bernshteyn [62] implies that *any* uniform distributed algorithm can be "sped up" to a deterministic local $O(\log n)$ complexity, as we prove in Theorem 10.56.

We show that there are local problems that can be solved by an uniform algorithm but only with a complexity of $\Omega(\log 1/\varepsilon)$. Namely, the problem of constructing a 2-hop

[1]Indeed, the descriptions of the problems have comparably large numbers of labels and do not behave like so-called "fixed points" (i.e., nicely) under round elimination, which suggests that it is hard to find a round elimination proof with the currently known approaches.

perfect matching on infinite Δ-regular trees for $\Delta \geq 3$ has an uniform local complexity between $\Omega(\log 1/\varepsilon)$ and $O(\text{poly} \log 1/\varepsilon)$. Formally, this proves the following theorem.

Theorem 10.2. *On bounded degree trees we have the following:* $\mathsf{OLOCAL}(O(\log\log 1/\varepsilon)) \subsetneq \mathsf{OLOCAL}(O(\text{poly}\log 1/\varepsilon))$.

The uniform algorithm for this problem is based on a so-called one-ended forest decomposition introduced in [107] in the descriptive combinatorics context. In a one-ended forest decomposition, each vertex selects exactly one of its neighbors as its parent by orienting the corresponding edge outwards. This defines a decomposition of the vertices into infinite trees. We refer to such a decomposition as a one-ended forest decomposition if the subtree rooted at each vertex only contains finitely many vertices. Having computed such a decomposition, 2-hop perfect matching can be solved inductively starting from the leaf vertices of each tree.

We leave the understanding of the uniform complexity landscape in the regime $\Omega(\log 1/\varepsilon)$ as an exciting open problem. In particular, does there exist a function $g(\varepsilon)$ such that each local problem that can be solved by an uniform algorithm has an uniform complexity of $O(g(\varepsilon))$?

Relationship of Distributed Classes with Descriptive Combinatorics:

Bernshteyn recently proved that $\mathsf{LOCAL}(\log^* n) \subseteq \mathsf{BOREL}$ [60]. That is, each local problem with a deterministic LOCAL complexity of $O(\log^* n)$ also admits a Borel-measurable solution. A natural question to ask is whether the converse also holds. Indeed, it is known that $\mathsf{LOCAL}(\log^* n) = \mathsf{BOREL}$ on paths with no additional input [184]. We show that on regular trees the situation is different. On one hand, a characterization of Bernshteyn [62] implies that $\mathsf{BOREL} \subseteq \mathsf{BAIRE} \subseteq \mathsf{LOCAL}(O(\log n))$. On the other hand, we show that this result cannot be strengthened by proving the following result.

Theorem 10.3. $\mathsf{BOREL} \not\subseteq \mathsf{RLOCAL}(o(\log n))$.

That is, there exists a local problem that admits a Borel-measurable solution but cannot be solved with a (randomized) LOCAL algorithm running in a sublogarithmic number of rounds.

Let us sketch a weaker separation, namely that $\mathsf{BOREL} \setminus \mathsf{LOCAL}(O(\log^* n)) \neq \emptyset$. Consider a version of Δ-coloring where a subset of vertices can be left uncolored. However, the subgraph induced by the uncolored vertices needs to be a collection of doubly-infinite paths (in finite trees, this means each path needs to end in a leaf vertex). The nonexistence of a fast distributed algorithm for this problem essentially follows from the celebrated $\Omega(\log n)$ deterministic lower bound for Δ-coloring of [73]. On the other hand, the problem allows a Borel solution. First, sequentially compute $\Delta - 2$ maximal independent sets, each time coloring all vertices in the MIS with the same color, followed by removing all the colored vertices from the graph. In that way, a total of $\Delta - 2$ colors are used. Moreover, each uncolored vertex has at most 2 uncolored neighbors. This implies that the set of uncolored vertices forms a disjoint union of finite paths, one ended infinite paths and doubly infinite paths. The first two classes can be colored inductively with two additional colors, starting at one endpoint of each path in a Borel way (namely it can be done by making use of the countably many MISes in larger and larger powers of the input graph). Hence, in the end only doubly infinite paths are left uncolored, as desired.

To show the stronger separation between the classes BOREL and $\mathsf{RLOCAL}(o(\log n))$ we use a variation of the 2-hop perfect matching problem. In this variation, some of the vertices can be left unmatched, but similar as in the variant of the Δ-coloring problem described above, the graph induced by all the unmatched vertices needs to satisfy some additional constraints.

We conclude the paragraph by noting that the separation between the classes BOREL and $\mathsf{LOCAL}(O(\log^* n))$ is not as simple as it may look in the following sense. This is because problems typically studied in the LOCAL model with a LOCAL complexity of $\omega(\log^* n)$ like Δ-coloring and perfect matching also do not admit a Borel-measurable solution due to the technique of Marks [244] that we discussed in Section 10.2.1.

10.2.3 Local Complexity and Baire solutions

We already discussed that one of complexity classes studied in descriptive combinatorics is the class BAIRE. Recently, Bernshteyn proved [62] that all local problems that are in the complexity class $\mathsf{BAIRE}, \mathsf{MEASURE}$ or fiid have to satisfy a simple combinatorial condition which we call being ℓ-full. On the other hand, all ℓ-full problems allow a BAIRE solution [62]. This implies a complete combinatorial characterization of the class BAIRE. We defer the formal definition of ℓ-fullness to Section 10.6 as it requires a formal definition of a local problem. Informally speaking, in the context of vertex labeling problems, a problem is ℓ-full if we can choose a subset S of the labels with the following property. Whenever we label two endpoints of a path of at least ℓ vertices with two labels from S, we can extend the labeling with labels from S to the whole path such that the overall labeling is valid. For example, proper 3-coloring is 3-full with $S = \{1, 2, 3\}$ because for any path of three vertices such that its both endpoints are colored arbitrarily, we can color the middle vertex so that the overall coloring is proper. On the other hand, proper 2-coloring is not ℓ-full for any ℓ.

We complement this result as follows. First, we prove that any ℓ-full problem has local complexity $O(\log n)$, thus proving that all complexity classes considered in the areas of factors of iids and descriptive combinatorics are contained in $\mathsf{LOCAL}(O(\log n))$. In particular, this implies that the existence of *any* uniform algorithm implies a local distributed algorithm for the same problem of local complexity $O(\log n)$. We obtain this result via the well-known rake-and-compress decomposition [254].

On the other hand, we prove that any problem in the class $\mathsf{LOCAL}(O(\log n))$ satisfies the ℓ-full condition. The proof combines a machinery developed by Chang and Pettie [88] with additional nontrivial ideas. In this proof we construct recursively a sequence of sets of rooted, layered, and partially labeled trees, where the partial labeling is computed by simulating any given $O(\log n)$-round distributed algorithm, and then the set S meeting the ℓ-full condition is constructed by considering all possible extensions of the partial labeling to complete correct labeling of these trees.

This result implies the following equality:

Theorem 10.4. $\mathsf{LOCAL}(O(\log n)) = \mathsf{BAIRE}$.

This equality is surprising in that the definitions of the two classes do not seem to have much in common on the first glance! Moreover, the proof of the equality relies on nontrivial results in both distributed algorithms (the technique of Chang and Pettie [88]) and descriptive combinatorics (the fact that a hierarchical decomposition, so-called toast, can be constructed in BAIRE, [105], see Proposition 10.54).

The combinatorial characterization of the local complexity class $\mathsf{LOCAL}(O(\log n))$ on Δ-regular trees is interesting from the perspective of distributed computing alone. This result can be seen as a part of a large research program aiming at classification of possible local complexities on various graph classes [27, 73, 94, 88, 85, 21, 31, 25, 20]. That is, we wish not only to understand possible complexity classes, but also to find combinatorial characterizations of problems in those classes that allow us to efficiently decide for a given problem which class it belongs to. Unfortunately, even for grids with input labels, it is *undecidable* whether a given local problem can be solved in $O(1)$ rounds [258, 77], since local problems on grids can be used to simulate a Turing machine. This undecidability result does not apply to paths and trees, hence for these graph classes it is still hopeful that we can find simple and useful characterizations for different classes of distributed problems.

In particular, on paths it is decidable what classes a given local problem belongs to, for all classes coming from the three areas considered here, and this holds even if we allow inputs [21, 184]. The situation becomes much more complicated when we consider trees. Recently, classification results on trees were obtained for so-called binary-labeling problems [25]. More recently, a complete classification was obtained in the case of *rooted regular trees* [31]. Although their algorithm takes exponential time in the worst case, the authors provided a practical implementation fast enough to classify many typical problems of interest.

Much less is known for general, *unoriented* trees, with an arbitrary number of labels. In general, deciding the optimal distributed complexity for a local problem on bounded-degree trees is $\mathsf{EXPTIME}$-hard [85], such a hardness result does not rule out the possibility for having a simple and polynomial-time characterization for the case of *regular trees*, where there is no input and the constraints are placed only on degree-Δ vertices. Indeed, it was stated in [31] as an open question to find such a characterization. Our characterization of $\mathsf{LOCAL}(O(\log n)) = \mathsf{BAIRE}$ by ℓ-full problems makes progress in better understanding the distributed complexity classes on trees and towards answering this open question.

Roadmap: In Section 10.3, we define formally all the three setups we consider in the chapter. In Section 10.4 we discuss the lower bound technique of Marks and the new concept of an ID graph. Next, in Section 10.5 we prove some basic results about the uniform complexity classes and give examples of problems separating some classes. Finally, in Section 10.6 we prove that a problem admits a Baire measurable solution if and only if it admits a distributed algorithm of local complexity $O(\log n)$.

The individual sections can be read largely independently of each other. Moreover, most of our results that are specific to only one of the three areas can be understood without reading the parts of the chapter that concern the other areas. We encourage the reader interested mainly in one of the areas to skip the respective parts.

10.3 Preliminaries

In this section, we explain the setup we work with, the main definitions and results. The class of graphs that we consider in this work are either infinite Δ-regular trees, or their finite analogue that we define formally in Section 10.3.1.

We sometimes explicitly assume $\Delta > 2$. The case $\Delta = 2$, that is, studying paths, behaves differently and seems much easier to understand [184]. Unless stated otherwise, we do not consider any additional structure on the graphs, but sometimes it is natural to work with trees with an input Δ-edge-coloring.

10.3.1 Local Problems on regular trees

Definition 10.5 (Δ-regular trees)**.** *A tree T, finite or infinite is a* Δ-regular tree *if either it is infinite and $T = T_\Delta$, where T_Δ is the unique infinite Δ-regular tree, that is each vertex has exactly Δ-many neighbors, or it is finite of maximum degree Δ and each vertex $v \in T$ of degree $d \leq \Delta$ is contained in $(\Delta - d)$-many virtual half-edges.*

Formally, we can view T as a triplet $(V(T), E(T), H(T))$, where $(V(T), E(T))$ is a tree of maximum degree Δ and $H(T)$ consists of real half-edges, that is pairs (v, e), where $v \in V(T)$, $e \in E(T)$ and e is incident to v, together with some virtual edges, in the case when T is finite, such that each vertex $v \in V(T)$ is contained in exactly Δ-many half-edges (real or virtual).

As we are considering trees in this work, each LCL problem can be described in a specific form that provides two lists, one describing all label combinations that are allowed on the half-edges incident to a vertex, and the other describing all label combinations that are allowed on the two half-edges belonging to an edge.[2] We arrive at the following definition

[2]Every problem that can be described in the form given by Naor and Stockmeyer [258] can be equivalently described as an LCL problem in this list form, by simply requiring each output label on some half-edge h to encode all output labels in a suitably large (constant) neighborhood of h in the form given in [258].

for LCLs on Δ-regular trees.[3]

Definition 10.6 (LCLs on Δ-regular trees)**.** *A* locally checkable labeling problem*, or* LCL *for short, is a triple* $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$*, where* Σ *is a finite set of labels,* $\mathcal{V}$ *is a subset of unordered cardinality-*Δ *multisets*[4] *of labels from* Σ *, and* $\mathcal{E}$ *is a subset of unordered cardinality-*2 *multisets of labels from* Σ*.*

We call $\mathcal{V}$ *and* $\mathcal{E}$ *the* vertex constraint *and* edge constraint *of* Π*, respectively. Moreover, we call each multiset contained in* $\mathcal{V}$ *a* vertex configuration *of* Π*, and each multiset contained in* $\mathcal{E}$ *an* edge configuration *of* Π*.*

Let T *be a* Δ*-regular tree and* $c : H(T) \to \Sigma$ *a half-edge labeling of* T *with labels from* Σ*. We say that* c *is a* Π-coloring*, or, equivalently, a correct solution for* Π*, if, for each vertex* v *of* T*, the multiset of labels assigned to the half-edges incident to* v *is contained in* $\mathcal{V}$*, and, for each edge* e *of* T*, the cardinality-*2 *multiset of labels assigned to the half-edges belonging to* e *is an element of* $\mathcal{E}$*.*

An equivalent way to define our setting would be to consider Δ-regular trees as commonly defined, that is, there are vertices of degree Δ and vertices of degree 1, i.e., leaves. In the corresponding definition of LCL one would consider leaves as unconstrained w.r.t. the vertex constraint, i.e., in the above definition of a correct solution the condition "for each vertex v" is replaced by "for each non-leaf vertex v". Equivalently, we could allow arbitrary trees of maximum degree Δ as input graphs, but, for vertices of degree $< \Delta$, we require the multiset of labels assigned to the half-edges to be extendable to some cardinality-Δ multiset in $\mathcal{V}$. When it helps the exposition of our ideas and is clear from the context, we may make use of these different but equivalent perspectives.

We illustrate the difference between our setting and "standard setting" without virtual half-edges on the perfect matching problem. A standard definition of the perfect matching problem is that some edges are picked in such a way that each vertex is covered by exactly one edge. It is easy to see that there is no local algorithm to solve this problem on the class of finite trees (without virtual half-edges), this is a simple parity argument. However, in our setting, every vertex needs to pick exactly one half-edge (real or virtual) in such a way that both endpoints of each edge are either picked or not picked. We remark that in our setting it is not difficult to see that (if $\Delta > 2$), then this problem can be solved by a local deterministic algorithm of local complexity $O(\log(n))$.

10.3.2 The Uniform Local Model

We discuss the general relation between the classical local complexity and the uniform local complexity. Recall that when we talk about distributed algorithm on Δ-regular trees, the algorithm has access to n, the size of the tree. The measure of complexity is

[3]Note that the defined LCL problems do not allow the correctness of the output to depend on input labels.

[4]Recall that a multiset is a modification of the concept of sets, where repetition is allowed.

the classical local complexity. On the other hand, when we talk about uniform distributed algorithms on Δ-regular trees, we talk about an infinite Δ-regular tree and the measure of complexity is the uniform local complexity. Recall that $B(v, t)$ is the t-hop neighborhood of a vertex v in an underlying graph G. Whenever we talk about *local* complexity on Δ-regular trees, we always tacitly think about the class of finite Δ-regular trees. We use the notation $\mathsf{LOCAL}(O(t(n)))$ whenever the deterministic and the randomized complexity of the problems are the same up to constant factor.

Uniform Algorithms: As we mentioned before, when talking about local complexities, we always have in mind that the underlying graph is finite. In particular, the corresponding algorithm knows the size of the graph. On infinite Δ-regular trees, or infinite graphs in general, we use the following notion [212, 223].

Definition 10.7. *An* uniform local algorithm $\mathcal{A}$ *is a function that is defined on all possible (finite) neighborhoods of a vertex. For some neighborhoods it outputs a special symbol $\emptyset$ instead of a labeling of the half-edges around the central vertex. Applying $\mathcal{A}$ on a graph G means that for each vertex u of G the function is applied to $B(u, t)$, where t is the minimal number such that $\mathcal{A}(u, t) \neq \emptyset$. We call t the coding radius of $\mathcal{A}$, and denote it, as a function on vertices, as $R_{\mathcal{A}}$.*

We define the corresponding notion of *uniform local complexity* for infinite Δ-regular trees where each vertex is assigned an infinite random string.

Definition 10.8 (Uniform local complexity [212])**.** *We say that the* uniform local (randomized) complexity *of an LCL problem Π is $t(\varepsilon)$ if there is an uniform local algorithm $\mathcal{A}$ such that the following hold on the infinite Δ-regular tree. Recall that $R_{\mathcal{A}}$ is the random variable measuring the coding radius of a vertex u, that is, the distance $\mathcal{A}$ needs to look at to decide the answer for u. Then, for any $0 < \varepsilon < 1$:*

$$P(R_{\mathcal{A}} \geq t(\varepsilon)) \leq \varepsilon.$$

We also say $\Pi \in \mathsf{ULOCAL}(O(t(\varepsilon))$.

We finish by stating the following lemma that bounds the uniform complexity of concatenation of two uniform algorithm (we need to be little bit more careful and cannot just add the complexites up).

Lemma 10.9 (Sequential Composition)**.** *Let A_1 and A_2 be two distributed uniform algorithms with an uniform local complexity of $t_1(\varepsilon)$ and $t_2(\varepsilon)$, respectively. Let A be the sequential composition of A_1 and A_2. That is, A needs to know the output that A_2 computes when the local input at every vertex is equal to the output of the algorithm A_1 at that vertex. Then, $t_A(\varepsilon) \leq t_{A_1}\left(\frac{\varepsilon/2}{\Delta^{t_{A_2}(\varepsilon/2)+1}}\right) + t_{A_2}(\varepsilon/2)$.*

Proof. Consider some arbitrary vertex u. Let E_1 denote the event that the coding radius of $\mathcal{A}_1$ is at most $t_{A_1}\left(\frac{\varepsilon/2}{\Delta^{t_{A_2}(\varepsilon/2)+1}}\right)$ for all vertices in the $t_{A_2}(\varepsilon/2)$-hop neighborhood around

u. As the $t_{A_2}(\varepsilon/2)$-hop neighborhood around u contains at most $\Delta^{t_{A_2}(\varepsilon/2)+1}$ vertices, a union bound implies that $P(E_1) \geq 1 - \varepsilon/2$. Moreover, let E_2 denote the event that the coding radius of algorithm $\mathcal{A}_2$ at vertex u is at most $t_{A_2}(\varepsilon/2)$. By definition, $P(E_2) \geq 1 - \varepsilon/2$. Moreover, if both events E_1 and E_2 occur, which happens with probability at least $1-\varepsilon$, then the coding radius of algorithm $\mathcal{A}$ is at most $t_{A_1}\left(\frac{\varepsilon/2}{\Delta^{t_{A_2}(\varepsilon/2)+1}}\right) + t_{A_2}(\varepsilon/2)$, thus finishing the proof. $\square$

10.3.3 Descriptive combinatorics

Before we define formally the descriptive combinatorics complexity classes, we give a high-level overview on their connection to distributing computing for the readers more familiar with the latter.

The complexity class that partially captures deterministic local complexity classes is called BOREL (see also Remark 10.13). First note that by a result of Kechris, Solecki and Todorčević [220] the *maximal independent set problem* (with any parameter $r \in \mathbb{N}$) is in this class for any bounded degree graph.[5] In particular, this yields that BOREL contains the class $\mathsf{LOCAL}(O(\log^* n))$ by the characterization of [94], see [60]. Moreover, as mentioned before, BOREL is closed under countably many iterations of the operations of finding maximal independent set (for some parameter that might grow) and of applying a constant local rule that takes into account what has been constructed previously.[6]

To get a better grasp of what this means, consider for example the proper vertex 2-coloring problem on half-lines. It is clear that no local algorithm can solve this problem. However, as it is possible to determine the starting vertex after countably many iterations of the maximal independent set operation, we conclude that this problem is in the class BOREL. The idea that BOREL can compute some unbounded, global, information will be implicitly used in all the construction in Section 10.5 that separate BOREL from local classes.

The intuition behind the class MEASURE is that it relates in the same way to the class BOREL, as randomized local algorithms relate to deterministic ones. In particular, the operations that are allowed in the class MEASURE are the same as in the class BOREL but the solution of a given LCL can be incorrect on a measure zero set.

The class BAIRE can be considered as a topological equivalent of the measure theoretic class MEASURE, that is, a solution can be incorrect on a topologically negligible set. The main difference between the classes MEASURE and BAIRE is that in the later there is a hierarchical decomposition that is called *toast*. (Note that this phenomenon is present in the case of MEASURE exactly on so-called amenable graphs. It is also tightly connected with the notion of hyperfiniteness [105, 145].) The independence of colorings on a

[5] That is, it is possible to find a Borel maximal independent set, i.e., a maximal independent set which is, moreover, a Borel subset of the vertex set.

[6] It is in fact an open problem, whether this captures fully the class BOREL. However, note that an affirmative answer to this question would yield that problems can be solved in an "effective" manner in the Borel context, which is known not to be the case in unbounded degree graphs [294].

tree together with this structure allows for a combinatorial characterization of the class BAIRE, which was proven by Bernshteyn [62], see also Section 10.6.

Next we formulate the precise definitions. We refer the reader to [265, 219, 60, 218], or to [184, Introduction, Section 4.1] and [185, Section 7.1, 7.2] for intuitive examples and standard facts of descriptive set theory. In particular, we do not define here the notion standard Borel/probability space, a Polish topology, a Borel probability measure, Baire property etc.

Let $\mathcal{G}$ be a Borel graph of bounded maximum degree on a standard Borel space X. In this chapter we consider exclusively acyclic Δ-regular Borel graphs and we refer to them as *Δ-regular Borel forests*. It is easy to see that the set of half-edges is naturally a standard Borel space, we denote this set by $H(\mathcal{G})$. Thus, it makes sense to consider Borel labelings of $H(\mathcal{G})$. Moreover, if $\mathcal{G}$ is a Δ-regular Borel forest and $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ is an LCL, we can also decide whether a coloring $f : \mathcal{H}(\mathcal{G}) \to \Sigma$ is a solution to Π as in Definition 10.6. Similarly, we say that the coloring f solves Π, e.g., on a μ-conull set for some Borel probability measure μ on X if there is a Borel set $C \subseteq X$ such that $\mu(C) = 1$, the vertex constraints are satisfied around every $x \in C$ and the edge constraints are satisfied for every $x, y \in C$ that form an edge in $\mathcal{G}$.

Definition 10.10 (Descriptive classes)**.** *Let $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ be an LCL. We say that Π is in the class* BOREL *if for every acyclic Δ-regular Borel graph $\mathcal{G}$ on a standard Borel space X, there is a Borel function $f : H(\mathcal{G}) \to \Sigma$ that is a Π-coloring of $\mathcal{G}$.*

We say that Π is in the class BAIRE *if for every acyclic Δ-regular Borel graph $\mathcal{G}$ on a standard Borel space X and every compatible Polish topology τ on X, there is a Borel function $f : H(\mathcal{G}) \to \Sigma$ that is a Π-coloring of $\mathcal{G}$ on a τ-comeager set.*

We say that Π is in the class $\mathsf{MEASURE}$ *if for every acyclic Δ-regular Borel graph $\mathcal{G}$ on a standard Borel space X and every Borel probability measure μ on X, there is a Borel function $f : H(\mathcal{G}) \to \Sigma$ that is a Π-coloring of $\mathcal{G}$ on a μ-conull set.*

The following claim follows directly from the definition.

Claim 10.11. *We have* $\mathsf{BOREL} \subseteq \mathsf{MEASURE}, \mathsf{BAIRE}$.

Recently Bernshteyn [60, 61] proved several results that connect distributed computing with descriptive combinatorics.

Theorem 10.12 ([60])**.** *We have*

- $\mathsf{LOCAL}(O(\log^*(n))) \subseteq \mathsf{BOREL}$,
- $\mathsf{RLOCAL}(o(\log(n))) \subseteq \mathsf{MEASURE}, \mathsf{BAIRE}$.

In fact, these inclusions hold for any reasonable class of bounded degree graphs.

Remark 10.13. *The "truly" local class in descriptive combinatorics is the class* $\mathsf{CONTINUOUS}$*. Even though we do not define this class here*[7]*, and we refer the reader to [61, 146, 185] for the definition and discussions in various cases, we mention that the inclusion*

$$\mathsf{LOCAL}(O(\log^* n)) \subseteq \mathsf{CONTINUOUS} \tag{*}$$

holds in most reasonable classes of bounded degree graphs, see [60]. This also applies to our situation Recently, it was shown by Bernshteyn [61] that $(*)$ *can be reversed for Cayley graphs of finitely generated groups. This includes, e.g., natural* Δ*-regular trees with proper edge* Δ*-coloring, as this is a Cayely graph of free product of* Δ*-many* $\mathbb{Z}_2$ *induced by the standard generating set. It is, however, not clear whether it* $(*)$ *can be reversed in our situation, i.e.,* Δ*-regular trees without any additional labels.*

10.3.4 Random processes

We start with an intutitve description of fiid processes. Let T_Δ be the infinite Δ-regular tree. Informally, *factors of iid processes (fiid)* on T_Δ are infinite analogues of local randomized algorithms in the following way. Let Π be an LCL and $u \in T_\Delta$. In order to solve Π, we are allowed to explore the whole graph, and the random strings that are assigned to vertices, and then output a labeling of half-edges around u. If such an assignment is a measurable function and produces a Π-coloring almost surely, then we say that Π is in the class fiid. Moreover, if every vertex needs to explore only finite neighborhood to output a solution, then we say that Π is in the class FFIID. Such processes are called *finitary* fiid (*ffiid*). There is also an intimate connection between ffiid and uniform algorithms. This is explained in [185, Section 2.2]. Informally, an ffiid process that almost surely solves Π is, in the language of distributed computing, an uniform local algorithm that solves Π. This allows us to talk about uniform local complexity of an ffiid. In the rest of the chapter we interchange both notions freely with slight preference for the distributed computing language.

Now we define formally these classes, using the language of probability. We denote by $\mathrm{Aut}(T_\Delta)$ the automorphism group of T_Δ. An *iid process* on T_Δ is a collection of iid random variables $Y = \{Y_v\}_{v \in V(T_\Delta)}$ indexed by vertices, or edges, half-edges etc, of T_Δ such that their joint distribution is invariant under $\mathrm{Aut}(T_\Delta)$. We say that X is a *factor of iid process (fiid)* if $X = F(Y)$, where F is a measurable $\mathrm{Aut}(T_\Delta)$-invariant map and Y is an iid process on T_Δ.[8] Moreover, we say that X is a *finitary factor if iid process (ffiid)* if F depends with probability 1 on a finite (but random) radius around each vertex. We denote as R_F the random variable that assigns minimal such radius to a given vertex, and call it the *coding radius* of F. We denote the space of all Π-colorings of T_Δ as X_Π. This is a subspace of $\Sigma^{H(T_\Delta)}$ that is invariant under $\mathrm{Aut}(T_\Delta)$.

[7]To define the class $\mathsf{CONTINUOUS}$, rather than asking for a continuous solution on all possible Δ-regular Borel graphs, one has to restrict to a smaller family of graphs, otherwise the class trivializes. To define precisely this family is somewhat inconvenient, and not necessary for our arguments.

[8]We always assume $Y = (2^{\mathbb{N}})^{T_\Delta}$ endowed with the product Lebesgue measure.

Definition 10.14. *We say that an LCL Π is in the class* fiid *(*FFIID*) if there is an fiid (ffiid) process X that almost surely produces elements of X_Π.*

Equivalently, we can define $\mathsf{FFIID} = \bigcup_f \mathsf{ULOCAL}(f(\varepsilon))$ *where f ranges over all functions. That is,* FFIID *is the class of problems solvable by any uniform distributed algorithm.*

It is obvious that $\mathsf{FFIID} \subseteq \mathsf{fiid}$. The natural connection between descriptive combinatorics and random processes is formulated by the inclusion $\mathsf{MEASURE} \subseteq \mathsf{fiid}$. While this inclusion is trivially satisfied, e.g., in the case of Δ-regular trees with proper edge Δ-coloring, in our situation we need a routine argument that we include for the sake of completeness in Section 10.7.

Lemma 10.15. *Let Π be an LCL such that $\Pi \in \mathsf{MEASURE}$. Then $\Pi \in \mathsf{fiid}$.*

10.3.5 Specific Local Problems

Here we list some well-known local problems that were studied in all three considered areas.

Edge Grabbing: We start by recalling the well-known problem of edge grabbing (a close variant of the problem is known as sinkless orientation[73]). In this problem, every vertex should mark one of its half-edges (that is, grab an edge) in such a way that no edge can be marked by two vertices. It is known that $\Pi_{\text{edgegrab}} \notin \mathsf{LOCAL}(O(\log^* n))$ [73] but $\Pi_{\text{edgegrab}} \in \mathsf{RLOCAL}(O(\log \log n))$.

Similarly, $\Pi_{\text{edgegrab}} \notin \mathsf{BOREL}$ by [244], but $\Pi_{\text{edgegrab}} \in \mathsf{MEASURE}$ by [107]: to see the former, just note that if a Δ-regular tree admits proper Δ-colorings of both edges and vertices, every vertex can grab an edge of the same color as the color of the vertex. Thus, $\Pi_{\text{edgegrab}} \in \mathsf{BOREL}$ would yield that Δ-regular Borel forests with Borel proper edge-colorings admit a Borel proper Δ-coloring, contradicting [244].

This completes the complexity characterization of Π_{edgegrab} as well as the proper vertex Δ-coloring.

Perfect Matching: Another notorious LCL problem is the perfect matching problem Π_{pm}. Recall that the perfect matching problem Π_{pm} asks for a matching that covers all vertices of the input tree.[9] It is known that $\Pi_{\text{pm}} \in \mathsf{fiid}$ [241], and it is easy to see that $\Pi_{\text{pm}} \notin \mathsf{RLOCAL}(O(\log\log(n)))$ (we will prove a stronger result in Theorem 10.39). Marks proved [244] that it is not in BOREL, even when the underlying tree admits a Borel bipartition. It is not clear if Π_{pm} is in FFIID, nor whether it is in $\mathsf{MEASURE}$.

Graph Homomorphism: We end our discussion with LCLs that correspond to graph homomorphisms (see also the discussion in Section 10.4). These are also known as edge constraint LCLs. Let G be a finite graph. Then we define Π_G to be the LCL that asks for a homomorphism from the input tree to G, that is, vertices are labeled with vertices

[9]In our formalism this means that around each vertex exactly one half-edge is picked.

of G in such a way that edge relations are preserved. There are not many positive results except for the class BAIRE. It follows from the result of Bernshteyn [62] (see Section 10.6) that $\Pi_G \in \mathsf{BAIRE}$ if and only if G is not bipartite. The main negative results can be summarized as follows. An easy consequence of the result of Marks [244] is that if $\chi(G) \leq \Delta$, then $\Pi_G \notin \mathsf{BOREL}$. In this chapter, we describe examples of graphs of chromatic number up to $2\Delta - 2$ such that the corresponding homomorphism problem is not in BOREL, see Section 10.4. The theory of *entropy inequalities* see [17] implies that if G is a cycle on more than 9 vertices, then $\Pi_G \notin \mathsf{fiid}$.

10.4 Generalization of Marks' technique

In this section, we first develop a new way of proving lower bounds in the LOCAL model based on a generalization of a technique of Marks [244]. Then, we use ideas arising in the finitary setting—connected to the adaptation of Marks' technique—to obtain new results back in the Borel context. For an introduction to Marks' technique and a high-level discussion about the challenges in adapting the technique to the standard distributed setting as well as our solution via the new notion of an ID graph, we refer the reader to Section 10.2.1.

The setting we consider in this section is Δ-regular trees that come with a proper Δ-edge coloring with colors from $[\Delta]$. All lower bounds discussed already hold under these restrictions (and therefore also in any more general setting).

Recall that an LCL $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ is given by specifying a list of allowed vertex configurations and a list of allowed edge configurations (see Definition 10.6). To make our lower bounds as widely applicable as possible, we replace the latter list by a separate list for each of the Δ edge colors; in other words, we consider LCLs where the correctness of the output is allowed to depend on the input that is provided by the edge colors. Hence, formally, in this section, an LCL is more generally defined: it is a tuple $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ where Σ and $\mathcal{V}$ are as before, while $\mathcal{E} = (\mathcal{E}_\alpha)_{\alpha \in [\Delta]}$ is now a Δ-tuple of sets $\mathcal{E}_\alpha$ consisting of cardinality-2 multisets. Similarly as before, a half-edge labeling (see Definition 10.10 for the Borel context) with labels from Σ is a correct solution for Π if, for each vertex v, the multiset of labels assigned to the half-edges incident to v is contained in $\mathcal{V}$, and, for each edge e, the multiset of labels assigned to the half-edges belonging to e is contained in $\mathcal{E}_\alpha$, where α is the color of the edge e.

The idea of our approach is to identify a condition that an LCL Π necessarily has to satisfy if it is solvable in $O(\log^* n)$ rounds in the LOCAL model. Showing that Π does not satisfy this condition then immediately implies an $\Omega(\log n)$ deterministic and $\Omega(\log\log n)$ randomized lower bound.

In order to define our condition we need to introduce the notion of a configuration graph: a *configuration graph* is a Δ-tuple $\mathbb{P} = (\mathbb{P}_\alpha)_{\alpha \in \Delta}$ of graphs, where the vertex set of each of the graphs $\mathbb{P}_\alpha$ is the set of subsets of Σ, and there is an edge in $\mathbb{P}_\alpha$ connecting two vertices S, T if and only if there are $\mathtt{a} \in S$ and $\mathtt{b} \in T$ such that $\{\mathtt{a}, \mathtt{b}\} \in \mathcal{E}_\alpha$, note that loops are allowed. (Naturally, we will consider any two vertices of different $\mathbb{P}_\alpha$ to be distinct, even if they correspond to the same subset of Σ.)

Now we are set to define the aforementioned condition. The intuition behind the playability condition is the following: assume that there exists a local algorithm $\mathcal{A}$ that solves Π using the t neighbourhood of a given vertex. We are going to define a family of two player games. The game will depend on some $S \subseteq \Sigma$. Alice and Bob will assign labels (or IDs) to the vertices in the t-neighbourhood (in some way specified later on, depending on $\alpha \in [\Delta]$). When the assignment is complete, we evaluate $\mathcal{A}$ on the root of the obtained labelled graph, this way obtaining an element s of Σ, and decide who is the winner based on $s \in S$ or not. Naturally, it depends on S and α, which player has a winning strategy. This gives rise to a two coloring of vertices of $\mathbb{P}_\alpha$ by colors Alice and Bob. The failure of the playability condition will guarantee that using a strategy stealing argument one can derive a contradiction.

Definition 10.16 (Playability condition)**.** *We say that an LCL Π is* playable *if for every $\alpha \in [\Delta]$ there is a coloring Λ_α of the vertices of $\mathbb{P}_\alpha$ with two colors* $\{\text{Alice}, \text{Bob}\}$ *such that the following conditions are satisfied:*

(A) For any tuple $(S_\alpha)_{\alpha\in[\Delta]} \in V(\mathbb{P}_1) \times \cdots \times V(\mathbb{P}_\Delta)$ satisfying $\Lambda_\alpha(S_\alpha) = \text{Alice}$ for each $\alpha \in [\Delta]$, there exist $\mathtt{a}_\alpha \notin S_\alpha$ such that $\{\mathtt{a}_\alpha\}_{\alpha\in\Delta} \in \mathcal{V}$, and

(B) for any $\alpha \in [\Delta]$, and any tuple $(S, T) \in V(\mathbb{P}_\alpha) \times V(\mathbb{P}_\alpha)$ satisfying $\Lambda_\alpha(S) = \Lambda_\alpha(T) = \text{Bob}$, we have that (S, T) is an edge of $\mathbb{P}_\alpha$.

Our aim in this section is to show the following general results.

Theorem 10.17. *Let Π be an LCL that is not playable. Then Π is not in the class $\mathsf{LOCAL}(O(\log^* n))$.*

Using ideas from the proof of this result, we can formulate an analogous theorem in the Borel context. Let us mention that while Theorem 10.17 is a consequence of Theorem 10.18 by [60], we prefer to state the theorems separately. This is because the proof of the Borel version of the theorem uses heavily complex results such as the Borel determinacy theorem, theory of local-global limits and some set-theoretical considerations about measurable sets. This is in a stark contrast with the proof of Theorem 10.17 that uses 'merely' the existence of large girth graphs and determinacy of finite games. However, the ideas surrounding these concepts are the same in both cases.

Theorem 10.18. *Let Π be an LCL that is not playable. Then Π is not in the class* BOREL.

The main application of these abstract results is to find lower bounds for *(graph) homomorphism LCLs* aka *edge constraint LCLs.* We already defined this class in Section 10.3.1 but we recall the definition in the formalism of this section to avoid any confusion. Let G be a finite graph. Then $\Pi_G = (\Sigma, \mathcal{V}, \mathcal{E})$ is defined by letting

1. $\mathcal{V} = \{(\mathtt{a}, \dots, \mathtt{a}) : \mathtt{a} \in \Sigma\}$,

2. $\Sigma = V(G)$,
3. $\forall \alpha \in [\Delta]$ $(\mathcal{E}_\alpha = E(G))$.

An LCL for which (1) holds is called a *vertex-LCL*. There is a one-to-one correspondence between vertex-LCLs for which $\forall \alpha, \beta$ $(\mathcal{E}_\alpha = \mathcal{E}_\beta)$ and LCLs of the form Π_G. (Indeed, to vertex-LCLs with this property one can associate a graph G whose vertices are the labels in Σ and where two vertices ℓ, ℓ' are connected by an edge if and only if $\{\ell, \ell'\} \in \mathcal{E}$.) Note that if Π is a vertex-LCL, then condition (A) in Definition 10.16 is equivalent to the statement that no tuple $\{S_\alpha\}_{\alpha \in \Delta}$ that satisfies the assumption of (A) can cover Σ, i.e., that $\Sigma \not\subseteq S_1 \cup \cdots \cup S_\Delta$ for such a tuple. Moreover, if Π_G is a homomorphism LCL, then $\mathbb{P}_\alpha = \mathbb{P}_\beta$ for every $\alpha, \beta \in \Delta$.

The simplest example of a graph homomorphism LCL is given by setting G to be the clique on k vertices; the associated LCL Π_G is simply the problem of finding a proper k-coloring of the vertices of the input graph. Now we can easily derive Marks' original result from our general theorem.

Corollary 10.19 (Marks [244]). *Let G be a finite graph that has chromatic number at most Δ. Then Π_G is not playable. In particular, there is a Δ-regular Borel forest that have Borel chromatic number $\Delta + 1$.*

Proof. Let $A_1, \ldots, A_\Delta$ be some independent sets that cover G, and $\Lambda_1, \ldots, \Lambda_\Delta$ arbitrary colorings of the vertices of $\mathbb{P}_1, \ldots, \mathbb{P}_\Delta$, respectively, with colors from $\{\text{Alice}, \text{Bob}\}$. It follows that $\Lambda_\alpha(A_\alpha) = \text{Alice}$ for every $\alpha \in \Delta$, since otherwise condition (B) in Definition 10.16 is violated with $S = T = A_\alpha$. But then condition (A) does not hold. □

As our main application we describe graphs with chromatic number larger than Δ such that Π_G is not playable. This rules out the hope that the complexity of Π_G is connected with chromatic number being larger than Δ. In Section 10.4.1 we show the following. Note that the case $k = \Delta$ is Theorem 10.19.

Theorem 10.20. *Let $\Delta > 2$ and $\Delta < k \leq 2\Delta - 2$. There exists a graph G_k with $\chi(G_k) = k$, such that Π_{G_k} is not playable and $\Pi_{G_k} \in \mathsf{RLOCAL}(O(\log\log n))$. In particular, $\Pi_{G_k} \notin \mathsf{BOREL}$ and $\Pi_{G_k} \notin \mathsf{LOCAL}(O(\log^*(n)))$.*

Interestingly, recent results connected to counterexamples to Hedetniemi's conjecture yield the same statement asymptotically, as $\Delta \to \infty$ (see Remark 10.26).

Remark 10.21. *It can be shown that for $\Delta = 3$ both the Chvátal and Grötsch graphs are suitable examples for $k = 4$.*

Remark 10.22. *As another application of his technique Marks showed in [244] that there is a Borel Δ-regular forest that does not admit Borel perfect matching. This holds even when we assume that the forest is Borel bipartite, i.e., it has Borel chromatic number 2. In order to show this result Marks works with free joins of colored hyperedges, that is,* *Cayley graphs of free products of cyclic groups. One should think of two types of triangles (3-hyperedges) that are joined in such a way that every vertex is contained in both types and there are no cycles. We remark that the playability condition can be formulated in this setting. Similarly, one can derive a version of Theorem 10.18. However, we do not have any application of this generalization.*

10.4.1 Applications of playability to homomorphism LCLs

In this section we find the graph examples from Theorem 10.20. First we introduce a condition Δ-(*) that is a weaker condition than having chromatic number at most Δ, but still implies that the homomorphism LCL is not playable. Then, we will show that–similarly to the way the complete graph on Δ-many vertices, K_Δ, is maximal among graphs of chromatic number $\leq \Delta$–there exists a maximal graph (under homorphisms) with property Δ-(*). Recall that we assume $\Delta > 2$.

Definition 10.23 (Property Δ-(*))**.** *Let $\Delta > 2$ and $G = (V, E)$ be a finite graph. We say that G satisfies property Δ-(*) if there are sets $S_0, S_1 \subseteq V$ such that G restricted to $V \setminus S_i$ has chromatic number at most $(\Delta - 1)$ for $i \in \{0, 1\}$, and there is no edge between S_0 and S_1.*

Note that $\chi(G) \leq \Delta$ implies Δ-(*): indeed, if $A_1, \ldots, A_\Delta$ are independent sets that cover $V(G)$, we can set $S_0 = S_1 = A_1$.

On the other hand, we claim that if G satisfies Δ-(*) then $\chi(G) \leq 2\Delta - 2$. In order to see this, take $S_0, S_1 \subseteq V(G)$ witnessing Δ-(*). Then, as there is no edge between S_0 and S_1, so in particular, between $S_0 \setminus S_1$ and $S_0 \cap S_1$, it follows that the chromatic number of G's restriction to S_0 is $\leq \Delta - 1$. But then we can construct proper $\Delta - 1$-colorings of S_0 and $V(G) \setminus S_0$, which shows our claim.

Proposition 10.24. *Let G be a graph satisfying Δ-(*). Then Π_G is not playable.*

Proof. Fix S_0, S_1 as in the definition of Δ-(*) and assume for a contradiction that colorings $\Lambda_1, \ldots, \Lambda_\Delta$ as described in Definition 10.16 exist. By Property Δ-(*), there exist independent sets $A_1, \ldots, A_{\Delta-1}$ such that S_0 together with the A_i covers G, i.e., such that $S_0 \cup A_1 \cup \cdots \cup A_{\Delta-1} = V(G)$. For each $\alpha \in [\Delta - 1]$, we must have $\Lambda_\alpha(A_\alpha) =$ Alice, since otherwise condition (B) in Definition 10.16 is violated. Consequently $\Lambda_\Delta(S_0) =$ Bob, otherwise condition (A) is violated. Similarly $\Lambda_\Delta(S_1) =$ Bob. This shows that Λ_Δ does not satisfy condition (B) with $S = S_0$ and $T = S_1$. □

Next we describe maximal examples of graphs that satisfy the condition Δ-(*). That is, we define a graph H_Δ that satisfies Δ-(*), its chromatic number is $2\Delta - 2$ and every other graph that satisfies Δ-(*) admits a homomorphism in H_Δ.

Recall that the (categorical) product $G \times H$ of graphs G, H is the graph on $V(G) \times V(H)$, such that $((g, h), (g', h')) \in E(G \times H)$ iff $(g, g') \in E(G)$ and $(h, h') \in E(H)$.

Write P for the product $K_{\Delta-1} \times K_{\Delta-1}$. Let V_0 and V_1 be vertex disjoint copies of $K_{\Delta-1}$. We think of vertices in V_i and P as having labels from $[\Delta-1]$ and $[\Delta-1] \times [\Delta-1]$, respectively. The graph H_Δ is the disjoint union of V_0, V_1, P and an extra vertex $\dagger$ that is connected by an edge to every vertex in P, and additionally, if v is a vertex in V_0 with label $i \in [\Delta-1]$, then we connect it by an edge with $(i', j) \in P$ for every $i' \neq i$ and $j \in [\Delta-1]$, and if v is a vertex in V_1 with label $j \in \Delta-1$, then we connect it by an edge with $(i, j') \in P$ for every $j' \neq j$ and $i \in [\Delta-1]$.

Proposition 10.25. *1. H_Δ satisfies Δ-(*).*

2. $\chi(H_\Delta) = 2\Delta - 2$.

3. A graph G satisfies Δ-(*) if and only if it admits a homomorphism to H_Δ.

Proof. (1) Set $S_0 = V(V_0) \cup \{\dagger\}$ and $S_1 = V(V_1) \cup \{\dagger\}$. By the definition there are no edges between S_0 and S_1. Consider now, e.g., $V(H_\Delta) \setminus S_0$. Let A_j consist of all elements in P that have second coordinate equal to j together with the vertex in V_1 that has the label j. By the definition, the set A_i is independent and $\bigcup_{i \in [\Delta-1]} A_i$ covers $H_\Delta \setminus S_0$, and similarly for S_1.

(2) By (1) and the claim after the definition of Δ-(*), it is enough to show that $\chi(H_\Delta) \geq 2\Delta - 2$. Towards a contradiction, assume that c is a proper coloring of H_Δ with $< 2\Delta - 2$-many colors. Note the vertex $\dagger$ guarantees that $|c(V(P))| \leq 2\Delta - 4$, and also $\Delta - 1 \leq |c(V(P))|$.

First we claim that there are no indices $i, j \in [\Delta-1]$ (even with $i = j$) such that $c(i, r) \neq c(i, s)$ and $c(r, j) \neq c(s, j)$ for every $s \neq r$: indeed, otherwise, by the definition of P we had $c(i, r) \neq c(s, j)$ for every r, s unless $(i, r) = (s, j)$, which would the upper bound on the size of $c(V(P))$.

Therefore, without loss of generality, we may assume that for every $i \in [\Delta-1]$ there is a color α_i and two indices $j_i \neq j'_i$ such that $c(i, j_i) = c(i, j'_i) = \alpha_i$. It follows form the definition of P and $j_i \neq j'_i$ that $\alpha_i \neq \alpha_{i'}$ whenever $i \neq i'$.

Moreover, note that any vertex in V_1 is connected to at least one of the vertices (i, j_i) and (i, j'_i), hence none of the colors $\{\alpha_i\}_{i \in [\Delta-1]}$ can appear on V_1. Consequently, since V_1 is isomorphic to $K_{\Delta-1}$ we need to use at least $\Delta - 1$ additional colors, a contradiction.

(3) First note that if G admits a homomorphism into H_Δ, then the pullbacks of the sets witnessing Δ-(*) will witness that G has Δ-(*).

Conversely, let G be a graph that satisfies Δ-(*). Fix the corresponding sets S_0, S_1 together with $(\Delta-1)$-colorings c_0, c_1 of their complements. We construct a homomorphism

Θ from G to H_Δ. Let

$$\Theta(v) = \begin{cases} \dagger & \text{if } v \in S_0 \cap S_1, \\ c_0(v) & \text{if } v \in S_1 \setminus S_0, \\ c_1(v) & \text{if } v \in S_0 \setminus S_1, \\ (c_0(v), c_1(v)) & \text{if } v \notin S_0 \cup S_1. \end{cases}$$

Observe that $S = S_0 \cap S_1$ is an independent set such that there is no edge between S and $S_0 \cup S_1$. Using this observation, one easily checks case-by-case that Θ is indeed a homomorphism. □

Now, combining what we have so far, we can easily prove Theorem 10.20.

Proof of Theorem 10.20. It follows that Π_{H_Δ} is not playable from Theorem 10.25, Theorem 10.24. It is easy to see that if for a graph G the LCL Π_G is not playable then $\Pi_{G'}$ is not playable for every subgraph G' of G. Since erasing a vertex decreases the chromatic number with at most one, for each $k \leq 2\Delta - 2$ there is a subgraph G_k of H_Δ with $\chi(G_k) = k$, such that Π_{G_k} is not playable.

It follows from Theorems 10.17 and 10.18 that there is a Δ-regular Borel forest that admits no Borel homomorphism to any graph of G_k and that $\Pi_{G_k} \notin \mathsf{LOCAL}(O(\log^*(n)))$.

Finally, note that if $k \geq \Delta$ then G_k can be chosen so that it contains K_Δ, yielding $\Pi_{G_k} \in \mathsf{RLOCAL}(\ O(\log\log(n))$. □

Remark 10.26. *Recall that Hedetniemi's conjecture is the statement that if G, H are finite graphs then $\chi(G \times H) = \min\{\chi(G), \chi(H)\}$. This conjecture has been recently disproven by Shitov [283], and strong counterexamples have been constructed later (see, [291, 298]). We claim that these imply for $\varepsilon > 0$ the existence of finite graphs H with $\chi(H) \geq (2-\varepsilon)\Delta$ to which Δ-regular Borel forests cannot have a homomorphism in BOREL, for every large enough Δ. Indeed, if a Δ-regular Borel forest admitted a Borel homomorphism to each finite graph of chromatic number at least $(2-\varepsilon)\Delta$, it would have such a homomorphism to their product as well. Thus, we would obtain that the chromatic number of the product of any graphs of chromatic number $(2-\varepsilon)\Delta$ is at least $\Delta+1$. This contradicts Zhu's result [298], which states that the chromatic number of the product of graphs with chromatic number n can drop to $\approx \frac{n}{2}$.*

Remark 10.27. *A natural attempt to construct graphs with large girth and not playable homomorphisms problem would be to consider random d-regular graphs of size n for a large enough n. However, it is not hard to see that setting $\Lambda(A) =$ Alice if and only if $|A| < \frac{n}{d}$ shows that this approach cannot work.*

10.4.2 Generalization of Marks' Technique

In this section we prove Theorem 10.17, by applying Marks' game technique in the LOCAL setting. In order to define our games, we will need certain auxiliary graphs, the so-called *ID graphs.* The purpose of these graphs is to define a "playground" for the games that we consider. Namely, vertices in the game are labeled by vertices from the ID graph in such a way that the at the end we obtain a homomorphism from our underlying graph to the ID graph.

Definition 10.28. *A pair $\mathbf{H}_{n,t,r} = (H_{n,t,r}, c)$ is called an* ID graph, *if*

1. *$H_{n,t,r}$ is graph with girth at least $2t + 2$,*
2. *$|V(H_{n,t,r})| \leq n$,*
3. *c is a Δ-edge-coloring of $H_{n,t,r}$, such that every vertex is adjacent to at least one edge of each color,*
4. *for each $\alpha \in [\Delta]$ the ratio of a maximal independent set of $H^{\alpha}_{n,t,r} = (V(H_{n,t,r}), E(H_{n,t,r}) \cap c^{-1}(\alpha))$ is at most r (i.e., $H^{\alpha}_{n,t,r}$ is the graph formed by α-colored edges).*

Before we define the game we show that ID graphs exist.

Proposition 10.29. *Let $t_n \in o(\log(n))$, $r > 0$, $\Delta \geq 2$. Then there is an ID graph $\mathbf{H}_{n,t_n,r}$ for every $n \in \mathbb{N}$ sufficiently large.*

Proof. We use the *configuration model* for regular random graphs, see [296]. This model is defined as follows. Let n be even. Then a d-regular random sample on n-vertices is just a union of d-many independent uniform random perfect matchings. Note that in this model we allow parallel edges.

It was proved by Bollobás [70] that the independence ratio of a random d-regular graph is at most $\frac{2\log(d)}{d}$ a.a.s. Moreover, this quantity is concentrated by an easy application of McDiarmid's result [249], i.e.,

$$\mathrm{P}\left(|X - \mathrm{E}(X)| \geq \sqrt{n}\right) < 2\exp\left(-\frac{n}{d}\right), \tag{10.1}$$

where X is the random variable that counts the size of a maximal independent set. Therefore for fixed n large enough we have that the independence ratio of a random sample is at most $\frac{3\log(d)}{d}$ with probability at least $\left(1 - 2\exp\left(-\frac{n}{d}\right)\right)$.

Pick a d large enough such that $\frac{3\log(d)}{d} < r$. Now for an n large enough take Δ-many independent samples of random d-regular graphs according to the configuration model. Note that this is a random sample from the configuration model for Δd-regular graphs. We define c to be equal to $\alpha \in [\Delta]$ on edges of the α-th sample. Then condition (4) is satisfied with probability at least

$$\left(1 - 2\exp\left(-\frac{n}{d}\right)\right)^{\Delta}.$$

It remains to show that the girth condition is satisfied. Recall that we assume $t_n \in o(\log n)$. Then we have $(\Delta d-1)^{2t_n-1} \in o(n)$. Using [250, Corollary 1] we have that the probability of having girth at least t_n is

$$\begin{aligned}\exp\left(-\sum_{a=3}^{t_n}\frac{(\Delta d-1)^a}{2a}+o(1)\right) &\geq \exp\left(-(\Delta d)^{t_n}\right)\\ &\geq \exp(-o(n)) \\ &> 1-\left(1-2\exp\left(-\frac{n}{d}\right)\right)^{\Delta}\end{aligned} \tag{10.2}$$

as $n \to \infty$. This shows that there exists such a graph $\mathbf{H}_{t_n,n,r}$ with non-zero probability. □

Next, we define the games. As mentioned before, the games are going to depend on the following parameters: an algorithm $\mathcal{A}_n$ of local complexity $t \in o(\log(n))$, an ID graph $\mathbf{H}_{n,t,r}$, $\alpha \in \Delta$, $\sigma \in V(H_{n,t,r})$ and $S \subseteq \Sigma$. (We will view $\mathcal{A}_n$, $\mathbf{H}_{n,t,r}$ as fixed, and the rest of the parameters as variables).

The game

$$\mathbb{G}(\mathcal{A}_n, n, t, \mathbf{H}_{n,t,r})[\alpha, \sigma, S]$$

is defined as follows: two players, Alice and Bob assign labels to the vertices of a rooted Δ-regular tree of diameter t. The labels are vertices of $H_{n,t,r}$ and the root is labeled σ. In the k-th round, where $0 < k \leq t$, first Alice labels vertices of distance k from the root on the side of the α edge. After that, Bob labels all remaining vertices of distance k, etc. We also require the assignment of labels to give rise to an edge-color preserving homomorphism to $\mathbf{H}_{n,t,r}$. (For example, if it is Alice's turn to label a neighbor of some vertex v, that has been assigned a label $\rho \in V(H_{n,t,r})$ in the previous round, along an edge that has color β, then the allowed labels are only those that span a β edge with ρ in $H_{n,t,r}$).

By property (2) of the ID graph, we can fix an injective map from $V(H_{n,t,r})$ to $[n]$. Now, we say that Alice wins an instance of the game iff $\mathcal{A}_n$ applied to the produced labeling of the rooted tree does **not** produce an element of S on the half edge that starts in the root and has edge color α. Note that this is well defined thanks to our assumption on the girth of $H_{n,t,r}$. Let us define $\Lambda^{\sigma}_{\alpha}(S)$ to be Alice or Bob depending on who has a winning strategy in the game

$$\mathbb{G}(\mathcal{A}_n, n, t, \mathbf{H}_{n,t,r})[\alpha, \sigma, S].$$

Since the game is finite and there is no draw, one of the players has a winning strategy. Thus, we have that

$$\Lambda^{\sigma}_{\alpha} : \mathbb{P}_{\alpha} \to \{\text{Alice}, \text{Bob}\}$$

is well-defined for all $\sigma \in V(H_{n,t,r})$ and $\alpha \in [\Delta]$.

Proposition 10.30. *Let $\mathcal{A}_n$ be an algorithm of local complexity $t \in o(\log(n))$ that solves Π and $\sigma \in V(H_{n,t,r})$. Then $(\Lambda^{\sigma}_{\alpha})_{\alpha\in\Delta}$ satisfies (A) in Theorem 10.16.*

Proof. Assume that $(S_\alpha)_{\alpha\in\Delta}$ is such that $\Lambda^\sigma_\alpha(S_\alpha) = \text{Alice}$. This means that Alice has winning strategy in all the games corresponding to S_α. Letting these strategies play against each other in the obvious way, produces a labeling of the tree. Since $\mathcal{A}_n$ solves Π it has to output labeling of half edges $(\mathsf{a}_\alpha)_{\alpha\in\Delta} \in \mathcal{N}$, where a_α is a label on the half edge that start at the root and has color α. Note that we must have $\mathsf{a}_\alpha \notin S_\alpha$ by the definition of winning strategy for Alice. This shows that (A) of Definition 10.16 holds. □

We are ready to prove Theorem 10.17.

Proof of Theorem 10.17. Let $(\mathcal{A}_n)_{n\in\mathbb{N}}$ be a sequence of algorithms of local complexity $t_n \in o(\log(n))$ that solve Π and $N \in \mathbb{N}$ be the number of all possible colorings of vertices of $(\mathbb{P}_\alpha)_{\alpha\in\Delta}$ with two colors, that is, $N = 2^{\sum_\alpha |V(\mathbb{P}_\alpha)|}$. Set $r := \frac{1}{N+1}$. By Theorem 10.29 there exists an ID graph $\mathbf{H}_{n,t,r}$. Thus, the games above are well-defined and we can construct the functions $(\Lambda^\sigma_\alpha)_{\alpha\in\Delta}$. Since there are only N possibilities for such a sequence, there exists a set $X \subseteq V(H_{n,t,r})$ of relative size greater than r, such that for all $\sigma \in X$ the sequence of functions $(\Lambda^\sigma_\alpha)_{\alpha\in\Delta}$ is the same.

Since Π is not playable, we can find an edge color $\alpha \in \Delta$ and sets $S, T \in \mathbb{P}_\alpha$ such that Λ^σ_α does not satisfy (B) from Theorem 10.16 with S, T for every $\sigma \in X$. Note that this is because (A) is always satisfied by Theorem 10.30.

By property (4) of the ID graph, there exist $\sigma_0, \sigma_1 \in X$ that span an α edge in $H_{n,t,r}$. We let the winning strategies of Bob in the games

$$\mathbb{G}(\mathcal{A}_n, n, t, \mathbf{H}_{n,t,r})[\alpha, \sigma_0, S],\ \mathbb{G}(\mathcal{A}_n, n, t, \mathbf{H}_{n,t,r})[\alpha, \sigma_1, T],$$

play against each other, where we start with an α edge with endpoints labeled by σ_0, σ_1. This produces a labeling of the vertices that have distance at most t from either of the endpoints of the edge (intuitively, the tree "rooted" at this edge of "diameter" $t+\frac{1}{2}$). Now applying $\mathcal{A}_n$ on the vertex with label σ_0 produces $\mathsf{a}_0 \in S$ on the the half edge that starts at this vertex and has color α. Similarly, we produce $\mathsf{a}_1 \in T$. However, $(\mathsf{a}_0, \mathsf{a}_1) \notin \mathbf{E}_\alpha$ by the definition of an edge in $\mathbb{P}_\alpha$. This shows that $\mathcal{A}_n$ does not solve Π. □

10.4.3 Proof of the Borel Impossibility Result

We show how to use an infinite analogue of the ID graph to prove our main result in the Borel context, Theorem 10.18. Finding such a graph/graphing is in fact much easier in this context.

Definition 10.31. *Let $r > 0$. $\mathbf{H}_r = (\mathcal{H}_r, c)$ is an* ID graphing*, if*

1. *$\mathcal{H}_r$ is an acyclic locally finite Borel graphing on a standard probability measure space (X, μ),*
2. *c is a Borel Δ-edge-coloring of $\mathcal{H}_r$, such that every vertex is adjacent to at least one edge of each color,*

3. *for each $\alpha \in [\Delta]$ the μ-measure of a maximal independent set of $\mathcal{H}_r^\alpha = (V(\mathcal{H}_r), E(\mathcal{H}_r) \cap c^{-1}(\alpha))$ is at most r.*

Proposition 10.32. *For each $r > 0$ there exists an ID graphing $\mathbf{H}_r$.*

Proof. Let $\mathcal{H}_r$ be a *local-global limit* of the random graphs constructed in Theorem 10.29 (see, e.g., [202] for the basic results about local-global convergence). It is not hard to check that this limit satisfies the required properties. □

Now we are ready to prove the theorem. The proof will closely follow the argument given in the proof of the LOCAL version, i.e., Theorem 10.17, but can be understood without reading the latter.

Proof of Theorem 10.18. Let Π be an LCL that is not playable, and $N \in \mathbb{N}$ be the number of all possible colorings of vertices of $(\mathbb{P}_\alpha)_{\alpha\in\Delta}$ with two colors, that is, $N = 2^{\sum_\alpha |V(\mathbb{P}_\alpha)|}$. Set $r := \frac{1}{N+1}$.

We define a Borel acyclic Δ-regular graph $\mathcal{G}$, with edges properly colored by Δ, that does not admit a Borel solution of Π. Vertices of $\mathcal{G}$ are pairs (x, A), where $x \in X$ is a vertex of $\mathcal{H}_r$ and A is a countable subgraph of $\mathcal{H}_r$ that is a Δ-regular tree that contains x and the edge coloring of $\mathbf{H}_r$ induces a proper edge coloring of A. We say that (x, A) and (y, B) are connected by an α-edge in $\mathcal{G}$ if $A = B$, x, y are adjacent in A and the edge that connects them has color $\alpha \in \Delta$.

Suppose for a contradiction that $\mathcal{A}$ is a Borel function that solves Π on $\mathcal{G}$.

Next, we define a family of games parametrized by $\alpha \in [\Delta]$, $x \in V(\mathcal{H}_r)$ and $S \subseteq \Sigma$. For the reader familiar with Marks' construction, let us point out that for a fixed x, the games are analogues to the ones he defines, with the following differences: allowed moves are vertices of the ID graphing $\mathcal{H}_r$ and restricted by its edge relation, and the winning condition is defined by a set of labels, not just merely one label.

So, the game

$$\mathbb{G}(\mathcal{A}, \mathbf{H}_r)[\alpha, x, S]$$

is defined as follows: Alice and Bob alternatingly label vertices of a Δ-regular rooted tree. The root is labelled by x, and the labels come from $V(\mathcal{H}_r)$. In the k-th round, first Alice labels vertices of distance k from the root on the side of the α edge. After that, Bob labels all remaining vertices of distance k, etc. We also require the assignment of labels to give rise to an edge-color preserving homomorphism to $\mathbf{H}_r$.

It follows from the acyclicity of $\mathcal{H}_r$ that a play of a game determines a Δ-regular rooted subtree of $\mathcal{H}_r$ to which the restriction of the edge-coloring is proper. That is, it determines a vertex (x, A) of $\mathcal{G}$. Let Alice win iff the output of $\mathcal{A}$ on the half-edge determined by (x, A) and the color α is **not** in S.

Define the function $\Lambda_\alpha^x : \mathbb{P}_\alpha \to \{\text{Alice}, \text{Bob}\}$ assigning the player to some $S \in V(\mathbb{P}_\alpha)$ who has a winning strategy in $\mathbb{G}(\mathcal{A}, \mathbf{H}_r)[\alpha, x, S]$. Note that, since $\mathcal{H}_r$ is locally finite, each player has only finitely many choices at each position. Thus, it follows from Borel Determinacy Theorem that Λ^x_α is well defined.

Now we show the analogue of Proposition 10.30.

Proposition 10.33. $(\Lambda^x_\alpha)_{\alpha\in\Delta}$ *satisfies (A) in Theorem 10.16.*

Proof. Assume that $(S_\alpha)_{\alpha\in\Delta}$ is such that $\Lambda^x_\alpha(S_\alpha) = \text{Alice}$. This means that Alice has winning strategy in all the games corresponding to S_α. Letting these strategies play against each other in the obvious way, produces a vertex (x, A) of $\mathcal{G}$. Since $\mathcal{A}$ solves Π it has to output labeling of half edges $(\mathsf{a}_\alpha)_{\alpha\in\Delta} \in \mathcal{N}$, where a_α is a label on the half edge that start at (x, A) and has color α. Note that we must have $\mathsf{a}_\alpha \notin S_\alpha$ by the definition of winning strategy for Alice. This shows that (A) of Definition 10.16 holds. $\square$

Let f be defined by

$$x \mapsto (\Lambda^x_\alpha)_{\alpha\in\Delta}.$$

Note that f has a finite range. Using the fact that the allowed moves for each player can be determined in a Borel way, uniformly in x, it is not hard to see that for each element s in the range, $f^{-1}(s)$ is in the algebra generated by sets that can be obtained by applying game quantifiers to Borel sets (for the definition see [218, Section 20.D]). It has been shown by Solovay and independently by Fenstad-Norman [130] that such sets are provably Δ^1_2, and consequently, measurable. Therefore, f is a measurable map.

As the number of sequences of functions $(\Lambda^x_\alpha)_{\alpha\in\Delta}$ is $\leq N$, by the choice of r, there exists a Borel set Y with $\mu(Y) > r$, such that f is constant on Y.

Since Π is not playable Proposition 10.33 gives that there is an $\alpha \in \Delta$ and $S, T \in \mathbb{P}_\alpha$ that violate condition (B). Recall that the measure of an independent Borel set of $\mathcal{H}^\alpha_r$ is at most r. That means that there are $x, y \in Y$ such that (x, y) is an α-edge in $\mathcal{H}_r$. We let the winning strategies of Bob in the games

$$\mathbb{G}(\mathcal{A}, \mathbf{H}_r)[\alpha, x, S],\ \mathbb{G}(\mathcal{A}, \mathbf{H}_r)[\alpha, y, T],$$

play against each other, where we start with an α edge with endpoints labeled by x, y. This produces a labeling of a Δ-regular tree A with labels from $\mathcal{H}_r$ such that (x, A) and (y, A) span an α-edge in $\mathcal{H}_r$. Applying $\mathcal{A}$ on the half-edge determined by (x, A) and the color α, we obtain $\mathsf{a}_0 \in S$. Similarly, we produce $\mathsf{a}_1 \in T$ for (y, A). However, $(\mathsf{a}_0, \mathsf{a}_1) \notin \mathbf{E}_\alpha$ by the definition of an edge in $\mathbb{P}_\alpha$. This shows that $\mathcal{A}$ does not produce a Borel solution to Π. $\square$

Remark 10.34. *One can give an alternative proof of the existence of a Borel Δ-regular forest that does not admit a Borel homomorphism to G_k that avoids using the graphing $\mathcal{H}_r$ and uses the original example described by Marks [244] instead. The reason is that the example graphs G_k contain K_Δ, as discussed in the proof of Theorem 10.20. This takes care of the "non-free" as in the argument of Marks. Then it is enough to use the pigeonhole principle on $\mathbb{N}$ instead of the independence ratio reasoning.*

10.5 Separation Of Various Complexity Classes

We first show $\mathsf{ULOCAL}(\operatorname{poly}\log 1/\varepsilon) \neq \mathsf{ULOCAL}(O(\log\log 1/\varepsilon))$. This shows that there are problems such that their worst case complexity is at least $\Theta(\log n)$ on finite Δ-regular trees, but their average local complexity is constant. Second, we show that there are problems in the class BOREL that are not in the class $\mathsf{RLOCAL}(O(\log\log n)) = \mathsf{RLOCAL}(o(\log n))$, on Δ-regular trees. This shows that one cannot in general hope that results from (Borel) measurable combinatorics can be turned into very efficient (sublogarithmic) distributed algorithms.

10.5.1 Preliminaries

We first show that there is a strong connection between randomized local complexities and uniform local complexities. Afterwards, we introduce a generic construction that turns any LCL into a new LCL. We later use this generic transformation to construct an LCL that is contained in the set $\mathsf{BOREL} \setminus \mathsf{RLOCAL}(O(\log\log n))$.

Uniform vs Local Randomized Complexity:

We will now discuss the connections between uniform and randomized complexities. Note that the easy part of the connection between the two concepts is turning uniform local algorithms into randomized ones, as formalized in the following proposition.

Proposition 10.35. *On bounded-degree graphs we have that* $\mathsf{OLOCAL}(t(\varepsilon)) \subseteq \mathsf{RLOCAL}(t(1/n^{O(1)}))$.

Proof. We claim that an uniform local algorithm $\mathcal{A}$ with an uniform local complexity of $t(\varepsilon)$ can be turned into a local randomized algorithm $\mathcal{A}'$ with a local complexity of $t(1/n^{O(1)})$.

The algorithm $\mathcal{A}'$ simulates $\mathcal{A}$ on an infinite Δ-regular tree – each vertex u of degree less than Δ in the original tree pretends that the tree continues past its virtual half-edges and the random bits of u are used to simulate the random bits in this virtual subtree. Choosing $\varepsilon = 1/n^{O(1)}$, one gets that the probability of $\mathcal{A}'$ needing to look further than $t(\varepsilon)$ for any vertex is bounded by $1/n^{O(1)}$, as needed in the definition of the randomized local complexity. □

On the other hand, we will use the following proposition from [185]. It informally states that the existence of *any* uniform local algorithm together with the existence of a sufficiently fast randomized local algorithm for a given LCL Π directly implies an upper bound on the uniform complexity of Π.

Proposition 10.36 ([185])**.** *Let $\mathcal{A}$ be an uniform local algorithm solving an LCL problem Π such that its* randomized *local complexity on finite Δ-regular trees is $t(n)$ for $t(n) = o(\log n)$. Then, the uniform local complexity of $\mathcal{A}$ on infinite Δ-regular trees is $O(t(1/\varepsilon))$.*

Theorem 10.36 makes our life simpler, since we can take a known randomized distributed local algorithm with local complexity $g(n) = o(\log n)$ and check if it works without the knowledge of n. If yes, this automatically implies that the uniform local complexity of the algorithm is $h(\varepsilon) = O(g(1/\varepsilon))$. In particular, combining Theorem 10.36 with the work of [94, 88] and verifying that the algorithms of Fischer and Ghaffari [133, Section 3.1.1] and Chang et al. [90, Section 5.1] work without the knowledge of n, we obtain the following result.

Theorem 10.37. *We have:*

- $\mathsf{LOCAL}(O(1)) = \mathsf{ULOCAL}(O(1))$
- $\mathsf{RLOCAL}(O(\log^* n)) = \mathsf{ULOCAL}(O(\log^* 1/\varepsilon))$
- $\mathsf{RLOCAL}(O(\log\log n)) = \mathsf{ULOCAL}(O(\log\log 1/\varepsilon))$

Moreover, there are no other possible uniform local complexities for $t(\varepsilon) = o(\log 1/\varepsilon)$.

We note that the first two items are proven in [185]. For the third item it suffices by known reductions to find an uniform algorithm solving a version of the distributed Lovász Local Lemma (LLL) on so-called tree-structured dependency graphs considered in [90].

Proof sketch. The proof follows from Theorem 10.35 and the following ideas. The first item follows from the fact that any local algorithm with local complexity $O(1)$ can simply be made uniform. More specifically, there exists a constant n_0 — depending only on the problem Π — such that the algorithm, being told the size of the graph is n_0, is correct on *any* graph of size $n \geq n_0$.

For the second item, by the work of [94, 88], it suffices to check that there is an uniform distributed $(\Delta + 1)$-coloring algorithm with uniform local complexity $O(\log^* 1/\varepsilon)$. Such an algorithm was given in [212], or follows from the work in [223] and Theorem 10.36.

Similarly, for the third item, by the work of [88] it suffices to check that there is an uniform distributed algorithm for a specific LLL problem on trees with uniform local complexity $O(\log\log 1/\varepsilon)$. Such an algorithm can be obtained by combining the randomized *pre-shattering* algorithm of Fischer and Ghaffari [133, Section 3.1.1] and the deterministic *post-shattering* algorithm of Chang et al. [90, Section 5.1] in a *graph shattering* framework [48], which solves the LLL problem with local complexity $O(\log\log n)$. By Theorem 10.36, it suffices to check that this algorithm can be made to work even if it does not know the size of the graph n. We defer the details to Section 10.8. This finishes the proof of Theorem 10.37. □

Adding Paths: Before we proceed to show some separation results, we define a certain construction that turns any LCL problem Π into a new LCL problem $\overline{\Pi}$ with the following property. If the original problem Π cannot be solved by a fast local algorithm, then the same holds for $\overline{\Pi}$. However, $\overline{\Pi}$ might be strictly easier to solve than Π for BOREL constructions.

Definition 10.38. *Let $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ be an LCL. We define an LCL $\overline{\Pi} = (\Sigma', \mathcal{V}', \mathcal{E}')$ as follows. Let Σ' be Σ together with one new label. Let $\mathcal{V}'$ be the union of $\mathcal{V}$ together with any cardinality-Δ multiset that contains the new label exactly two times. Let $\mathcal{E}'$ be the union of $\mathcal{E}$ together with the cardinality-2 multiset that contains the new label twice.*

In other words, the new label determines doubly infinite lines in infinite Δ-regular trees, or lines that start and end in virtual half-edges in finite Δ-regular trees. Moreover, a vertex that is on such a line does not have to satisfy any other vertex constraint. We call these vertices *line-vertices* and each edge on a line a *line-edge.*

Proposition 10.39. *Let Π be an LCL problem such that $\overline{\Pi} \in \mathsf{RLOCAL}(t(n))$ for $t(n) = o(\log(n))$. Then also $\Pi \in \mathsf{RLOCAL}(O(t(n)))$.*

Proof. Let $\overline{\mathcal{A}}$ be a randomized LOCAL algorithm that solves $\overline{\Pi}$ in $t(n) = o(\log n)$ rounds with probability at least $1-1/n^C$ for some sufficiently large constant C. We will construct an algorithm $\mathcal{A}$ for Π with complexity $t(n)$ that is correct with probability $1 - \frac{4}{n^{C/3-2}}$. The success probability of $\mathcal{A}$ can then be boosted by "lying to it" that the number of vertices is $n^{O(1)}$ instead of n; this increases the running time by at most a constant factor and boosts the success probability back to $1 - 1/n^C$.

Consider a Δ-regular rooted finite tree T of depth $10t(n)$ and let u be its root vertex. Note that $|T| \leq \Delta^{10t(n)+1} < n$, for n large enough.

We start by proving that when running $\overline{\mathcal{A}}$ on the tree T, then u is most likely not a line-vertex. This observation then allows us to turn $\overline{\mathcal{A}}$ into an algorithm $\mathcal{A}$ that solves Π on any Δ-regular input tree. Let X be the indicator of $\mathcal{A}$ marking u as a line-vertex. Moreover, for $i \in [\Delta]$, let Y_i be the indicator variable for the following event. The number of line edges in the i-th subtree of u with one endpoint at depth $5t(n)$ and the other endpoint at depth $5t(n) + 1$ is odd.

By a simple parity argument we can relate the value of X with the values of $Y_1, Y_2, \ldots, Y_\Delta$ as follows. If $X = 0$, that is, u is not a line-vertex, then all of the Y_i's have to be 0, as each path in the tree T is completely contained in one of the Δ subtrees of u. On the other hand, if u is a line-vertex, then there exists exactly one path, the one containing u, that is not completely contained in one of the Δ subtrees of u. This in turn implies that exactly two of the Δ variables $Y_1, Y_2, \ldots, Y_\Delta$ are equal to 1.

The random variables $Y_1, Y_2, \ldots, Y_\Delta$ are identically distributed and mutually independent. Hence, if $\mathrm{P}(Y_i = 1) > \frac{1}{n^{C/3}}$, then the probability that there are at least 3 Y_i's equal to 1 is strictly greater than $\left(\frac{1}{n^{C/3}}\right)^3 = \frac{1}{n^C}$. This is a contradiction, as in that case $\overline{\mathcal{A}}$ does not produce a valid output, which according to our assumption happens with probability at most $\frac{1}{n^C}$.

Thus, we can conclude that $\mathrm{P}(Y_i = 1) \leq \frac{1}{n^{C/3}}$. By a union bound, this implies that all of the Y_i's are zero with probability at least $\frac{1}{n^{C/3-1}}$, and in that case u is not a line-vertex.

Finally, the algorithm $\mathcal{A}$ simulates $\overline{\mathcal{A}}$ as if the neighborhood of each vertex was a Δ-regular branching tree up to depth at least $t(n)$.

It remains to analyze the probability that $\mathcal{A}$ produces a valid solution for the LCL problem Π. To that end, let v denote an arbitrary vertex of the input tree. The probability that the output of v's half edges satisfy the vertex constraint of the LCL problem $\overline{\Pi}$ is at least $1 - 1/n^C$. Moreover, in the case that v is not a line-vertex, which happens with probability at least $1 - \frac{1}{n^{C/3-1}}$, the output of v's half edges even satisfy the vertex constraint of the LCL problem Π. Hence, by a union bound the vertex constraint of the LCL problem Π around v is satisfied with probability at least $1 - \frac{2}{n^{C/3-1}}$. With exactly the same reasoning, one can argue that the probability that the output of $\mathcal{A}$ satisfies the edge constraint of the LCL problem Π at a given edge is also at least $1 - \frac{2}{n^{C/3-1}}$. Finally, doing a union bound over the n vertex constraints and $n-1$ edge constraints it follows that $\mathcal{A}$ produces a valid solution for Π with probability at least $1 - \frac{4}{n^{C/3-2}}$. □

Remark 10.40. *The ultimate way how to solve LCLs of the form $\overline{\Pi}$ is to find a spanning forest of lines. This is because once we have a spanning forest of lines, then we might declare every vertex to be a line-vertex and label every half-edge that is contained on a line with the new symbol in $\overline{\Pi}$. The remaining half-edges can be labeled arbitrarily in such a way that the edge constraints are satisfied.*

Put otherwise, once we are able to construct a spanning forest of lines and $\mathcal{E} \neq \emptyset$, where $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$, then $\overline{\Pi}$ can be solved. Moreover, in that case the complexity of $\overline{\Pi}$ is upper bounded by the complexity of finding the spanning forest of lines.

Lyons and Nazarov [241] showed that $\Pi_{pm} \in \mathsf{fiid}$ and it is discussed in [240] that the construction can be iterated $(\Delta - 2)$-many times. After removing each edge that is contained in one of the $\Delta - 2$ perfect matchings each vertex has a degree of two. Hence, it is possible to construct a spanning forest of lines as fiid. Consequently, $\overline{\Pi} \in \mathsf{fiid}$ for every $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ with $\mathcal{E} \neq \emptyset$.

We now formalize the proof that $\mathsf{LOCAL}(O(\log^* n)) \neq \mathsf{BOREL}$ from Section 10.2.2. Recall that $\Pi_{\chi,\Delta}$ is the proper vertex Δ-coloring problem. The following claim directly follows by Theorem 10.39 together with the fact that $\Pi_{\chi,\Delta} \notin \mathsf{LOCAL}(O(\log^* n))$.

Claim 10.41. *We have $\overline{\Pi_{\chi,\Delta}} \notin \mathsf{LOCAL}(O(\log^* n))$.*

On the other hand we show that adding lines helps to find a Borel solution. This already provides a simple example of a problem in the set $\mathsf{BOREL} \setminus \mathsf{LOCAL}(O(\log^* n))$.

Proposition 10.42. *We have $\overline{\Pi_{\chi,\Delta}} \in \mathsf{BOREL}$.*

Proof. By [220], every Borel graph of finite maximum degree admits a Borel maximal independent set. Let $\mathcal{G}$ be a Borel Δ-regular acyclic graph on a standard Borel space X. By an iterative application of the fact above we find Borel sets $A_1, \dots, A_{\Delta-2}$ such that A_1 is a maximal independent set in $\mathcal{G}$ and A_i is a maximal independent set in $\mathcal{G} \setminus \bigcup_{j<i} A_j$

for every $i > 1$. We think of $A_1, \ldots, A_{\Delta-2}$ as color classes for the first $\Delta - 2$ colors. Let $B = X \setminus \bigcup_{i\in[\Delta-2]} A_i$ and let $\mathcal{H}$ be the subgraph of $\mathcal{G}$ determined by B. It is easy to see that the maximum degree of $\mathcal{H}$ is 2. In particular, $\mathcal{H}$ consists of finite paths, one-ended infinite paths or doubly infinite paths. We use the extra label in $\overline{\Pi_{\chi,\Delta}}$ to mark the doubly infinite paths. It remains to use the remaining 2 colors in $[\Delta]$ to define a proper vertex 2-coloring of the finite and one-ended paths. It is a standard argument that this can be done in a Borel way and it is easy to verify that, altogether, we found a Borel $\overline{\Pi_{\chi,\Delta}}$-coloring of $\mathcal{G}$. □

10.5.2 Examples and Lower Bound

In this subsection we define two LCLs and show that they are not in the class $\mathsf{RLOCAL}(o(\log n))$. In the next subsection we show that one of them is in the class $\mathsf{ULOCAL}(\operatorname{poly}\log 1/\varepsilon)$ and the other in the class BOREL. Both examples that we present are based on the following relaxation of the perfect matching problem.

Definition 10.43 (Perfect matching in power-2 graph). *Let Π^2_{pm} be the perfect matching problem in the power-2 graph, i.e., in the graph that we obtain from the input graph by adding an edge between any two vertices that have distance at most 2 in the input graph.*

We show that Π^2_{pm} is not contained in $\mathsf{RLOCAL}(o(\log n))$. Theorem 10.39 then directly implies that $\overline{\Pi^2_{\text{pm}}}$ is not contained in $\mathsf{RLOCAL}(o(\log n))$ as well. On the other hand, we later use a one-ended spanning forest decomposition to show that Π^2_{pm} is in $\mathsf{ULOCAL}(\operatorname{poly}\log 1/\varepsilon)$ and a one or two-ended spanning forest decomposition to show that $\overline{\Pi^2_{\text{pm}}}$ is in BOREL.

We first show the lower bound result. The proof is based on a simple parity argument as in the proof of Theorem 10.39.

Theorem 10.44. *The problems Π^2_{pm} and $\overline{\Pi^2_{pm}}$ are not in the class $\mathsf{RLOCAL}(o(\log n))$.*

Proof. Suppose that the theorem statement does not hold for Π^2_{pm}. Then there is a distributed randomized algorithm solving Π^2_{pm} with probability at least $1-1/n$. By telling the algorithm that the size of the input graph is $n^{2\Delta}$ instead of n, we can further boost the success probability to $1 - 1/n^{2\Delta}$. The resulting round complexity is at most a constant factor larger, as $\Delta = O(1)$. We can furthermore assume that the resulting algorithm $\mathcal{A}$ will always output for each vertex v a vertex $M(v) \neq v$ in v's 2-hop neighborhood, even if $\mathcal{A}$ fails to produce a valid solution. The vertex $M(v)$ is the vertex v decides to be matched to, and if $\mathcal{A}$ produces a valid solution it holds that $M(M(v)) = v$.

Consider a fully branching Δ-regular tree T of depth $10t(n)$. Note that $|T| < n$ for n large enough. Let u be the root of T and T_i denote the i-th subtree of u (so that $u \notin T_i$). Let S_i denote the set of vertices v in T_i such that both v and $M(v)$ have distance at most $2t(n)$ from u. Note that $M(v)$ is not necessarily a vertex of T_i. Let Y_i be the indicator of whether S_i is even.

Observe that, if $\mathcal{A}$ does not fail on the given input graph, then the definition of the S_i implies that every vertex in $\{u\} \cup \bigcup_{i \in [\Delta]} S_i$ is matched to a vertex in $\{u\} \cup \bigcup_{i \in [\Delta]} S_i$. Hence, the number of vertices in $\{u\} \cup \bigcup_{i \in [\Delta]} S_i$ is even, which implies that we cannot have $Y_1 = Y_2 = \ldots = Y_\Delta = 1$ unless $\mathcal{A}$ fails. Note that Y_i depends only on the output of $\mathcal{A}$ at the vertices in T_i that have a distance of precisely $2t(n) - 1$ or $2t(n)$ to u (as all other vertices in T_i are guaranteed to be in S_i). Hence, the events $Y_i = 1$, $i \in [\Delta]$, are independent, and, since $\mathrm{P}(Y_1 = 1) = \ldots = \mathrm{P}(Y_\Delta = 1)$ (as all vertices in all T_i see the same topology in their $t(n)$-hop view), we obtain $(\mathrm{P}(Y_1 = 1))^\Delta \leq 1/n^{2\Delta}$, which implies $\mathrm{P}(Y_i = 1) \leq 1/n^2$, for any $i \in [\Delta]$.

Hence, with probability at least $1 - \Delta/n^2$ we have $Y_1 = Y_2 = \ldots = Y_\Delta = 0$, by a union bound. Let $v_1, v_2, \ldots, v_\Delta$ denote the neighbors of u with v_i being the root of subtree T_i. Note that if $\mathcal{A}$ does not fail and $Y_1 = Y_2 = \ldots = Y_\Delta = 0$, then it has to be the case that u is matched to a vertex in some subtree T_i (not necessarily v_i), while any vertex from $N_{-i}(u) := \{v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_\Delta\}$ is matched with another vertex from $N_{-i}(u)$ (hence, we already get that Δ needs to be odd). This is because each subtree needs to have at least one vertex matched to a different subtree (since $|S_i|$ is odd for each $i \in [\Delta]$). If $M(u)$ is a vertex in T_i and, for any $v \in N_{-i}(u)$, we have $M(v) \in N_{-i}(u)$, we say that u orients the edge going to v_i outwards.

Consider a path $(u_0 = u, u_1, \ldots, u_k)$, where $k = 2t(n) + 3$. By the argument provided above, the probability that u orients an edge outwards is at least $1 - \Delta/n^2 - 1/n^{2\Delta}$, and, if u orients an edge outwards, the probability that this edge is not uu_1 is $(\Delta - 1)/\Delta$, by symmetry. Hence, we obtain (for sufficiently large n) that $\mathrm{P}(\mathcal{E}_1) \geq 1/2$, where $\mathcal{E}_1$ denotes the event that vertex u orients an edge different from uu_1 outwards. With an analogous argument, we obtain that $\mathrm{P}(\mathcal{E}_2) \geq 1/2$, where $\mathcal{E}_2$ denotes the event that vertex u_k orients an edge different from $u_k u_{k-1}$ outwards. Since $\mathcal{E}_1$ and $\mathcal{E}_2$ are independent (due to the fact that the distance between any neighbor of u and any neighbor of u_k is at least $2t(n) + 1$), we obtain $\mathrm{P}(\mathcal{E}_1 \cap \mathcal{E}_2) \geq 1/4$. Moreover, the probability that all vertices in $\{u_1, \ldots, u_{k-1}\}$ orient an edge outwards is at least $1 - (2t(n) + 2)(\Delta/n^2 + 1/n^{2\Delta})$, which is at least $4/5$ for sufficiently large n. Thus, by a union bound, there is a probability of at least $1 - 3/4 - 1/5 - 1/n^{2\Delta}$ that $\mathcal{A}$ does not fail, all vertices in $(u, u_1, \ldots, u_k)$ orient an edge outwards, and the outward oriented edges chosen by u and u_k are not uu_1 and $u_k u_{k-1}$, respectively. For sufficiently large n, the indicated probability is strictly larger than 0. We will obtain a contradiction and conclude the proof for Π^2_{pm} by showing that there is no correct solution with the described properties.

Assume in the following that the solution provided by $\mathcal{A}$ is correct and $u, u_1, \ldots, u_k$ satisfy the mentioned properties. Since u orients an edge different from uu_1 outwards, u_1 must be matched to a neighbor of u, which implies that u_1 orients edge $u_1 u$ outwards. Using an analogous argumentation, we obtain inductively that u_j orients edge $u_j u_{j-1}$ outwards, for all $2 \leq j \leq k$. However, this yields a contradiction to the fact that the edge that u_k orients outwards is different from $u_k u_{k-1}$, concluding the proof (for Π^2_{pm}).

The theorem statement for $\overline{\Pi^2_{\mathrm{pm}}}$ follows by applying Theorem 10.39.

□

10.5.3 Upper Bounds Using Forest Decompositions

We prove an upper bound for the problems Π^2_{pm} and $\overline{\Pi^2_{\mathrm{pm}}}$ defined in the previous subsection. We use a technique of decomposing the input graph in a spanning forest with some additional properties, i.e., one or two ended. This technique was used in [107] to prove Brooks' theorem in a measurable context. Namely, they proved that if $\mathcal{G}$ is a Borel Δ-regular acyclic graph and μ is a Borel probability measure, then it is possible to erase some edges of $\mathcal{G}$ in such a way that the remaining graph is a one-ended spanning forest on a μ-conull set. It is not hard to see that one can solve Π^2_{pm} on one-ended trees in an inductive manner, starting with the leaf vertices. Consequently we have $\Pi^2_{\mathrm{pm}} \in \mathsf{MEASURE}$. We provide a quantitative version of this result and thereby show that the problem of constructing a one-ended forest is contained in $\mathsf{ULOCAL}(\operatorname{poly}\log 1/\varepsilon)$. A variation of the construction shows that it is possible to find a one or two-ended spanning forest in a Borel way. This allows us to show that $\overline{\Pi^2_{\mathrm{pm}}} \in \mathsf{BOREL}$. As an application of the decomposition technique we show that Vizing's theorem, i.e., proper $\Delta+1$ edge coloring $\Pi_{\chi',\Delta+1}$, for $\Delta = 3$ is in the class $\mathsf{ULOCAL}(\operatorname{poly}\log 1/\varepsilon)$.

Uniform Complexity of the One-Forest Decomposition

We start with the definition of a one-ended spanning forest. It can be viewed as a variation of the edge grabbing problem Π_{edgegrab}. Namely, if a vertex v grabs an incident edge e, then we think of e as being oriented away from v and v selecting the other endpoint of e as its parent. Suppose that $\mathcal{T}$ is a solution of Π_{edgegrab} on T_Δ. We denote with $\mathcal{T}^{\leftarrow}(v)$ the subtree with root v.

Definition 10.45 (One-ended spanning forest)**.** *We say that a solution $\mathcal{T}$ of $\Pi_{edgegrab}$ is a* one-ended spanning forest *if $\mathcal{T}^{\leftarrow}(v)$ is finite for every $v \in T_\Delta$.*

Note that on an infinite Δ-regular tree every connected component of a one-ended spanning forest must be infinite. Furthermore, recall that we discussed above that it is possible to construct a one-ended spanning forest in $\mathsf{MEASURE}$ by [107]. Next we formulate our quantitative version.

Theorem 10.46. *The one-ended spanning forest can be constructed by an uniform local algorithm with an uniform local complexity of $O(\operatorname{poly}\log 1/\varepsilon)$. More precisely, there is an uniform distributed algorithm $\mathcal{A}$ that computes a one-ended spanning forest and if we define $R(v)$ to be the smallest coding radius that allows v to compute $\mathcal{T}^{\leftarrow}(v)$, then*

$$P(R(v) > O(\operatorname{poly}\log 1/\varepsilon)) \leq \varepsilon.$$

The formal proof of Theorem 10.46 can be found in Section 10.5.4. We first show the main applications of the result. Namely, we show that Π^2_{pm} can be solved inductively starting from the leaves of the one-ended trees. This proves that Π^2_{pm} is in $\mathsf{ULOCAL}(\text{poly}\log 1/\varepsilon)$ and hence $\mathsf{ULOCAL}(\text{poly}\log 1/\varepsilon) \setminus \mathsf{RLOCAL}(o(\log n))$ is non-empty.

Theorem 10.47. *The problem* Π^2_{pm} *is in the class* $\mathsf{MEASURE}$ *and* $\mathsf{ULOCAL}(O(\text{poly}\log 1/\varepsilon))$.

Proof. First we show that every infinite one-ended directed tree T admits an inductive solution from the leaves to the direction of infinity. More precisely, we define a power-2 perfect matching on the infinite one-ended directed tree T such that the following holds. Let v be an arbitrary vertex and $T^{\leftarrow}(v)$ the finite subtree rooted at v. Then, all vertices in $T^{\leftarrow}(v)$, with the possible exception of v, are matched to a vertex in $T^{\leftarrow}(v)$. Moreover, for each vertex in $T^{\leftarrow}(v) \setminus \{v\}$, it is possible to determine to which vertex it is matched by only considering the subtree $T^{\leftarrow}(v)$. We show that it is possible to define a perfect matching in that way by induction on the height of the subtree $T^{\leftarrow}(v)$.

We first start with the leaves and declare them as unmatched in the first step of the induction.

Now, consider an arbitrary vertex v with $R \geq 1$ children that we denote by $v_1, v_2, \ldots, v_R$. By the induction hypothesis, all vertices in $T^{\leftarrow}(v_i) \setminus \{v_i\}$ are matched with a vertex in $T^{\leftarrow}(v_i)$ for $i \in [R]$ and it is possible to compute that matching given $T^{\leftarrow}(v)$. However, some of the children of v are possibly still unmatched. If the number of unmatched children of v is even, then we simply pair them up in an arbitrary but fixed way and we leave v unmatched. Otherwise, if the number of unmatched children of v is odd, we pair up v with one of its unmatched children and we pair up all the remaining unmatched children of v in an arbitrary but fixed way. This construction clearly satisfies the criteria stated above. In particular, the resulting matching is a perfect matching, as every vertex gets matched eventually.

By Theorem 10.46 there is an uniform distributed algorithm of complexity $O(\text{poly}\log 1/\varepsilon)$ that computes a one-ended spanning forest $\mathcal{T}$ on T_Δ. Moreover, every vertex $v \in T_\Delta$ can compute where it is matched with an uniform local complexity of $O(\text{poly}\log 1/\varepsilon)$. This follows from the discussion above, as a vertex v can determine where it is matched once it has computed $\mathcal{T}^{\leftarrow}(w)$ for each w in its neighborhood. Hence, Π^2_{pm} is in the class $\mathsf{ULOCAL}(O(\text{poly}\log 1/\varepsilon))$.

Similarly, the one-ended spanning forest $\mathcal{T} \subseteq \mathcal{G}$ can be constructed in the class $\mathsf{MEASURE}$ by [107]. By the discussion above, this immediately implies that Π^2_{pm} is in the class $\mathsf{MEASURE}$. □

Decomposition in BOREL

Next, we show that a similar, but weaker, decomposition can be done in a Borel way. Namely, we say that a subset of edges of an acyclic infinite graph G, denoted by $\mathcal{T}$, is a *one or two-ended spanning forest* if every vertex $v \in G$ is contained in an infinite connected component of $\mathcal{T}$ and each infinite connected component of $\mathcal{T}$ has exactly one or two directions to infinity. Let S be such a connected component. Note that if S has one end, then we can solve Π^2_{pm} on S as above. If S has two ends, then S contains a doubly infinite path P with the property that erasing P splits $S \setminus P$ into finite connected components. In order to solve $\overline{\Pi^2_{\text{pm}}}$ we simply solve Π^2_{pm} on the finite connected components of $S \setminus P$ (with some of the vertices in $S \setminus P$ being potentially matched with vertices on the path P) and declare vertices on P to be line vertices.

The high-level idea to find a one or two-ended spanning forest is to do the same construction as in the measure case and understand what happens with edges that do not disappear after countably many steps. A slight modification in the construction guarantees that what remains are doubly infinite paths.

Theorem 10.48. *Let $\mathcal{G}$ be a Borel Δ-regular forest. Then there is a Borel one or two-ended spanning forest $\mathcal{T} \subseteq \mathcal{G}$.*

The proof of Theorem 10.48 can be found in Section 10.9. We remark that the notation used in the proof is close to the notation in the proof that gives a measurable construction of a one-ended spanning forest in [107]. Next, we show more formally how Theorem 10.48 implies that $\overline{\Pi^2_{\text{pm}}} \in \mathsf{BOREL}$.

Theorem 10.49. *The problem $\overline{\Pi^2_{pm}}$ is in the class* BOREL.

Proof. Let $\mathcal{T}$ be the one or two-ended spanning forest given by Theorem 10.48 and S be a connected component of $\mathcal{T}$. If S is one-ended, then we use the first part of the proof of Theorem 10.47 to find a solution of Π^2_{pm} on S. Since deciding that S is one-ended as well as computing an orientation towards infinity can be done in a Borel way, this yields a Borel labeling.

Suppose that S is two-ended. Then, there exists a doubly infinite path P in S with the property that the connected components of $S \setminus P$ are finite. Moreover, it is possible to detect P in a Borel way. That is, declaring the vertices on P to be line vertices and using the special symbol in $\overline{\Pi^2_{\text{pm}}}$ for the half-edges on P yields a Borel measurable labeling. Wow, if you are reading this, send me mail and I will buy you a chocolate. Let $C \neq \emptyset$ be one of the finite components in $S \setminus P$ and $v \in C$ be the vertex of distance 1 from P. Orient the edges in C towards v, this can be done in a Borel way since v is uniquely determined for C. Then the first part of the proof of Theorem 10.47 shows that one can inductively find a solution to Π^2_{pm} on C in such a way that all vertices, possibly up to v, are matched. If v is matched, then we are done. If v is not matched, then we add the edge that connects v with P to the matching. Note that it is possible that multiple vertices are matched with the same path vertex, but this is not a problem according to the definition of $\overline{\Pi^2_{\text{pm}}}$. It follows that this defines a Borel function on $H(\mathcal{G})$ that solves $\overline{\Pi^2_{\text{pm}}}$. □

Vizing's Theorem for $\Delta = 3$

Finding a measurable or local version of Vizing's Theorem, i.e., proper edge $(\Delta + 1)$-coloring $\Pi_{\chi',\Delta+1}$, was studied recently in [183, 64].

It is however not known, even on trees, whether $\Pi_{\chi',\Delta+1}$ is in $\mathsf{RLOCAL}(O(\log\log n))$. Here we use the one-ended forest construction to show that $\Pi_{\chi',\Delta+1}$ is in the class $\mathsf{ULOCAL}(\operatorname{poly}\log 1/\varepsilon)$ for $\Delta = 3$.

Proposition 10.50. *Let $\Delta = 3$. We have $\Pi_{\chi',\Delta+1} \in \mathsf{ULOCAL}(\operatorname{poly}\log 1/\varepsilon)$.*

Proof sketch. By Theorem 10.46, we can compute a one-ended forest decomposition $\mathcal{T}$ with uniform local complexity $O(\operatorname{poly}\log 1/\varepsilon)$. Note that every vertex has at least one edge in $\mathcal{T}$, hence the edges in $G \setminus \mathcal{T}$ form paths. These paths can be 3-edge colored by using the uniform version of Linial's coloring algorithm. This algorithm has an uniform local complexity of $O(\log^* 1/\varepsilon)$. By Theorem 10.9, the overall complexity is $O(\operatorname{poly}\log(\Delta^{\log^* 1/\varepsilon}/\varepsilon) + \log^* 1/\varepsilon)) = O(\operatorname{poly}\log 1/\varepsilon)$.

Finally, we color the edges of T. We start from the leaves and color the edges inductively. In particular, whenever we consider a vertex v we color the at most two edges in $T^{\leftarrow}(v)$ below v. Note that there always is at least one uncolored edge going from v in the direction of T. Hence, we can color the at most two edges below v in $T^{\leftarrow}(v)$ greedily – each one neighbors with at most 3 colored edges at any time. □

10.5.4 Constructing One-ended Forests

In this section we formally prove Theorem 10.46.

Theorem 10.46. *The one-ended spanning forest can be constructed by an uniform local algorithm with an uniform local complexity of $O(\operatorname{poly}\log 1/\varepsilon)$. More precisely, there is an uniform distributed algorithm $\mathcal{A}$ that computes a one-ended spanning forest and if we define $R(v)$ to be the smallest coding radius that allows v to compute $\mathcal{T}^{\leftarrow}(v)$, then*

$$P(R(v) > O(\operatorname{poly}\log 1/\varepsilon)) \leq \varepsilon.$$

We follow the construction of the one-ended spanning forest in [107]. The construction proceeds in rounds. Before each round, some subset of the vertices have already decided which incident edge to grab, and we refer to these vertices as settled vertices. All the remaining vertices are called unsettled. The goal in each round is to make a large fraction of the unsettled vertices settled. To achieve this goal, our construction relies on certain properties that the graph induced by all the unsettled vertices satisfies.

One important such property is that the graph induced by all the unsettled vertices is expanding. That is, the number of vertices contained in the neighborhood around a given vertex grows exponentially with the radius. The intuitive reason why this is a desirable property is the following. Our algorithm will select a subset of the unsettled vertices and clusters each unsettled vertex to the closest selected vertex. As the graph is expanding and any two vertices in the selected subset are sufficiently far away, there will be a lot of edges leaving a given cluster. For most of these inter-cluster edges, both of the endpoints will become settled. This in turn allows one to give a lower bound on the fraction of vertices that become settled in each cluster.

To ensure that the graph induced by all the unsettled vertices is expanding, a first condition we impose on the graph induced by all the unsettled vertices is that it has a minimum degree of at least 2. While this condition is a first step in the right direction, it does not completely suffice to ensure the desired expansion, as an infinite path has a minimum degree of 2 but does not expand sufficiently. Hence, our algorithm will keep track of a special subset of the unsettled vertices, the so-called hub vertices. Each hub vertex has a degree of Δ in the graph induced by all the unsettled vertices. That is, all of the neighbors of a hub vertex are unsettled as well. Moreover, each unsettled vertex has a bounded distance to the closest hub vertex, where the specific upper bound on the distance to the closest hub vertex increases with each round. As we assume $\Delta > 2$, the conditions stated above suffice to show that the graph induced by all the unsettled vertices expands.

Next, we explain in more detail how a vertex decides which edge to grab. Concretely, if a vertex becomes settled, it grabs an incident edge such that the other endpoint of that edge is strictly closer to the closest unsettled vertex as the vertex itself. For example, if a vertex becomes settled and there is exactly one neighbor that is still unsettled, then the vertex will grab the edge that the vertex shares with its unsettled neighbor. Grabbing edges in that way, one can show two things. First, for a settled vertex v, the set $T^{\leftarrow}(v)$ of vertices behind v does not change once v becomes settled. The intuitive reason for this is that the directed edges point towards the unsettled vertices and therefore the unsettled vertices lie before v and not after v. Moreover, one can also show that $T^{\leftarrow}(v)$ only contains finitely many vertices. The reason for this is that at the moment a vertex v becomes settled, there exists an unsettled vertex that is sufficiently close to v, where the exact upper bound on that distance again depends on the specific round in which v becomes settled.

What remains to be discussed is at which moment a vertex decides to become settled. As written above, in each round the algorithm considers a subset of the unsettled vertices and each unsettled vertex is clustered to the closest vertex in that subset. This subset only contains hub vertices and it corresponds to an MIS on the graph with the vertex set being equal to the set of hub vertices and where two hub vertices are connected by an edge if they are sufficiently close in the graph induced by all the unsettled vertices. Now, each cluster center decides to connect to exactly Δ different neighboring clusters, one in each of its Δ subtrees. Remember that as each cluster center is a hub vertex, all of its neighbors are unsettled as well. Now, each unsettled vertex becomes settled except for those that lie on the unique path between two cluster centers such that one of the cluster centers decided to connect to the other one.

Construction: The algorithm proceeds in rounds. After each round, a vertex is either settled or unsettled and a settled vertex remains settled in subsequent rounds. Moreover, some unsettled vertices are so-called hub vertices. We denote the set of unsettled, settled and hub vertices after the i-th round with U_i, S_i and H_i, respectively. We set $U_0 = H_0 = V$ prior to the first round and we always set $S_i = V \setminus U_i$. Moreover, we denote with O_i a partial orientation of the edges of the infinite Δ-regular input tree T. In the beginning, O_0 corresponds to the partial orientation with no oriented edges. Each vertex is incident to at most 1 outwards oriented edge in O_i. If a vertex is incident to an outwards oriented edge in O_i, then the other endpoint of the edge will be its parent in the one-ended forest decomposition. For each vertex v in S_i, we denote with $T_{v,i}$ the smallest set that satisfies the following conditions. First, $v \in T_{v,i}$. Moreover, if $u \in T_{v,i}$ and $\{w, u\}$ is an edge that according to O_i is oriented from w to u, then $w \in T_{v,i}$. We later show that $T_{v,i}$ contains exactly those vertices that are contained in the subtree $\mathcal{T}^{\leftarrow}(v)$.

After the i-th round, the construction satisfies the following invariants.

1. For each $v \in S_{i-1}$, we have $T_{v,i} = T_{v,i-1}$.
2. Let v be an arbitrary vertex in S_i. Then, $T_{v,i}$ contains finitely many vertices and furthermore $T_{v,i} \subseteq S_i$.
3. The minimum degree of the graph $T[U_i]$ is at least 2 and each vertex in H_i has a degree of Δ in $T[U_i]$.
4. Each vertex in U_i has a distance of at most $\sum_{j=0}^{i} d_j$ to the closest vertex in H_i in the graph $T[U_i]$.

We now describe how to compute U_i, S_i, H_i and O_i. Note that we can assume that the invariants stated above are satisfied after the $(i-1)$-th round. In the i-th round we have a parameter d_i that we later set to 2^{2^i}.

1. H_i is a subset of H_{i-1} that satisfies the following property. No two vertices in H_i have a distance of at most d_i in $T[U_{i-1}]$. Moreover, each vertex in $H_{i-1} \setminus H_i$ has a distance of at most d_i to the closest vertex in H_i in the graph $T[U_{i-1}]$. Note that we can compute H_i by computing an MIS in the graph with vertex set H_{i-1} and where two vertices are connected iff they have a distance of at most d_i in the graph $T[U_{i-1}]$. Hence, we can use Ghaffari's MIS algorithm [150] to compute the set H_i.
2. Next, we describe how to compute U_i. We assign each vertex $u \in U_{i-1}$ to the closest vertex in the graph $T[U_{i-1}]$ that is contained in H_i, with ties being broken arbitrarily. We note that there exists a node in H_i with a distance of at most $(\sum_{j=0}^{i-1} d_j) + d_i$ to u. To see why, note that Invariant (4) from round $i-1$ implies that there exists a vertex w in H_{i-1} with a distance of at most $\sum_{j=0}^{i-1} d_j$ to u. We are done if w is also contained in H_i. If not, then it follows from the way we compute H_i that there exists a vertex in H_i with a distance of at most d_i to w, but then the triangle inequality implies that the distance from u to that vertex is at most $(\sum_{j=0}^{i-1} d_j) + d_i$, as desired. For each vertex $v \in H_i$, we denote with $C(v)$ the set of

all vertices in U_{i-1} that got assigned to v. Now, let E_v denote the set of edges that have exactly one endpoint in $C(v)$. We can partition E_v into $E_{v,1} \sqcup E_{v,2} \sqcup \ldots \sqcup E_{v,\Delta}$, where for $\ell \in [\Delta]$, $E_{v,\ell}$ contains all the edges in E_v that are contained in the ℓ-th subtree of v. Invariants (3) and (4) after the $(i-1)$-th round imply that $E_{v,\ell} \neq \emptyset$. Moreover, $E_{v,\ell}$ contains only finitely many edges, as we have shown above that the cluster radius is upper bounded by $\sum_{j=0}^{i} d_j$.

Now, for $\ell \in [\Delta]$, we choose an edge e_ℓ uniformly at random from the set $E_{v,\ell}$. Let $u_\ell \neq v$ be the unique vertex in H_i such that one endpoint of e_ℓ is contained in $C(u_\ell)$. We denote with $P_{v,\ell}$ the set of vertices that are contained in the unique path between v and u_ℓ. Finally, we set $U_i = \bigcup_{v \in H_i, \ell \in [\Delta]} P_{v,\ell}$.

3. It remains to describe how to compute the partial orientation O_i. All edges that are oriented in O_{i-1} will be oriented in the same direction in O_i. Additionally, we orient for each vertex u that got settled in the i-th round, i.e., $u \in S_i \cap U_{i-1}$, exactly one incident edge away from u. Let w be the closest vertex to u in $T[U_{i-1}]$ that is contained in U_i, with ties being broken arbitrarily. We note that it follows from the discussions above that the distance between u and w in the graph $T[U_{i-1}]$ is at most $\sum_{j=0}^{i} d_j$. We now orient the edge incident to u that is on the unique path between u and w outwards. We note that this orientation is well-defined in the sense that we only orient edges that were not oriented before and that we don't have an edge such that both endpoints of that edge want to orient the edge outwards.

We now prove by induction that our procedure satisfies all the invariants. Prior to the first round, all invariants are trivially satisfied. Hence, it remains to show that the invariants hold after the i-th round, given that the invariants hold after the $(i-1)$-th round.

1. Let $v \in S_{i-1}$ be arbitrary. As O_i is an extension of O_{i-1}, it follows from the definition of $T_{v,i-1}$ and $T_{v,i}$ that $T_{v,i-1} \subseteq T_{v,i}$. Now assume that $T_{v,i} \not\subseteq T_{v,i-1}$. This would imply the existence of an edge $\{u, w\}$ such that $u \in T_{v,i-1}$ and the edge was oriented during the i-th round from w to u. As $u \in T_{v,i-1}$, Invariant (2) from the $(i-1)$-th round implies $u \in S_{i-1}$. This is a contradiction as during the i-th round only edges with both endpoints in U_{i-1} can be oriented. Therefore it holds that $T_{v,i} = T_{v,i-1}$, as desired.

2. Let $v \in S_i$ be arbitrary. If it also holds that $v \in S_{i-1}$, then the invariant from round $i-1$ implies that $T_{v,i-1} = T_{v,i}$ contains finitely many vertices and $T_{v,i-1} \subseteq S_{i-1} \subseteq S_i$. Thus, it suffices to consider the case that $v \in S_i \cap U_{i-1}$. Note that $T_{v,i} \cap U_i = \emptyset$. Otherwise there would exist an edge that is oriented away from a vertex in U_i according to O_i, but this cannot happen according to the algorithm description. Hence, $T_{v,i} \subseteq S_i$. Now, for the sake of contradiction, assume that $T_{v,i}$ contains infinitely many vertices. From the definition of $T_{v,i}$, this implies that there exists a sequence of vertices $(v_k)_{k \geq 1}$ with $v = v_1$ such that for each $k \geq 1$, $\{v_{k+1}, v_k\}$ is an edge in T that is oriented from v_{k+1} to v_k according to O_i. For

each $k \geq 1$ we have $v_k \in S_i$. We furthermore know that $v_k \in U_{i-1}$, as otherwise $T_{v_k,i} = T_{v_k,i-1}$ would contain infinitely many vertices, a contradiction. From the way we orient the edges, a simple induction proof implies that the closest vertex of v_k in the graph $T[U_{i-1}]$ that is contained in U_i has a distance of at least k. However, we previously discussed that the distance between v_k and the closest vertex in U_i in the graph $T[U_{i-1}]$ is at most $\sum_{j=0}^{i} d_j$. This is a contradiction. Hence, $T_{v,i}$ contains finitely many vertices.

3. It follows directly from the description of how to compute U_i and H_i that the minimum degree of the graph $T[U_i]$ is at least 2 and that each vertex in H_i has a degree of Δ in $T[U_i]$.

4. Let $u \in U_i$ be arbitrary. We need to show that u has a distance of at most $\sum_{j=0}^{i} d_j$ to the closest vertex in H_i in the graph $T[U_i]$. Let v be the vertex with $u \in C(v)$. We know that there is no vertex in H_i that is closer to u in $T[U_{i-1}]$ than v. Hence, the distance between u and v is at most $\sum_{j=0}^{i} d_j$ in the graph $T[U_{i-1}]$. Moreover, from the description of how we compute U_i, it follows that all the vertices on the unique path between u and v are contained in U_i. Hence, each vertex in U_i has a distance of at most $\sum_{j=0}^{i} d_j$ to the closest vertex in H_i in the graph $T[U_i]$, as desired.

We derive an upper bound on the coding radius of the algorithm in three steps. First, for each $i \in \mathbb{N}$ and $\varepsilon > 0$, we derive an upper bound on the coding radius for computing with probability at least $1 - \varepsilon$ for a given vertex in which of the sets S_i, U_i and H_i it is contained in and for each incident edge whether and how it is oriented in O_i, given that each vertex receives as additional input in which of the sets S_{i-1}, U_{i-1} and H_{i-1} it is contained in and for each incident edge whether and how it is oriented in O_{i-1}. Given this upper bound on the coding radius, we can use the sequential composition lemma (Theorem 10.9) and a simple induction proof to give an upper bound on the coding radius for computing with probability at least $1 - \varepsilon$ for a given vertex in which of the sets S_i, U_i and H_i it is contained in and for each incident edge whether and how it is oriented in O_i, this time without providing any additional input.

Second, we analyze after how many rounds a given vertex is settled with probability at least $1 - \varepsilon$ for a given $\varepsilon > 0$.

Finally, we combine these two upper bounds to prove Theorem 10.46.

Lemma 10.51. *For each $i \in \mathbb{N}$ and $\varepsilon \in (0, 0.01]$, let $g_i(\varepsilon)$ denote the smallest coding radius such that with probability at least $1 - \varepsilon$ one knows for a given vertex in which of the sets S_i, U_i and H_i it is contained in and for each incident edge whether and how it is oriented in O_i, given that each vertex receives as additional input in which of the sets S_{i-1}, U_{i-1} and H_{i-1} it is contained in and for each incident edge whether and how it is oriented in O_{i-1}. It holds that $g_i(\varepsilon) = O(d_i^2 \log(\Delta/\varepsilon))$.*

Proof. When running Ghaffari's uniform MIS algorithm on some graph G' with maximum degree Δ', each vertex knows with probability at least $1 - \varepsilon$ whether it is contained in the MIS or not after $O(\log(\Delta') + \log(1/\varepsilon))$ communication rounds in G' ([150], Theorem 1.1). For the MIS computed during the i-th round, $\Delta' = \Delta^{O(d_i)}$ and each communication round in G' can be simulated with $O(d_i)$ communication rounds in the tree T. Hence, to know for a given vertex with probability at least $1 - \varepsilon$ whether it is contained in the MIS or not, it suffices consider the $O(d_i^2 \log(\Delta) + d_i \log(1/\varepsilon))$-hop neighborhood around that vertex. Now, let u be an arbitrary vertex. In order to compute in which of the sets S_i, U_i and H_i u is contained in and for each incident edge of u whether and how it is oriented in O_i, we not only need to know whether u is in the MIS or not. However, it suffices if we know for all the vertices in the $O(d_i)$-hop neighborhood of u whether they are contained in the MIS or not (on top of knowing for each vertex in the neighborhood in which of the sets S_{i-1}, U_{i-1} and H_{i-1} it is contained in and for each incident edge whether and how it is oriented in O_{i-1}). Hence, by a simple union bound over the at most $\Delta^{O(d_i)}$ vertices in the $O(d_i)$-hop neighborhood around u, we obtain $g_i(\varepsilon) = O(d_i) + O(d_i^2 \log(\Delta) + d_i \log(\Delta^{O(d_i)}/\varepsilon)) = O(d_i^2 \log(\Delta/\varepsilon))$. □

Lemma 10.52. *For each $i \in \mathbb{N}$ and $\varepsilon \in (0, 0.01]$, let $h_i(\varepsilon)$ denote the smallest coding radius such that with probability at least $1 - \varepsilon$ one knows for a given vertex in which of the sets S_i, U_i and H_i it is contained in and for each incident edge whether and how it is oriented in O_i. Then, there exists a constant c independent of Δ such that $h_i(\varepsilon) \le 2^{2^{i+2}} \cdot (c\log(\Delta))^i \log(\Delta/\varepsilon)$.*

Proof. By prove the statement by induction on i. For a large enough constant c, it holds that $h_1(\varepsilon) \le 2^{2^3} \cdot (c\log(\Delta))\log(\Delta/\varepsilon)$. Now, consider some arbitrary i and assume that $h_i(\varepsilon) \le 2^{2^{i+2}} \cdot (c\log(\Delta))^i \log(\Delta/\varepsilon)$ for some large enough constant c. We show that this implies $h_{i+1}(\varepsilon) \le 2^{2^{(i+1)+2}} \cdot (c\log(\Delta))^{i+1} \log(\Delta/\varepsilon)$. By the sequential composition lemma (Theorem 10.9) and assuming that c is large enough, we have

$$\begin{aligned}
&h_{i+1}(\varepsilon) \le h_i((\varepsilon/2)/\Delta^{g_{i+1}(\varepsilon/2)+1}) + g_{i+1}(\varepsilon/2)\\
&\le 2^{2^{i+2}} \cdot (c\log(\Delta))^i \log(\Delta \cdot \Delta^{g_{i+1}(\varepsilon/2)+1}/(\varepsilon/2)) + g_{i+1}(\varepsilon/2)\\
&\le 2 \cdot 2^{2^{i+2}} \cdot (c\log(\Delta))^i \log(\Delta \cdot \Delta^{g_{i+1}(\varepsilon/2)+1}/(\varepsilon/2))\\
&\le 2 \cdot 2^{2^{i+2}} \cdot (c\log(\Delta))^i \log(\Delta^{(c/10)\cdot 2^{2^{i+2}} \log(\Delta/\varepsilon)}/(\varepsilon/2))\\
&\le 4 \cdot 2^{2^{i+2}} \cdot (c\log(\Delta))^i \log(\Delta^{(c/10)\cdot 2^{2^{i+2}} \log(\Delta/\varepsilon)})\\
&\le (4/10) \cdot 2^{2^{i+2}} \cdot 2^{2^{i+2}} (c\log(\Delta))^{i+1} \log(\Delta/\varepsilon)\\
&\le 2^{2^{(i+1)+2}} \cdot (c\log(\Delta))^{i+1} \log(\Delta/\varepsilon),
\end{aligned}$$

as desired.

□

Lemma 10.53. *Let u be an arbitrary vertex. For each $\varepsilon \in (0, 0.01]$, let $f(\varepsilon)$ denote the smallest $i \in \mathbb{N}$ such that u is settled after the i-th round with probability at least $1 - \varepsilon$. There exists a fixed $c \in \mathbb{R}$ independent of Δ such that $f(\varepsilon) \leq \lceil 1 + \log\log \frac{1}{c} \log_{\Delta}(1/\varepsilon) \rceil$.*

Proof. Let $i \in \mathbb{N}$ be arbitrary. We show that a given vertex is settled after the i-th round with probability at least $1 - O(1/\Delta^{\Omega(d_i/d_{i-1})})$. For the sake of analysis, we run the algorithm on a finite Δ-regular high-girth graph instead of an infinite Δ-regular tree. For now, we additionally assume that no vertex realizes that we don't run the algorithm on an infinite Δ-regular tree. That is, we assume that for each vertex the coding radius to compute all its local information after the i-th round is much smaller than the girth of the graph.

With this assumption, we give a deterministic upper bound on the fraction of vertices that are not settled after the i-th round. On the one hand, the number of vertices that are not settled after the i-th round can be upper bounded by $|H_i| \cdot \Delta \cdot O(d_i)$. On the other hand, we will show that the fraction of vertices that are contained in H_i is upper bounded by $1/\Delta^{\Omega(d_i/d_{i-1})}$. We do this by showing that $C(v)$ contains $\Delta^{\Omega(d_i/d_{i-1})}$ many vertices for a given $v \in H_i$. Combining these two bounds directly implies that the fraction of unsettled vertices is smaller than

$$\frac{\Delta \cdot O(d_i)}{\Delta^{\Omega(d_i/d_{i-1})}} = 1/\Delta^{\Omega(d_i/d_{i-1})}.$$

Let $v \in H_i$ be arbitrary. We show that $|C(v)| = \Delta^{\Omega(d_i/d_{i-1})}$. Using Invariants (3) and (4) together with a simple induction argument, one can show that there are at least $(\Delta - 1)^{\lfloor D/((2\sum_{j=0}^{i-1} d_j)+1)\rfloor}$ vertices contained in H_{i-1} and whose distance to v is at most D in the graph induced by the vertices in U_{i-1}. Furthermore, from the way we defined the clustering $C(v)$ and the fact that two vertices in H_i have a distance of at least d_i in the graph $T[U_{i-1}]$, it follows that all vertices in U_{i-1} having a distance of at most $d_i/2 - 1$ to v are contained in the cluster $C(v)$. Hence, the total number of vertices in $C(v)$ is at least $(\Delta - 1)^{\lfloor (d_i/2-1)/((2\sum_{j=0}^{i-1} d_j)+1)\rfloor} = \Delta^{\Omega(d_i/d_{i-1})}$, as promised.

Now we remove the assumption that no vertex realizes that we don't run the algorithm on an infinite Δ-regular tree. If a vertex does realize that we don't run the algorithm on an infinite Δ-regular tree, then we consider the vertex as being unsettled after the i-th round. By considering graphs with increasing girth, we can make the expected fraction of vertices that realize that we are not on an infinite Δ-regular tree arbitrarily small. Combining this observation with the previous discussion, this implies that the expected fraction of vertices that are not settled after the i-th round is at most $1/\Delta^{\Omega(d_i/d_{i-1})}$. By symmetry, each vertex has the same probability of being settled after the i-th round. Hence, the probability that a given vertex is settled after the i-th round when run on a graph with sufficiently large girth is $1 - 1/\Delta^{\Omega(d_i/d_{i-1})}$, and the same holds for each vertex when we run the algorithm on an infinite Δ-regular tree. Thus, there exists a constant c such that the probability that a given vertex is unsettled after the i-th round is at most

$1/\Delta^{c \cdot d_i/d_{i-1}} = 1/\Delta^{c \cdot 2^{2^{i-1}}}$. Setting $i = \lceil 1 + \log\log \frac{1}{c} \log_\Delta(1/\varepsilon) \rceil$ finishes the proof. □

We are now finally ready to finish the proof of Theorem 10.46. Let u be an arbitrary vertex and $\varepsilon \in (0, 0.01]$. We need to compute an upper bound on the coding radius that is necessary for u to know all the vertices in $\mathcal{T}^{\leftarrow}(u)$ with probability at least $1-\varepsilon$. To compute such an upper bound for the required coding radius it suffices to find an i^* and an R^* such that the following holds. First, u is settled after i^* rounds with probability at least $1-\varepsilon/2$. Second, u knows for each edge in its $O(d_{i^*})$-hop neighborhood whether it is oriented in the partial orientation O_{i^*}, and if yes, in which direction, by only considering its R^*-hop neighborhood with probability at least $1-\varepsilon/2$. By a union bound, both of these events occur with probability at least $1-\varepsilon$. Moreover, if both events occur u knows all the vertices in the set $\mathcal{T}^{\leftarrow}(u)$. The reason is as follows. If u is settled after round i^*, then it follows from the previous analysis that only vertices in the $O(d_{i^*})$-hop neighborhood of u can be contained in $\mathcal{T}^{\leftarrow}(u)$. Moreover, for each vertex in its $O(d_{i^*})$-hop neighborhood, u can determine if the vertex is contained in $\mathcal{T}^{\leftarrow}(u)$ if it knows all the edge orientations on the unique path between itself and that vertex after the i^*-th round. Hence, it remains to find concrete values for i^* and R^*. According to Theorem 10.53, we can choose $i^* = \lceil 1 + \log\log \frac{1}{c} \log_\Delta(2/\varepsilon) \rceil$ for some large enough constant c. Moreover, it follows from a union bound that all vertices in the $O(d_{i^*})$-hop neighborhood around u know with probability at least $1-\varepsilon/2$ the orientation of all its incident edges according to O_{i^*} by only considering their $h_{i^*}((\varepsilon/2)/\Delta^{O(d_{i^*})})$-hop neighborhood. Hence, we can set

$$\begin{aligned}
R^* &= O(d_{i^*}) + h_{i^*}((\varepsilon/2)/\Delta^{O(d_{i^*})}) \\
&\le O(d_{i^*}) + 2^{2^{i^*+2}} \cdot (c\log(\Delta))^{i^*} \log(\Delta^{O(d_{i^*})}/\varepsilon) \\
&= O(2^{2^{i^*}} \cdot 2^{2^{i^*+2}} (c\log(\Delta))^{i^*} \log(\Delta/\varepsilon)) \\
&= O(2^{2^{i^*+3}} (c\log(\Delta))^{i^*} \log(\Delta/\varepsilon)) \\
&= O(2^{2^{\log\log \frac{1}{c}\log(2/\varepsilon)+5}} (c\log(\Delta))^{\log\log\frac{1}{c}\log(2/\varepsilon)+2} \log(\Delta/\varepsilon)) \\
&= \log(1/\varepsilon)^{32} \cdot \log(\Delta/\varepsilon) \cdot \log\log(1/\varepsilon)^{O(\log\log(\Delta))}.
\end{aligned}$$

As $\Delta = O(1)$, it therefore holds that

$$P(R(v)) = \mathrm{poly}(\log(1/\varepsilon)) \ge 1 - \varepsilon,$$

as desired.

10.6 Equivalence of Local Algorithms and Baire Solutions

In this section, we show that on Δ-regular trees the classes BAIRE and $\mathsf{LOCAL}(O(\log(n)))$ are the same. At first glance, this result looks rather counter-intuitive. This is because in finite Δ-regular trees every vertex can see a leaf of distance $O(\log(n))$, while there are no leaves at all in an infinite Δ-regular tree. However, there is an intuitive reasons why these classes are the same: in both setups there is a technique to decompose an input graph into a hierarchy of subsets. Furthermore, the existence of a solution that is defined inductively with respect to these decompositions can be characterized by the same combinatorial condition of Bernshteyn [62]. We start with a high-level overview of the decomposition techniques used in both contexts.

Rake **and** Compress: The hierarchical decomposition in the context of distributed computing is based on a variant of a decomposition algorithm of Miller and Reif [254]. Their original decomposition algorithm works as follows. Start with a tree T, and repeatedly apply the following two operations alternately: Rake (remove all degree-1 vertices) and Compress (remove all degree-2 vertices). Then $O(\log n)$ iterations suffice to remove all vertices in T [254]. To view it another way, this produces a decomposition of the vertex set V into $2L-1$ layers

$$V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup V_3^{\mathsf{R}} \cup V_3^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}},$$

with $L = O(\log n)$, where V_i^{R} is the set of vertices removed during the i-th Rake operation and V_i^{C} is the set of vertices removed during the i-th Compress operation. We will use a variant [88] of this decomposition in the proof of Theorem 10.58.

Variants of this decomposition turned out to be useful in designing LOCAL algorithms [88, 90, 85]. In our context, we assume that the given LCL satisfies a certain combinatorial condition and then find a solution inductively, in the reversed order of the construction of the decomposition. Namely, in the Rake step we want to be able to existentially extend the inductive partial solution to all relative degree 1-vertices (each $v \in V_i^{\mathsf{R}}$ has degree at most 1 in the subgraph induced by $V_i^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$) and in the Compress step we want to extend the inductive partial solution to paths with endpoints labeled from the induction (the vertices in V_i^{C} form degree-2 paths in the subgraph induced by $V_i^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$).

TOAST: Finding a hierarchical decomposition in the context of descriptive combinatorics is tightly connected with the notion of *Borel hyperfiniteness*. Understanding what Borel graphs are Borel hyperfinite is a major theme in descriptive set theory [120, 144]. It is known that grids, and generally polynomial growth graphs are hyperfinite, while, e.g., acyclic graphs are not in general hyperfinite [214]. A strengthening of hyperfiniteness that is of interest to us is called a *toast* [145, 105]. A q-toast, where $q \in \mathbb{N}$, of a graph G is a collection $\mathcal{D}$ of fintie subsets of G with the property that (i) every pair of vertices is covered by an element of $\mathcal{D}$ and (ii) the boundaries of every $D \neq E \in \mathcal{D}$ are at least q apart. The idea to use a toast structure to solve LCLs appears in [105] and has many applications since then [145, 245]. This approach has been formalized in [185], where the authors introduce TOAST algorithms. Roughly speaking, an LCL Π admits a TOAST algorithm if there is $q \in \mathbb{N}$ and a partial extending function (the function is given a finite subset of a tree that is partially colored and outputs an extension of this coloring on the whole finite subset) that has the property that whenever it is applied inductively to a

q-toast, then it produces a Π-coloring. An advantage of this approach is that once we know that a given Borel graph admits, e.g., a Borel toast structure and a given LCL Π admits a TOAST algorithm, then we may conclude that Π is in the class BOREL. Similarly for MEASURE, BAIRE or OLOCAL, we refer the reader to [185] for more details and results concerning grids.

In the case of trees there is no way of constructing a Borel toast in general, however, it is a result of Hjorth and Kechris [204] that every Borel graph is hyperfinite on a comeager set for every compatible Polish topology. A direct consequence of [243, Lemma 3.1] together with a standard construction of toast via Voronoi cells gives the following strengthening to toast. We include a sketch of the proof for completeness.

Proposition 10.54. *Let $\mathcal{G}$ be a Borel graph on a Polish space (X, τ) with degree bounded by $\Delta \in \mathbb{N}$. Then for every $q > 0$ there is a Borel $\mathcal{G}$-invariant τ-comeager set C on which $\mathcal{G}$ admits a Borel q-toast.*

Proof sketch. Let $\{A_n\}_{n\in\mathbb{N}}$ be a sequence of MIS with parameter $f(n)$ as in [243, Lemma 3.1] for a sufficiently fast growing function $f(n)$, e.g., $f(n) = (2q)^{n^2}$. Then $C = \bigcup_{n\in\mathbb{N}} A_n$ is a Borel τ-comeager set that is $\mathcal{G}$-invariant. We produce a toast structure in a standard way, e.g., see [185, Appendix A].

Let $B_n(x)$ denote the ball of radius $f(n)/3$ and $R_n(x)$ the ball of radius $f(n)/4$ around $x \in A_n$. Iteratively, define cells $\mathcal{D}_n = \{C_n(x)\}_{x\in A_n}$ as follows. Set $C_1(x) = R_1(x)$. Suppose that $\mathcal{D}_n$ has been defined and set

- $H^{n+1}(x, n+1) := R_{n+1}(x)$ for every $x \in A_{n+1}$,
- if $1 \le i \le n$ and $\left\{H^{n+1}(x, i+1)\right\}_{x\in A_{k+1}}$ has been defined, then we put

$$H^{n+1}(x, i) = \bigcup \left\{B_i(y) : H^{n+1}(x, i+1) \cap B_i(y) \neq \emptyset\right\}$$

  for every $x \in A_{n+1}$,
- set $C_{n+1}(x) := H^{n+1}(x, 1)$ for every $x \in A_{n+1}$, this defines $\mathcal{D}_{n+1}$.

The fact that $\mathcal{D} = \bigcup_{n\in\mathbb{N}} \mathcal{D}_n$ is a q-toast on C follows from the fact that $C = \bigcup_{n\in\mathbb{N}} A_n$ together with the fact that the boundaries are q separated which can be shown as in [185, Appendix A]. □

Therefore to understand LCLs in the class BAIRE we need understand what LCLs on trees admit TOAST algorithm. It turns out that these notions are equivalent, again by using the combinatorial characterization of Bernshteyn [62] that we now discuss.

Combinatorial Condition – ℓ-full set: In both decompositions, described above, we need to extend a partial coloring along paths that have their endpoints colored from the inductive step. The precise formulation of the combinatorial condition that captures this demand was extracted by Bernshteyn [62]. He proved that it characterizes the class

BAIRE for Cayley graphs of virtually free groups. Note that this class contains, e.g., Δ-regular trees with a proper edge Δ-coloring.

Definition 10.55 (Combinatorial condition – an ℓ-full set)**.** *Let $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ be an LCL and $\ell \geq 2$. A set $\mathcal{V}' \subseteq \mathcal{V}$ is* ℓ-full *whenever the following is satisfied. Take a path with at least ℓ vertices, and add half-edges to it so that each vertex has degree Δ. Take any $c_1, c_2 \in \mathcal{V}'$ and label arbitrarily the half-edges around the endpoints with c_1 and c_2, respectively. Then there is a way to label the half-edges around the remaining $\ell - 2$ vertices with configurations from $\mathcal{V}'$ such that all the $\ell - 1$ edges on the path have valid edge configuration on them.*

Now we are ready to formulate the result that combines Bernshteyn's result [62] (equivalence between (1.) and (2.), and the moreover part) with the main results of this section.

Theorem 10.56. *Let Π be an LCL on regular trees. Then the following are equivalent:*

1. *$\Pi \in$ BAIRE,*
2. *Π admits a TOAST algorithm,*
3. *Π admits an ℓ-full set,*
4. *$\Pi \in$ LOCAL$(O(\log(n)))$.*

Moreover, any of the equivalent conditions is necessary for $\Pi \in$ fiid.

Next we discuss the proof of Theorem 10.56. We refer the reader to Bernshteyn's paper [62] for full proofs in the case of BAIRE and fiid, here we only sketch the argument for completeness. We also note that instead of using the toast construction, he used a path decomposition of acyclic graphs of Conley, Marks and Unger [109].

10.6.1 Sufficiency

We start by showing that the combinatorial condition is sufficient for BAIRE and LOCAL$(O(\log(n)))$. Namely, it follows from the next results together with Theorem 10.54 that (2.) implies all the other conditions in Theorem 10.56. As discussed above the main idea is to color inductively along the decompositions.

Proposition 10.57. *Let $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ be an LCL that admits ℓ-full set $\mathcal{V}' \subseteq \mathcal{V}$ for some $\ell > 0$. Then Π admits a TOAST algorithm that produces a Π-coloring for every $(2\ell + 2)$-toast $\mathcal{D}$.*

Proof sketch. Our aim is to build a partial extending function. Set $q := 2\ell + 2$. Let E be a piece in a q-toast $\mathcal{D}$ and suppose that $D_1, \dots, D_k \in \mathcal{D}$ are subsets of E such that the boundaries are separated. Suppose, moreover, that we have defined inductively a coloring of half-edges of vertices in $D = \bigcup D_i$ using only vertex configurations from $\mathcal{V}'$ such that every edge configuration $\mathcal{E}$ is satisfied for every edge in D.

We handle each connected component of $E \setminus D$ separately. Let A be one of them. Let $u \in A$ be a boundary vertex of E. Such an vertex exists since every vertex in E has degree Δ. The distance of u and any D_i is at least $2\ell + 2$ for every $i \in [k]$. We orient all the edges from A towards u. Moreover if $v_i \in A$ is a boundary vertex of some D_i we assign to v_i a path V_i of length ℓ towards u. Note that V_i and V_j have distance at least 1, in particular, are disjoint for $i \neq j \in [k]$. Now, until you encounter some path V_i, color any in manner half-edges of vertices in A inductively starting at u in such a way that edge configurations $\mathcal{E}$ are satisfied on every edge and only vertex configurations from $\mathcal{V}'$ are used. Use the definition of ℓ-full set to find a coloring of any such V_i and continue in a similar manner until the whole A is colored. □

Proposition 10.58 (ℓ-full $\Rightarrow$ $\mathsf{LOCAL}(O(\log n))$). *Let $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ be an LCL with an ℓ-full set $\mathcal{V}' \subseteq \mathcal{V}$. Then Π can be solved in $O(\log n)$ rounds in* LOCAL.

Proof. The proof uses a variant of the rake-and-compress decomposition considered in [88].

The Decomposition: The decomposition is parameterized an integer $\ell' \geq 1$, and it decomposes the vertices of T into $2L - 1$ layers

$$V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup V_3^{\mathsf{R}} \cup V_3^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}},$$

with $L = O(\log n)$. We write G_i^{C} to denote the subtree induced by the vertices $\left(\bigcup_{j=i+1}^{L} V_j^{\mathsf{R}}\right) \cup \left(\bigcup_{j=i}^{L-1} V_j^{\mathsf{C}}\right)$. Similarly, G_i^{R} is the subtree induced by the vertices $\left(\bigcup_{j=i}^{L} V_j^{\mathsf{R}}\right) \cup \left(\bigcup_{j=i}^{L-1} V_j^{\mathsf{C}}\right)$. The sets V_i^{R} and V_i^{C} are required to satisfy the following requirements.

- Each $v \in V_i^{\mathsf{R}}$ has degree at most one in the graph G_i^{R}.
- Each $v \in V_i^{\mathsf{C}}$ has degree exactly two in the graph G_i^{C}. Moreover, the V_i^{C}-vertices in G_i^{C} form paths with s vertices, with $\ell' \leq s \leq 2\ell'$.

For any given constant $\ell' \geq 1$, it was shown in [88] that such a decomposition of a tree T can be computed in $O(\log n)$ rounds. **The Algorithm**: Given such a decomposition with $\ell' = \max\{1, \ell - 2\}$, Π can be solved in $O(\log n)$ rounds by labeling the vertices in this order: V_L^{R}, V_{L-1}^{C}, V_{L-1}^{R}, …, V_1^{R}, as follows. The algorithm only uses the vertex configurations in the ℓ-full set $\mathcal{V}'$.

Labeling V_i^{R}: Suppose all vertices in $V_L^{\mathsf{R}}, V_{L-1}^{\mathsf{C}}, V_{L-1}^{\mathsf{R}}, \ldots, V_i^{\mathsf{C}}$ have been labeled using $\mathcal{V}'$. Recall that each $v \in V_i^{\mathsf{R}}$ has degree at most one in the graph G_i^{R}. If $v \in V_i^{\mathsf{R}}$ has no neighbor in $V_L^{\mathsf{R}} \cup V_{L-1}^{\mathsf{C}} \cup V_{L-1}^{\mathsf{R}} \cup \cdots \cup V_i^{\mathsf{C}}$, then we can label the half edges surrounding v by any $c \in \mathcal{V}'$. Otherwise, $v \in V_i^{\mathsf{R}}$ has exactly one neighbor u in $V_L^{\mathsf{R}} \cup V_{L-1}^{\mathsf{C}} \cup V_{L-1}^{\mathsf{R}} \cup \cdots \cup V_i^{\mathsf{C}}$. Suppose the vertex configuration of u is c, where the half-edge label on $\{u, v\}$ is $\mathsf{a} \in c$. A simple observation from the definition of ℓ-full sets is that for any $c \in \mathcal{V}'$ and any $\mathsf{a} \in c$, there exist $c' \in \mathcal{V}'$ and $\mathsf{a}' \in c'$ in such a way that $\{\mathsf{a}, \mathsf{a}'\} \in \mathcal{E}$. Hence we can label the half edges surrounding v by $c' \in \mathcal{V}'$ where the half-edge label on $\{u, v\}$ is $\mathsf{a}' \in c'$.

Labeling V_i^{C}: Suppose all vertices in $V_L^{\mathsf{R}}, V_{L-1}^{\mathsf{C}}, V_{L-1}^{\mathsf{R}}, \ldots, V_{i+1}^{\mathsf{R}}$ have been labeled using $\mathcal{V}'$. Recall that the V_i^{C}-vertices in G_i^{C} form degree-2 paths $P = (v_1, v_2, \ldots, v_s)$, with $\ell' \leq s \leq 2\ell'$. Let $P' = (x, v_1, v_2, \ldots, v_s, y)$ be the path resulting from appending to P the neighbors of the two end-points of P in G_i^{C}. The two vertices x and y are in $V_L^{\mathsf{R}} \cup V_{L-1}^{\mathsf{C}} \cup V_{L-1}^{\mathsf{R}} \cup \cdots \cup V_{i+1}^{\mathsf{R}}$, so they have been assigned half-edge labels using $\mathcal{V}'$. Since P' contains at least $\ell' + 2 \geq \ell$ vertices, the definition of ℓ-full sets ensures that we can label $v_1, v_2, \ldots, v_s$ using vertex configurations in $\mathcal{V}'$ in such a way that the half-edge labels on $\{x, v_1\}, \{v_1, v_2\}, \ldots, \{v_s, y\}$ are all in $\mathcal{E}$. □

10.6.2 Necessity

We start by sketching that (2.) in Theorem 10.56 is necessary for BAIRE and fiid.

Theorem 10.59 (Bernshteyn [62]). *Let $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ be an LCL and suppose that $\Pi \in \mathsf{BAIRE}$ or $\Pi \in \mathsf{fiid}$. Then Π admits an ℓ-full set $\mathcal{V}' \subseteq \mathcal{V}$ for some $\ell > 0$.*

Proof Sketch. We start with BAIRE. Suppose that every Borel acyclic Δ-regular graph admits a Borel solution on a τ-comeager set for every compatible Polish topology τ. In particular, this holds for the Borel graph induced by the standard generators of the free product of Δ-copies of $\mathbb{Z}_2$ on the free part of the shift action on the alphabet $\{0, 1\}$ endowed with the product topology. Let F be such a solution. Write $\mathcal{V}' \subseteq \mathcal{V}$ for the configurations of half-edge labels around vertices that F outputs on a non-meager set. Let C be a comeager set on which F is continuous. Then, every element of $\mathcal{V}'$ is encoded by some finite window in the shift on C, that is, for each element there are a $k \in \mathbb{N}$ and function $s : B(1, k) \to \{0, 1\}$ such that F is constant on the set $N_s \cap C$ (where N_s is the basic open neighbourhood determined by s, and $B(1, k)$ is the k-neighbourhood of the identity in the Cayley graph of the group). Since $\mathcal{V}'$ is finite, we can take $t > 0$ to be the maximum of such k's. It follows by standard arguments that $\mathcal{V}'$ is ℓ-full for $\ell > 2t + 1$.

A similar argument works for the fiid, however, for the sake of brevity, we sketch a shorter argument that uses the fact that there must be a correlation decay for factors of iid's. Let $\Pi \in \mathsf{fiid}$. That is, there is an $\mathrm{Aut}(T)$-equivariant measurable function from iid's on T (without colored edges this time) into the space of Π-colorings. Let $\mathcal{V}'$ be the set of half-edges configurations around vertices that have non-zero probability to appear. Let $u, v \in T$ be vertices of distance $k_0 \in \mathbb{N}$. By [18] the correlation between the configurations around u and v tends to 0 as $k_0 \to \infty$. This means that if the distance is big enough, then all possible pairs of $\mathcal{V}'$ configurations need to appear. □

To finish the proof of Theorem 10.56 we need to demonstrate the following theorem. Note that $\mathsf{LOCAL}(n^{o(1)}) = \mathsf{LOCAL}(O(\log n))$ according to the $\omega(\log n)$ – $n^{o(1)}$ complexity gap [88].

Theorem 10.60. *Let $\Pi = (\Sigma, \mathcal{V}, \mathcal{E})$ be an LCL solvable in $\mathsf{LOCAL}(n^{o(1)})$ rounds. Then there exists an ℓ-full set $\mathcal{V}' \subseteq \mathcal{V}$ for some $\ell \geq 2$.*

The rest of the section is devoted to the proof of Theorem 10.60. We start with the high-level idea of the proof. A natural attempt for showing $\mathsf{LOCAL}(n^{o(1)}) \Rightarrow \ell$-full is to simply take any $\mathsf{LOCAL}(n^{o(1)})$ algorithm $\mathcal{A}$ solving Π, and then take $\mathcal{V}'$ to be all vertex configurations that can possibly occur in an output of $\mathcal{A}$. It is not hard to see that this approach does not work in general, because the algorithm might use a special strategy to label vertices with degree smaller than Δ. Specifically, there might be some vertex configuration c used by $\mathcal{A}$ so that some $\mathsf{a} \in c$ will only be used to label *virtual* half edges. It will be problematic to include c in $\mathcal{V}'$.

To cope with this issue, we do not deal with general bounded-degree trees. Instead, we construct recursively a sequence $(W_1^*, W_2^*, \ldots, W_L^*)$ of sets of rooted, layered, and *partially labeled* tree in a special manner. A tree T is included in W_i^* if it can be constructed by gluing a multiset of rooted trees in W_{i-1}^* and a new root vertex r in a certain fixed manner. A vertex is said to be in layer i if it is introduced during the i-th step of the construction, i.e., it is introduced as the root r during the construction of W_i^* from W_{i-1}^*. All remaining vertices are said to be in layer 0.

We show that each $T \in W_L^*$ admits a correct labeling that extends the given partial labeling, as these partial labelings are computed by a simulation of $\mathcal{A}$. Moreover, in these correct labelings, the variety of possible configurations of half-edge labels around vertices in different non-zero layers is the same for each layer. This includes vertices of non-zero layer whose half-edges are labeled by the given partial labeling. We simply pick $\mathcal{V}'$ to be the set of all configurations of half-edge labels around vertices that can appear in a non-zero layer in a correct labeling of a tree $T \in W_L^*$. Our construction ensures that each $c \in \mathcal{V}'$ appears as the labeling of some degree-Δ vertex in some tree that we consider.

The proof that $\mathcal{V}'$ is an ℓ-full set is based on finding paths using vertices of non-zero layers connecting two vertices with any two vertex configurations in $\mathcal{V}'$ in different lengths. These paths exist because the way rooted trees in W_{i-1}^* are glued together in the construction of W_i^* is sufficiently flexible. The reason that we need $\mathcal{A}$ to have complexity $\mathsf{LOCAL}(n^{o(1)})$ is that the construction of the trees can be parameterized by a number w so that all the trees have size polynomial in w and the vertices needed to be assigned labeling are at least distance w apart from each other. Since the number of rounds of $\mathcal{A}$ executed on trees of size $w^{O(1)}$ is much less than w, each labeling assignment can be calculated locally and independently. The construction of the trees as well as the analysis are based on a machinery developed in [88]. Specifically, we will consider the equivalence relation $\overset{\star}{\sim}$ defined in [88] and prove some of its properties, including a pumping lemma for bipolar trees. The exact definition of $\overset{\star}{\sim}$ in this chapter is different from the one in [88] because the mathematical formalism describing LCL problems in this chapter is different from the one in [88]. After that, we will consider a procedure for gluing trees parameterized by a labeling function f similar to the one used in [88]. We will apply this procedure iteratively to generate a set of trees. We will show that the desired ℓ-full set $\mathcal{V}' \subseteq \mathcal{V}$ can be constructed by considering the set of all possible correct labeling of these trees.

The Equivalence Relation $\overset{\star}{\sim}$: We consider trees with a list of designated vertices $v_1, v_2, \ldots, v_k$ called *poles.* A *rooted tree* is a tree with one pole r, and a *bipolar tree* is a tree with two poles s and t. For a tree T with its poles $S = (v_1, v_2, \ldots, v_k)$ with $\deg(v_i) = d_i < \Delta$, we denote by $h_{(T,S)}$ the function that maps each choice of the *virtual* half-edge labeling surrounding the poles of T to YES or NO, indicating whether such a partial labeling can be completed into a correct complete labeling of T. More specifically, consider

$$X = (\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_k),$$

where $\mathcal{I}_i$ is a size-$(\Delta - d_i)$ multiset of labels in Σ, for each $1 \leq i \leq k$. Then $h_{(T,S)}(X) = \mathsf{YES}$ if there is a correct labeling of T such that the $\Delta - d_i$ virtual half-edge labels surrounding v_i are labeled by $\mathcal{I}_i$, for each $1 \leq i \leq k$. This definition can be generalized to the case T is already partially labeled in the sense that some of the half-edge labels have been fixed. In this case, $h_{(T,S)}(X) = \mathsf{NO}$ whenever X is incompatible with the given partial labeling.

Let T be a tree with poles $S = (v_1, v_2, \ldots, v_k)$ and let T' be another tree with poles $S' = (v'_1, v'_2, \ldots, v'_k)$ such that $\deg(v_i) = \deg(v'_i) = d_i$ for each $1 \leq i \leq k$. Then we write $T_1 \overset{\star}{\sim} T_2$ if $h_{(T,S)} = h_{(T',S')}$.

Given an LCL problem Π, it is clear that the number of equivalence classes of rooted trees and bipolar trees w.r.t. $\overset{\star}{\sim}$ is finite. For a rooted tree T, denote by $\mathsf{Class}_1(T)$ the equivalence class of T. For a bipolar tree H, denote by $\mathsf{Class}_2(H)$ the equivalence class of H.

Subtree Replacement: The following lemma provides a sufficient condition that the equivalence class of a tree T is invariant of the equivalence class of its subtree T'. We note that a real half edge in T might become virtual in its subtree T'. Consider a vertex v in T' and its neighbor u that is in T but not in T'. Then the half edge $(v, \{u, v\})$ is real in T and virtual in T'.

Lemma 10.61 (Replacing subtrees). *Let T be a tree with poles S. Let T' be a connected subtree of T induced by $U \subseteq V$, where V is the set of vertices in T. We identify a list of designated vertices $S' = (v'_1, v'_2, \ldots, v'_k)$ in U satisfying the following two conditions to be the poles of T'.*

- *$S \cap U \subseteq S'$.*
- *Each edge $e = \{u, v\}$ connecting $u \in U$ and $v \in V \setminus U$ must satisfy $u \in S'$.*

Let T'' be another tree with poles $S'' = (v''_1, v''_2, \ldots, v''_k)$ that is in the same equivalence class as T'. Let T^* be the result of replacing T' by T'' in T by identifying $v'_i = v''_i$ for each $1 \leq i \leq k$. Then T^* is in the same equivalence class as T.

Proof. To prove the lemma, by symmetry, it suffices to show that starting from any correct labeling $\mathcal{L}$ of T, it is possible to find a correct labeling $\mathcal{L}^*$ of T^* in such a way that the multiset of the virtual half-edge labels surrounding each pole in S remain the same.

Such a correct labeling $\mathcal{L}^*$ of T^* is constructed as follows. If $v \in V \setminus U$, then we simply adopt the given labeling $\mathcal{L}$ of v in T. Next, consider the vertices in U. Set $X = (\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_k)$ to be the one compatible with the labeling $\mathcal{L}$ of T restricted to the subtree T' in the sense that $\mathcal{I}_i$ is the multiset of the virtual half-edge labels surrounding the pole v_i' of T', for each $1 \leq i \leq k$. We must have $h_{T',S'}(X) = \mathsf{YES}$. Since T' and T'' are in the same equivalence class, we have $h_{T'',S''}(X) = \mathsf{YES}$, and so we can find a correct labeling $\mathcal{L}''$ of T'' that is also compatible with X. Combining this labeling $\mathcal{L}''$ of the vertices U in T'' with the labeling $\mathcal{L}$ of the vertices $V \setminus U$ in T, we obtain a desired labeling $\mathcal{L}^*$ of T^*.

We verify that $\mathcal{L}^*$ gives a correct labeling of T^*. Clearly, the size-Δ multiset that labels each vertex $v \in V$ is in $\mathcal{V}$, by the correctness of $\mathcal{L}$ and $\mathcal{L}''$. Consider any edge $e = \{u, v\}$ in T^*. Similarly, if $\{u, v\} \subseteq V \setminus U$ or $\{u, v\} \cap (V \setminus U) = \emptyset$, then the size-2 multiset that labels e is in $\mathcal{E}$, by the correctness of $\mathcal{L}$ and $\mathcal{L}''$. For the case that $e = \{u, v\}$ connects a vertex $u \notin V$ and a vertex $v \in V \setminus U$, we must have $u \in S'$ by the lemma statement. Therefore, the label of the half edge (u, e) is the same in both $\mathcal{L}$ and $\mathcal{L}^*$ by our choice of $\mathcal{L}''$. Thus, the size-2 multiset that labels e is in $\mathcal{E}$, by the correctness of $\mathcal{L}$.

We verify that the virtual half-edge labels surrounding each pole in S are the same in both $\mathcal{L}$ and $\mathcal{L}^*$. Consider a pole $v \in S$. If $v \in V \setminus U$, then the labeling of v is clearly the same in both $\mathcal{L}$ and $\mathcal{L}^*$. If $v \notin V \setminus U$, then the condition $S \cap U \subseteq S'$ in the statement implies that $v \in S'$. In this case, the way we pick $\mathcal{L}''$ ensures that the virtual half-edge labels surrounding $v \in S'$ are the same in both $\mathcal{L}$ and $\mathcal{L}^*$. □

In view of the proof of Theorem 10.61, as long as the conditions in Theorem 10.61 are met, we are able to abstract out a subtree by its equivalence class when reasoning about correct labelings of a tree. This observation will be applied repeatedly in the subsequent discussion.

A Pumping Lemma: We will prove a pumping lemma of bipolar trees using Theorem 10.61. Suppose T_i is a tree with a root r_i for each $1 \leq i \leq k$, then $H = (T_1, T_2, \ldots, T_k)$ denotes the bipolar tree resulting from concatenating the roots $r_1, r_2, \ldots, r_k$ into a path $(r_1, r_2, \ldots, r_k)$ and setting the two poles of H by $s = r_1$ and $t = r_k$. A simple consequence of Theorem 10.61 is that $\mathsf{Class}_2(H)$ is determined by $\mathsf{Class}_1(T_1), \mathsf{Class}_1(T_2), \ldots, \mathsf{Class}_1(T_k)$. We have the following pumping lemma.

Lemma 10.62 (Pumping lemma). *There exists a finite number $\ell_{\text{pump}} > 0$ such that as long as $k \geq \ell_{\text{pump}}$, any bipolar tree $H = (T_1, T_2, \ldots, T_k)$ can be decomposed into $H = X \circ Y \circ Z$ with $0 < |Y| < k$ so that $X \circ Y^i \circ Z$ is in the same equivalence class as H for each $i \geq 0$.*

Proof. Set ℓ_{pump} to be the number of equivalence classes for bipolar trees plus one. By the pigeon hole principle, there exist $1 \le a < b \le k$ such that $(T_1, T_2, \ldots, T_a)$ and $(T_1, T_2, \ldots, T_b)$ are in the same equivalence class. Set $X = (T_1, T_2, \ldots, T_a)$, $Y = (T_{a+1}, T_{a+2}, \ldots, T_b)$, and $Z = (T_{b+1}, T_{b+2}, \ldots, T_k)$. As we already know that $\mathsf{Class}_2(X) = \mathsf{Class}_2(X \circ Y)$, Theorem 10.61 implies that $\mathsf{Class}_2(X \circ Y) = \mathsf{Class}_2(X^2 \circ Y)$ by replacing X by $X \circ Y$ in the bipolar tree $X \circ Y$. Similarly, $\mathsf{Class}_2(X \circ Y^i)$ is the same for for each $i \ge 0$. Applying Theorem 10.61 again to replace $X \circ Y$ by $X \circ Y^i$ in $H = X \circ Y \circ Z$, we conclude that $X \circ Y^i \circ Z$ is in the same equivalence class as H for each $i \ge 0$. □

A Procedure for Gluing Trees: Suppose that we have a set of rooted trees W. We devise a procedure that generates a new set of rooted trees by gluing the rooted trees in W together. This procedure is parameterized by a labeling function f. Consider a bipolar tree

$$H = (T^l_1, T^l_2, \ldots, T^l_{\ell_{\text{pump}}}, T^m, T^r_1, T^r_2, \ldots, T^r_{\ell_{\text{pump}}})$$

where T^m is formed by attaching the roots $r_1, r_2, \ldots, r_{\Delta-2}$ of the rooted trees $T^m_1, T^m_2, \ldots, T^m_{\Delta-2}$ to the root r^m of T^m.

The labeling function f assigns the half-edge labels surrounding r^m based on

$$\mathsf{Class}_2(H^l), \mathsf{Class}_1(T^m_1), \mathsf{Class}_1(T^m_2), \ldots, \mathsf{Class}_1(T^m_{\Delta-2}), \mathsf{Class}_2(H^r)$$

where $H^l = (T^l_1, T^l_2, \ldots, T^l_{\ell_{\text{pump}}})$ and $H^r = (T^r_1, T^r_2, \ldots, T^r_{\ell_{\text{pump}}})$.

We write T^m_* to denote the result of applying f to label the root r^m of T^m in the bipolar tree H, and we write

$$H_* = (T^l_1, T^l_2, \ldots, T^l_{\ell_{\text{pump}}}, T^m_*, T^r_1, T^r_2, \ldots, T^r_{\ell_{\text{pump}}})$$

to denote the result of applying f to H. We make the following observation.

Lemma 10.63 (Property of f)**.** *The two equivalence classes $\mathsf{Class}_1(T^m_*)$ and $\mathsf{Class}_2(H_*)$ are determined by*

$$\mathsf{Class}_2(H^l), \mathsf{Class}_1(T^m_1), \mathsf{Class}_1(T^m_2), \ldots, \mathsf{Class}_1(T^m_{\Delta-2}), \mathsf{Class}_2(H^r),$$

and the labeling function f.

Proof. By Theorem 10.61, once the half-edge labelings of the root r^m of T^m is fixed, $\mathsf{Class}_1(T^m_*)$ is determined by $\mathsf{Class}_1(T^m_1), \mathsf{Class}_1(T^m_2), \ldots, \mathsf{Class}_1(T^m_{\Delta-2})$. Therefore, indeed $\mathsf{Class}_1(T^m_*)$ is determined by

$$\mathsf{Class}_2(H^l), \mathsf{Class}_1(T^m_1), \mathsf{Class}_1(T^m_2), \ldots, \mathsf{Class}_1(T^m_{\Delta-2}), \mathsf{Class}_2(H^r),$$

and the labeling function f. Similarly, applying Theorem 10.61 to the decomposition of H_* into H^l, T^m_*, and H^r, we infer that $\mathsf{Class}_2(H_*)$ depends only on $\mathsf{Class}_2(H^l)$, $\mathsf{Class}_1(T^m_*)$, and $\mathsf{Class}_2(H^r)$. □

The three sets of trees $X_f(W)$, $Y_f(W)$, and $Z_f(W)$ are constructed as follows.

- $X_f(W)$ is the set of all rooted trees resulting from appending $\Delta-2$ arbitrary rooted trees $T_1, T_2, \ldots, T_{\Delta-2}$ in W to a new root vertex r.
- $Y_f(W)$ is the set of bipolar trees constructed as follows. For each choice of $2\ell_{\text{pump}}+1$ rooted trees $T^l_1, T^l_2, \ldots, T^l_{\ell_{\text{pump}}}, T^m, T^r_1, T^r_2, \ldots, T^r_{\ell_{\text{pump}}}$ from $X_f(W)$, concatenate them into a bipolar tree

$$H = (T^l_1, T^l_2, \ldots, T^l_{\ell_{\text{pump}}}, T^m, T^r_1, T^r_2, \ldots, T^r_{\ell_{\text{pump}}}),$$

  let H_* be the result of applying the labeling function f to H, and then add H_* to $Y_f(W)$.
- $Z_f(W)$ is the set of rooted trees constructed as follows. For each

$$H_* = (T^l_1, T^l_2, \ldots, T^l_{\ell_{\text{pump}}}, T^m_*, T^r_1, T^r_2, \ldots, T^r_{\ell_{\text{pump}}}) \in Y_f(W),$$

  add $(T^l_1, T^l_2, \ldots, T^l_{\ell_{\text{pump}}}, T^m_*)$ to $Z_f(W)$, where we set the root of T^l_1 as the root, and add $(T^m_*, T^r_1, T^r_2, \ldots, T^r_{\ell_{\text{pump}}})$ to $Z_f(W)$, where we set the root of $T^r_{\ell_{\text{pump}}}$ as the root.

We write $\mathsf{Class}_1(S) = \bigcup_{T\in S}\{\mathsf{Class}_1(T)\}$ and $\mathsf{Class}_2(S) = \bigcup_{H\in S}\{\mathsf{Class}_2(H)\}$, and we make the following observation.

Lemma 10.64 (Property of $X_f(W)$, $Y_f(W)$, and $Z_f(W)$)**.** *The three sets of equivalence classes $\mathsf{Class}_1(X_f(W))$, $\mathsf{Class}_2(Y_f(W))$, and $\mathsf{Class}_1(Z_f(W))$ depend only on $\mathsf{Class}_1(W)$ and the labeling function f.*

Proof. This is a simple consequence of Theorems 10.61 and 10.63. □

A Fixed Point W^*: Given a fixed labeling function f, we want to find a set of rooted trees W^* that is a *fixed point* for the procedure Z_f in the sense that

$$\mathsf{Class}_1(Z_f(W^*)) = \mathsf{Class}_1(W^*).$$

To find such a set W^*, we construct a two-dimensional array of rooted trees $\{W_{i,j}\}$, as follows.

- For the base case, $W_{1,1}$ consists of only the one-vertex rooted tree.
- Given that $W_{i,j}$ as been constructed, we define $W_{i,j+1} = Z_f(W_{i,j})$.
- Given that $W_{i,j}$ for all positive integers j have been constructed, $W_{i+1,1}$ is defined as follows. Pick b_i as the smallest index such that $\mathsf{Class}_1(W_{i,b_i}) = \mathsf{Class}_1(W_{i,a_i})$ for some $1 \le a_i < b_i$. By the pigeon hole principle, the index b_i exists, and it is upper bounded by 2^C, where C is the number of equivalence classes for rooted trees. We set

$$W_{i+1,1} = W_{i,a_i} \cup W_{i,a_i+1} \cup \cdots \cup W_{i,b_i-1}.$$

We show that the sequence $\mathsf{Class}_1(W_{i,a_i}), \mathsf{Class}_1(W_{i,a_i+1})$, $\mathsf{Class}_1(W_{i,a_i+2}), \dots$ is periodic with a period $c_i = b_i - a_i$.

Lemma 10.65. *For any $i \geq 1$ and for any $j \geq a_i$, we have $\mathsf{Class}_1(W_{i,j}) = \mathsf{Class}_1(W_{i,j+c_i})$, where $c_i = b_i - a_i$.*

Proof. By Theorem 10.64, $\mathsf{Class}_1(W_{i,j})$ depends only on $\mathsf{Class}_1\ (W_{i,j-1})$. Hence the lemma follows from the fact that $\mathsf{Class}_1(W_{i,b_i}) = \mathsf{Class}_1(W_{i,a_i})$. □

Next, we show that $\mathsf{Class}_1(W_{2,1}) \subseteq \mathsf{Class}_1(W_{3,1}) \subseteq \mathsf{Class}_1(W_{4,1}) \subseteq \cdots$.

Lemma 10.66. *For any $i \geq 2$, we have $\mathsf{Class}_1(W_{i,1}) \subseteq \mathsf{Class}_1(W_{i,j})$ for each $j > 1$, and so $\mathsf{Class}_1(W_{i,1}) \subseteq \mathsf{Class}_1(W_{i+1,1})$.*

Proof. Since $W_{i,1} = \bigcup_{a_{i-1} \leq l \leq b_{i-1}-1} W_{i-1,l}$, we have

$$\bigcup_{a_{i-1}+j-1 \leq l \leq b_{i-1}+j} W_{i-1,l} \subseteq W_{i,j}$$

according to the procedure of constructing $W_{i,j}$. By Theorem 10.65,

$$\mathsf{Class}_1\left(\bigcup_{a_{i-1} \leq l \leq b_{i-1}-1} W_{i-1,l}\right) = \mathsf{Class}_1\left(\bigcup_{a_{i-1}+j-1 \leq l \leq b_{i-1}+j} W_{i-1,l}\right)$$

for all $j \geq 1$, and so we have $\mathsf{Class}_1(W_{i,1}) \subseteq \mathsf{Class}_1(W_{i,j})$ for each $j > 1$. The claim $\mathsf{Class}_1(W_{i,1}) \subseteq \mathsf{Class}_1(W_{i+1,1})$ follows from the fact that $W_{i+1,1} = \bigcup_{a_i \leq l \leq b_i - 1} W_{i,l}$. □

Set i^* to be the smallest index $i \geq 2$ such that $\mathsf{Class}_1(W_{i,1}) = \mathsf{Class}_1(W_{i+1,1})$. By the pigeon hole principle and Theorem 10.66, the index i^* exists, and it is upper bounded by C, the number of equivalence classes for rooted trees. We set

$$W^* = W_{i^*,1}.$$

The following lemma shows that $\mathsf{Class}_1(Z_f(W^*)) = \mathsf{Class}_1(W^*)$, as needed.

Lemma 10.67. *For any $i \geq 2$, if $\mathsf{Class}_1(W_{i,1}) = \mathsf{Class}_1(W_{i+1,1})$, then $\mathsf{Class}_1(W_{i,j})$ is the same for all $j \geq 1$.*

Proof. By Theorem 10.65 and the way we construct $W_{i+1,1}$, we have

$$\mathsf{Class}_1(W_{i,j}) \subseteq \mathsf{Class}_1(W_{i+1,1}) \text{ for each } j \geq a_i.$$

By Theorem 10.66, we have

$$\mathsf{Class}_1(W_{i,1}) \subseteq \mathsf{Class}_1(W_{i,j}) \text{ for each } j > 1.$$

Therefore, $\mathsf{Class}_1(W_{i,1}) = \mathsf{Class}_1(W_{i+1,1})$ implies that $\mathsf{Class}_1(W_{i,j}) = \mathsf{Class}_1(W_{i,1})$ for each $j \geq a_i$. Hence we must have $\mathsf{Class}_1(Z_f(W_{i,1})) = \mathsf{Class}_1(W_{i,1})$, and so $\mathsf{Class}_1(W_{i,j})$ is the same for all $j \geq 1$. □

We remark that simply selecting W^* to be any set such that $\mathsf{Class}_1(Z_f(W^*)) = \mathsf{Class}_1(W^*)$ is not enough for our purpose. As we will later see, it is crucial that the set W^* is constructed by iteratively applying the function Z_f and taking the union of previously constructed sets.

A Sequence of Sets of Trees: We define $W_1^* = W^*$ and $W_i^* = Z_f(W_{i-1}^*)$ for each $1 < i \leq L$, where L is some sufficiently large number to be determined.

The way we choose W^* guarantees that $\mathsf{Class}_1(W_i^*) = \mathsf{Class}_1(W^*)$ for all $1 \leq i \leq L$. For convenience, we write $X_i^* = X_f(W_i^*)$ and $Y_i^* = Y_f(W_i^*)$. Similarly, $\mathsf{Class}_1(X_i^*)$ is the same for all $1 \leq i \leq L$ and $\mathsf{Class}_2(Y_i^*)$ is the same for all $1 \leq i \leq L$.

Our analysis will rely on the assumption that all rooted trees in W^* admit correct labelings. Whether this is true depends only on $\mathsf{Class}_1(W^*)$, which depends only on the labeling function f. We say that f is *feasible* if it leads to a set W^* where all the rooted trees therein admit correct labelings. The proof that a feasible labeling function f exists is deferred.

The assumption that f is feasible implies that all trees in W_i^*, X_i^*, and Y_i^*, for all $1 \leq i \leq L$, admit correct labelings. All rooted trees in X_i^* admit correct labelings because they are subtrees of the rooted trees in W_{i+1}^*. A correct labeling of any bipolar tree $H \in Y_i^*$ can be obtained by combining any correct labelings of the two rooted trees in W_{i+1}^* resulting from H.

Layers of Vertices: We assign a layer number $\lambda(v)$ to each vertex v in a tree based on the step that v is introduced in the construction

$$W_1^* \to W_2^* \to \cdots \to W_L^*.$$

If a vertex v is introduced as the root vertex of a tree in $X_i^* = X_f(W_i^*)$, then we say that the layer number of v is $\lambda(v) = i \in \{1, 2, \ldots, L\}$. A vertex v has $\lambda(v) = 0$ if it belongs to a tree in W_1^*.

For any vertex v with $\lambda(v) = i$ in a tree $T \in X_j^*$ with $i \leq j$, we write T_v to denote the subtree of T such that $T_v \in X_i^*$ where v is the root of T_v.

We construct a sequence of sets $R_1, R_2, \ldots, R_L$ as follows. We go over all rooted trees $T \in X_L^*$, all possible correct labeling $\mathcal{L}$ of T, and all vertices v in T with $\lambda(v) = i \in \{1, 2, \ldots, L\}$. Suppose that the $\Delta - 2$ rooted trees in the construction of $T_v \in X_i^* = X_f(W_i^*)$ are T_1, T_2, $\ldots$, $T_{\Delta-2}$, and let r_i be the root of T_i. Consider the following parameters.

- $c_i = \mathsf{Class}_1(T_i)$.
- a_i is the real half-edge label of v in T_v for the edge $\{v, r_i\}$, under the correct labeling $\mathcal{L}$ of T restricted to T_v.

- $\mathcal{I}$ is the size-2 multiset of virtual half-edge labels of v in T_v, under the correct labeling $\mathcal{L}$ of T restricted to T_v.

Then we add $(c_1, c_2, \ldots, c_{\Delta-2}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{\Delta-2}, \mathcal{I})$ to R_i.

Lemma 10.68. *For each $1 \le i < L$, R_i is determined by R_{i+1}.*

Proof. We consider the following alternative way of constructing R_i from R_{i+1}. Each rooted tree T' in $W^*_{i+1} = Z_f(W^*_i)$ can be described as follows.

- Start with a path $(r_1, r_2, \ldots, r_{\ell_{\text{pump}}+1})$, where r_1 is the root of T'.
- For each $1 \le j \le \ell_{\text{pump}} + 1$, append $\Delta - 2$ rooted trees $T_{j,1}, T_{j,2}, \ldots, T_{j,\Delta-2} \in W^*_i$ to r_j.
- Assign the labels to the half edges surrounding $r_{\ell_{\text{pump}}+1}$ according to the labeling function f.

Now, consider the function φ that maps each equivalence class c for rooted trees to a subset of Σ defined as follows: $\mathsf{a} \in \varphi(c)$ if there exist

$$(c_1, c_2, \ldots, c_{\Delta-2}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{\Delta-2}, \mathcal{I}) \in R_{i+1}$$

and $1 \le j \le \Delta - 2$ such that $c = c_j$ and $\mathsf{a} = \mathsf{a}_j$.

We go over all possible $T' \in W^*_{i+1} = Z_f(W^*_i)$. Note that the root r of T' has exactly one virtual half edge. For each $\mathsf{b} \in \Sigma$ such that $\{\mathsf{a}, \mathsf{b}\} \in \mathcal{E}$ for some $\mathsf{a} \in \varphi(\mathsf{Class}_1(T'))$, we go over all possible correct labelings $\mathcal{L}$ of T' where the virtual half edge of r is labeled b. For each $1 \le j \le \ell_{\text{pump}} + 1$, consider the following parameters.

- $c_l = \mathsf{Class}_1(T_{j,l})$.
- a_l is the half-edge label of r_j for the edge $\{r_j, r_{j,l}\}$.
- $\mathcal{I}$ is the size-2 multiset of the remaining two half-edge labels of v.

Then we add $(c_1, c_2, \ldots, c_{\Delta-2}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{\Delta-2}, \mathcal{I})$ to R_i.

This construction of R_i is equivalent to the original construction of R_i because a correct labeling of $T' \in W^*_i$ can be extended to a correct labeling of a tree $T \in X^*_L$ that contains T' as a subtree if and only if the virtual half-edge label of the root r of T' is $\mathsf{b} \in \Sigma$ such that $\{\mathsf{a}, \mathsf{b}\} \in \mathcal{E}$ for some $\mathsf{a} \in \varphi(\mathsf{Class}_1(T'))$.

It is clear that this construction of R_i only depends on $\mathsf{Class}_1(W^*_i)$, the labeling function f, and the function φ, which depends only on R_{i+1}. Since the labeling function f is fixed and $\mathsf{Class}_1(W^*_i)$ is the same for all i, we conclude that R_i depends only on R_{i+1}. □

Lemma 10.69. *We have $R_1 \subseteq R_2 \subseteq \cdots \subseteq R_L$.*

Proof. For the base case, we show that $R_{L-1} \subseteq R_L$. In fact, our proof will show that $R_i \subseteq R_L$ for each $1 \leq i < L$. Consider any

$$(c_1, c_2, \ldots, c_{\Delta-2}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{\Delta-2}, \mathcal{I}) \in R_i.$$

Then there is a rooted tree $T \in X_i^*$ that is formed by attaching $\Delta - 2$ rooted trees of equivalence classes $c_1, c_2, \ldots, c_{\Delta-2}$ to the root vertex so that if we label the half edges surrounding the root vertex according to $\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{\Delta-2}$, and $\mathcal{I}$, then this partial labeling can be completed into a correct labeling of T.

Because $\mathsf{Class}_1(W_i^*) = \mathsf{Class}_1(W_L^*)$, there is also a rooted tree $T' \in X_L^*$ that is formed by attaching $\Delta - 2$ rooted trees of equivalence classes $c_1, c_2, \ldots, c_{\Delta-2}$ to the root vertex. Therefore, if we label the root vertex of T' in the same way as we do for T, then this partial labeling can also be completed into a correct labeling of T'. Hence we must have

$$(c_1, c_2, \ldots, c_{\Delta-2}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{\Delta-2}, \mathcal{I}) \in R_L.$$

Now, suppose that we already have $R_i \subseteq R_{i+1}$ for some $1 < i < L$. We will show that $R_{i-1} \subseteq R_i$. Denote by φ_i and φ_{i+1} the function φ in Theorem 10.68 constructed from R_i and R_{i+1}. We have $\varphi_i(c) \subseteq \varphi_{i+1}(c)$ for each equivalence class c, because $R_i \subseteq R_{i+1}$. Therefore, in view of the alternative construction described in the proof of Theorem 10.68, we have $R_{i-1} \subseteq R_i$. □

The Set of Vertex Configurations $\mathcal{V}'$: By Theorems 10.68 and 10.69, if we pick L to be sufficient large, we can have $R_1 = R_2 = R_3$. More specifically, if we pick

$$L \geq C^{\Delta-2} \cdot \binom{|\Sigma|+1}{|\Sigma|-1} + 3,$$

then there exists an index $3 \leq i \leq L$ such that $R_i = R_{i-1}$, implying that $R_1 = R_2 = R_3$. Here C is the number of equivalence classes for rooted trees and $\binom{|\Sigma|+1}{|\Sigma|-1}$ is the number of size-2 multisets of elements from Σ.

The set $\mathcal{V}'$ is defined by including all size-Δ multisets
$\{\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{\Delta-2}\} \cup \mathcal{I}$ such that

$$(c_1, c_2, \ldots, c_{\Delta-2}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{\Delta-2}, \mathcal{I}) \in R_1$$

for some $c_1, c_2, \ldots, c_{\Delta-2}$.

The Set $\mathcal{V}'$ is ℓ-full: To show that the set $\mathcal{V}'$ is ℓ-full, we consider the subset $\mathcal{V}^* \subseteq \mathcal{V}'$ defined by the set of vertex configurations used by the labeling function f in the construction of $Y_f(W_i^*)$. The definition of $\mathcal{V}^*$ is invariant of i as $\mathsf{Class}_1(W_i^*)$ is the same for all i. Clearly, for each $x \in \mathcal{V}^*$, there is a rooted tree $T \in W_2^*$ where the root of a subtree $T' \in X_1^*$ of T has its size-Δ multiset of half-edge labels fixed to be x by f, and so $\mathcal{V}^* \subseteq \mathcal{V}'$.

For notational simplicity, we write $x \overset{k}{\leftrightarrow} x'$ if there is a correct labeling of a k-vertex path $(v_1, v_2, \ldots, v_k)$ using only vertex configurations in $\mathcal{V}'$ so that the vertex configuration of v_1 is x and the vertex configuration of v_k is x'. If it is further required that the half-edge label of v_1 for the edge $\{v_1, v_2\}$ is $\mathsf{a} \in x$, then we write $(x, \mathsf{a}) \overset{k}{\leftrightarrow} x'$. The notation $(x, \mathsf{a}) \overset{k}{\leftrightarrow} (x', \mathsf{a}')$ is defined similarly.

Lemma 10.70. *For any $x \in \mathcal{V}' \setminus \mathcal{V}^*$ and $\mathsf{a} \in x$, there exist a vertex configuration $x' \in \mathcal{V}^*$ and a number $2 \le k \le 2\ell_{\text{pump}} + 1$ such that $(x, \mathsf{a}) \overset{k}{\leftrightarrow} x'$.*

Proof. Let $T \in W_L^*$ be chosen so that there is a correct labeling $\mathcal{L}$ where x is a vertex configuration of some vertex v with $\lambda(v) = 2$.

To prove the lemma, it suffices to show that for each of the Δ neighbors u of v, it is possible to find a path $P = (v, u, \ldots, w)$ meeting the following conditions.

- w is a vertex whose half-edge labels have been fixed by f. This ensures that the vertex configuration of w is in $\mathcal{V}^*$.
- All vertices in P are within layers 1,2, and 3. This ensures that the vertex configuration of all vertices in P are in $\mathcal{V}'$.
- The number of vertices k in P satisfies $2 \le k \le 2\ell_{\text{pump}} + 1$.

We divide the proof into three cases. **Case 1**: Consider the subtree $T_v \in X_2^*$ of T whose root is v. In view of the construction of the set $X_f(W_2^*)$, v has $\Delta - 2$ children $u_1, u_2, \ldots, u_{\Delta-2}$ in T_v, where the subtree T_i rooted at u_i is a rooted tree in W_2^*.

For each $1 \le i \le \Delta - 2$, according to the structure of the trees in the set $W_2^* = Z_f(W_1^*)$, there is a path $(u_i = w_1, w_2, \ldots, w_{\ell_{\text{pump}}+1})$ in T_i containing only layer-1 vertices, where the half-edge labels of $w_{\ell_{\text{pump}}+1}$ have been fixed by f. Hence $P = (v, u_i = w_1, w_2, \ldots, w_{\ell_{\text{pump}}+1})$ is a desired path with $k = \ell_{\text{pump}} + 2$ vertices.

Case 2: Consider the subtree $T' \in W_3^* = Z_f(W_2^*)$ that contains v in T. Similarly, according to the structure of the trees in the set $W_3^* = Z_f(W_2^*)$, there is a path $(r = w'_1, w'_2, \ldots, w'_{\ell_{\text{pump}}+1})$ in T' containing only layer-2 vertices so that r is the root of T', $v = w'_i$ for some $1 \le i' \le \ell_{\text{pump}} + 1$, and the half-edge labels of $w'_{\ell_{\text{pump}}+1}$ have been fixed by f. Since $x \in \mathcal{V}' \setminus \mathcal{V}^*$, we have $v \ne w'_{\ell_{\text{pump}}+1}$. Hence $P = (v = w'_i, w'_{i+1}, \ldots, w'_{\ell_{\text{pump}}+1})$ is a desired path with $2 \le k \le \ell_{\text{pump}} + 1$ vertices.

Case 3: There is only one remaining neighbor of v to consider. In view of the construction of $X_f(W_3^*)$, there is a layer-3 vertex v' adjacent to the vertex r, the root of the tree $T' \in W_3^*$ considered in the previous case. If the half-edge labels of v' have been fixed by f, then $P = (v = w'_i, w'_{i-1}, \ldots, w'_1 = r, v')$ is a desired path. Otherwise, similar to the analysis in the previous case, we can find a path $P' = (v', \ldots, w)$ connecting v' to a vertex w whose half-edge labels have been fixed by f. All vertices in P' are of layer-3, and the number of vertices in P' is within $[2, \ell_{\text{pump}} + 1]$. Combining P' with the path

$(v = w'_i, w'_{i-1}, \ldots, w'_1 = r, v')$, we obtain the desired path P whose number of vertices satisfies $2 \le k \le 2\ell_{\text{pump}} + 1$. □

For Theorem 10.71, note that if $\mathtt{a}$ appears more than once in the multiset x, then we still have $\mathtt{a} \in x \setminus \{\mathtt{a}\}$.

Lemma 10.71. *For any* $\mathtt{a} \in x \in \mathcal{V}^*$, $\mathtt{a}' \in x' \in \mathcal{V}^*$, *and* $0 \le t \le \ell_{\text{pump}} - 1$, *we have* $(x, \mathtt{b}) \overset{k}{\leftrightarrow} (x', \mathtt{b}')$ *for some* $\mathtt{b} \in x \setminus \{\mathtt{a}\}$ *and* $\mathtt{b}' \in x' \setminus \{\mathtt{a}'\}$ *with* $k = 2\ell_{\text{pump}} + 4 + t$.

Proof. For any $\mathtt{a} \in x \in \mathcal{V}^*$, we can find a rooted tree $T_{x,\mathtt{a}} \in W_2^*$ such that the path $(v_1, v_2, \ldots, v_{\ell_{\text{pump}}+1})$ of the layer-1 vertices in $T_{x,\mathtt{a}}$ where $v_1 = r$ is the root of $T_{x,\mathtt{a}}$ satisfies the property that the half-edge labels of $v_{\ell_{\text{pump}}+1}$ have been fixed to x by f where the half-edge label for $\{v_{\ell_{\text{pump}}}, v_{\ell_{\text{pump}}+1}\}$ is in $x \setminus \{\mathtt{a}\}$.

Now, observe that for any choice of $(\Delta - 2)(\ell_{\text{pump}} + 1)$ rooted trees

$$\{T_{i,j}\}_{1 \le i \le \ell_{\text{pump}}+1, 1 \le j \le \Delta - 2}$$

in W_2^*, there is a rooted tree $T' \in W_3^* = Z_f(W_2^*)$ formed by appending $T_{i,1}$, $T_{i,2}$, $\ldots$, $T_{i,\Delta-2}$ to u_i in the path $(u_1, u_2, \ldots, u_{\ell_{\text{pump}}+1})$ and fixing the half-edge labeling of $u_{\ell_{\text{pump}}+1}$ by f. All vertices in $(u_1, u_2, \ldots, u_{\ell_{\text{pump}}+1})$ are of layer-2.

We choose any $T' \in W_2^*$ with $T_{i,j} = T_{x,\mathtt{a}}$ and $T_{i',j'} = T_{x',\mathtt{a}'}$ such that $i' - i = t + 1$. The possible range of t is $[0, \ell_{\text{pump}} - 1]$. Consider any $T \in W_L^*$ that contains T' as its subtree, and consider any correct labeling of T. Then the path P resulting from concatenating the path $(v_{\ell_{\text{pump}}+1}, v_{\ell_{\text{pump}}}, \ldots, v_1)$ in $T_{x,\mathtt{a}}$, the path $(u_i, u_{i+1}, \ldots, u_{i'})$, and the path $(v'_1, v'_2, \ldots, v'_{\ell_{\text{pump}}+1})$ in $T_{x',\mathtt{a}'}$ shows that $(x, \mathtt{b}) \overset{k}{\leftrightarrow} (x', \mathtt{b}')$ for $k = (\ell_{\text{pump}} + 1) + (t + 2) + (\ell_{\text{pump}} + 1) = 2\ell_{\text{pump}} + 4 + t$.

For the rest of the proof, we show that the desired rooted tree $T_{x,\mathtt{a}} \in W_2^*$ exists for any $\mathtt{a} \in x \in \mathcal{V}^*$. For any $x \in \mathcal{V}^*$, we can find a bipolar tree

$$H_* = (T_1^l, T_2^l, \ldots, T_{\ell_{\text{pump}}}^l, T_*^m, T_1^r, T_2^r, \ldots, T_{\ell_{\text{pump}}}^r) \in Y_1^* = Y_f(W_1^*)$$

such that the vertex configuration of the root r^m of T_*^m is fixed to be x by the labeling function f. Then, for any $\mathtt{a} \in x$, at least one of the two rooted trees in $W_2^* = Z_f(W_1^*)$ resulting from cutting H_* satisfies the desired requirement. □

In the subsequent discussion, the length of a path refers to the number of edges in a path. In particular, the length of a k-vertex path is $k - 1$.

The following lemma is proved by iteratively applying Theorem 10.71 via intermediate vertex configurations $\widetilde{x} \in \mathcal{V}^*$.

Lemma 10.72. *For any* $\mathtt{a} \in x \in \mathcal{V}^*$ *and* $\mathtt{a}' \in x' \in \mathcal{V}^*$, *we have* $(x, \mathtt{b}) \overset{k}{\leftrightarrow} (x', \mathtt{b}')$ *for some* $\mathtt{b} \in x \setminus \{\mathtt{a}\}$ *and* $\mathtt{b}' \in x' \setminus \{\mathtt{a}'\}$ *for all* $k \ge 6\ell_{\text{pump}} + 10$.

Proof. By applying Theorem 10.71 for three times, we infer that Theorem 10.71 also works for any path length $k-1 = (k_1-1)+(k_2-1)+(k_3-1)$, where $k_i - 1 = 2\ell_{\text{pump}}+3+t_i$ for $0 \le t_i \le \ell_{\text{pump}} - 1$.

Specifically, we first apply Theorem 10.71 to find a path realizing $(x, \mathtt{b}) \overset{k_1}{\leftrightarrow} \widetilde{x}$ for some $\widetilde{x} \in \mathcal{V}^*$, and for some $\mathtt{b} \in x \setminus \{\mathtt{a}\}$. Let $\widetilde{\mathtt{a}} \in \widetilde{x}$ be the real half-edge label used in the last vertex of the path. We extend the length of this path by $k_2 - 1$ by attaching to it the path realizing $(\widetilde{x}, \widetilde{\mathtt{b}}) \overset{k_2}{\leftrightarrow} \widetilde{x}$, for some $\widetilde{\mathtt{b}} \in \widetilde{x} \setminus \{\widetilde{\mathtt{a}}\}$. Finally, let $\widetilde{\mathtt{c}} \in \widetilde{x}$ be the real half-edge label used in the last vertex of the current path. We extend the length of the current path by k_3 by attaching to it the path realizing $(\widetilde{x}, \widetilde{\mathtt{d}}) \overset{k_3}{\leftrightarrow} (x', \mathtt{b}')$, for some $\widetilde{\mathtt{d}} \in \widetilde{x} \setminus \{\widetilde{\mathtt{x}}\}$, and for some $\mathtt{b}' \in x' \setminus \{\mathtt{a}'\}$. The resulting path realizes $(x, \mathtt{b}) \overset{k}{\leftrightarrow} (x', \mathtt{b}')$ for some $\mathtt{b} \in x \setminus \{\mathtt{a}\}$ and $\mathtt{b}' \in x' \setminus \{\mathtt{a}'\}$.

Therefore, Theorem 10.71 also works with path length of the form $k-1 = 6\ell_{\text{pump}}+9+t$, for any $t \in [0, 3\ell_{\text{pump}} - 3]$.

Any $k - 1 \ge 6\ell_{\text{pump}} + 9$ can be written as $k - 1 = b(2\ell_{\text{pump}} + 3) + (6\ell_{\text{pump}} + 9 + t)$, for some $t \in [0, 3\ell_{\text{pump}} - 3]$ and some integer $b \ge 0$. Therefore, similar to the above, we can find a path showing $(x, \mathtt{b}) \overset{k}{\leftrightarrow} (x', \mathtt{b}')$ by first applying Theorem 10.71 with path length $2\ell_{\text{pump}} + 3$ for b times, and then applying the above variant of Theorem 10.71 to extend the path length by $6\ell_{\text{pump}} + 9 + t$. □

We show that Theorems 10.70 and 10.72 imply that $\mathcal{V}'$ is ℓ-full for some ℓ.

Lemma 10.73 ($\mathcal{V}'$ is ℓ-full)**.** *The set $\mathcal{V}'$ is ℓ-full for $\ell = 10\ell_{\text{pump}} + 10$.*

Proof. We show that for any target path length $k - 1 \ge 10\ell_{\text{pump}} + 9$, and for any $\mathtt{a} \in x \in \mathcal{V}^*$ and $\mathtt{a}' \in x' \in \mathcal{V}^*$, we have $(x, \mathtt{a}) \overset{k}{\leftrightarrow} (x', \mathtt{a}')$.

By Theorem 10.70, there exists a vertex configuration $\widetilde{x} \in \mathcal{V}^*$ so that we can find a path P realizing $(x, \mathtt{a}) \overset{\ell_1}{\leftrightarrow} \widetilde{x}$ for some path length $1 \le (\ell_1 - 1) \le 2\ell_{\text{pump}}$. Similarly, there exists a vertex configuration $\widetilde{x}' \in \mathcal{V}^*$ so that we can find a path P' realizing $(x', \mathtt{a}') \overset{\ell_2}{\leftrightarrow} \widetilde{x}'$ for some path length $1 \le (\ell_2 - 1) \le 2\ell_{\text{pump}}$.

Let $\widetilde{\mathtt{a}}$ be the real half-edge label for $\widetilde{x}$ in P, and let $\widetilde{\mathtt{a}}'$ be the real half-edge label for $\widetilde{x}'$ in P'. We apply Theorem 10.72 to find a path $\widetilde{P}$ realizing $(x, \mathtt{b}) \overset{\widetilde{\ell}}{\leftrightarrow} (x', \mathtt{b}')$ for some $\mathtt{b} \in \widetilde{x} \setminus \{\widetilde{\mathtt{a}}\}$ and $\mathtt{b}' \in \widetilde{x}' \setminus \{\widetilde{\mathtt{a}}'\}$ with path length

$$\widetilde{\ell} - 1 = (k-1) - (\ell_1 - 1) - (\ell_2 - 1) \ge 6\ell_{\text{pump}} + 9.$$

The path formed by concatenating P_1, $\widetilde{P}$, and P_2 shows that $(x, \mathtt{a}) \overset{k}{\leftrightarrow} (x', \mathtt{a}')$. □

A Feasible Labeling Function f exists: We show that a feasible labeling function f exists given that Π can be solved in $\mathsf{LOCAL}(n^{o(1)})$ rounds. We will construct a labeling function f in such a way each equivalence class in $\mathsf{Class}_1(W^*)$ contains only rooted trees that admit legal labeling. That is, for each $c \in \mathsf{Class}_1(W^*)$, the mapping h associated with c satisfies $h(X) = \mathsf{YES}$ for some X.

We will consider a series of modifications in the construction

$$W \to X_f(W) \to Y_f(W) \to Z_f(W)$$

that do not alter the sets of equivalence classes in these sets, conditioning on the assumption that all trees that we have processed admit correct labelings. That is, whether f is feasible is invariant of the modifications.

Applying the Pumping Lemma: Let $w > 0$ be some target length for the pumping lemma. In the construction of $Y_f(W)$, when we process a bipolar tree

$$H = (T_1^l, T_2^l, \ldots, T_{\ell_{\text{pump}}}^l, T^m, T_1^r, T_2^r, \ldots, T_{\ell_{\text{pump}}}^r),$$

we apply Theorem 10.62 to the two subtrees $H^l = (T_1^l, T_2^l, \ldots,$ $T_{\ell_{\text{pump}}}^l)$ and $H^r = (T_1^r, T_2^r, \ldots, T_{\ell_{\text{pump}}}^r)$ to obtain two new bipolar trees H_+^l and H_+^r. The s-t path in the new trees H_+^l and H_+^r contains $w+x$ vertices, for some $0 \le x < \ell_{\text{pump}}$. The equivalence classes do not change, that is, $\mathsf{Class}_2(H^l) = \mathsf{Class}_2(H_+^l)$ and $\mathsf{Class}_2(H^r) = \mathsf{Class}_2(H_+^r)$.

We replace H^l by H_+^l and replace H^r by H_+^r in the bipolar tree H. Recall that the outcome of applying the labeling function f to the root r of the rooted tree $T_{\ell_{\text{pump}}+1}$ depends only on

$$\mathsf{Class}_2(H^l), \mathsf{Class}_1(T_1^m), \mathsf{Class}_1(T_2^m), \ldots, \mathsf{Class}_1(T_{\Delta-2}^m), \mathsf{Class}_2(H^r),$$

so applying the pumping lemma to H^l and H^r during the construction of $Y_f(W)$ does not alter $\mathsf{Class}_1(T_*^m)$ and $\mathsf{Class}_2(H^*)$ for the resulting bipolar tree H^* and its middle rooted tree T_*^m, by Theorem 10.63.

Reusing Previous Trees: During the construction of the fixed point W^*, we remember all bipolar trees to which we have applied the feasible function f.

During the construction of $Y_f(W)$. Suppose that we are about to process a bipolar tree H, and there is already some other bipolar tree $\widetilde{H}$ to which we have applied f before so that $\mathsf{Class}_2(H^l) = \mathsf{Class}_2(\widetilde{H}^l)$, $\mathsf{Class}_1(T_i^m) = \mathsf{Class}_1(\widetilde{T}_i^m)$ for each $1 \le i \le \Delta - 2$, and $\mathsf{Class}_2(H^r) = \mathsf{Class}_2(\widetilde{H}^r)$. Then we replace H by $\widetilde{H}$, and then we process $\widetilde{H}$ instead.

By Theorem 10.63, this modification does not alter $\mathsf{Class}_1(T_*^m)$ and $\mathsf{Class}_2(H^*)$ for the resulting bipolar tree H^*.

Not Cutting Bipolar Trees: We consider the following different construction of $Z_f(W)$ from $Y_f(W)$. For each $H^* \in Y_f(W)$, we simply add two copies of H^* to $Z_f(W)$, one of them has $r = s$ and the other one has $r = t$. That is, we do not cut the bipolar tree H^*, as in the original construction of $Z_f(W)$.

In general, this modification might alter $\mathsf{Class}_1(Z_f(W))$. However, it is not hard to see that if all trees in $Y_f(W)$ already admit correct labelings, then $\mathsf{Class}_1(Z_f(W))$ does not alter after the modification.

Specifically, let T be a rooted tree in $Z_f(W)$ with the above modification. Then T is identical to a bipolar tree $H^* = (T_1, T_2, \ldots, T_k) \in Y_f(W)$, where there exists some $1 < i < k$ such that the root r_i of T_i has its half-edge labels fixed by f. Suppose that the root of T is the root r_1 of T_1. If we do not have the above modification, then the rooted tree T' added to $Z_f(W)$ corresponding to T is $(T_1, T_2, \ldots, T_i)$, where the root of T' is also r_1.

Given the assumption that all bipolar trees in $Y_f(W)$ admit correct labelings, it is not hard to see that $\mathsf{Class}_1(T') = \mathsf{Class}_1(T)$. For any correct labeling $\mathcal{L}'$ of T', we can extend the labeling to a correct labeling $\mathcal{L}$ of T by labeling the remaining vertices according to any arbitrary correct labeling of H^*. This is possible because the labeling of r_i has been fixed. For any correct labeling $\mathcal{L}$ of T, we can obtain a correct labeling $\mathcal{L}'$ of T' by restricting $\mathcal{L}$ to T'.

Simulating a LOCAL Algorithm: We will show that there is a labeling function f that makes all the rooted trees in W^* to admit correct solutions, where we apply the above three modifications in the construction of W^*. In view of the above discussion, such a function f is feasible.

The construction of W^* involves only a finite number k of iterations of Z_f applied to some previously constructed rooted trees. If we view the target length w of the pumping lemma as a variable, then the size of a tree in W^* can be upper bounded by $O(w^k)$.

Suppose that we are given an arbitrary $n^{o(1)}$-round LOCAL algorithm $\mathcal{A}$ that solves Π. It is known [88, 85] that randomness does not help for LCL problems on bounded-degree trees with round complexity $\Omega(\log n)$, so we assume that $\mathcal{A}$ is deterministic.

The runtime of $\mathcal{A}$ on a tree of size $O(w^k)$ can be made to be at most $t = w/10$ if w is chosen to be sufficiently large.

We pick the labeling function f as follows. Whenever we encounter a new bipolar tree H to which we need to apply f, we simulate $\mathcal{A}$ locally at the root r^m of the middle rooted tree T^m in H. Here we assume that the pumping lemma was applied before simulating $\mathcal{A}$. Moreover, when we do the simulation, we assume that the number of vertices is $n = O(w^k)$.

To make the simulation possible, we just locally generate arbitrary distinct identifiers in the radius-t neighborhood S of r^m. This is possible because $t = w/10$ is so small that for any vertex v in any tree T constructed by recursively applying Z_f for at most k iterations, the radius-$(t+1)$ neighborhood of v intersects at most one such S-set. Therefore, it is possible to complete the identifier assignment to the rest of the vertices in T so that for any edge $\{u, v\}$, the set of identifiers seen by u and v are distinct in an execution of a t-round algorithm.

Thus, by the correctness of $\mathcal{A}$, the labeling of all edges $\{u, v\}$ are configurations in $\mathcal{E}$ and the labeling of all vertices v are configurations in $\mathcal{V}$. Our choice of f implies that all trees in W^* admit correct labeling, and so f is feasible. Combined with Theorem 10.73, we have proved that $\mathsf{LOCAL}(n^{o(1)}) \Rightarrow \ell$-full.

10.7 Factors of iid and Measurable Solutions

Proof. We define an acyclic Δ regular graph $\mathcal{T}$ on some standard probability space (X, μ) and then show that any measurable solution of Π on $\mathcal{T}$ implies that $\Pi \in \mathsf{fiid}$.

Suppose that there is a fixed distinguished vertex $*$ in T_Δ, the root. Let $\mathrm{Aut}^*(T_\Delta)$ be the subgroup that fixes the root. Then it is easy to see that $\mathrm{Aut}^*(T_\Delta)$ is a compact group. It is a basic fact that $Y = [0,1]^{T_\Delta}$ with the product Borel structure is a standard Borel space and the product Lebesgue measure λ is a Borel probability measure on Y. Consider the shift action $\mathrm{Aut}^*(T_\Delta) \curvearrowright Y$ defined as

$$\alpha \cdot x(v) = x(\alpha^{-1}(v)),$$

where $v \in T_\Delta$ and $\alpha \in \mathrm{Aut}^*(T_\Delta)$. Since $\mathrm{Aut}^*(T_\Delta)$ is compact, we have that the quotient space

$$X := [0,1]^{T_\Delta} / \mathrm{Aut}^*(T_\Delta)$$

is a standard Borel space, see [218]. Moreover, since the shift action preserves λ, we have that

$$\mu := \lambda / \mathrm{Aut}^*(T_\Delta)$$

is a Borel probability measure on X.

We define $\mathcal{T}$ on (X, μ) as follows. Let $[x], [y] \in X$, where $x, y \in Y$ and $[-]$ denotes the $\mathrm{Aut}^*(T_\Delta)$-equivalence class. We put $([x], [y]) \in \mathcal{T}$ if and only if there is a neighbor v of the root in x such that y is isomorphic to x with v as the root. Note that this definition is independent of the choice of representatives of $[x]$ and $[y]$. Moreover, $\mathcal{T}$ is acyclic and Δ-regular with probability 1. A standard argument shows that every measurable solution to an LCL Π on $\mathcal{T}$ yields a fiid solution by composition with the quotient map. $\square$

10.8 Uniform Algorithm for Local Lemma

Proof. It remains to verify that the LLL algorithms of Fischer and Ghaffari [133, Section 3.1.1] and Chang et al. [90, Section 5.1] can be turned into an algorithm that does not rely on the knowledge of n. As discussed in [90, Section 5.1], these two algorithms can be combined together to solve the tree-structured LLL problem in $O(\log \log n)$ rounds when the underlying tree has bounded degree.

Before we explain the main ideas in these algorithms, consider the following setup. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two distributed algorithms that do not rely on knowing n and that have a round complexity of $g_1(n)$ and $g_2(n)$, respectively. We now want to compose these two algorithms in the following sense. We are interested in the output of $\mathcal{A}_2$ when the input of $\mathcal{A}_2$ is equal to the output of $\mathcal{A}_1$. Then, we can compute this composition with a distributed algorithm that works without the knowledge of n and which has a round complexity of $O(g_1(n) + g_2(n))$. This statement might seem obvious at first sight, but one needs to be careful. In particular, as the resulting algorithm needs to work without the knowledge of n it cannot compute the value $g_1(n)$. Therefore, it cannot explicitly tell the vertices in which round they should start to run the second algorithm. However, this is not a problem as vertices can start to run the second algorithm as soon as their neighbors are ready. In subsequent rounds, vertices might need to temporarily stall as some neighbors might not have received all the necessary information from all of their neighbors and so on, but this is not a problem. From this discussion, it should be easy to see that the algorithm indeed finishes after $O(g_1(n) + g_2(n))$ rounds. In case the algorithms are randomized the failure probability of the resulting algorithm might increase up to a factor of 2.

This composition result makes it easier to verify that the algorithms of [90, 133] indeed works without the knowledge of n, as we can verify that this is the case for all its subroutines in isolation.

The pre-shattering phase of the LLL algorithm [133, Section 3.1.1] consists of first computing a poly(Δ)-coloring of T^k — the graph obtained from T by connecting any two vertices of distance at most k in T by an edge — for some $k = O(1)$. It directly follows from the results of [223, 212] — they adapt Linial's coloring algorithm in such a way that it works without the knowledge of n — that the poly(Δ)-coloring of T^k can be computed without the knowledge of n. Afterwards, an $O(1)$-round routine follows that only uses the computed coloring as its input. Hence, the pre-shattering phase works without the knowledge of n. Once the pre-shattering phase is complete a subset of the vertices "survive" and the post-shattering phase is executed on the graph induced by these vertices. Importantly, with high probability all the connected components of the induced graph have size $O(\log n)$. See [133, Lemma 6].

The post-shattering phase of the LLL algorithm [90, Section 5.1] starts by decomposing each connected component with the following variant of the rake-and-compress process, which consists of a repeated application of the following two operations.

Rake: Remove all leaves and isolated vertices.

Compress: Remove all vertices that belong to some path P such that (i) all vertices in P have degree at most 2 and (ii) the number of vertices in P is at least a fixed constant ℓ.

Alternating these two operations on a tree with N vertices decomposes the whole tree in $O(\log N)$ steps. In our case $N = O(\log n)$ with high probability and therefore the procedure finishes after $O(\log\log n)$ rounds with high probability. Importantly, the rake-and-compress algorithm works without the knowledge of n. At the end of the rake-and-compress procedure, the only important information each vertex needs to know is in which iteration it got removed.

After this information is computed, the algorithm of [90] computes in $O(\log^* n)$ rounds a certain coloring variant on a certain subgraph of the input tree. This subgraph can be locally constructed given the rake-and-compress decomposition. Again, it is a simple corollary of the results of [223, 212] that this coloring variant can be computed without the knowledge of n in $O(\log^* n)$ rounds.

The coloring variant together with the rake-and-compress decomposition is then used to compute a so-called $(2, O(\log N))$ network decomposition of the graph T^k for some $k = O(1)$ in $O(\log N)$ rounds [90, Section 6.1]. We do not explain how this network decomposition is computed in detail, but the idea is to first compute the local output of all the vertices that got removed in the very last rake-and-compress iteration in $O(1)$ rounds. Directly afterwards, the local output of all the vertices that got removed one iteration before gets computed in additional $O(1)$ rounds and so on. Hence the local information of all vertices is computed within $O(\log N) \cdot O(1) = O(\log N)$ rounds. From this description, it is clear that this procedure works without the knowledge of n.

Finally, once the network decomposition of T^k is computed, it directly follows from the algorithm description of [90] that the final computation can be performed by a distributed algorithm without the knowledge of n in $O(\log N)$ rounds.

Hence we have shown that the $O(\log\log n)$-round LLL algorithm [90, 133] works without the knowledge of n. □

10.9 Missing Borel Construction

Proof of Theorem 10.48. Let x be a vertex of degree strictly bigger than 2. A *star* $S(x)$ around x consists of x and vertices of degree 2 that are connected with x by a path with all inner vertices of degree 2. It is clear that $S(x)$ is connected in $\mathcal{G}$.

Recall that if F is a function we define the iterated preimage of a vertex x as $F^{\leftarrow}(x) = \bigcup_{n\in\mathbb{N}} F^{-n}(x)$. We first specify a canonical one ended orientation on $S(x)$, or, equivalently, a function F with finite iterated preimages. (In what follows we interchange freely the notion of one ended orientation and function with finite iterated preimages.) That is to say, if $S(x)$ is infinite we orient things towards infinity in a one ended fashion (that is always possible), otherwise we fix an orientation towards any of the boundary points, i.e., there is exactly one point that is directed outside of $S(x)$. We refer to this orientation as *the canonical orientation.*

The inductive construction produces an orientation of some vertices together with doubly infinite lines. The orientation points either to infinity (is one-ended) or towards these doubly infinite lines, this takes $(\aleph_0 + 1)$-many steps.

For a graph $\mathcal{G}'$ we define a graph $\mathcal{H}'$ on the vertices of degree strictly bigger than 2, where $(x, y) \in \mathcal{H}'$ if there is a path from x to y in $\mathcal{G}'$ that has at most one inner vertex of degree strictly bigger than 2. Since, in our situation, $\mathcal{H}'$ is always Borel and has degree bounded by Δ^2, we can pick a Borel maximal independent set $\mathcal{M}'$ of $\mathcal{H}'$ by [220].

Inductively along $\mathbb{N}$ do the following: Suppose that we are in stage $k \in \mathbb{N}$ and we have a Borel set $\mathcal{O}_k$ and $\mathcal{G}_k := \mathcal{G} \upharpoonright \mathcal{O}_k$ with the property that every vertex has degree at least 2. We start with $\mathcal{G}_0 = \mathcal{G}$ and $\mathcal{O}_0 = X$. Pick a Borel maximal independent set $\mathcal{M}_k$ in $\mathcal{H}_k$ that is defined from $\mathcal{G}_k$. Add to the domain of F every vertex from the star $S_k(x)$ in $\mathcal{G}_k$, where $x \in \mathcal{M}_k$, and set

$$\mathcal{O}_{k+1} = \mathcal{O}_k \setminus \bigcup_{x \in \mathcal{M}_k} S(x).$$

Define F so that it corresponds to the canonical orientation on each $S_k(x)$. By the definition we have that $F(y) \in S_k(x)$ for every $y \in S_k(x)$, possibly up to one point $z \in S_k(x)$ that satisfies $F(z) \in \mathcal{O}_{k+1}$. It is easy to see that every $x \in \mathcal{O}_{k+1}$ has degree at least 2 in $\mathcal{G}_{k+1} = \mathcal{G} \upharpoonright \mathcal{O}_{k+1}$. This follows from the definition of $\mathcal{M}_k$.

Set $\mathcal{O}_\infty = \bigcap_{k \in \mathbb{N}} \mathcal{O}_k$ and write $\mathcal{G}_\infty := \mathcal{G} \upharpoonright \mathcal{O}_\infty$. It follows from the inductive (finitary) construction that every vertex $x \in \mathcal{O}_\infty$ has degree at least 2 in $\mathcal{G}_\infty$. Moreover, the function F satisfies $\mathrm{dom}(F) = X \setminus \mathcal{O}_\infty$ and has finite iterated preimages. This is because, first, for every $x \in \mathrm{dom}(F)$ there is $k \in \mathbb{N}$ such that $x \in \mathcal{O}_k \setminus \mathcal{O}_{k+1}$ and $F^{-1}(x) \subseteq X \setminus \mathcal{O}_{k+1}$. Second, $F^{\leftarrow}(x) \cap \mathcal{O}_k$ is finite by the definition of F on stars. It follows inductively that $F^{\leftarrow}(x) \cap \mathcal{O}_l$ is finite for every $l \leq k$, hence, $F^{\leftarrow}(x)$ is finite, whenever $x \in \mathrm{dom}(F)$. If $x \notin \mathrm{dom}(F)$, then $F^{-1}(x) \subseteq \mathrm{dom}(F)$ is finite and the claim follows.

Let $x \in \mathcal{O}_\infty$ have degree strictly bigger than 2 and write $C_1, \ldots, C_\ell$ for the connected components of $\mathcal{G}_\infty \setminus \{x\}$ in the connected component of x in $\mathcal{G}_\infty$. Note that $\ell \leq \Delta$. We claim that at least one of the sets C_i is a one ended line. Suppose not, and write $z_1, \ldots, z_\ell$ for the closest splitting points in $C_1, \ldots, C_\ell$ and p_i for the paths that connect x with z_i for every $i \leq \ell$. There is $k \in \mathbb{N}$ large enough such that the degree of $x, z_1, \ldots, z_\ell$ is the same in $\mathcal{G}_\infty$ as in $\mathcal{G}_k$ and p_i is a path in $\mathcal{G}_k$ whose inner vertices have degree 2. Note that $\mathcal{G}_\infty \subseteq \mathcal{G}_k$ because $\mathcal{O}_\infty \subseteq \mathcal{O}_k$. Since $\mathcal{M}_k$ was maximal in $\mathcal{H}_k$, there is $y \in \mathcal{M}_k$ such that $(x, y) \in \mathcal{H}_k$. This is because $x \notin \mathcal{M}_k$. Similar reasoning implies that that $y \neq z_i$ for any $i \leq \ell$. Let q be the path that connects x and y in $\mathcal{G}_k$. By the choice of k we have that q extends one of the paths p_i. Let y' be the last point on q such that $y' \in \mathcal{O}_\infty$. Since the degree of z_i is the same in $\mathcal{G}_k$ as in $\mathcal{G}_\infty$ we have that $y' \neq z_i$. The degree of y' in $\mathcal{G}_\infty$ is at least 2. Suppose it were 3 in $\mathcal{G}_k$, then $y' = y$ because we must have $(x, y) \in \mathcal{H}_k$. In that case removing $S(y)$ would decrease the degree of z_i in $\mathcal{G}_{k+1}$ and consequently in $\mathcal{G}_\infty$. Therefore y' has degree 2 in $\mathcal{G}_k$. But then y' is not the last vertex on q such that $y' \in \mathcal{O}_\infty$, i.e., the other neighbor of y', that is the same in $\mathcal{G}_k$ and in $\mathcal{G}_\infty$ must be a vertex on q, a contradiction.

Consider now the graph $\mathcal{H}_\infty$, where $(x, y) \in \mathcal{H}_\infty$ if and only if $x, y \in \mathcal{O}_\infty$ have degree strictly bigger than 2 and there is a path from x to y in $\mathcal{G}_\infty$ with all inner points having degree 2 in $\mathcal{G}_\infty$. Consider any Borel $(\Delta+1)$-coloring of $\mathcal{H}_\infty$ a decomposition of all vertices of degree strictly bigger than 2 in $\mathcal{G}_\infty$ into $\mathcal{H}_\infty$-independent Borel sets $D_0, \dots, D_\Delta$. Define a one ended orientation of stars of the form $S(x)$, where $x \in D_i$, inductively according to $i \leq \Delta$ using the canonical orientation, i.e., in step $i \leq \Delta$ we work with the graph

$$\mathcal{G} \upharpoonright \left(\mathcal{O}_\infty \setminus \bigcup_{j<i} \bigcup_{x \in D_j} S(x) \right).$$

Note that by the previous argument we have that every such star $S(x)$ is infinite and consequently, this defines a valid extension of F, i.e., iterated preimages are still finite. It remains to realize that complement of $\operatorname{dom}(F)$ consists of doubly infinite lines. This finishes the proof. □

Bibliography

[1] Ittai Abraham, Yair Bartal, and Ofer Neiman. Embedding metrics into ultrametrics and graphs into spanning trees with constant average distortion. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 502–511. SIAM, 2007. URL http://dl.acm.org/citation.cfm?id=1283383.1283437.

[2] Amirreza Akbari, Navid Eslami, Henrik Lievonen, Darya Melnyk, Joona Särkijärvi, and Jukka Suomela. Locality in Online, Dynamic, Sequential, and Distributed Graph Algorithms. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-278-5. doi: 10.4230/LIPIcs.ICALP.2023.10. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2023.10.

[3] Amirreza Akbari, Xavier Coiteux-Roy, Francesco d'Amore, François Le Gall, Henrik Lievonen, Darya Melnyk, Augusto Modanese, Shreyas Pai, Marc-Olivier Renou, Václav Rozhoň, et al. Online locality meets distributed quantum computing. *arXiv preprint arXiv:2403.01903*, 2024.

[4] Amirreza Akbari, Xavier Coiteux-Roy, Francesco d'Amore, François Le Gall, Henrik Lievonen, Darya Melnyk, Augusto Modanese, Shreyas Pai, Marc-Olivier Renou, Václav Rozhoň, et al. Online locality meets distributed quantum computing. *arXiv preprint arXiv:2403.01903*, 2024.

[5] Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008. ISBN 978-0-470-17020-5.

[6] Noga Alon, Laszlo Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4): 567–583, 1986.

[7] Noga Alon, Richard M Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995.

[8] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1132–1139. SIAM, 2012.

[9] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On

sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1): 81–100, 1993.

[10] Omer Angel, Itai Benjamini, Ori Gurel-Gurevich, Tom Meyerovitch, and Ron Peled. Stationary map coloring. *Ann. Inst. Henri Poincaré Probab. Stat.*, 48(2): 327–342, 2012.

[11] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009.

[12] Matti Åstrand, Valentin Polishchuk, Joel Rybicki, Jukka Suomela, and Jara Uitto. Local algorithms in (weakly) coloured graphs. *arXiv preprint arXiv:1002.0125*, 2010.

[13] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. 30th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 364–369, 1989.

[14] Ágnes Backhausz and Balázs Szegedy. On large girth regular graphs and random processes on trees. *Random Structures Algorithms*, 53(3):389–416, 2018.

[15] Ágnes Backhausz and Bálint Virag. Spectral measures of factor of i.i.d. processes on vertex-transitive graphs. *Ann. Inst. Henri Poincaré Probab. Stat.*, 53(4):2260–2278, 2017.

[16] Ágnes Backhausz, Balázs Szegedy, and Bálint Virág. Ramanujan graphings and correlation decay in local algorithms. *Random Structures Algorithms*, 47(3):424–435, 2015. ISSN 1042-9832. doi: 10.1002/rsa.20562. URL https://doi.org/10.1002/rsa.20562.

[17] Ágnes Backhausz, Balázs Gerencsér, and Viktor Harangi. Entropy inequalities for factors of IID. *Groups Geom. Dyn.*, 13(2):389–414, 2019. ISSN 1661-7207. doi: 10.4171/GGD/492. URL https://doi.org/10.4171/GGD/492.

[18] Ágnes Backhausz, Balázs Gerencsér, Viktor Harangi, and Máté Vizer. Correlation bounds for distant parts of factor of iid processes. *Combin. Probab. Comput.*, 27 (1):1–20, 2018.

[19] K. Ball. Poisson thinning by monotone factors. *Electron. Comm. Probab.*, 10:60–69, 2005.

[20] Alkida Balliu, Juho Hirvonen, Janne H Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1307–1318, 2018.

[21] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. The distributed complexity of locally checkable problems on paths is decidable. In *Proceedings of the 2019 ACM Symposium on*

Principles of Distributed Computing, PODC '19, page 262–271, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362177. doi: 10.1145/3293611.3331606. URL https://doi.org/10.1145/3293611.3331606.

[22] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In *Proc. Foundations of Computer Science (FOCS)*, 2019.

[23] Alkida Balliu, Juho Hirvonen, Christoph Lenzen, Dennis Olivetti, and Jukka Suomela. Locality of not-so-weak coloring. In *International Colloquium on Structural Information and Communication Complexity*, pages 37–51. Springer, 2019.

[24] Alkida Balliu, Juho Hirvonen, Dennis Olivetti, and Jukka Suomela. Hardness of minimal symmetry breaking in distributed computing. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 369–378, 2019.

[25] Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Classification of distributed binary labeling problems. In *34th International Symposium on Distributed Computing (DISC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[26] Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. Distributed lower bounds for ruling sets. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 365–376, 2020. doi: 10.1109/FOCS46700.2020.00042. URL https://doi.org/10.1109/FOCS46700.2020.00042.

[27] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. Almost global problems in the local model. *Distributed Computing*, pages 1–23, 2020.

[28] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. How much does randomness help with locally checkable problems? In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, page 299–308, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375825. doi: 10.1145/3382734.3405715. URL https://doi.org/10.1145/3382734.3405715.

[29] Alkida Balliu, Fabian Kuhn, and Dennis Olivetti. Distributed edge coloring in time quasi-polylogarithmic in delta. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 289–298, 2020.

[30] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Improved distributed lower bounds for mis and bounded (out-) degree dominating sets in trees. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 283–293, 2021.

[31] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, Jan Studený, Jukka Suomela, and Aleksandr Tereshchenko. Locally checkable problems in rooted trees. In *Proceedings*

of the 2021 ACM Symposium on Principles of Distributed Computing, pages 263–272, 2021.

[32] Alkida Balliu, Keren Censor-Hillel, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Locally checkable labelings with small messages. In *35th International Symposium on Distributed Computing*, 2021.

[33] Alkida Balliu, Fabian Kuhn, and Dennis Olivetti. Improved distributed fractional coloring algorithms. *arXiv preprint arXiv:2112.04405*, 2021.

[34] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, and Jukka Suomela. Efficient classification of locally checkable problems in regular trees. In *36th International Symposium on Distributed Computing*, 2022.

[35] Alkida Balliu, Sebastian Brandt, Manuela Fischer, Rustam Latypov, Yannic Maus, Dennis Olivetti, and Jara Uitto. Exponential speedup over locality in mpc with optimal memory. *arXiv preprint arXiv:2208.09453*, 2022.

[36] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed Δ-coloring plays hide-and-seek. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 464–477, 2022.

[37] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed edge coloring in time polylogarithmic in δ. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 15–25, 2022.

[38] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed maximal matching and maximal independent set on hypergraphs. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2632–2676. SIAM, 2023.

[39] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, Dennis Olivetti, and Gustav Schmid. On the node-averaged complexity of locally checkable problems on trees. *arXiv preprint arXiv:2308.04251*, 2023.

[40] Alkida Balliu, Janne H Korhonen, Fabian Kuhn, Henrik Lievonen, Dennis Olivetti, Shreyas Pai, Ami Paz, Joel Rybicki, Stefan Schmid, Jan Studený, et al. Sinkless orientation made simple. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 175–191. SIAM, 2023.

[41] Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Efficient deterministic distributed coloring with small bandwidth. In *Proc. Principles of Distributed Computing (PODC)*, pages to appear, arXiv:1912.02814, 2020.

[42] Leonid Barenboim. Deterministic $(\Delta + 1)$-coloring in sublinear (in Δ) time in static, dynamic and faulty networks. In *Proc. 34th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 345–354, 2015.

[43] Leonid Barenboim and Michael Elkin. Sublogarithmic distributed mis algorithm

for sparse graphs using nash-williams decomposition. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 25–34, 2008.

[44] Leonid Barenboim and Michael Elkin. Deterministic distributed vertex coloring in polylogarithmic time. In *Proc. 29th Symp. on Principles of Distributed Computing (PODC)*, pages 410–419, 2010.

[45] Leonid Barenboim and Michael Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *Journal of the ACM (JACM)*, 58(5):1–25, 2011.

[46] Leonid Barenboim and Yaniv Tzur. Distributed symmetry-breaking with improved vertex-averaged complexity, 2018.

[47] Leonid Barenboim, Michael Elkin, and Fabian Kuhn. Distributed $(\Delta+1)$-coloring in linear (in Δ) time. *SIAM Journal on Computing*, 43(1):72–95, 2015.

[48] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of the ACM*, 63:20:1–20:45, 2016.

[49] Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed $(\Delta+\ 1)$: -coloring below szegedy-vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, page 437–446, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357951. doi: 10.1145/3212734.3212769. URL `https://doi.org/10.1145/3212734.3212769`.

[50] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 184–193. IEEE, 1996.

[51] Yair Bartal. On approximating arbitrary metrices by tree metrics. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 161–168. ACM, 1998. doi: 10.1145/276698.276725. URL `https://doi.org/10.1145/276698.276725`.

[52] Yair Bartal. Advances in metric ramsey theory and its applications, 2021.

[53] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

[54] József Beck. An algorithmic approach to the lovász local lemma. i. *Random Structures & Algorithms*, 2(4):343–365, 1991.

[55] Ruben Becker, Yuval Emek, Mohsen Ghaffari, and Christoph Lenzen. Distributed algorithms for low stretch spanning trees. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146, page 4. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

[56] Ruben Becker, Yuval Emek, and Christoph Lenzen. Low diameter graph decompositions by approximate distance computation. In *ITCS*, 2020.

[57] Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 873–884. IEEE, 2022.

[58] Ferenc Bencs, Aranka Hrušková, and László Márton Tóth. Factor of iid schreier decoration of transitive graphs. *arXiv:2101.12577*, 2021.

[59] Itai Benjamini and Oded Schramm. Recurrence of distributional limits of finite planar graphs. In *Selected Works of Oded Schramm*, pages 533–545. Springer, 2011.

[60] Anton Bernshteyn. Distributed algorithms, the lovász local lemma, and descriptive combinatorics. *arXiv preprint arXiv:2004.04905*, 2020.

[61] Anton Bernshteyn. Probabilistic constructions in continuous combinatorics and a bridge to distributed algorithms, 2021.

[62] Anton Bernshteyn. Work in progress, 2022.

[63] Anton Bernshteyn. Descriptive combinatorics and distributed algorithms. *NOTICES OF THE AMERICAN MATHEMATICAL SOCIETY*, 69(9), 2022.

[64] Anton Bernshteyn. A fast distributed algorithm for $(\Delta + 1)$-edge-coloring. *Journal of Combinatorial Theory, Series B*, 152:319–352, 2022.

[65] Anton Bernshteyn and Abhishek Dhawan. Borel vizing's theorem for graphs of subexponential growth. *arXiv e-prints*, pages arXiv–2307, 2023.

[66] Anton Bernshteyn and Abhishek Dhawan. Fast algorithms for vizing's theorem on bounded degree graphs. *arXiv preprint arXiv:2303.05408*, 2023.

[67] Anton Bernshteyn and Felix Weilacher. Borel versions of the local lemma and local algorithms for graphs of finite asymptotic separation index. *arXiv preprint arXiv:2308.14941*, 2023.

[68] Anton Bernshteyn and Jing Yu. Coarse embeddings into grids and asymptotic dimension for borel graphs of polynomial growth. *arXiv preprint arXiv:2302.04727*, 2023.

[69] Marcel Bezdrighin, Michael Elkin, Mohsen Ghaffari, Christoph Grunau, Bernhard Haeupler, Saeed Ilchi, and Václav Rozhoň. Deterministic distributed sparse and ultra-sparse spanners and connectivity certificates. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 1–10, 2022.

[70] Béla Bollobás. The independence ratio of regular graphs. *Proc. Amer. Math. Soc.*, 83(2):433–436, 1981. ISSN 0002-9939. doi: 10.2307/2043545. URL https://doi.org/10.2307/2043545.

[71] Lewis Bowen. A measure-conjugacy invariant for free group actions. *Ann. of Math.*, 171(2):1387–1400, 2010.

[72] Lewis Bowen. The ergodic theory of free group actions: entropy and the f -invariant. *Groups Geom. Dyn.*, 4(3):419–432, 2010.

[73] S. Brandt, O. Fischer, J. Hirvonen, B. Keller, T. Lempiäinen, J. Rybicki, J. Suomela, and J. Uitto. A lower bound for the distributed Lovász local lemma. In *Proc. 48th ACM Symp. on Theory of Computing (STOC)*, pages 479–488, 2016.

[74] Sebastian Brandt. An automatic speedup theorem for distributed problems. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 379–388, 2019. doi: 10.1145/3293611.3331611. URL https://doi.org/10.1145/3293611.3331611.

[75] Sebastian Brandt. An automatic speedup theorem for distributed problems. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 379–388, 2019.

[76] Sebastian Brandt and Dennis Olivetti. Truly tight-in-δ bounds for bipartite maximal matching and variants. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 69–78, 2020.

[77] Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R.J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemysław Uznański. Lcl problems on grids. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, page 101–110, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349925. doi: 10.1145/3087801.3087833. URL https://doi.org/10.1145/3087801.3087833.

[78] Sebastian Brandt, Yannic Maus, and Jara Uitto. A sharp threshold phenomenon for the distributed complexity of the lovász local lemma. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 389–398, 2019.

[79] Sebastian Brandt, Christoph Grunau, and Václav Rozhoň. Generalizing the sharp threshold phenomenon for the distributed complexity of the Lovász local lemma. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 329–338, 2020.

[80] Sebastian Brandt, Yi-Jun Chang, Jan Grebík, Christoph Grunau, Václav Rozhoň, and Zoltán Vidnyánszky. On homomorphism graphs. *arXiv preprint arXiv:2111.03683*, 2021.

[81] Sebastian Brandt, Christoph Grunau, and Václav Rozhoň. The randomized local computation complexity of the Lovász local lemma. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 307–317, 2021.

[82] Sebastian Brandt, Rustam Latypov, and Jara Uitto. Towards a complexity classification of lcl problems in massively parallel computation. *arXiv preprint arXiv:2112.09479*, 2021.

[83] Sebastian Brandt, Yi-Jun Chang, Jan Grebík, Christoph Grunau, Václav Rozhoň, and Zoltán Vidnyánszky. Local problems on trees from the perspectives of distributed algorithms, finitary factors, and descriptive combinatorics. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[84] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. In *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[85] Yi-Jun Chang. The Complexity Landscape of Distributed Locally Checkable Problems on Trees. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISBN 978-3-95977-168-9. doi: 10.4230/LIPIcs.DISC.2020.18. URL https://drops.dagstuhl.de/opus/volltexte/2020/13096.

[86] Yi-Jun Chang. The distributed complexity of locally checkable labeling problems beyond paths and trees. *arXiv preprint arXiv:2311.06726*, 2023.

[87] Yi-Jun Chang and Mohsen Ghaffari. Strong-diameter network decomposition. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 273–281, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385480. doi: 10.1145/3465084.3467933. URL https://doi.org/10.1145/3465084.3467933.

[88] Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the local model. *SIAM Journal on Computing*, 48(1):33–69, 2019.

[89] Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 66–73, 2019.

[90] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2633–2652. SIAM, 2018.

[91] Yi-Jun Chang, Tsvi Kopelowitz, Seth Pettie, Ruosong Wang, and Wei Zhan. Exponential separations in the energy complexity of leader election, 2018.

[92] Yi-Jun Chang, Wenzheng Li, and Seth. Pettie. An optimal distributed $(\Delta + 1)$-

coloring algorithm? In *Proc. 50th ACM Symp. on Theory of Computing (STOC)*, 2018.

[93] 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 *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 471–480. ACM, 2019.

[94] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the local model. *SIAM Journal on Computing*, 48(1):122–143, 2019.

[95] Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 821–840. SIAM, 2019.

[96] Yi-Jun Chang, Varsha Dani, Thomas P. Hayes, and Seth Pettie. The energy complexity of bfs in radio networks, 2020.

[97] Yi-Jun Chang, Jan Studenỳ, and Jukka Suomela. Distributed graph problems through an automata-theoretic lens. In *International Colloquium on Structural Information and Communication Complexity*, pages 31–49. Springer, 2021.

[98] Yi-Jun Chang, Gopinath Mishra, Thuan Hung Nguyen, Mingyang Yang, and Yu-Cheng Yeh. A tight lower bound for 3-coloring grids in the online-local model. *arXiv preprint arXiv:2312.01384*, 2023.

[99] Soumyottam Chatterjee, Robert Gmyr, and Gopal Pandurangan. Sleeping is efficient: Mis in $o(1)$-rounds node-averaged awake complexity, 2020.

[100] Sourav Chatterjee, Ron Peled, Yuval Peres, and Dan Romik. Gravitational allocation to poisson points. *Ann. of Math.*, 172(1):617–671, 2010.

[101] Davin Choo, Christoph Grunau, Julian Portmann, and Václav Rozhon. k-means++: few more steps yield constant approximation. In *International Conference on Machine Learning*, pages 1909–1917. PMLR, 2020.

[102] Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. Distributed algorithms for the lovász local lemma and graph coloring. *Distributed Computing*, 30(4):261–280, 2017.

[103] Xavier Coiteux-Roy, Francesco d'Amore, Rishikesh Gajjala, Fabian Kuhn, François Le Gall, Henrik Lievonen, Augusto Modanese, Marc-Olivier Renou, Gustav Schmid, and Jukka Suomela. No distributed quantum advantage for approximate graph coloring. *arXiv preprint arXiv:2307.09444*, 2023.

[104] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[105] Clinton T. Conley and Benjamin D. Miller. A bound on measurable chromatic numbers of locally finite Borel graphs. *Math. Res. Lett.*, 23(6):1633–1644, 2016. ISSN 1073-2780. doi: 10.4310/MRL.2016.v23.n6.a3. URL https://doi.org/10.4310/MRL.2016.v23.n6.a3.

[106] Clinton T. Conley and Benjamin D. Miller. Measure reducibility of countable Borel equivalence relations. *Ann. of Math. (2)*, 185(2):347–402, 2017. ISSN 0003-486X. doi: 10.4007/annals.2017.185.2.1. URL https://doi.org/10.4007/annals.2017.185.2.1.

[107] Clinton T. Conley, Andrew S. Marks, and Robin D. Tucker-Drob. Brooks' theorem for measurable colorings. *Forum Math. Sigma*, e16(4):23pp, 2016.

[108] Clinton T. Conley, Jan Grebík, and Oleg Pikhurko. Divisibility of spheres with measurable pieces, 2020.

[109] Clinton T. Conley, Andrew S. Marks, and Spencer T. Unger. Measurable realizations of abstract systems of congruences. In *Forum of Mathematics, Sigma*, volume 8. Cambridge University Press, 2020.

[110] Endre Csóka, Balázs Gerencsér, Viktor Harangi, and Bálint Virág. Invariant gaussian processes and independent sets on regular graphs of large girth. *Random Structures & Algorithms*, 47(2):284–303, 2015.

[111] Endre Csóka, Łukasz Grabowski, András Máthé, Oleg Pikhurko, and Konstantinos Tyros. Moser-tardos algorithm with small number of random bits. *arXiv preprint arXiv:2203.05888*, 2022.

[112] Endre Csóka and András Pongrácz. Local algorithms for the maximum flow and minimum cut in bounded-degree networks, 2023.

[113] Artur Czumaj and Christian Scheideler. Coloring Non-uniform Hypergraphs: A New Algorithmic Approach to the General Lovász Local Lemma. In *Proc. Symposium on Discrete Algorithms (SODA)*, pages 30–39, 2000. ISBN 0-89871-453-2.

[114] Artur Czumaj and Christian Scheideler. A new algorithmic approach to the general Lovász local lemma with applications to scheduling and satisfiability problems. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 38–47, 2000.

[115] Sameep Dahal and Jukka Suomela. Distributed half-integral matching and beyond. *Theoretical Computer Science*, 982:114278, 2024.

[116] Peter Davies. Improved distributed algorithms for the lovász local lemma and edge coloring. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4273–4295. SIAM, 2023.

[117] Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the locality of distributed sparse spanner construction. In *Proceedings of the Twenty-*

Seventh ACM Symposium on Principles of Distributed Computing, PODC '08, page 273–282, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781595939890. doi: 10.1145/1400751.1400788. URL https://doi.org/10.1145/1400751.1400788.

[118] Janosch Deurer, Fabian Kuhn, and Yannic Maus. Deterministic distributed dominating set approximation in the congest model. In *Proc. Principles of Distributed Computing (PODC)*, pages 94–103, 2019.

[119] Randall Dougherty and Matthew Foreman. Banach-Tarski paradox using pieces with the property of Baire. *Proc. Nat. Acad. Sci. U.S.A.*, 89(22):10726–10728, 1992. ISSN 0027-8424. doi: 10.1073/pnas.89.22.10726. URL https://doi.org/10.1073/pnas.89.22.10726.

[120] Randall Dougherty, Steve Jackson, and Alexander S. Kechris. The structure of hyperfinite Borel equivalence relations. *Trans. Amer. Math. Soc.*, 341(1):193–225, 1994. ISSN 0002-9947. doi: 10.2307/2154620. URL https://doi.org/10.2307/2154620.

[121] Gábor Elek and Gábor Lippner. Borel oracles. an analytical approach to constant-time algorithms. *Proceedings of the American Mathematical Society*, 138(8):2939–2947, 2010.

[122] Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 652–669. SIAM, 2017.

[123] Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.

[124] Michael Elkin, Seth Pettie, and Hsin-Hao Su. (2δ—l)-edge-coloring is much easier than maximal matching in the distributed setting. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 355–370. SIAM, 2014.

[125] Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Infinite and finite sets*, 10(2):609–627, 1975.

[126] Paul Erdös, Peter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union of two others. *Journal of Combinatorial Theory, Series A*, 33(2):158–166, 1982.

[127] Guy Even, Moti Medina, and Dana Ron. Deterministic stateless centralized local algorithms for bounded degree graphs. In *European Symposium on Algorithms*, pages 394–405. Springer, 2014.

[128] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497,

2004. doi: 10.1016/j.jcss.2004.04.011. URL https://doi.org/10.1016/j.jcss.2004.04.011.

[129] Salwa Faour, Mohsen Ghaffari, Christoph Grunau, Fabian Kuhn, and Václav Rozhoň. Local distributed rounding: Generalized to mis, matching, set cover, and beyond. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4409–4447. SIAM, 2023.

[130] Jens E. Fenstad and Dag Normann. On absolutely measurable sets. *Fund. Math.*, 81(2):91–98, 1973/74. ISSN 0016-2736. doi: 10.4064/fm-81-2-91-98. URL https://doi.org/10.4064/fm-81-2-91-98.

[131] Laurent Feuilloley. How long it takes for an ordinary node with an ordinary id to output?, 2017.

[132] Laurent Feuilloley. Introduction to local certification. *CoRR*, abs/1910.12747, 2019. URL http://arxiv.org/abs/1910.12747.

[133] Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for lovász local lemma, and the complexity hierarchy. In *31 International Symposium on Distributed Computing*, 2017.

[134] Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2017.

[135] Manuela Fischer, Slobodan Mitrović, and Jara Uitto. Deterministic $(1+\varepsilon)$-approximate maximum matching with poly $(1/\varepsilon)$ passes in the semi-streaming model and beyond. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 248–260, 2022.

[136] P. Fraigniaud, M. Heinrich, and A. Kosowski. Local conflict coloring. In *Proc. 57th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 625–634, 2016.

[137] Pierre Fraigniaud, Magnús M Halldórsson, and Amos Korman. On the impact of identifiers on local decision. In *International Conference On Principles Of Distributed Systems*, pages 224–238. Springer, 2012.

[138] Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. What can be decided locally without identifiers? In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, page 157–165, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320658. doi: 10.1145/2484239.2484264. URL https://doi.org/10.1145/2484239.2484264.

[139] Marc Fuchs and Fabian Kuhn. List defective colorings: Distributed algorithms and applications. *arXiv preprint arXiv:2304.09666*, 2023.

[140] Damien Gaboriau. Coût des relations d'équivalence et des groupes. *Invent. Math.*,

139(1):41–98, 2000. ISSN 0020-9910. doi: 10.1007/s002229900019. URL https://doi.org/10.1007/s002229900019.

[141] Damien Gaboriau and Russell Lyons. A measurable-group-theoretic solution to von neumann's problem. *Invent. Math.*, 177(3):533–540, 2009.

[142] François Le Gall, Harumichi Nishimura, and Ansis Rosmanis. Quantum advantage for the local model in distributed computing. *arXiv preprint arXiv:1810.10838*, 2018.

[143] David Gamarnik and Madhu Sudan. Limits of local algorithms over sparse random graphs. *Ann. Probab.*, 45(4):2353–2376, 2017.

[144] Su Gao and Steve Jackson. Countable abelian group actions and hyperfinite equivalence relations. *Inventiones Math.*, 201(1):309–383, 2015.

[145] Su Gao, Steve Jackson, Edward Krohne, and Brandon Seward. Forcing constructions and countable borel equivalence relations. *arXiv:1503.07822*, 2015.

[146] Su Gao, Steve Jackson, Edward Krohne, and Brandon Seward. Continuous combinatorics of abelian group actions. *arXiv preprint arXiv:1803.03872*, 2018.

[147] Su Gao, Steve Jackson, Edward Krohne, and Brandon Seward. Continuous combinatorics of abelian group actions. *arXiv preprint arXiv:1803.03872*, 2018.

[148] Cyril Gavoille, Ralf Klasing, Adrian Kosowski, Łukasz Kuszner, and Alfredo Navarra. On the complexity of distributed graph coloring with local minimality constraints. *Networks: An International Journal*, 54(1):12–19, 2009.

[149] Cyril Gavoille, Adrian Kosowski, and Marcin Markiewicz. What can be observed locally? round-based models for quantum distributed computing. In *Proceedings of the 23rd International Conference on Distributed Computing*, DISC'09, page 243–257, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 3642043542.

[150] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 270–277, 2016.

[151] Mohsen Ghaffari. Distributed maximal independent set using small messages. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–820. SIAM, 2019.

[152] Mohsen Ghaffari. Local computation of maximal independent set. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 438–449. IEEE, 2022.

[153] Mohsen Ghaffari and Christoph Grunau. Faster deterministic distributed mis and approximate matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1777–1790, 2023.

[154] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks ii: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 202–219, 2016.

[155] Mohsen Ghaffari and Bernhard Haeupler. Low-congestion shortcuts for graphs excluding dense minors. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 213–221, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385480. doi: 10.1145/3465084.3467935. URL https://doi.org/10.1145/3465084.3467935.

[156] Mohsen Ghaffari and Bernhard Haeupler. A time-optimal randomized parallel algorithm for mis. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2892–2903. SIAM, 2021.

[157] Mohsen Ghaffari and Fabian Kuhn. Derandomizing distributed algorithms with small messages: Spanners and dominating set. In *32nd International Symposium on Distributed Computing (DISC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[158] Mohsen Ghaffari and Fabian Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1009–1020. IEEE, 2022.

[159] Mohsen Ghaffari and Julian Portmann. Improved network decompositions using small messages with applications on mis, neighborhood covers, and beyond. In *33rd International Symposium on Distributed Computing*, 2019.

[160] Mohsen Ghaffari and Julian Portmann. Average awake complexity of mis and matching. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '22, page 45–55, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391467. doi: 10.1145/3490148.3538566. URL https://doi.org/10.1145/3490148.3538566.

[161] Mohsen Ghaffari and Julian Portmann. Distributed mis with low energy and time complexities. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, PODC '23, page 146–156, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701214. doi: 10.1145/3583668.3594587. URL https://doi.org/10.1145/3583668.3594587.

[162] Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 2505–2523, 2017.

[163] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Pro-*

ceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 1636–1653. SIAM, 2019.

[164] Mohsen Ghaffari and Goran Zuzic. Universally-optimal distributed exact min-cut. *arXiv preprint*, 2022.

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

[166] Mohsen Ghaffari, David Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *Proc. Foundations of Computer Science (FOCS)*, pages 662–673, 2018.

[167] Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. Deterministic distributed edge-coloring with fewer colors. In *Proc. 50th ACM Symp. on Theory of Computing (STOC)*, 2018.

[168] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In *Proc. Foundations of Computer Science (FOCS)*, 2019.

[169] Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, and Yannic Maus. Improved distributed δ-coloring, 2020.

[170] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. Improved deterministic network decomposition. In *Proc. of the 32nd ACM-SIAM Symp. on Discrete Algorithms (SODA)*, page 2904–2923, USA, 2021. Society for Industrial and Applied Mathematics. ISBN 9781611976465.

[171] Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. Hop-constrained oblivious routing. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1208–1220, 2021.

[172] Mohsen Ghaffari, Christoph Grunau, Bernhard Haeupler, Saeed Ilchi, and Václav Rozhoň. Improved distributed network decomposition, hitting sets, and spanners, via derandomization. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2532–2566. SIAM, 2023.

[173] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. Work-efficient parallel derandomization i: Chernoff-like concentrations via pairwise independence. *arXiv preprint arXiv:2311.13764*, 2023.

[174] Ghaffari, Mohsen. Distributed graph algorithms, 2020. URL https://disco.ethz.ch/courses/fs20/podc/lecturenotes/DGA.pdf. Lecture notes.

[175] Andrew V. Goldberg, Sergey A. Plotkin, and Gregory E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988.

[176] Mika Göös and Jukka Suomela. Locally checkable proofs. In *Proc. 30th Symp. on Principles of Distributed Computing (PODC)*, pages 159–168, 2011.

[177] Mika Göös, Juho Hirvonen, and Jukka Suomela. Lower bounds for local approximation. *J. ACM*, 60(5), oct 2013. ISSN 0004-5411. doi: 10.1145/2528405. URL https://doi.org/10.1145/2528405.

[178] Mika Göös, Juho Hirvonen, and Jukka Suomela. Linear-in-delta lower bounds in the local model. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, page 86–95, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329446. doi: 10.1145/2611462.2611467. URL https://doi.org/10.1145/2611462.2611467.

[179] Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, and Jukka Suomela. Non-local probes do not help with many graph problems. In *Distributed Computing: 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings 30*, pages 201–214. Springer, 2016.

[180] Lukasz Grabowski, András Máthé, and Oleg Pikhurko. Measurable circle squaring. *Ann. of Math.*, 185(2):671–710, 2017.

[181] Ronald L Graham, Bruce L Rothschild, and Joel H Spencer. *Ramsey theory*, volume 20. John Wiley & Sons, 1991.

[182] Jan Grebík. Measurable Vizing's theorem. *arXiv preprint arXiv:2303.16440*, 2023.

[183] Jan Grebík and Oleg Pikhurko. Measurable versions of vizing's theorem. *Advances in Mathematics*, 374:107378, 2020.

[184] Jan Grebík and Václav Rozhoň. Classification of local problems on paths from the perspective of descriptive combinatorics. In *Extended Abstracts EuroComb 2021: European Conference on Combinatorics, Graph Theory and Applications*, pages 553–559. Springer, 2021.

[185] Jan Grebík and Václav Rozhoň. Local problems on grids from the perspective of distributed algorithms, finitary factors, and descriptive combinatorics. *Advances in Mathematics*, 431:109241, 2023.

[186] Jan Grebík, Rachel Greenfeld, Václav Rozhoň, and Terence Tao. Measurable tilings by abelian group actions. *International Mathematics Research Notices*, 2023(23): 20211–20251, 2023.

[187] Christoph Grunau and Václav Rozhoň. Adapting k-means algorithms for outliers, 2020. URL https://arxiv.org/abs/2007.01118.

[188] Christoph Grunau, Václav Rozhoň, and Sebastian Brandt. The landscape of distributed complexities on trees and beyond. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 37–47, 2022.

[189] Christoph Grunau, Ahmet Alper Özüdoğru, and Václav Rozhoň. Noisy k-means++ revisited. *arXiv preprint arXiv:2307.13685*, 2023.

[190] Christoph Grunau, Ahmet Alper Özüdoğru, Václav Rozhoň, and Jakub Tětek. A nearly tight analysis of greedy k-means++. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1012–1070. SIAM, 2023.

[191] Ori Gurel-Gurevich and Ron Peled. Poisson thickening. *Israel J. Math.*, 196(1): 215–234, 2013.

[192] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the Lovász local lemma. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 397–406, 2010.

[193] Bernhard Haeupler, D Ellis Hershkowitz, and Goran Zuzic. Tree embeddings for hop-constrained network design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 356–369, 2021.

[194] Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1166–1179, 2021.

[195] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 1325–1338, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392648. doi: 10.1145/3519935.3520026. URL https://doi.org/10.1145/3519935.3520026.

[196] Bernhard Haeupler, Richard Hladík, Václav Rozhoň, Robert Tarjan, and Jakub Tětek. Universal optimality of dijkstra via beyond-worst-case heaps. *arXiv preprint arXiv:2311.11793*, 2023.

[197] Magnús M Halldórsson, Fabian Kuhn, and Yannic Maus. Distance-2 coloring in the congest model. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 233–242, 2020.

[198] M. Hańćkowiak, M. Karoński, and A. Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Math.*, 15(1):41–57, 2001.

[199] Viktor. Harangi and Bálint Virág. Independence ratio and random eigenvectors in transitive graphs. *Ann. Probab.*, 43(5):2810–2840, 2015.

[200] David G Harris. Distributed local approximation algorithms for maximum matching in graphs and hypergraphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 700–724. IEEE, 2019.

[201] David G. Harris, Johannes Schneider, and Hsin-Hao Su. Distributed (Δ+1)-coloring in sublogarithmic rounds. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 465–478, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341325. doi: 10.1145/2897518.2897533. URL https://doi.org/10.1145/2897518.2897533.

[202] Hamed Hatami, László Lovász, and Balázs Szegedy. Limits of locally-globally convergent graph sequences. *Geom. Funct. Anal.*, 24(1):269–296, 2014. ISSN 1016-443X. doi: 10.1007/s00039-014-0258-7. URL https://doi.org/10.1007/s00039-014-0258-7.

[203] Juho Hirvonen and Jukka Suomela. Distributed maximal matching: Greedy is optimal. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, page 165–174, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450314503. doi: 10.1145/2332432.2332464. URL https://doi.org/10.1145/2332432.2332464.

[204] Greg Hjorth and Alexander S. Kechris. Borel equivalence relations and classifications of countable models. *Annals of pure and applied logic*, 82(3):221–272, 1996.

[205] Alexander Holroyd, Thomas Liggett, et al. Symmetric 1-dependent colorings of the integers. *Electronic Communications in Probability*, 20, 2015.

[206] Alexander E. Holroyd. Geometric properties of poisson matchings. *Probab. Theory Related Fields*, 150(3–4):511–527, 2011.

[207] Alexander E. Holroyd. One-dependent coloring by finitary factors. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 53(2):753 – 765, 2017. doi: 10.1214/15-AIHP735. URL https://doi.org/10.1214/15-AIHP735.

[208] Alexander E Holroyd. Symmetrization for finitely dependent colouring. *arXiv preprint arXiv:2305.13980*, 2023.

[209] Alexander E. Holroyd and Yuval Peres. Trees and matchings from point processes. *Electron. Comm. Probab.*, 8:17–27, 2003.

[210] Alexander E. Holroyd, Robin Pemantle, Y. Peres, and Oded Schramm. Poisson matching. *Ann. Inst. Henri Poincaré Probab. Stat.*, 45(1):266–287, 2009.

[211] Alexander E. Holroyd, Russel Lyons, and Terry Soo. Poisson splitting by factors. *Ann. Probab.*, 39(5):1938–1982, 2011.

[212] Alexander E. Holroyd, Oded Schramm, and David B. Wilson. Finitary coloring. *Ann. Probab.*, 45(5):2867–2898, 09 2017. doi: 10.1214/16-AOP1127. URL https://doi.org/10.1214/16-AOP1127.

[213] Carlos Hoppen and Nicholas Wormald. Local algorithms, regular graphs of large girth, and random regular graphs. *Combinatorica*, 38(3):619–664, 2018.

[214] Steve Jackson, Alexander S. Kechris, and Alain Louveau. Countable Borel equivalence relations. *J. Math. Logic*, 2(01):1–80, 2002.

[215] Joseph JáJá. *An introduction to parallel algorithms*. Addison Wesley Longman Publishing Co., Inc., 1992.

[216] Ojvind Johansson. Simple distributed $\Delta+1$-coloring of graphs. *Information Processing Letters*, 70(5):229–232, 1999. ISSN 0020-0190. doi: https://doi.org/10.1016/S0020-0190(99)00064-2. URL https://www.sciencedirect.com/science/article/pii/S0020019099000642.

[217] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. SIAM, 2010.

[218] Alexander S. Kechris. *Classical descriptive set theory*, volume 156 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995. ISBN 0-387-94374-9. doi: 10.1007/978-1-4612-4190-4. URL https://doi.org/10.1007/978-1-4612-4190-4.

[219] Alexander S Kechris and Andrew S Marks. Descriptive graph combinatorics. *preprint*, 2020(9), 2016.

[220] Alexander S. Kechris, Sławomir Solecki, and Stevo Todorčević. Borel chromatic numbers. *Adv. Math.*, 141(1):1–44, 1999.

[221] Daniel J. Kleitman, J Shearer, and Dean Sturtevant. Intersections of k-element sets. *Combinatorica*, 1(4):381–384, 1981.

[222] Janne H. Korhonen, Ami Paz, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Brief Announcement: Sinkless Orientation Is Hard Also in the Supported LOCAL Model. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing (DISC 2021)*, volume 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 58:1–58:4, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-210-5. doi: 10.4230/LIPIcs.DISC.2021.58. URL https://drops.dagstuhl.de/opus/volltexte/2021/14860.

[223] Amos Korman, Jean-Sébastien Sereni, and Laurent Viennot. Toward more localized local algorithms: removing assumptions concerning global knowledge. *Distributed Computing*, 26(5-6):289–308, Sep 2012. ISSN 1432-0452. doi: 10.1007/s00446-012-0174-8. URL http://dx.doi.org/10.1007/s00446-012-0174-8.

[224] Fabian Kuhn. Local weak coloring algorithms and implications on deterministic symmetry breaking. In *Proc. 21st ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, 2009.

[225] Fabian Kuhn. Faster deterministic distributed coloring through recursive list coloring. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1244–1259. SIAM, 2020.

[226] Fabian Kuhn and Rogert Wattenhofer. On the complexity of distributed graph coloring. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, page 7–15, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933840. doi: 10.1145/1146381.1146387. URL https://doi.org/10.1145/1146381.1146387.

[227] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *Journal of the ACM*, 63(2), 2016.

[228] Gábor Kun. Expanders have a spanning lipschitz subgraph with large girth. *preprint*, 2013.

[229] Jakub Lacki, Bernhard Haeupler, Christoph Grunau, Václav Rozhoň, and Rajesh Jayaram. Fully dynamic consistent k-center clustering. *arXiv preprint arXiv:2307.13747*, 2023.

[230] Miklós Laczkovich. Equidecomposability and discrepancy; a solution of Tarski's circle-squaring problem. *J. Reine Angew. Math.*, 404:77–117, 1990.

[231] Tom Leighton, Bruce Maggs, and Andréa W. Richa. Fast algorithms for finding O(congestion + dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999.

[232] Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 59–61, 2015.

[233] Jason Li. Faster parallel algorithm for approximate shortest path. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 308–321. ACM, 2020.

[234] Henrik Lievonen, Timothé Picavet, and Jukka Suomela. Distributed binary labeling problems in high-degree graphs. *arXiv preprint arXiv:2312.12243*, 2023.

[235] Henrik Lievonen, Timothé Picavet, and Jukka Suomela. Distributed binary labeling problems in high-degree graphs. *arXiv preprint arXiv:2312.12243*, 2023.

[236] Nati Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[237] Nati Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.

[238] László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012.

[239] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[240] Russell Lyons. Factors of iid on trees. *Combin. Probab. Comput.*, 26(2):285–300, 2017.

[241] Russell Lyons and Fedor Nazarov. Perfect matchings as iid factors on non-amenable groups. *European Journal of Combinatorics*, 32(7):1115–1125, 2011.

[242] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

[243] Andrew Marks and Spencer Unger. Baire measurable paradoxical decompositions via matchings. *Advances in Mathematics*, 289:397–410, 2016.

[244] Andrew S Marks. A determinacy approach to borel combinatorics. *Journal of the American Mathematical Society*, 29(2):579–600, 2016.

[245] Andrew S Marks and Spencer T Unger. Borel circle squaring. *Annals of Mathematics*, 186(2):581–605, 2017.

[246] Donald A Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

[247] Yannic Maus. Distributed graph coloring made easy. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 362–372, 2021.

[248] Yannic Maus and Tigran Tonoyan. Local Conflict Coloring Revisited: Linial for Lists. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-168-9. doi: 10.4230/LIPIcs.DISC.2020.16. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.DISC.2020.16.

[249] Colin McDiarmid. Concentration for independent permutations. *Combinatorics Probability and Computing*, 11(2):163–178, 2002.

[250] Brendan D. McKay, Nicholas C. Wormald, and Beata Wysocka. Short cycles in random regular graphs. *Electron. J. Combin.*, 11(1):Research Paper 66, 12, 2004. URL http://www.combinatorics.org/Volume_11/Abstracts/v11i1r66.html.

[251] Darya Melnyk, Jukka Suomela, and Neven Villani. Mending partial solutions with few changes. *arXiv preprint arXiv:2209.05363*, 2022.

[252] Darya Melnyk, Jukka Suomela, and Neven Villani. Mending partial solutions with few changes. In *26th International Conference on Principles of Distributed Systems*, 2023.

[253] Péter Mester. A factor of i.i.d with uniform marginals and infinite clusters spanned by equal labels. *preprint*, 2011.

[254] Gary L. Miller and John H. Reif. Parallel tree contraction–Part I: fundamentals. *Advances in Computing Research*, 5:47–72, 1989.

[255] Gary L Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 196–203, 2013.

[256] Robin A Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):1–15, 2010.

[257] Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991.

[258] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

[259] Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336, 2008. doi: 10.1109/FOCS.2008.81.

[260] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1123–1131. SIAM, 2012.

[261] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed computing*, 14(2):97–100, 2001.

[262] Alessandro Panconesi and Aravind Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Proc. 24th ACM Symp. on Theory of Computing (STOC)*, pages 581–592, 1992.

[263] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

[264] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[265] Oleg Pikhurko. Borel combinatorics of locally finite graphs, 2021.

[266] Long Qian and Felix Weilacher. Descriptive combinatorics, computable combinatorics, and asi algorithms. *arXiv preprint arXiv:2206.08426*, 2022.

[267] Mustazee Rahman. Factor of iid percolation on trees. *SIAM J. Discrete Math.*, 30 (4):2217–2242, 2016.

[268] Mustazee Rahman and Bálint Virág. Local algorithms for independent sets are half-optimal. *Ann. Probab.*, 45(3):1543–1577, 2017.

[269] Omer Reingold and Shai Vardi. New techniques and tighter bounds for local computation algorithms. *Journal of Computer and System Sciences*, 82(7):1180–1200, 2016.

[270] Will Rosenbaum and Jukka Suomela. Seeing far vs. seeing wide: Volume complexity of local graph problems. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, page 89–98, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375825. doi: 10.1145/3382734.3405721. URL https://doi.org/10.1145/3382734.3405721.

[271] Václav Rozhoň. Simple and sharp analysis of k-means||. In *International Conference on Machine Learning*, pages 8266–8275. PMLR, 2020.

[272] Václav Rozhoň, Michael Elkin, Christoph Grunau, and Bernhard Haeupler. Deterministic low-diameter decompositions for weighted graphs and distributed and parallel applications. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1114–1121. IEEE, 2022.

[273] Václav Rozhoň, Bernhard Haeupler, Anders Martinsson, Christoph Grunau, and Goran Zuzic. Parallel breadth-first search and exact shortest paths and stronger notions for approximate distances. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 321–334, 2023.

[274] Václav Rozhoň, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. Undirected (1+eps)-shortest paths via minor-aggregates: Near-optimal deterministic parallel and distributed algorithms. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 478–487, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392648. doi: 10.1145/3519935.3520074. URL https://doi.org/10.1145/3519935.3520074.

[275] Václav Rozhoň, Bernhard Haeupler, and Christoph Grunau. A simple deterministic distributed low-diameter clustering. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 166–174. SIAM, 2023.

[276] Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *STOC*, 2020.

[277] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. *arXiv preprint arXiv:1104.1377*, 2011.

[278] Joel Rybicki and Jukka Suomela. Exact bounds for distributed graph colouring. In *International Colloquium on Structural Information and Communication Complexity*, pages 46–60. Springer, 2015.

[279] Stefan Schmid and Jukka Suomela. Exploiting locality in distributed sdn control. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, page 121–126, New York, NY,

USA, 2013. Association for Computing Machinery. ISBN 9781450321785. doi: 10.1145/2491185.2491198. URL https://doi.org/10.1145/2491185.2491198.

[280] Christian Schmidt, Nils-Eric Guenther, and Lenka Zdeborová. Circular coloring of random graphs: statistical physics investigation. *Journal of Statistical Mechanics: Theory and Experiment*, 2016(8):083303, 2016.

[281] Johannes Schneider and Roger Wattenhofer. A new technique for distributed symmetry breaking. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, page 257–266, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605588889. doi: 10.1145/1835698.1835760. URL https://doi.org/10.1145/1835698.1835760.

[282] James B. Shearer. On a problem of Spencer. *Combinatorica*, 5(3):241–245, 1985.

[283] Yaroslav Shitov. Counterexamples to Hedetniemi's conjecture. *Ann. of Math. (2)*, 190(2):663–667, 2019. ISSN 0003-486X. doi: 10.4007/annals.2019.190.2.6. URL https://doi.org/10.4007/annals.2019.190.2.6.

[284] Joel Spencer. Asymptotic lower bounds for ramsey functions. *Discrete Mathematics*, 20:69–76, 1977. doi: 10.1016/0012-365X(77)90044-9. URL https://doi.org/10.1016/0012-365X(77)90044-9.

[285] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90. ACM, 2004. doi: 10.1145/1007352.1007372. URL https://doi.org/10.1145/1007352.1007372.

[286] Yinon Spinka. Finitely dependent processes are finitary. *Ann. Probab.*, 48(4): 2088–2117, 2020.

[287] Jukka Suomela. Distributed algorithms for edge dominating sets. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, page 365–374, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605588889. doi: 10.1145/1835698.1835783. URL https://doi.org/10.1145/1835698.1835783.

[288] Jukka Suomela. Survey of local algorithms. *ACM Computing Surveys*, 45(2), 2013.

[289] Suomela, Jukka. Open problems related to locality in distributed graph algorithms, 2023. URL https://jukkasuomela.fi/open/.

[290] M. Szegedy and S. Vishwanathan. Locality based graph coloring. In *Proc. 25th ACM Symp. on Theory of Computing (STOC)*, pages 201–207, 1993.

[291] Claude Tardif and Xuding Zhu. A note on Hedetniemi's conjecture, Stahl's conjecture and the Poljak-Rödl function. *arXiv preprint arXiv:1906.03748*, 2019.

[292] Ádám Timár. Tree and grid factors for general point processes. *Electron. Comm. Probab.*, 9(53–59), 2004.

[293] Ádám Timár. Invariant colorings of random planar maps. *Ergodic Theory Dynam. Systems*, 31(2):549–562, 2011.

[294] Stevo Todorčević and Zoltán Vidnyánszky. A complexity problem for Borel graphs. *Invent. Math.*, 226:225–249, 2021.

[295] Frederick A. (Frederick Albert) Valentine. *Convex sets.* Huntington, N.Y. : R. E. Krieger Pub. Co, 1975. ISBN 0882752898. Reprint of the ed. published by McGraw-Hill in series: McGraw-Hill series in higher mathematics.

[296] Nicholas C. Wormald. Models of random regular graphs. *London Mathematical Society Lecture Note Series*, pages 239–298, 1999.

[297] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 225–234, 2009.

[298] Xuding Zhu. Note on Hedetniemi's conjecture and the Poljak-Rödl function. *2019-20 MATRIX Annals*, pages 499–511, 2021.

[299] Goran Zuzic, Goramoz Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. Universally-optimal distributed shortest paths and transshipment via graph-based l1-oblivious routing. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022.