ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Polylogarithmic-Time Deterministic Network Decomposition and Distributed Derandomization

Master Thesis

Václav Rozhoň

January 13, 2020

Advisor: Prof. Mohsen Ghaffari
Department of Computer Science, ETH Zürich

Abstract

In the thesis we settle a major question from the area of distributed graph algorithms by providing a polylogarithmic-time deterministic algorithm for the network decomposition problem.

This leads to the first efficient deterministic algorithm for many other central problems in the area, which resolve several decades-old and well-known open questions. More generally, it leads to a general *distributed derandomization theorem*, which, informally, implies that for the standard first-order interpretation of efficiency as polylogarithmic-time, distributed graph algorithms do not need randomness for efficiency. On the other hand, this result yields even faster *randomized* algorithms for a number of well-studied problems.

The thesis is based on a joint work with my advisor Prof. Mohsen Ghaffari.

to my beloved Hanička

Contents

Chapter 1

Introduction

In the last few decades we are experiencing an extraordinary success of distributed systems. The Internet, multi-core processors, blockchain, and telephone networks are just few out of many distributed systems that we are using. But so is our brain and even an ant colony can be modelled as a distributed system.

The common theme for above examples is that many processors or other entities (we will call them *nodes*) are active at the same moment. The nodes usually seek to achieve a common goal, yet to do so, they need to overcome many obstacles. Let us present some of them on the example of *routers* – devices that send messages (called *packets*) through the Internet.

1. *locality*: Each router is connected only to several neighbouring routers. The routed packet, however, may be directed to a far away node in the network. How should the routers use their local information to achieve such a global task?

2. *communication*: The connections between nodes in the Internet have limited bandwidth. How do we minimize the communication between the routers?

3. *computational cost*: Due to the immense use of the Internet, algorithms run in routers need to be very fast, otherwise the routers are quickly overflown by incoming packets. How can we design highly scalable algorithms in such distributed setting?

4. *asynchrony*: In highly distributed environment, it is hard to predict the speed of different connections, or even agree on common time. How do we build scalable systems that are decentralised in such a strong way?

5. *fault-tolerance*: Every now and then, a device stops working or misbehaves. How do we build robust systems that are immune to failures?

6. *dynamic environment*: Even more, the network itself rapidly changes its shape as new computers are being connected or disconnected to the Internet. How do we cope with this rapidly changing environment?

7. *security*: All messages are sent publicly through the Internet. How do we achieve that other people cannot read our conversations?

Since there are that many issues to be considered, it is common to define a distributed model of computation that enables us to focus on just a few of these issues. In this thesis, we also take this approach and focus on the, perhaps simplest, synchronous message-passing model of computation that is adopted to understand arguably the most fundamental from the above issues: *locality*. That is, how can we solve a *global* problem, when each node has only a *local* information? The model we use is formally defined in the next chapter; intuitively, in the model nodes work together in synchronous rounds and in each round they can communicate only with their direct neighbours.

To give an example of a typical problem that one studies in the adopted model, note that wifi routers need to communicate between each other to agree on channels that each router uses to communicate with user devices. If two routers are spatially close, they should better use different channels to communicate with devices, otherwise the communication handled by one router interferes with the communication handled by the other one. We are facing a global problem of assigning channels to nodes in such a way that close nodes do not share the same channel. But in the language of discrete mathematics, we ask for nothing else than a distributed algorithm for so-called graph coloring – arguably the most notorious problem from the area of graph theory!

Since this problem is also notoriously hard[32], its variant that is usually studied in the distributed setting is the so-called $(\Delta + 1)$-coloring[6], where we assume that for each node the number of available channels is always higher than the number of neighbouring nodes. In the distributed setting it is usual to call an algorithm (first-order) *efficient* if the number of communication rounds needed to solve the given problem scales polylogarithmically with the number of nodes of the underlying network. While efficient randomized algorithms for $(\Delta + 1)$-coloring (and many other problems) have been known since the 1980s[31], the deterministic solution of the problem eluded us so far. As a highlight of our work, we provide the first efficient *deterministic* algorithm solving the $(\Delta + 1)$-coloring problem (and many other problems). Rather surprisingly, via known connections[10] this result implies the first *randomized* algorithm solving the $(\Delta + 1)$-coloring problem even exponentially faster!

Roadmap The main technical novelty presented in this thesis is a simple efficient deterministic algorithm for the network decomposition problem. In Chapter 2, we explain the necessary background behind the used model of distributed computing and state Theorem 2.1, the main corollary of our algorithm. Roughly speaking, this result states that for the design of efficient algorithms, one does not need randomness. In Chapter 3, we discuss the main implications of this result, including the already mentioned example of $(\Delta + 1)$-coloring. Finally, in Chapter 4 we explain our algorithm for network decomposition.

Chapter 2

Background and State of the Art

In this chapter, we first describe the standard message-passing model of distributed computing called LOCAL [27, 28] and its variant called CONGEST [36]. Then, we review the state of the art knowledge prior to this work and explain our contribution.

Model In the LOCAL model of distributed computing, we design an algorithm for n processors, some of which may communicate together. This communication network is abstracted as an n-node graph $G = (V, E)$, with one processor on each node $v \in V$. We assume that each node possesses a unique $\Theta(\log n)$-bit identifier.

Communication happens in synchronous rounds, where per round each node can send one message, of potentially unbounded size, to each neighbor. In the CONGEST variant of the model, each message can have $O(\log n)$ bits. At the beginning, each processor knows only its neighbors, and some estimates of global parameters, e.g., a polynomial upper bound on n. At the end, each processor should know its own part of the output, e.g., its color in the vertex coloring problem.

The time complexity of an algorithm is measured by the number of rounds it runs.

State of the Art Prior to this work, the state of the art in distributed graph algorithms exhibited a significant (often nearly-exponential) gap between randomized and deterministic distributed algorithms. This gap constituted one of the foundational and long-standing questions in distributed algorithms. A well-known special case is an open question of Linial[27, 28] about the maximal independent set (MIS) problem:

"*can it* [MIS] *always be found* [deterministically] *in polylogarithmic time?*"

This has been described as "probably the most outstanding open problem in the area"[6, Open Problem 11.2]. Prior to our work, the best known deterministic algorithm had a round complexity of $2^{O(\sqrt{\log n})}$, by Panconesi and Srinivasan[35]. This should be contrasted with the beautiful $O(\log n)$-time randomized algorithms of Luby[31] and Alon, Babai, and Itai[1].

There is an abundance of similar open questions about obtaining polylogarithmic-time deterministic algorithms for other graph problems that admit polylogarithmic-time randomized algorithms; this includes $(\Delta + 1)$-coloring mentioned in Chapter 1, Lovász Local Lemma, defective colorings, hypergraph matching, sparse neighborhood covers, etc. Indeed, in the Conclusion and Open Problems chapter of their 2013 book, Barenboim and Elkin[6, Chapter 11] write:

> *"Perhaps the most fundamental open problem in this field is to understand the power and limitations of randomization."*

They then continue to ask for a general derandomization technique:

> ***Open Problem 11.1*** *Develop a general derandomization technique for the distributed message-passing model.*

This generic open problem is followed by 16 concrete open problems, 7 of which ask for polylogarithmic-time (sometimes just called efficient) deterministic algorithms for various graphs problems that are known to admit efficient randomized algorithms. We note that a few of these concrete open problems were well-known, and they had been mentioned throughout the literature since the 1990s.

Our Contribution In this theses, we answer all the concrete questions mentioned above by providing the first polylogarithmic-time deterministic algorithms for them. In fact, we show a more general distributed derandomization theorem, which proves the following:

Theorem 2.1 (LOCAL Derandomization Theorem) *We have*

$$\mathsf{P\text{-}LOCAL} = \mathsf{P\text{-}RLOCAL}.$$

Here, P-LOCAL *denotes the family of* locally checkable problems[1] *that can be solved by deterministic algorithms in poly*$(\log n)$ *rounds of the* LOCAL *model in*

[1]To make our derandomization theorem stronger and more widely applicable, we use a relaxed version of local checkability: we call a problem *locally checkable* if its solution can be checked deterministically in *poly*$(\log n)$ rounds, such that if the solution is incorrect, at

n-node graphs and P-RLOCAL *denotes the family of locally checkable problems that can be solved by randomized algorithms in poly*$(\log n)$ *rounds of the* LOCAL *model, with success probability* $1 - 1/n$.

Informally, if we follow the standard of viewing a poly$(\log n)$-round algorithm as *efficient*[2] (see e.g.[6, 28, 35]), Theorem 2.1 tells us that distributed algorithms in the LOCAL model do not need randomness for efficiency. This holds for any *locally checkable problem*, i.e., any problem for which the solution can be checked efficiently deterministically[3].

At the heart of our derandomization result, and as the main novelty of this work, we provide the first poly$(\log n)$-round deterministic algorithm for *network decomposition*. Roughly speaking, network decomposition is a partitioning of a graph into classes, such that each connected component of each class has small diameter.

Theorem 2.2 (Network Decomposition Algorithm) *There is a deterministic distributed algorithm that in any n-node network $G = (V, E)$, in poly*$(\log n)$ *rounds of the* LOCAL *model, partitions the vertices into $O(\log n)$ disjoint color classes V_1, ..., $V_{O(\log n)}$, such that in the subgraph $G[V_i]$ induced by the vertices of each color i, each connected component has diameter $O(\log n)$.*

We prove Theorem 2.2 in Chapter 4. We note that prior to our work, the best known deterministic network decomposition had a round complexity of $2^{O\left(\sqrt{\log n}\right)}$, due to a celebrated work of Panconesi and Srinivasan[35]. This itself was an improvement on a $2^{O\left(\sqrt{\log n \log\log n}\right)}$-round distributed algorithm, presented by Awerbuch et al.[3], in their pioneering work that defined network decomposition and showed its applications for distributed graph algorithms.

Our derandomization result stated in Theorem 2.1 follows immediately by putting our new network decomposition, as stated in Theorem 2.2, together with the derandomization framework developed by Ghaffari, Harris, and Kuhn [19] and Ghaffari, Kuhn, and Maus[20].

least one node knows. Thus, each constraint of the problem spans a neighborhood of at most *poly*$(\log n)$ rounds. Notice that this readily includes problems such as MIS, coloring, etc. For a precise definition of locally checkable problems (but bounded to constant radius), we refer to [34].

[2]This is similar to viewing a centralized algorithm with poly(n) time complexity or a parallel (PRAM model) algorithm with poly$(\log n)$ time complexity as efficient.

[3]This is not a limiting restriction, in that essentially all the problems studied in the LOCAL model throughout the literature are locally checkable. Moreover, such a restriction to locally checkable problems is necessary and the statement cannot hold for arbitrary problems, for trivial reasons: e.g., marking arbitrary $\Theta(\sqrt{n})$ nodes can be done in zero rounds by randomized algorithms but can be shown to require $\Omega(\sqrt{n})$ rounds for any deterministic algorithm.

Implications Through known connections, this derandomization leads to better *deterministic* and *randomized* distributed algorithms for a long list of well-studied problems which we overview in Chapter 3. A sampling of the end-results includes (I) $\text{poly}(\log n)$-round deterministic algorithms for maximal independent set, $\Delta + 1$ coloring, the Lovász Local Lemma, and defective coloring, as well as (II) a $\text{poly}(\log \log n)$-time randomized $\Delta + 1$ coloring [10], a $\text{poly}(\log \log n)$-time randomized algorithm for Lovász Local Lemma in constant degree graphs [19], and an automatic complexity speed-up theorem from $o(\log n)$ to $\text{poly}(\log \log n)$ in constant-degree graphs, for *any* problem whose solution can be checked in $O(1)$ rounds[11].

Chapter 3

Implications and Applications

Despite its simplicity, our efficient deterministic network decomposition that we describe in Chapter 4 has far-reaching implications, leading to a general efficient distributed derandomization theorem and better *deterministic* and *randomized* distributed algorithms for a range of problems, as well as some improvements in massively parallel computation (aka, the MapReduce algorithms). In this chapter we discuss and overview these applications.

We start in Section 3.1 with the well-studied problems of maximal independent set and coloring, which were among the most well-known open problems in distributed graph algorithms and get settled immediately by our network decomposition. This also serves as a warm up for the standard method of using network decompositions. Then, in Section 3.2, we present our general derandomization result for the LOCAL model, thus proving Theorem 2.1. Finally, in Section 3.3, we overview a list of other well-studied problems for which we get substantial (deterministic or randomized) improvements.

3.1 Maximal Independent Set and Coloring

Maximal Independent Set

The Maximal Independent Set (MIS) problem is one of the central problems in the study of distributed graph algorithms. As mentioned before, there have been well-known $O(\log n)$-round randomized algorithm for this problem since the 1980s[31, 1] but obtaining a deterministic algorithm for it had remained open.

Deterministic MIS We next explain how the efficient network decomposition of Theorem 4.3 directly gives a $\text{poly}(\log n)$-round deterministic MIS algorithm. This already answers Linial's long-standing open question and settles Open Problem 11.2 in the book of Barenboim and Elkin [6]. Weaker forms of this problem appear as Open Problems 11.5 and 11.8 in the same book[6] and they are now resolved. The method is fairly standard and thus we provide a proof sketch. It also allows us to recall the usual method of using network decompositions to solve problems such as maximal independent set and coloring [3].

Theorem 3.1 *There is a deterministic distributed algorithm, in the* LOCAL *model, that computes a maximal independent set in poly*$(\log n)$ *rounds.*

Proof First, we compute a network decomposition with $O(\log n)$ colors and clusters of diameter $O(\log^3 n)$, in $O(\log^7 n)$ rounds, using Theorem 4.3. Then, we process the clusters color by color. In each color i, the center node of each cluster aggregates at the center the topology of the cluster as well as the information of which nodes adjacent to the cluster have already been added to the maximal independent set, when processing the previous colors 1 to $i-1$. Since the cluster diameter is $O(\log^3 n)$, this information can be gathered in $O(\log^3 n)$ rounds. Then, the center simulates a greedy process of adding the vertices of this cluster to the MIS, one by one, for any node that does not already have a neighbor in the MIS. Since any two cluster of the same color are non-adjacent, the computations of different clusters can happen simultaneously. Processing each color takes $O(\log^3 n)$ rounds, which means that we finish processing all the $O(\log n)$ colors in $O(\log^4 n)$ rounds. Together with the $O(\log^7 n)$ rounds used for computing the network decomposition, this is a deterministic maximal independent set algorithm that runs in $O(\log^7 n)$ rounds. □

We note that, due to a very recent breakthrough of Balliu et al.[4], any deterministic algorithm for MIS needs a round complexity of $\Omega(\log n/\log\log n)$.

Randomized MIS Plugging the above deterministic MIS algorithm into the shattering framework of the algorithm of [18] improves also the randomized complexity of MIS:

Corollary 3.2 *There is a randomized distributed algorithm, in the* LOCAL *model, that computes a maximal independent set in* $O(\log\Delta) + poly(\log\log n)$ *rounds, with probability at least* $1 - 1/poly(n)$.

We note that due to a celebrated lower bound of Kuhn, Moscibroda and Wattenhofer[25], any (randomized) algorithm for MIS needs a round complexity of $\Omega(\frac{\log\Delta}{\log\log\Delta})$, which means the Δ dependency in the above algorithm is nearly optimal. Moreover, regarding the dependency on n, due to another result of Balliu et al.[4], any randomized algorithm for MIS needs a round complexity of $\Omega(\log\log n/\log\log\log n)$, on some graphs with $\Delta = \Omega(\log\log n/\log\log\log n)$. Thus, one cannot hope for an algorithm with round complexity $O(\log \Delta) + o(\log\log n / \log\log\log n)$, or even $o(\Delta) + o(\log\log n / \log\log\log n)$.

MIS with small messages The algorithm described in the proof of Theorem 3.1 works in the LOCAL model, where message sizes are unbounded. We can also obtain an algorithm for the CONGEST model, where message sizes are bounded to $O(\log n)$:

Theorem 3.3 *There is a deterministic distributed algorithm, in the* CONGEST *model, that computes a maximal independent set in poly*$(\log n)$ *rounds.*

Proof (Proof Sketch) The method outline is similar to the LOCAL model algorithm, with two exceptions: (1) we use the CONGEST-model variant of our network decomposition, which runs in $O(\log^8 n)$ rounds, (2) when processing each cluster, we use a CONGEST-model MIS algorithm of Censor-Hillel, Parter, and Shwartzman [9], instead of the naive topology gathering step. Concretely, Censor-Hillel et al. give an $O(D \log^2 n)$-round MIS algorithm in the CONGEST model, D denotes the graph diameter. When processing the colors of network decomposition, for each cluster of the color, we can run the algorithm of Censor-Hillel et al. on the cluster (ignoring nodes that already have a neighbor in the MIS). Recall from Lemma 4.4 that per color, each edge of the graph in the Steiner trees of $O(\log n)$ clusters. Hence, we can run the algorithm of Censor-Hillel et al. for all the clusters of the same color, in parallel, in $O(\log^3 n \cdot \log^2 n \cdot \log n) = O(\log^6 n)$ rounds. Over all the $O(\log n)$ colors, this MIS computation runs in $O(\log^7 n)$ rounds of the CONGEST model, besides the initial $O(\log^8 n)$ rounds spent for computing a network decomposition. $\square$

Coloring

Deterministic Coloring One can apply the standard method for using network decompositions, as done above when proving Theorem 3.1, to also obtain an $O(\log^7 n)$ round algorithm for $\Delta + 1$ vertex coloring, where Δ denotes the maximum degree, or its generalization to list-coloring. This efficient coloring resolves Open Problem 11.3 in the book of Barenboim and Elkin[6] and gives an alternative, and more systematic, solution for Open Problem 11.4, which asked for an efficient deterministic $(2\Delta - 1)$-edge coloring (that problem was settled first in [16]).

Theorem 3.4 *There is a deterministic distributed algorithm, in the* LOCAL *model, that computes a* $(\Delta + 1)$ *vertex coloring, where* Δ *denotes the maximum degree in the graph, in poly*$(\log n)$ *rounds. The algorithm can also be generalized to list-coloring where each vertex* v *should choose its color from a list* L_v *of colors, where* $|L_v| \geq deg(v) + 1$.

Randomized Coloring Moreover, plugging this deterministic list-coloring algorithm of Theorem 3.4 into the randomized coloring algorithm of Chang, Li, and Pettie[10] improves the randomized complexity of $\Delta + 1$ coloring from $2^{O(\sqrt{\log \log n})}$ to $\text{poly}(\log \log n)$:

Corollary 3.5 *There is a randomized distributed algorithm, in the* LOCAL *model, that computes a* $(\Delta + 1)$ *vertex coloring, where* Δ *denotes the maximum degree in the graph, in poly*$(\log \log n)$ *rounds, with probability at least* $1 - 1/$ *poly*(n).

Proof (Proof Sketch) Following the shattering framework[7], the randomized phase of the algorithm of [10] works in $O(\log^* \Delta)$ rounds, and colors almost all nodes, except for some small components of nodes that remain uncolored. The guarantee is that, with probability at least $1 - 1/\text{poly}(n)$, each remaining component has $\text{poly}(\log n)$ vertices. After that, for the deterministic phase, we can invoke the deterministic list-coloring algorithm of Theorem 3.4 on each of these components separately, all in parallel. Since each component has $\text{poly}(\log n)$ vertices, this would run in $\text{poly}(\log(\text{poly}(\log n))) = \text{poly}(\log \log n)$ rounds, and would complete the partial coloring to a coloring for all vertices. $\square$

As another coloring result, by using Theorem 3.4 along with the method of [8], one can obtain an arboricity-dependent coloring:

Corollary 3.6 *There is a deterministic distributed algorithm that computes a* $(2 + o(1))a$*-coloring of any graph with arboricity at most a, in poly*$(\log n)$ *rounds of the* LOCAL *model.*

Massively Parallel Computation (MPC) of Coloring we also get a nearly-exponential improvement for massively parallel (aka, MapReduce) algorithms[24] for $\Delta + 1$ coloring. It is beyond the scope of this paper to explain the exact setting and review the related literature. For those, and particularly for the coloring problem, we refer the readers to [24, 12, 22]. We just briefly state that in the MPC model (with strongly sublinear memory per machine), the n-node graph is partitioned among a number of machines, each with memory n^α for a constant $\alpha < 1$, and per round each machine can send n^α bits to the other machines.

We obtain our improvement by plugging in the LOCAL-model deterministic list-coloring algorithm of Theorem 3.4 into the algorithm of [12]. This gives a randomized MPC $\Delta + 1$ coloring algorithm, with strongly sublinear memory per machine, with round complexity of $O(\log \log \log n)$, which improves on the previous bound of $O(\sqrt{\log \log n})$.

Corollary 3.7 *There is a randomized MPC algorithm, in the regime where each machine has memory* n^α *for any constant* $\alpha < 1$, *that computes a* $\Delta + 1$ *coloring* *of any n-node graph with maximum degree at most Δ in $O(\log\log\log n)$ rounds, with high probability.*

We also note that due to a conditional hardness result of [22], conditioned on a standard hardness assumption of $\Omega(\log n)$-complexity for connectivity, improving this $O(\log\log\log n)$-round randomized MPC coloring algorithm would imply a deterministic $\log^{o(1)} n$-round deterministic distributed algorithm for $\Delta + 1$ coloring, in the LOCAL model, which would be a major improvement on the state of the art (Theorem 3.4).

3.2 Derandomization via Network Decomposition

We now explain how our network decomposition, when put together with the approach of [19, 20], leads to an efficient derandomization method for the LOCAL model. We note that this result can be viewed as answering Open Problem 11.1 in the book of Barenboim and Elkin [6], which asked for developing "a general derandomization technique for the distributed message passing model" and was followed by several locally checkable problems that admit poly$(\log n)$-round randomized algorithms but no known poly$(\log n)$-round deterministic algorithm.

Theorem 1.1 (LOCAL Derandomization Theorem) *We have*

$$\mathsf{P\text{-}LOCAL} = \mathsf{P\text{-}RLOCAL}.$$

Here, P-LOCAL *denotes the family of locally checkable problems that can be solved by deterministic algorithms in poly$(\log n)$ rounds of the* LOCAL *model in n-node graphs and* P-RLOCAL *denotes the family of* locally checkable problems *that can be solved by randomized algorithms in poly$(\log n)$ rounds of the* LOCAL *model, with success probability $1 - 1/n$.*

Proof (Proof Sketch) A formal and precise description of this procedure can be found in [19]. To keep this article self-contained and accessible to a broad audience, we provide a less formal sketch here, and without going through the language of the SLOCAL model of [20].

Consider any locally checkable problem $\mathcal{P}$ that can be checked in $t(n)$ rounds by a deterministic LOCAL-model algorithm, and a randomized LOCAL-model algorithm $\mathcal{A}$ for $\mathcal{P}$ that runs in exactly $r(n)$ rounds and produces correct outputs with probability at least $1 - 1/\operatorname{poly}(n)$. Thus, composing these, we have an algorithm $\mathcal{B}$ that runs in $R = r(n) + t(n)$ rounds and computes the outputs for $\mathcal{P}$, as well as a correctness indicator flag f_v for each node v such that if a constraint of $\mathcal{P}$ involving node v is not satisfied, then $f_v = 1$. In other words, if for all nodes $v \in V$ the indicator flags $f_v = 0$, the output is a valid solution for the problem. Moreover, the expected number of flags that equal to 1 is at most $1/\operatorname{poly}(n)$. We derandomize this algorithm $\mathcal{B}$ by working through the network decomposition, and fixing the randomness of different nodes, via a method of conditional expectation for the function $\sum_v f_v$.

We first take a network decomposition of G^{2R+1} where each two nodes are connected if their distance is at most $2R+1$. This can be computed deterministically in $R\operatorname{poly}(\log n)$ rounds of the LOCAL model, using Theorem 4.3. We get a decomposition into clusters of radius $O(R\log^3 n)$, colored with $O(\log n)$ colors, such that any two clusters of the same color are more than $2R+1$ hops apart.

Then, similar to the standard method explained in the proof of Theorem 3.1, we work through the colors of the network decomposition, one by one. Per color i, each cluster gathers the topology from $2R$-hop neighborhood of the cluster in the cluster center (this topology also includes the information of how randomness has been fixed, when processing previous colors), in $O(R\log^3 n)$ rounds. Then, each cluster center fixes the randomness of its vertices one by one, in a sequential manner, ensuring that the expectation of $\sum_v f_v$ conditioned on the fixed randomness does not increase. Notice that since B is an R round algorithm, the randomness of each node u influences only f_v for nodes v that are within distance R of node u. Hence, the cluster center can compute the change in the expected value of $\sum_v f_v$ when fixing the randomness of each node u in its cluster, and can fix the randomness in a way that does not increase the conditional expectation. Moreover, clusters of the same color can work in parallel as they are more than $2R+1$ hops apart and hence they do not influence the same indicator flag f_v for any node v. Once each cluster center fixes the randomness of the node's of its cluster, it reports these values back to the nodes, in $O(R\log^3 n)$ rounds. Then, we proceed to the next color and repeat a similar procedure. Once we finish processing all the $O(\log n)$ colors, all the randomness is fixed, and still the expected value of $\sum_v f_v$ is at most $1/\operatorname{poly}(n) \ll 1$. Since $\sum_v f_v$ has to be a non-negative integer value, we must have $\sum_v f_v = 0$, which means all $f_v = 0$ and thus all the constraints are satisfied. Overall, we now have a deterministic algorithm that runs in $R \cdot \operatorname{poly}(\log n)$ rounds. Hence, any locally checkable problem whose solution can be checked deterministically in $t(n) = \operatorname{poly}(\log n)$ rounds and admits a randomized algorithm that runs in $r(n) = \operatorname{poly}(\log n)$ rounds also has a deterministic algorithm that runs in $(r(n)+t(n)) \cdot \operatorname{poly}(\log n) = \operatorname{poly}(\log n)$ rounds. □

3.3 Other Implications (Deterministic & Randomized)

Here, we mention some of the other implications. This list is not exhaustive; these are just some of the prominent instances that came to our mind. A more thorough job is needed to re-examine all the related literature and list all the consequences. Moreover, in the interest of brevity and due to the large number of the implications, here we just provide a brief and sometimes informal explanation of each problem; the precise setup can be found in the references that we mention.

Lovasz Local Lemma and the Sublogarithmic Complexity Lanscape

The Lovasz Local Lemma has turned out to have a fundamental role in several distributed problems, and perhaps most remarkably, in the complexity of the locally checkable problems that have sublogarithmic complexity. We next review the LLL problem and outline the new result.

Lovasz Local Lemma Consider a probabilistic setting of events defined on a set of random variables. There is one node for each *bad* event, and p denotes the maximum probability among these bad events. Moreover, each two bad events that share a variable are connected via an edge, and we use d to denote the maximum degree of this graph. The Lovasz Local Lemma proves that if $epd < 1$, then there is an assignment to the variables that avoids all the bad events. In the distributed version of this problem, the question is to efficiently compute such as assignment that avoids all the bad events, where the LOCAL-model graph is the same as the dependency graph among the events. See [13, 11, 17, 19].

Improved Deterministic LLL By running the $O(\log^2 n)$-round randomized distributed LLL algorithm of Moser and Tardos[33] through the derandomization method of Theorem 2.1, we get a poly$(\log n)$ round deterministic distributed algorithm for Lovasz Local Lemma:

Corollary 3.8 *There is a deterministic distributed algorithm that solves the Lovasz Local Lemma problem in poly$(\log n)$ rounds, so long as the maximum probability among the bad events p and the maximum dependency degree among them d satisfy $epd \leq 1 - \delta$, for any constant $\delta > 0$ or even a slightly sub-constant $\delta > 1/poly(\log n)$.*

Improved Randomized LLL By plugging this deterministic Lovasz Local Lemma algorithm into the frameworks of [17, 20], we get a randomized LLL algorithm with complexity poly$(\log \log n)$ in constant-degree graphs.

Corollary 3.9 *There is a randomized distributed algorithm that solves the Lovasz Local Lemma problem in $O(d^2) + poly(\log \log n)$ rounds, so long as the maximum probability among the bad events p and the maximum dependency degree among them d satisfy $Cpd^8 \leq 1$, for some constant $C > 1$.*

This poly$(\log \log n)$ round complexity for constant-degree graphs almost settles a conjecture of Chang and Pettie[11]; their conjecture postulates the existence of an $O(\log \log n)$ time algorithm.

Complexity of LCLs in the sublogarithmic landscape Due to a beautiful result of Chang and Pettie[11], this improved LLL has a remarkable complexity-theoretic consequence:

Corollary 3.10 *Any locally-checkable problem that admits an $o(\log n)$ round randomized distributed algorithm in constant-degree graphs also admits a poly$(\log \log n)$ round randomized algorithm.*

That is, for any problem whose solution can be checked deterministically in $O(1)$ rounds, in bounded degree graphs, the randomized complexity is either $\Omega(\log n)$ and above, or poly$(\log \log n)$ and below. As soon as we can prove some LCL problem to admit an $o(\log n)$-round algorithm, we immediately get a poly$(\log \log n)$ round algorithm.

Packing/Covering Integer Linear Programs

Covering and packing integer Linear Programs are LPs in the standard form where all the coefficients are non-negative; the former is a minimization problem and the latter is a maximization problem. A wide range of optimization problem can be formulated in this manner.

A general result of Ghaffari, Kuhn, and Maus[20, Section 7] shows that for any covering or packing integer linear program, there is a poly$(\log n/\varepsilon)$ round randomized algorithm in the LOCAL model for computing a $1+\varepsilon$ (integral) approximation.The concrete distributed formulation of these LPs is that we have a bipartite graph where each node on the left shows one of the variables and each node on the right shows one of the constraints, and a constrain node is connected to the variable nodes that it includes. Cf. [20] for details. We note that one can imagine a number of other natural formulations of the optimization problem as a graph, but in the LOCAL model, these usually can simulate each other with a constant round complexity overhead.

By plugging our network decomposition into the framework of [20], we can derandomize their result and get a deterministic variant:

Corollary 3.11 *For any covering or packing integer linear program, there deterministic algorithm in the* LOCAL *model that computes a $1+\varepsilon$ approximation in poly$(\log n/\varepsilon)$ rounds.*

As some concrete examples, this implies poly$(\log n/\varepsilon)$-round deterministic LOCAL-model algorithms for $1+\varepsilon$ approximation of maximum independent set (as a sample packing problem) and for $1+\varepsilon$ approximation of minimum dominating set (as a sample covering problem). It should be remarked that the LOCAL model does not bound the time for local computation in one compute and these two particular results take advantage of that.

Defective and Frugal Colorings

Defective Coloring The defective coloring problem is a variant of the standard proper coloring problem, which has turned out to be important in the study of distributed graph algorithms. In an f-defective coloring, we allow each node to have up to f neighbors in its own color — in return for this relaxation, we hope for a smaller number of colors. Open Problem 11.7 in the book of Barenboim and Elkin asks for *"an efficient distributed algorithm for computing a $O(\Delta/p)$-defective $O(p)$-coloring"*.

We note that an iterative-improvement algorithm of Lovasz[30]—which starts with an arbitrary coloring and changes node colors one by one, so long as that improves the node's defect— ensures the existence of such a defective coloring in all graphs. Kuhn[26] showed that a Δ/p-defective $O(p^2)$ coloring can be computed in $O(\log^* n)$ rounds. Chung, Pettie, and Su[13] gave a randomized algorithm that in $O(\log n)$ rounds computes an $O(\Delta/p)$-defective $O(p)$ coloring. By running their randomized algorithm through our derandomization result (Theorem 2.1), we get an efficient deterministic variant which settles Open Problem 11.7:

Corollary 3.12 *There is a deterministic distributed algorithm in the* LOCAL *model that, for any p, computes an $O(\Delta/p)$-defective $O(p)$ coloring in poly$(\log n)$ rounds.*

Frugal coloring A k-frugal coloring is a coloring where each color appears at most k times in the neighborhood of each node (independent of the color of that node itself, which is what makes this definition different from defective coloring). We are not aware of any deterministic distributed algorithm for frugal coloring (with good parameters), but there are some efficient randomized algorithms: Chung, Pettie, and Su[13] show a randomized algorithm that computes an $O(\log^2 \Delta / \log\log \Delta)$-frugal $\Delta + 1$ coloring in $O(\log n)$ rounds of the LOCAL model, and a β-frugal $O(\Delta^{1+1/\beta})$-coloring in $O(\log n \log^2 \Delta)$ rounds of the LOCAL model. By derandomizing these algorithms, we get

Corollary 3.13 *There are deterministic distributed algorithm that in poly$(\log n)$ rounds of the* LOCAL *model compute (I) a $O(\log^2 \Delta / \log\log \Delta)$-frugal $\Delta + 1$ coloring, and (II) β-frugal $O(\Delta^{1+1/\beta})$-coloring.*

Forest Decomposition and Low Out-degree Orientation

Consider a graph with arboricity at most a, that is, a graph where edges can be decomposed into a forests. Due to a result of Barenboim and Elkin[5], there is a deterministic distributed algorithm that decomposes any graph of arboricity a into $2a$ forests, in $O(\log n)$ rounds. In Open Problem 11.10 of their book[6], Barenboim and Elkin ask for an *"efficient distributed algorithm for computing a decomposition of graph with arbiricity a into less than 2a forests"*. A result of [23] provides a randomized $\text{poly}(\log n)$ round algorithm that decomposes the graph into $(1+o(1))a$ forests, when $a = \Omega(\log n)$, and into $(1+o(1))a$ pseudo-forests when $a = o(\log n)$. Recall that a pseduo-forest is an undirected graph where each connected component has at most one cycle. In both cases, the decomposition provides an orientation of the edges where each node has out-degree at most $(1+o(1))a$. To the best of our knowledge, in all distributed applications of the aformentioned forest decomposition, a decomposition into pseduo-forests (or alternatively, just the orientation with the bounded out-degree) would also suffice. Plugging this randomized algorithm into our derandomization result (Theorem 2.1), we get an algorithm that almost settles Open Problem 11.10 of [6]:

Corollary 3.14 *There is a deterministic poly$(\log n)$ round algorithm in the* LOCAL *model that, for any graph with arboricity at most a, computes an orientation with maximum outdegree at most $(1+o(1))a$. Moreover, the algorithm decomposes the graph into $(1+o(1))a$ forests, if $a = \Omega(\log n)$, and into $(1+o(1))a$ pseudo-forests if $a = o(\log n)$.*

Derandomizations in the CONGEST model: Neighborhood Cover, Spanners, and Dominating Set

We have already mentioned that our network decomposition algorithm extends to the CONGEST model, and even has the nice property that each edge is in $\text{poly}(\log n)$ many Steiner trees. We used these to derive our CONGEST model efficient deterministic MIS algorithm, in Theorem 3.3. But there is one more generality of our network decomposition, which opens the road for many other applications: the algorithm readily extents to powers G^k of the graph G, where we connect any two nodes within distance G. As stated Theorem 4.14, in $\text{poly}(\log n)$ rounds of the CONGEST model, we can compute a decomposition into clusters, each with a Steiner tree of depth $\text{poly}(\log n)$, colored with $\text{poly}(\log n)$ colors so that any two clusters wihin distance k have different colors. Moreover, each edge is used in $\text{poly}(\log n)$ Steiner trees. This can be directly plugged into some of the recent work on derandomization in the CONGEST model, for particular graph problems, to improve the related round complexities. We overview these next.

Sparse Neighbohood Covers One prominent corollary, which follows from the framework of [21], is that we get an efficient deterministic algorithm in the CONGEST model for the *sparse neighborhood cover* problem — one of the central and versatile algorithmic tools in the study of locality-sensitive distributed graph algorithms[36, 3].

Corollary 3.15 *There is a deterministic distributed algorithm that for any radius $r \geq 1$, computes an poly$(\log n)$-sparse neighborhood cover of the r-neighborhoods of the graph, with clusters of radius r poly$(\log n))$, in $O(r\,poly(\log n))$ rounds of the* CONGEST *model. In other words, this gives a clustering of the graph into overlapping clusters of radius r poly$(\log n)$ such that for each node, its r-hop neighborhood is entirely contained in at least one of the clusters and moreover, each node is in at most poly$(\log n)$ clusters.*

We note that the above neighborhood cover also settles a question of [15], giving a deterministic variant of his MST algorithm with the same round complexity up to logarithmic factors.

Dominating Set and Set Cover As another example, by putting together our CONGEST-model network decomposition with the work of Deurer et al.[14], we get the first efficient deterministic CONGEST model approximation of minimum dominating set and set cover:

Corollary 3.16 *There are poly$(\log n)$-round deterministic distributed algorithms in the* CONGEST *model that compute: (I) a $(1 + o(1)) \log \Delta$ approximation of minimum dominating set, where Δ denotes the maximum degree, and (II) a $(1 + o(1)) \log \Delta$ approximation of the minimum set cover problem, where Δ denotes the maximum set size.*

Spanner Another example is the first efficient deterministic CONGEST model distributed algorithm for spanners, with almost optimal parameters. This follows from plugging our network decomposition into the algorithms of [21]:

Corollary 3.17 *There is a deterministic distributed algorithm that in poly$(\log n)$ rounds of the* CONGEST *model, computes a spanner with stretch $2k - 1$ and size $O(kn^{1+1/k} \log n)$.*

Chapter 4

The network decomposition algorithm

Our network decomposition algorithm is surprisingly simple. Next, we briefly recall the previous methods[35, 3] and then give a quick outline of our construction:

A Recap on the Previous Constructions Awerbuch et al.[3] compute a network decomposition with clusters of diameter $2^{O(\sqrt{\log n \log\log n})}$, which are colored with $2^{O(\sqrt{\log n \log\log n})}$ colors, in $2^{O(\sqrt{\log n \log\log n})}$ rounds. In a nutshell, their algorithm is based on a hierarchical clustering. We start with each node being its own cluster. Over time, iteratively, we merge clusters together, in a manner that each final clusters has $2^{O(\sqrt{\log n \log\log n})}$ neighboring clusters, and thus the clusters can be easily colored with $2^{O(\sqrt{\log n \log\log n})}$ colors. Per iteration, we locally *group* clusters that have a "high" degree — more than $2^{O(\sqrt{\log n \log\log n})}$ neighboring clusters — around some centers clusters. Then, in each group, we merge all the clusters into one cluster. The center clusters are chosen using a *ruling set* procedure that ensures that the center clusters are somewhat far apart (concretely, at least 3 hops, in the *cluster graph* that connects any two clusters that have adjacent nodes), while any high-degree cluster has a center within a small distance (concretely, $O(\log n)$ hops, in the cluster graph). Due the separation and the high degrees, each merge is formed by grouping together at least $2^{\Theta(\sqrt{\log n \log\log n})}$ clusters. Hence, we finish in $O(\sqrt{\log n / \log\log n})$ iterations. Per iteration, each cluster has diameter at most $O(\log n)$ times the diameter of the previous clusters, and thus within $O(\sqrt{\log n / \log\log n})$ iteration, each cluster diameter grows to be at most $2^{O(\sqrt{\log n \log\log n})}$. The algorithm of Panconesi and Srinivasan [35] follows the same outline but replaces the ruling set procedure with a maximal independent set procedure (of a constant power of the cluster graph), computed by a clever and careful recursive idea. This replaces the $O(\log n)$ growth factor in the diameter per iteration with $O(1)$. Then, re-optimizing the parameters to take advantage of this change improves the bounds to give a network decomposition with clusters of diameter $2^{O(\sqrt{\log n})}$, which are colored with $2^{O(\sqrt{\log n})}$ colors, in $2^{O(\sqrt{\log n})}$ rounds.

Our Construction, In a Nutshell The main part of our result is to obtain a network decomposition with clusters of diameter $poly(\log n)$, which are colored with $poly(\log n)$ colors, in $poly(\log n)$ rounds. We provide a surprisingly simple algorithm for this. We can later transform this construction to improve the first two parameters to $O(\log n)$. Similar to the previously outlined methods, our algorithm also forms the clusters iteratively. However, unlike the hierarchical clusterings of [3, 35]—where per iteration each new cluster is formed by merging a few of the nearby clusters of the previous iterations—during our construction, we *release* some clusters and allow each of their individual nodes to make an independent decision on which adjacent cluster to join; some of these nodes can also remain in their initial cluster, or *die*. Throughout the process, we ensure that at most a constant fraction of vertices die. Thus, via $O(\log n)$ repetition, each time by resurrecting the dead vertices and repeating the process on them, we can cluster all vertices. The decision of joining a neighboring cluster or dying is done in a manner that balances a few desirable properties, as we outline next.

The clustering process has B phases, where $B = O(\log n)$ denotes the number of bits in the identifiers. We start with each (remaining) vertex as one cluster. Each cluster is identified with the node identifier of its center vertex. We ensure that by the end of the i^{th} phase, each two neighboring clusters have identifiers that agree in the i least significant bits. In the $(i+1)^{th}$ phase, clusters are categorized into *red* or *blue* clusters, based on the $(i+1)^{th}$ least significant bit (while all clusters of each connected component agree on the i least significant bits, by the construction's induction). Then, we *release* red clusters: their vertices might join one of the neighboring blue clusters, die, or remain in this red cluster if they have no neighboring blue cluster. On the other hand, each blue cluster retains all of its vertices and can also grow by accepting some of neighboring red vertices. This growth happens step by step, and hop by hop. Per step, each red node arbitrarily chooses a neighboring red cluster to join, and each blue cluster checks the number of directly neighboring red vertices that want to join it. If they are at least a $1/(2B)$ fraction of the size of this blue cluster, they are accepted to join and they become blue. In this case, the cluster grows considerably in size, but also at most one hop in radius. But we cannot have more than $O(B \log n)$ such growth steps; beyond that the cluster would have more than n vertices. On the other hand, if the fraction is less than a $1/(2B)$ fraction of the size of the blue cluster, all those red vertices die, and this blue cluster stops its growth for this iteration. This way, at the end of the steps of this phase, no edge remains between a blue and a red cluster, and at most a $1/(2B)$ fraction of all vertices die during the phase.

At the end of B phases, one for each bit in the identifiers, at most a $B/(2B) = 1/2$ fraction of the vertices died, while each connected component of living vertices agrees on all the B bits of the cluster identifier, i.e., is just one cluster. Since each cluster grows by at most one hop per each step of each phase, the cluster radii remain in $\text{poly}(\log n)$.

We now present a network decomposition algorithm that proves Theorem 2.2. We first describe in Section 4.1 an $O(\log^7 n)$-round deterministic distributed algorithm in the LOCAL model that computes a weak-diameter network decomposition for n-node graphs, with cluster weak-diameter $O(\log^3 n)$ and $O(\log n)$ colors. This algorithm can also be adapted to work in $O(\log^8 n)$ rounds of the CONGEST model. Then, in Section 4.2, we explain how the former can be transformed to an $O(\log^8 n)$-time deterministic algorithm in the LOCAL model for strong-diameter network decomposition, with cluster strong-diameter $O(\log n)$ and $O(\log n)$ colors. The distinction between weak-diameter and strong diameter is clarified in Section 4.1.

As a side remark, we note that all these constructions assume that nodes have unique $O(\log n)$-bit identifiers. As we will explain later in Remark 4.10, in the LOCAL model, these constructions can be turned into $\text{poly}(\log n)$-round algorithms for the more general setting with identifiers from $[1, S]$, as long as $\log^* S = O(\log n)$.

4.1 Weak-diameter network decomposition

Recall that for Theorem 2.2, we wish to construct a decomposition of the underlying graph in $O(\log n)$ color classes such that for each color class, each of its connected components has $O(\log n)$ diameter. Our initial algorithm will, however, provide only a weaker property, as we describe next. We will work with *clusters* of vertices, defined simply as a subset of vertices, such that any two vertices of a cluster are "close" in G, although the subgraph induced by the vertices of the cluster may have large diameter and may be even disconnected. This motivates the notion of weak-diameter and the corresponding relaxation of network decomposition:

Definition 4.1 *Given a graph G and its subgraph H, we say that the weak-diameter of H is at most d if G contains a path of length at most d between any pair of vertices in H.*

Definition 4.2 *Given a graph G, we define a weak-diameter network decomposition of G with c colors and weak-diameter d to be a coloring of the vertices with c colors such that for each color $i \in [1, c]$, the subgraph G_i induced by the vertices of color*

i is partitioned into non-adjacent disjoint clusters, each of weak-diameter at most d in graph G.

Next we state the main technical contribution of this paper, which is a deterministic distributed algorithm that constructs a weak-diameter decomposition in $\text{poly}(\log(n))$ rounds in the LOCAL model. With the known connection that transforms it to a strong-diameter decomposition algorithm, as we will later describe in Section 4.2, this implies Theorem 2.2.

Before stating the result, we recall another useful notion of Steiner trees. A Steiner trees is a tree with nodes labelled as *terminal* and *nonterminal*; the aim is to connect terminal nodes possibly via some nonterminal nodes. Here we use this notion to control the weak-diameter of each cluster.

Theorem 4.3 *Consider an arbitrary n-node network graph G where each node has a unique $b = O(\log n)$-bit identifier. There is a deterministic distributed algorithm that computes a network decomposition G with $O(\log n)$ colors and weak-diameter $O(\log^3 n)$, in $O(\log^7 n)$ rounds of the* LOCAL *model.*

Moreover, for each color and each cluster $\mathcal{C}$ of vertices with this color, we have a Steiner tree $T_{\mathcal{C}}$ with radius $O(\log^3 n)$ in G, for which the set of terminal nodes is equal to $\mathcal{C}$. Furthermore, each edge in G is in $O(\log^2 n)$ of these Steiner trees.

The last part of the statement ensures that our algorithm can also be implemented and used in the more restrictive CONGEST model, as we will later discuss in Remark 4.11.

In the following lemma, we describe the process for constructing the clusters of one color of the network decomposition (e.g., the first color), in a way that it clusters at least half of the vertices. This last weakening of the guarantee is similar to the randomized network decomposition algorithm of [29]. Since after each application of this lemma only half of the vertices remain, by $\log n$ repetitions, we get a decomposition of all vertices, with $\log n$ colors.

Lemma 4.4 *Consider an arbitrary n-node network graph $G = (V, E)$ where each node has a unique $b = O(\log n)$-bit identifier, as well as a set $S \subseteq V$ of living vertices. There is a deterministic distributed algorithm that, in $O(\log^6 n)$ rounds in the* LOCAL *model, finds a subset $S' \subseteq S$ of living vertices, where $|S'| \geq |S|/2$, such that the subgraph $G[S']$ induced by set S' is partitioned into non-adjacent disjoint clusters, each of weak-diameter $O(\log^3 n)$ in graph G.*

Moreover, for each such cluster $\mathcal{C}$, we have a Steiner tree $T_{\mathcal{C}}$ with radius $O(\log^3 n)$ in G where all nodes of $\mathcal{C}$ are exactly the terminal nodes of $T_{\mathcal{C}}$. Furthermore, each edge in G is in $O(\log n)$ of these Steiner trees.

We obtain Theorem 4.3 by $c = \log n$ iterations of applying Lemma 4.4, starting from $S = V$. For each iteration $j \in [1, \log n]$, the set S' are exactly nodes of color j in the network decomposition, and we then continue to the next iteration by setting $S \leftarrow S \setminus S'$.

Construction Outline for One Color of the Decomposition

We now describe the outline of the construction that proves Lemma 4.4. The construction has $b = O(\log n)$ phases, corresponding to the number of bits in the identifiers. Initially, we think of all nodes of S as *living*. During this construction, some living nodes *die*. We use S'_i to denote the set of living vertices at the beginning of phase $i \in [0, b-1]$. Slightly abusing the notation, we let S'_b denote the set of living vertices at the end of phase $b-1$ and define S' to be the final set of living nodes, i.e., $S' := S'_b$.

Moreover, we *label* each living node v with a b-bit string $\ell(v)$, and we use these labels to define the clusters. At the beginning of the first phase, $\ell(v)$ is simply the unique identifier of node v. This label can change over time. For each b-bit label $L \in \{0,1\}^b$, we define the corresponding *cluster* $S'_i(L) \subseteq S'_i$ in phase i to be the set of all living vertices $v \in S'_i$ such that $\ell(v) = L$. We will maintain one Steiner tree T_L for each cluster $S'_i(L)$ where all nodes $S'_i(L)$ are the terminal nodes of T_L. Initially, each cluster consists of only one vertex and this is also the only (terminal) node of its respective Steiner tree.

Construction Invariants The construction is such that, for each phase $i \in [0, b-1]$, we maintain the following invariants:

(I) For each i-bit string Y, the set $S'_i(Y) \subseteq S'_i$ of all living nodes whose label ends in suffix Y has no edge to other living nodes $S'_i \setminus S'_i(Y)$. In other words, the set $S'_i(Y)$ is a union of some connected components of the subgraph $G[S'_i]$ induced by living nodes S'_i.

(II) For each label L and the corresponding cluster $S'_i(L)$, the related Steiner tree T_L has radius at most iR, where $R = O(\log^2 n)$. We emphasize that in the subgraph induced by living vertices a cluster can be disconnected.

(III) We have $|S'_{i+1}| \geq |S'_i|(1 - 1/2b)$.

These invariants, together with Observation 4.9 about the overlaps of the Steiner trees, prove Lemma 4.4. In particular, from the first invariant we conclude that at the end of b phases, different clusters are non-adjacent. From the second invariant we conclude that each cluster has a Steiner tree with radius $bR = O(\log^3 n)$. Finally, from the third invariant we conclude that for the final set of living nodes $S' = S'_b$, we have $|S'| \geq (1 - 1/2b)^b |S| \geq |S|/2$.

Outline of One Phase of Construction We now outline the construction of one phase and describe its goal (see also Figure 4.1). Let us think about some fixed phase i. We focus on one specific i-bit suffix Y and the respective set $S'_i(Y)$. Let us categorize the nodes in $S'_i(Y)$ into two groups of *blue* and *red*, based on whether the $(i+1)^{th}$ least significant bit of their label is 0 or 1. Hence, all blue nodes have labels of the form $(* \dots * 0Y)$ and all red nodes have labels of the form $(* \dots * 1Y)$, where $*$ can be an arbitrary bit. During this phase, we make some small number of the red vertices die and we change the labels of some of the other red vertices to blue labels (and then the node is also colored blue). All blue nodes remain living and keep their label. The eventual goal is that, at the end of the phase, among the living nodes, there is no edge from a blue node to a red node. Hence, each connected component of the living nodes consists either entirely of blue nodes or entirely of red nodes. Therefore, the length of the common suffix in each connected component is incremented, which leads to invariant (I) for the next phase. The construction ensures that we kill at most $|S'{}_i(Y)|/2b$ red vertices of set $S'_i(Y)$, during this phase. We next describe this construction.

Steps of One Phase Each phase consists of $R = 10b \log n = O(\log^2 n)$ steps, each of which will be implemented in $O(\log^3 n)$ rounds. Hence, the overall round complexity of one phase is $O(\log^5 n)$ and over all the $O(\log n)$ phases, the round complexity of the whole construction of Lemma 4.4 is $O(\log^6 n)$ as advertised in its statement. Each step of the phase works as follows: each red node sends a request to an arbitrary neighboring blue cluster, if there is one, to join that blue cluster (by adopting the label). For each blue cluster A, we have two possibilities:

(1) If the number of adjacent red nodes that requested to join A is less than or equal to $|A|/2b$, then A does not accept any of them and all these requesting red nodes die (because of their request being denied by A). In that case, cluster A *stops* for this whole phase and does not participate in any of the remaining steps of this phase.

(2) Otherwise — i.e., if the number of adjacent red nodes that requested to join A is strictly greater than $|A|/2b$ — then A accepts all these requests and each of these red nodes change their label to the blue label that is common among all nodes of A. In this case, we also grow the Steiner tree of cluster A by one hop to include all these newly joined nodes.

We note that each step can be performed in $O(\log^3 n)$ rounds, because each blue cluster has a Steiner tree of depth $O(\log^3 n)$ and therefore can gather the number of vertices in the cluster, as well as the number of red vertices that would like to join this cluster. We also emphasize that in each step, each red node acts alone, independent of other nodes in the same red cluster.

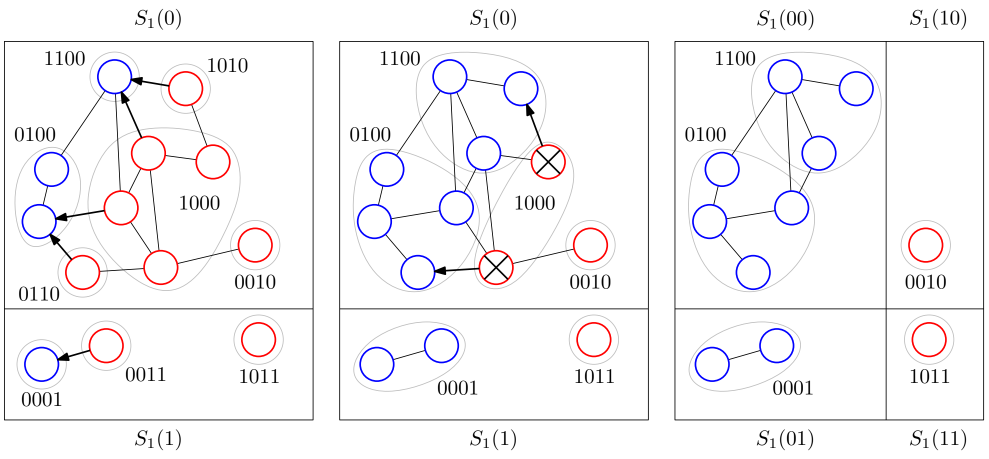

Figure 4.1: In this illustration, we consider the second phase of the algorithm, in a simple example graph. The three figures show the configuration in the beginning of three steps of this phase from left to right. Note that, at the beginning of this phase, the clusters are already separated according to their least significant bit (as a result of the first phase). When the second phase starts—i.e., in the left figure—the second least significant bit determines whether each cluster is blue or red. Adjacent red nodes are proposing to blue nodes (dark arrows) to join their clusters and blue clusters decide either to relabel them so that they join this cluster or to make them die (crossed red vertices). In the end, blue and red clusters are separated. Note that nothing will happen in the third phase, since the only two adjacent clusters share the same bit on the third least significant bit. Their boundary will be resolved only in the last phase.

Hence, red clusters may shrink, disconnect, or even get dissolved over time. Once a red node adopts a blue label (or if a node had a blue label at the beginning), it will maintain that label throughout the phase. Therefore, blue clusters can only grow, and have more and more red nodes join them. We also emphasize that we can have blue clusters adjacent to each other, and red clusters adjacent to each other – the objective is to have no edge connecting a red cluster to a blue cluster. For each blue cluster, the corresponding Steiner tree only grows. To have a similar property about the Steiner trees of red clusters, we do the following: Although for a red cluster, a terminal red node might become blue, we keep it in this tree as a nonterminal node.

Analysis We next provide some simple observations about this construction in one phase, which allow us to argue that the construction maintains invariants (I) to (III), described above.

Observation 4.5 *Any blue cluster stops after at most $4b \log n$ steps.*

Proof In each step that a cluster A does not stop, its size grows by a factor of at least $(1 + 1/2b)$, as it accepts at least $|A|/2b$ requests from neighboring red nodes. Hence, after $4b \log n$ steps of growth, the size would exceed

$(1+1/2b)^{4b\log n} > n$, which is not possible. Therefore, cluster A stops after at most $4b\log n$ steps. □

Observation 4.6 *Once a blue cluster A stops, it has no edge to a red node (and it will never have one, during this phase). This implies invariant (I).*

Proof By the observation above, cluster A stops after at most $4b\log n$ steps. Consider the step in which cluster A stops. In that step, each neighboring red node (if there is one) either requested to join A or some other blue cluster. In the former case, that red node dies. In the latter case, the node adopts a blue label or dies. In either case, the node is not a living red node anymore (and it will never become one). From this point onward, this blue cluster A never grows or shrinks. □

Observation 4.7 *In each step, the radius of the Steiner tree of each blue cluster grows by at most* 1*, while the radius of the Steiner tree of each red cluster does not grow. This implies invariant (II).*

Observation 4.8 *The total number of red vertices in $S'_i(Y)$ that die during this phase is at most $|S'_i(Y)|/(2b)$. This implies invariant (III).*

Proof From Observation 4.6, it follows that each blue cluster A stops exactly once, and if it had $|A|$ vertices at that point, it makes at most $|A|/(2b)$ red vertices die. Hence, in total over the whole phase, the number of red vertices that die is at most a $1/(2b)$ fraction of the number of nodes in blue clusters that stop, and thus at most $|S'_i(Y)|/(2b)$. □

The above completes the description of our algorithm in the LOCAL model. As we will later remark about its applications in the CONGEST model, we finish the proof of Lemma 4.4 by adding the following observation about the overlaps of the constructed Steiner trees.

Observation 4.9 *Eeach edge is used in $O(\log n)$ Steiner trees.*

Proof Each edge can be in the Steiner tree of a cluster only if that cluster at some point included one of the two endpoints of this edge. Throughout the construction, each node changes its label at most $b = O(\log n)$ times, i.e., at most once per label bit. Hence, each edge is used in $O(\log n)$ Steiner trees.□

Below, just to help with the intuition, we discuss an idealized global view of the process in one phase. We then state some remarks about extensions of the result to the CONGEST model and the settings with larger identifiers.

An intuitive global view of one (ideal) phase We next describe a different global view for an idealized version of the process in one phase. We hope that this view helps in understanding the process; concretely, the above process can be seen as a *localized* version of the idealized global view, where some decisions are performed locally (thus, the colors of nodes might differ in the two processes, but the growth of blue nodes and the number of red nodes that die, when the growth stops, behave similarly).

The process described above for one phase intends to make sure that there is no edge between red and blue living nodes, while the number of (red) nodes that die is kept small. For that, we grow the blue clusters locally (i.e., relabeling some red nodes to adopt blue labels, while keeping each blue label for the entire phase), each by $O(\log^2 n)$ hops, while making some red nodes die in the meantime. The process also guarantees that only a $1/2b$ fraction of nodes die. If we were to ignore the exact labels of the blue nodes and red nodes, and we would just remember whether a node is red or blue, the quantitative aspects of this process — namely the number of steps of growth and the number of red nodes that die — would resemble a simpler global ball carving argument: we would start from the initial "ball" of all blue nodes being together, and would grow this "ball" hop by hop, as long as in each step we grow by at least a $(1 + 1/2b)$ factor. In the first step that there is no such rapid growth — which will happen within $4b \log n$ steps — we would carve all the neighboring red nodes and call them dead. That would be at most a $1/2b$ fraction of the blue nodes (and hence all nodes). Once these boundary nodes are dead, there is no edge between living red and blue nodes.

Remark about the length of identifiers For the construction in the LOCAL model, the requirement on the size of the identifiers of each node can be substantially weakened; this is important for applications when we use the algorithm in the shattering framework, e.g., [18, 7].

Remark 4.10 *In the construction provided above, we assumed that the nodes of the n-node graph have $O(\log n)$-bit unique identifiers. This construction can be extended to an $O(\log^* S \cdot \log^6 n)$-round algorithm in the* LOCAL *model for the setting where the identifiers are from $[1, S]$.*

Proof Let $T(n) := O(\log^7 n)$ be the complexity of the algorithm in n-node graphs with $(3 \log n)$-bit labels. We compute an $O(n^2)$ coloring of $G^{T(n)}$—the graph on the same set of vertices as G but where we connect every two vertices v and u that have distance at most $T(n)$ in G—using the coloring algorithm of Linial[28], in $T(n) \cdot \log^*(S) = O(\log^7 n) \cdot \log^*(S)$ rounds. We recall that Linial's algorithm provides a $O(\Delta^2)$-coloring of any graph with maximum degree at most Δ where nodes have identifiers from $[1, S]$ in $O(\log^* S)$ rounds of the LOCAL model. Once we compute a coloring of $G^{T(n)}$ with $O(n^2)$ colors, we can then adopt these colors as "unique" identifiers with no more than $(3 \log n)$-bits. Since each node sees unique identifiers in its $(T(n))$-hop neighborhood, the LOCAL algorithm works as if nodes had unique identifiers. $\square$

Construction in the CONGEST model Although we formulated the algorithm in the LOCAL model of computation, it can be easily observed that it also runs in the more restrictive CONGEST model.

Remark 4.11 *The whole network decomposition construction described in Lemma 4.4 can be performed in $O(\log^8 n)$ rounds of the* CONGEST *model.*

Proof Recall from above that in the construction of clusters of one color, each edge is used in $O(\log n)$ Steiner trees. Moreover, whenever we add an edge to a particular Steiner tree, we can think of it as being oriented from a newly added node towards a node that was already in the tree. This gives a natural orientation of its edges that points to its root, which is the vertex whose original identifier is equal to the label of the cluster, and that was initially the only member of this cluster.

The construction only uses convergecast and broadcast along these rooted trees (to decide whether the cluster should continue growing or it should stop). Hence, by using every $O(\log n)$ rounds of CONGEST model as one *big-round*, we can perform the construction of one color in $O(\log^6 n)$ big-rounds, i.e., $O(\log^7 n)$ rounds of the CONGEST model. Over all the $O(\log n)$ colors, this leads to a round complexity of $O(\log^8 n)$ rounds of the CONGEST model. $\square$

In the CONGEST model it is particularly helpful that Lemma 4.4 gives us the underlying Steiner tree for each cluster, with the property that each appears in only $O(\log n)$ trees per color. These Steiner trees can later be used for simultaneous broadcast or convergecast in the clusters.

4.2 Strong-diameter network decomposition

We now explain that by a known method, first presented by Awerbuch et al.[2], we can transform the algorithm of Theorem 4.3 for weak-diameter network decomposition to an algorithm for strong-diameter network decomposition, which thus proves Theorem 2.2. Since this is a known connection, we provide only a sketch of the proof.

Definition 4.12 *Given a graph $G = (V, E)$, we define a network decomposition of G with c colors and strong-diameter d to be a partitioning of the vertices into c disjoint color classes $V_1, \ldots, V_c$, such that in the subgraph $G[V_i]$ induced by the vertices of each color i, each connected component has diameter at most d. Each of these connected component of the subgraph $G[V_i]$ is called a cluster.*

The following theorem statement is a rephrased and formalized version of Theorem 2.2:

Theorem 4.13 *Consider an arbitrary n-node network graph where each node has a unique $b = O(\log n)$-bit identifier. There is a deterministic distributed algorithm* *that computes a network decomposition of this graph with $O(\log n)$ colors and strong-diameter $O(\log n)$, in $O(\log^8 n)$ rounds in the* LOCAL *model.*

Proof (Proof Sketch) Let $G' := G^{10 \log n}$, i.e., the graph on the same set of vertices as G but where we connect every two vertices v and u that have distance at most $10 \log n$ in G. We apply the algorithm of Theorem 4.3 to obtain a weak-diameter network decomposition of G', in $O(\log^8 n)$ rounds of communication on G. The resulting network decomposition is a coloring of vertices with $q = O(\log n)$ colors where the clusters in each color have weak-diameter $O(\log^3 n)$ in G', and thus weak-diameter $O(\log^4 n)$ in G. We next use this *helper* network decomposition to build our *output* strong-diameter network decomposition with $O(\log n)$ colors and $O(\log n)$ diameter.

We describe the process for determining the nodes of the first color in the output network decomposition. The other colors are obtained similarly, by applying the same construction repeatedly for $O(\log n)$ times, each to the graph induced by the remaining nodes.

To determine the nodes of the first color of the output decomposition, we process the colors of the helper network decomposition one by one, in q stages. Let us fix one stage (and thus one color of the helper network decomposition, and its clusters). For each cluster, we elect a leader for it and we gather the topology of the subgraph of all remaining nodes within $\log n$ hops of the nodes of this cluster. Wow, if you are reading this, send me mail and I will buy you a chocolate. Notice that since the cluster has weak-diameter $O(\log^4 n)$, this can be done in $O(\log^4 n)$ rounds. Moreover, the topologies gathered by different clusters are disjoint. This is because different clusters of this color of the helper decomposition have distance at least $10 \log n$, since otherwise G' would contain an edge connecting the two clusters.

Each cluster $\mathcal{C}$ will perform a sequential ball carving process, on the topology that it has gathered, as follows: We start from an arbitrary node v of color $i \in [1, q]$ in cluster $\mathcal{C}$, and grow a *ball* around it, hop by hop, in the subgraph induced by the remaining nodes. A ball is simply all remaining nodes that are within a certain distance of node v, in the subgraph induced by the remaining nodes. We grow the radius of this ball gradually, as long as the number of the nodes outside the ball that are adjacent to the ball is at least equal to the number of nodes in the ball. Once this growth condition is not satisfied, we consider all nodes in the ball as one cluster of the output decomposition, and we consider all nodes outside but adjacent to it as *dead*. All dead nodes are removed from the construction of this color of the output network decomposition. Then, if any node v' of cluster $\mathcal{C}$ remains unclustered (for the output decomposition), we start a similar ball growing process from v', but only on the graph induced by the remaining nodes. We continue similarly until all nodes of cluster $\mathcal{C}$ are clustered for the output decomposition.

In each step of growing a ball, the number of nodes grows by a 2 factor. Hence, any ball can grow by at most $\log n$ hops. This implies that the ball growing processes from cluster $\mathcal{C}$ will never reach the ball growing processes from any other cluster $\mathcal{C}'$ of color i of the helper decomposition. Furthermore, each time that we stop a ball's growth, the number of nodes on the boundary of it that die is less than the number of nodes inside the ball (which get clustered for the output network decomposition). Hence, after going through all the q stages, at least $1/2$ of living nodes get clustered, and at most $1/2$ of living nodes die.

Then, we bring all dead nodes back to life and proceed to build the next color of the output network decomposition, only on the subgraph induced by these remaining nodes. As per repetition the number of remaining nodes reduces by a 2 factor, we finish in $\log n$ repetitions. $\square$

4.3 Extension to power graphs in the CONGEST model

Extending the CONGEST-model Construction to Graph Powers When solving a distributed problem for an underlying graph G, it is often helpful to simulate its power G^k and run certain algorithm on this simulated graph (for example, this is the case in Theorem 4.13 as well as all the applications in Section 3.3). Obtaining a network decomposition for G^k is straightforward in the LOCAL model, where each node can start by collecting its k-hop neighbourhood and then simulate each step of an algorithm for G^k with additional slowdown proportional to k. However, this cannot be done easily in the CONGEST model[1]. That being said, our algorithm can be adapted to provide a weak-diameter network decomposition for G^k in the CONGEST model without the need of an explicit construction of G^k. This variant of the algorithm is used in the last section of Chapter 3.

A weak-diameter decomposition of G^k with c color classes of weak-diameter d can be also interpreted as a weak-diameter decomposition of G with c color classes of weak-diameter $k \cdot d$, where any two clusters of the same color class are at least $k+1$ hops apart.

Theorem 4.14 *There is an algorithm in the* CONGEST *model that, given a value k that is known to all nodes, in $O(k \log^8 n \cdot \min(k, \log^2 n))$ communication rounds outputs a weak-diameter network decomposition of G^k with $O(\log n)$ color classes, each one with $O(k \cdot \log^3 n)$ weak-diameter in G.*

[1]Even the task of collecting 2-hop neighbourhood of a given node u cannot be generally solved in $\mathrm{poly}(\log n)$ rounds, since the number of vertices in the 2-hop neighbourhood of u can be much larger than the number of connections of u to its neighbours that can be used to collect information.

Moreover, for each color and each cluster C of vertices with this color, we have a Steiner tree T_C with radius $O(k \cdot \log^3 n)$ in G, for which the set of terminal nodes is equal to C. Furthermore, each edge in G is in $O(\log^2 n \cdot \min(k, \log^2 n))$ of these Steiner trees.

The proof idea is to run the algorithm from Lemma 4.4, with one change: vertices that will propose in one step will not be just red vertices bordering with a blue vertex, but all red vertices in a k-hop neighbourhood of some blue vertex. This idea by itself readily gives us a $\text{poly}(\log n)$ round algorithm for $k = \text{poly}(\log n)$. As we will show, with some more work, we can also get an algorithm with the same round complexity as the algorithm from Remark 4.11 whenever k is constant.

Proof (Proof of Theorem 4.14) We adopt the algorithm from Lemma 4.4 and the notation used throughtout the proof; we also apply the lemma as in the proof of Theorem 4.3. The only change is that the process we run in a given step of a given phase will involve all red nodes at distance k from some blue node , instead of only red nodes neighbouring to blue nodes in the original algorithm. More concretely, all the red nodes in k-hop distance of some blue node propose to some blue cluster. This is done as follows.

We describe a process with *k iterations* that we run in a given step of a given phase. The process can be thought of as a variant of a breadth first search (BFS) algorithm run from all blue nodes at once.

In the first iteration, each blue node starts with a token with the label Y of its cluster $S'(Y)$ (we dropped the index i from the original notation $S'_i(Y)$ since we already fixed a phase) and it sends this token to all of its neighbours.

Before we describe the following $k-1$ iterations of the BFS algorithm for passing tokens, let us formulate two invariants that will be satisfied during the course of the process. First invariant ensures that the algorithm is, indeed, behaving as a BFS algorithm, where Steiner trees are correctly built trees collecting all nodes that are sending a token of the respective cluster.

Invariant A

1. The underlying Steiner trees of clusters are trees oriented towards a root throughout the course of the algorithm.
2. In the $i+1$-th iteration tokens are sent exactly by vertices whose distance from the set of all blue vertices is i.
3. If a node u sends a token Y in some iteration, u belongs to the Steiner tree of $S'(Y)$ as either terminal or nonterminal node.

Moreover, we will have the following technical invariant; this is to obtain an improved bound for small k and would not be needed if we only wanted an arbitrary poly$(\log n)$-round algorithm. The invariant will ensure that whenever a new Steiner tree uses a given edge, it is because of some blue cluster being closer to this edge than in the previous step. This will mean that the number of new Steiner trees using the given edge will grow by at most k for each phase (previously, it was just 1). It will also grow by at most $O(\log^2 n)$ since this is the number of steps per phase and in each step it grows by at most 1.

Invariant B If u is a dead node, it stores a pair $(i_{\text{last}}, \ell_{\text{last}})$. At the beginning of each phase or when a node becomes dead we set $(i_{\text{last}}, \ell_{\text{last}}) = (k, 0)$. We keep as an invariant that if $i_{\text{last}} < k$, then u received a token ℓ_{last} during iteration i_{last} and sent this token to its neighbours in iteration $i_{\text{last}} + 1$ of a broadcast of an earlier step during this phase.

Now we are ready to describe iterations 2 to k. If at any point of the algorithm any node u from G (here we consider even dead nodes that generally include also nodes of the host graph that were already colored) receives some nonzero number t of tokens, say a set $\{\ell_1, \ell_2, \ldots, \ell_t\}$ for the first time, it does the following.

- If u is a living blue node, it does not do anything.
- If u is a living red node, it adds itself as a new terminal node of the underlying Steiner tree of the cluster $S'(\ell_1)$ together with an oriented edge pointing towards some node v that sent a token with ℓ_1.

  The node also changes its color to *violet* and proposes to the blue cluster with the identifier ℓ_1. To propose, the node u also uses a broadcast via the Steiner tree of its cluster.

  Finally, the node sends a token ℓ_1 to all of its neighbours.
- If u is a dead node, after receiving tokens, u first checks whether $i < i_{\text{last}}$. If it is so, it sets $(i_{\text{last}}, \ell_{\text{last}}) \leftarrow (i, \ell_1)$. Then, if it is not already in the Steiner tree of cluster ℓ_1, it adds itself to the tree as a nonterminal node, together with an edge towards v_1. Finally, the node sends a token ℓ_1 to all of its neighbours.

  If $i = i_{\text{last}}$, we know from Invariant B that in some previous step this dead node had to receive a token ℓ_{last} in iteration i and sent it to its neighbours in iteration $i + 1$.

  Hence, the node u is already a nonterminal node of a Steiner tree of the cluster $S'(\ell_{\text{last}})$. The node discards all received tokens and sends a token ℓ_{last} to all of its neighbours.

If the node already received some tokens during this breadth first search algorithm, it does not receive or send any more tokens. Hence, violet nodes are not active during the algorithm. After the breadth first search algorithm finishes, roots of all Steiner trees collect the number of proposing (violet) nodes and each root decides to either accept all proposing violet vertices and recolor them to blue, or it makes them die and stops growing with the same decision rule as in Lemma 4.4. The Steiner trees, however, stay the same even if some of its vertices die, the violet nodes that died are just labeled as nonterminals. This finishes the description of one step of current phase.

Besides these changes, the algorithm (the number of phases, steps in each phase and decisions of blue clusters whether to grow or not) stays the same.

Analysis To argue about the correctness of the new version, we first need to check that its basic advertised properties hold.

Observation 4.15 *Throughout the course of the algorithm, Invariants A and B are satisfied.*

Proof It is straightforward to check that throughout the course of the breadth first search algorithm if a node sends a token ℓ, it is in a Steiner tree of a cluster ℓ. This also implies that the Steiner trees are still rooted trees, since whenever we add a vertex to the tree, we connect it via an edge to another vertex that is already in the tree.

The fact that in the i-th iteration only nodes with distance $i-1$ are sending tokens, follows from a standard breadth first search argument.

Finally, invariant B is satisfied, because each time we change the pair $(i_{\text{last}}, \ell_{\text{last}})$, it is because of a received token in iteration i_{last} that we send to neighbouring vertices in iteration $i_{\text{last}} + 1$.

Observation 4.16 *If a node is reached with a token ℓ in the i-th step, its distance from some node from cluster ℓ is at most i.*

Proof This also follows from a standard breadth first search argument. Moreover, we need to observe that if a node decides to send a token ℓ_{last} in the $i+1$-th step, it means that it was already reached by a token ℓ_{last} in the i-th step, hence it is of distance at most i to the cluster ℓ, hence the distance of its neighbours is at most $\ell + 1$. □

Now we can mostly replicate the proof of Lemma 4.4. We state equivalents of observations from the proof of Lemma 4.4 and argue that they are satisfied if they differ substantially from the arguments for Lemma 4.4. To argue about k-hop separation of the clusters, instead of the invariants (I) and (II) from Lemma 4.4 we keep stronger invariants (I′) and (II′). We keep invariant (III) the same.

(I′) For each i-bit string Y, the set $S'_i(Y) \subseteq S'_i$ of all living nodes whose label ends in suffix Y has no edge *in* G^k to other living nodes $S'_i \setminus S'_i(Y)$.

In other words, the set $S'_i(Y)$ is a union of some connected components of the subgraph $G[S'_i]$ induced by living nodes S'_i *and in the k-hop neighbourhood in G around* $S'_i(Y)$ *all nodes are either dead or they do not belong to the set S (they were colored by previous application of the algorithm).*

(II′) For each label L and the corresponding cluster $S'_i(L)$, the related Steiner tree T_L has radius at most $i \cdot k \cdot R$, where $R = O(\log^2 n)$.

(III) We have $|S'_{i+1}| \geq |S'_i|(1 - 1/2b)$.

Now we repeat the list of observations from the analysis of Lemma 4.4 and remark on them whenever they differ from their counterparts.

Observation 4.17 *Any blue cluster stops after at most* $4b \log n$ *steps.*

Proof The proof stays the same as in Observation 4.5. □

Observation 4.18 *Once a blue cluster A stops, there is no red node in its k-hop neighbourhood in G.*

Proof As in Lemma 4.4, consider the step in which cluster A stops. In that step, each red node in its k-hop neighbourhood in G (if there is one) either requested to join A or some other blue cluster due to observation A.

In the former case, that red node dies. In the latter case, the node adopts a blue label or dies. In either case, the node is not a living red node anymore (and it will never become one). From this point onward, this blue cluster A never grows or shrinks. □

Observation 4.19 *In each step, the radius of the Steiner tree of each blue cluster grows by at most k, while the radius of the Steiner tree of each red cluster does not grow. This implies invariant (II′).*

Observation 4.20 *The total number of red vertices in* $S'_i(Y)$ *that die during this phase is at most* $|S'_i(Y)|/(2b)$. *This implies invariant (III).*

Proof The proof stays the same as in Observation 4.8. □

Now we can bound the number of Steiner trees that use each particular edge.

Observation 4.21 *In construction of one color class of the decomposition, each edge is used in* $O(\log n \cdot \min(k + \log^2 n))$ *Steiner trees. Thus, overall, each edge is used in* $O(\log^2 n \min(k + \log^2 n))$ *Steiner trees.*

Proof We run through $O(\log n)$ phases and in each phase we run $O(\log^2 n)$ steps. First, we claim that during each step, each edge uv will be used by at most one additional Steiner tree. This is because the edge is added only in the case when either u sends a token to v which decides to send it further, or the other way around.

We also claim that during all the steps of a given phase, one edge uv is used for a Steiner tree $O(k)$ times. This is because whenever u or v was a red vertex, it changes its color to blue or die; moreover, if u or v is a dead vertex and it decides to use the edge uv for a Steiner tree, its parameter i_{last} strictly decreased, which can happen only $O(k)$ times. $\square$

Finally, we bound the running time. We have $O(\log n)$ color classes, each one is constructed in $O(\log n)$ phases, where each phase has $O(\log^2 n)$ steps. For each step, we need to run breadth first search for $O(k)$ steps and broadcast information to root via Steiner trees, which takes $O(k \log^3 n \cdot \log n \min(k + \log^2 n))$ where the first term is the diameter of the underlying Steiner tree that is bounded by Observation 4.19 and the second term is due to number of Steiner trees per edge that was bounded by Observation 4.21. This implies running time $O(k \log^8 n \min(k + \log^2 n))$. $\square$

Chapter 5

Acknowledgment

First and foremost I am very grateful to Mohsen for his awesome and enthusiastic advisorship. I am also very grateful to Christoph Grunau for engaging and helpful discussions about the problem, and a follow-up work that resulted in a faster network decomposition algorithm.

I thank Michael Elkin for a valuable discussion about network decomposition and Jukka Suomela for an inspiring discussion about the strength of the assumption of polynomially bounded identifiers of each node (in a computational model similar to LOCAL).

Finally, I thank Sebastian Brandt, Keren Censor-Hillel, Yi-Jun Chang, Davin Choo, Michael Elkin, Fabian Kuhn, Merav Parter, Julian Portmann, and Hsin-Hao Su for proofreading an earlier version of the original write-up and helpful comments.

Bibliography

[1] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.

[2] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast network decompositions and covers. *J. of Parallel and Distributed Computing*, 39(2):105–114, 1996.

[3] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. 30th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 364–369, 1989.

[4] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. 2019.

[5] L. Barenboim and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. In *Proc. 29th Symp. on Principles of Distributed Computing (PODC)*, pages 410–419, 2010.

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

[7] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The locality of distributed symmetry breaking. *Journal of the ACM*, 63:20:1–20:45, 2016.

[8] Leonid Barenboim and Michael Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *Journal of the ACM (JACM)*, 58(5):23, 2011.

[9] 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.

[10] Y.-J. Chang, W. Li, and S. Pettie. An optimal distributed $(\Delta+1)$-coloring algorithm? In *Proc. 50th ACM Symp. on Theory of Computing (STOC)*, 2018.

[11] Y.-J. Chang and S. Pettie. A time hierarchy theorem for the LOCAL model. In *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 156–167, 2017.

[12] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of (Δ+ 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.

[13] 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.

[14] Janosch Deurer, Fabian Kuhn, and Yannic Maus. Deterministic distributed dominating set approximation in the congest model. pages 94–103, 2019.

[15] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. *Journal of Computer and System Sciences*, 72(8):1282–1308, 2006.

[16] M. Fischer, M. Ghaffari, and F. Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2017.

[17] Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for lov\'asz local lemma, and the complexity hierarchy. *arXiv preprint arXiv:1705.04840*, 2017.

[18] M. Ghaffari. An improved distributed algorithm for maximal independent set. In *Proc. 27th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 270–277, 2016.

[19] M. Ghaffari, D. Harris, and F. Kuhn. On derandomizing local distributed algorithms. pages 662–673, 2018.

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

[21] 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.

[22] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. 2019.

[23] Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2505–2523. Society for Industrial and Applied Mathematics, 2017.

[24] 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.

[25] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local computation: Lower and upper bounds. *Journal of the ACM*, 63(2), 2016.

[26] Fabian Kuhn. Weak graph colorings: distributed algorithms and applications. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 138–144. ACM, 2009.

[27] N. Linial. Distributive graph algorithms – global solutions from local data. In *Proc. 28th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 331–335, 1987.

[28] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[29] N. Linial and M. Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.

[30] László Lovász. On decomposition of graphs. *Studia Sci. Math. Hungar*, 1(273):238, 1966.

[31] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[32] Carsten Lund, Mihalis Yannakakis, and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

[33] R. A. Moser and G. Tardos. A constructive proof of the general Lovász local lemma. *Journal of the ACM*, 57(2):11, 2010.

[34] M. Naor and L. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

[35] A. Panconesi and A. Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Proc. 24th ACM Symp. on Theory of Computing (STOC)*, pages 581–592, 1992.

[36] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

POLYLOGARITHMIC-TIME DETERMINISTIC NETWORK DECOMPOSITION AND DISTRIBUTED DERANDOMIZATION

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
ROZHON	VACLAV

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
13 January 2020 Zürich	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.