DISS. ETH NO. 27521

Local Algorithms for Classic Graph Problems

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

Manuela Fischer
M.Sc. ETH in Computer Science

born on 14 February 1992
citizen of Switzerland

accepted on the recommendation of

Prof. Dr. Mohsen Ghaffari, examiner
Prof. Dr. Fabian Kuhn, co-examiner
Prof. Dr. David Peleg, co-examiner
Prof. Dr. Seth Pettie, co-examiner

2021

Acknowledgments

I wish to express my deepest gratitude to my advisor Mohsen Ghaffari for the enormous trust and effort he put into me by allowing me to be his first Ph.D. student. He not only provided me with plenty of scientific opportunities but also gave me the freedom to pursue my own research interests. Without his encouragement, support, guidance, and expertise, this thesis would not have been possible. His staggering enthusiasm for problem-solving and his seemingly infinite supply of knowledge have never ceased to amaze me.

I also want to thank my co-examiners Fabian Kuhn, David Peleg, and Seth Pettie for agreeing to join my thesis committee and for their invaluable feedback.

Next, I would like to thank all past and present members of DaDA[1] who made our research group feel like a big family. In particular, I want to mention Jara Uitto for making every single day at work incredibly fun; Sebastian Brandt for being the best office mate imaginable; Julian Portmann for our entertaining conversations; and Bernhard Haeupler, Christoph Grunau, Goran Zuzic, Saeed Ilchi, and Václav Rozhoň for the delightful coffee breaks.

[1]the group of Discrete and Distributed Algorithms

A very special thanks goes to the group of Emo Welzl. From the very first day, they adopted me into their group as if I belonged to them, and invited me, besides daily lunch, to all their fun activities, such as after-work beers at bQm, board game evenings, and hikes. In particular, I am greatly indebted to Emo Welzl for inspiring me with his fascination with research and for helping me to find the right path for my Ph.D.; and to Manuel Wettstein for never failing to make me laugh.

Further, I would like to thank everyone else I had the privilege of interacting with during my time at ETH, especially Andreas Noever for becoming my bouldering buddy; Ralph Keusch for joining me for runs, even before sunrise and in the pouring rain; Yannic Maus and Krzysztof Nowicki for interesting discussions during conferences and their research visits; the group of Angelika Steger for inviting me to their workshop in Buchboden twice; and Andrea Salow not only for being extremely helpful with any administrative matters but also for motivating me for swimming.

I wish to thank all people who helped me proofreading parts of this thesis. A big shout-out to Manuel Wettstein for designing the amazing cover graphic.

Last but not least, thank you to my co-authors MohammadHossein Bateni, Soheil Behnezhad, Sebastian Brandt, Yi-Jun Chang, Mahsa Derakhshan, Hossein Esfandiari, Mohsen Ghaffari, MohammadTaghi Hajiaghayi, Richard M. Karp, Fabian Kuhn, Vahab Mirrokni, Slobodan Mitrović, Andreas Noever, Nemanja Škorić, Angelika Steger, Miloš Trujić, Jara Uitto, and Yufan Zheng for many enlightening discussions and fruitful collaborations; and the teams from Google Research New York, IBM Research Europe, and the IBM T. J. Watson Research Center in New York for hosting me during my research internships.

Zürich, June 2021 *Manuela Fischer*

Contents

Abstract

Graph algorithms have been studied extensively for decades, if not centuries. A slightly more recent development, initiated by the seminal work of Linial in 1987, is the study of *local* graph algorithms: algorithms that, as opposed to centralized ones, have access only to a small part of the graph. Understanding what can and what cannot be computed locally is one of the oldest branches of distributed graph algorithms.

In this thesis, we develop novel local techniques that lead to the resolution of long-standing open questions at the heart of the theory of distributed computing with ramifications way beyond.

Part I is dedicated to the LOCAL model where there is a direct correspondence between complexity and locality—faster LOCAL algorithms are more local. We advance on the state of the art in several directions: we introduce a deterministic local rounding technique for linear programs that results in the first improvement for maximal matching in over twenty years and in the first efficient deterministic edge coloring algorithm, answering a question that goes back to the very beginning of the area; we devise a local algorithm for Lovász Local Lemma that gives rise to an exponential improvement on its predecessor; we show that a simple local greedy maximal independent set algorithm is as fast as the celebrated algorithm of Luby from 1986, confirming a wide-spread belief; and we present a strikingly simple local sampling technique that fully parallelizes its centralized counterpart.

In Part II, we demonstrate the importance of local algorithms for the global communication models Congested Clique and Massively Parallel Computation. Although in some sense they are orthogonal to LOCAL, local approaches turn out to be absolutely essential for the design of algorithms in these models. By adopting local techniques and adapting LOCAL algorithms, we settle the complexity of the vertex coloring problem in the Congested Clique model and exponentially improve on its complexity (or, alternatively, polynomially on its memory requirement) in the Massively Parallel Computation model; and we break a seemingly fundamental barrier in the Massively Parallel Computation model with the first efficient algorithms with sublinear memory for a number of classic graph problems.

Zusammenfassung

Graphalgorithmen werden seit mehreren Jahrzehnten, wenn nicht Jahrhunderten, ausgiebig studiert. Eine etwas jüngere Entwicklung, angestossen durch die bahnbrechende Arbeit von Linial in 1987, ist das Studieren von *lokalen* Graphalgorithmen: Algorithmen, die – im Gegensatz zu zentralisierten – nur auf einen kleinen lokalen Teil des Graphen Zugriff haben. Zu verstehen, was lokal berechnet werden kann und was nicht, ist eine der ältesten Fragen im Bereich von verteilten Graphalgorithmen.

In dieser Dissertation entwickeln wir neue lokale Techniken, die zur Lösung von lange offenstehenden Fragen im Herzen der Theorie von verteilten Graphalgorithmen mit Implikationen für zahlreiche andere Bereiche führen.

Teil I ist dem Modell LOCAL gewidmet, wo es eine direkte Beziehung zwischen Komplexität und Lokalität gibt: schnellere Algorithmen sind lokaler. Wir bringen den Forschungsstand in verschiedene Richtungen voran: wir führen eine deterministischte lokale Rundungstechnik für lineare Programme ein, die zu der ersten Verbesserung für das Problem *Maximal Matching* seit mehr als zwanzig Jahren und zu den ersten effizienten deterministischen Kantenfärbalgorithmen führt, was eine zum Beginn des Forschungsbereichs zurückgehende Frage positiv beantwortet; wir entwerfen einen lokalen Algorithmus für das Problem *Lovász Local Lemma*, der expoentiell schneller ist als sein Vorgänger; wir zeigen dass der einfache lokale *greedy* Algorithmus für das Problem *Maximal Independent Set* gleich schnell ist wie der berühmte Algorithmus von Luby in 1986, was eine weit-verbreitete These bestätigt; und wir präsentieren eine auffallend einfache lokale *Sampling*-Technik, die ihr zentralisiertes Gegenstück vollständig parallelisiert.

In Teil II demonstrieren wir die Bedeutung von lokalen Algorithmen für die Modelle *Congested Clique* und *Massively Parallel Computation* mit globaler Kommunikation. Obwohl diese Modelle in einem gewissen Sinne orthogonal zu LOCAL sind, stellt sich heraus, dass lokale Strategien absolut essentiell sind für das Entwerfen von effizienteren Algorithmen. Indem wir lokale Techniken übernehmen und lokale Algorithmen adaptieren, entwickeln wir einen optimalen Knotenfärbalgorithmus in *Congested Clique* und wir durchbrechen eine scheinbar fundamentale Barriere in *Massively Parallel Computation* mit dem ersten effizienten Algorithmus mit sublinearem Speicher für zahlreiche klassische Graphprobleme.

CHAPTER 1

Background

Graph problems play an extraordinarily important role in mathematics and computer science. They have been studied extensively for centuries—their origins can be traced back to the famous problem of the Königsberg bridges by Euler in 1736 [90]; they were among the first problems examined in complexity theory [42, 85, 152]; and there are numerous books devoted to the topic of graph theory and graph algorithms [154, 91, 41].

A vast majority of these classic graph algorithms, however, is *centralized*: they assume a single computing entity to have knowledge of the whole graph. Recently, there has been an increased interest in *local* algorithms, that is, algorithms that can see only a small part of the graph, yet have to come up with a solution to the whole problem. This is particularly essential when faced with massive graphs that cannot be stored and accessed in the memory of a single computer, which is becoming more and more common.

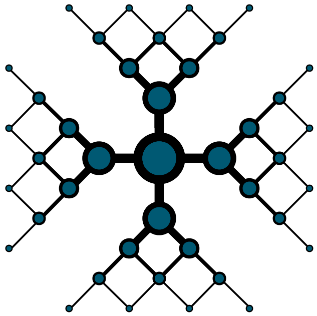


Figure 1.1: Local (as opposed to centralized) graph algorithms have access only to a small part of the input graph.

Understanding the capabilities and limitations of local approaches, hence, as Linial [171] phrased it,

> *"to what extent a global solution to a computational problem can be obtained from locally available data",*

is the key challenge and, in fact, one of the oldest branches of distributed graph algorithms.

1.1 Distributed Graph Algorithms

Complex networks composed of a multitude of autonomous entities have emerged at every scale in a variety of areas. Examples range from biological structures like neurons in a brain, insects in a colony, fishes in a school, birds in a flock, zebras in a herd, and human beings in a society to technological systems like railcars in a fleet, drones in a swarm, computers in the Internet, mobile devices in a wireless network, processors in a supercomputer, cores on a chip, and gates in a logic circuit.

Over the past decades, such distributed systems have experienced an unprecedented growth; they have become an imperative part of modern computing. There are a plethora of reasons for this development.

Firstly, trends such as digitalization, the Internet of things, and swarm robotics have led to an increased need for geographically distributed devices to communicate. From self-driving cars to activity trackers to smart contact lenses: they all have to be capable of interacting with their environment.

Secondly, cloud storage services like *Dropbox* and *Google Drive* have seen steady user growth ever since their inception. Similarly, *Google Compute Engine*, *Microsoft Azur*, and other cloud computing platforms have gained a lot of popularity. The groundbreaking innovation of crowd-sourced resource sharing and pooling has further pushed this progress. For instance, *Storj* leverages underutilized hard drive capacity and *Golem* harnesses idle machines all over the world. Such virtual supercomputers have played an important role in recent projects such as *Folding@Home* which impressively demonstrate the power of this advancement. In general, for many applications, combining cheap computers—be it crowd-sourced or not—is more beneficial from an economic perspective than deploying one expensive specialized machine.

In addition, multicore computing has become more important than ever. Moore's law has approached physical limits; the process of doubling the number of transistors on a chip has been hindered and eventually brought to an end by barriers in electron tunneling and heat extraction. To still keep up with its predicted speed increase, instead of accumulating more sequential computing power in one core, multiple cores need to be incorporated.

This is further exacerbated in today's era of Big Data, where the growth of speed and memory of a single computer is outpaced by the surge in the amount of information available. Immense data sets originating from various sources, such as gene sequences from bioinformatics and stock charts from computational finance prevalently are too massive to admit efficient centralized algorithms and to even fit into the random access memory of a single computer. In other cases, data is naturally spread across multiple machines, as it emerges in a distributed manner.

Despite the diversity of their applications, these distributed systems have one thing in common: there is no central control—they all depend on communication to establish coordination. Generally speaking, under a distributed system we understand a collection of autonomous entities that cooperate by exchanging messages. They can collaborate to achieve a shared goal or they can pursue their own interests but rely on sharing information to do so. While *time complexity* serves as a good indicator for the running time of centralized algorithms in practice, this is not accurate for distributed computing. In fact, *offline* computation—that is, computation performed locally by one entity without interacting with others—often is negligible compared to the cost incurred by communication [153]. This is conceptually very different from traditional computational complexity, and hence constitutes a fundamental change.

To address this challenge, many different computational models have been proposed, specifically tailored to distributed computing and its inherent need for communication. These models differ in how messages are exchanged—from broadcasting in radio-based systems to point-to-point in telecommunication; in how reliable they are—whether messages can be delayed, corrupted, and lost; and in how fault-tolerant and secure they are—how they handle hardware and software failures and how they deal with malicious and selfish entities.

Synchronous Distributed Message-Passing

In the area of *synchronous distributed message-passing*, a point-to-point communication network is modeled as a graph: there is a node with a unique ID for every computing entity, and an edge connecting two nodes if there is a bidirectional communication channel between them. Communication happens in rounds and is assumed to be completely synchronous and reliable—there are no clock drifts, no lost messages, and no crashes.

In this setting, an algorithm simply is a synchronous round-based protocol that specifies for every round and every node what offline computation it performs (based on knowledge it has at that point) and then what messages it sends to its neighbors. Initially, a node solely knows about its neighbors. Eventually, the solution to a problem has to be output in a distributed manner: every node only has to output a part of the solution but all these partial solutions combined have to be consistent and complete. The goal is to minimize the number of rounds needed until every node terminates. This quantity is referred to as *round complexity* or *running time*. Since the focus lies on communication, offline computation is allowed to be unbounded[1]

There are two main challenges for efficient communication in this synchronous distributed message-passing setting: *locality* and *congestion*. The challenge of locality is to deal with the shortcoming that every node only has partial information about the system's state. Due to the constraint of direct communication being restricted to adjacent nodes, there is no fast dissemination to and from far-away nodes. Accessing remote information requires intermediate nodes to transceive messages, resulting in communication costs that are quickly growing with (hop) distance. As a consequence, an

[1]Most algorithms, however, do not abuse this offline computing power; they often even run in constant time.

individual node cannot afford to gain and maintain global knowledge about the entire network; it has to base its decisions on local information only. The challenge of congestion, on the other hand, is to cope with the limited bandwidth of the network. Every edge can transfer messages only at a certain rate and every node can only hold a certain amount of information—if these limits are exceeded, delays and failures have to be expected.

To sum up, locality and congestion can be understood as orthogonal concepts characterizing the detriment in communication efficiency due to the sheer size as well as the scarce bandwidth of the network, respectively. As illustrated in Table 1.2, there are several models, all of them capturing different (combinations of) aspects of locality and congestion. To help us better understand their role as obstacles to fast distributed algorithms, it seems natural to decouple them by studying the influence of each issue separately.

At one extreme, to analyze the pure effect of congestion, and hence to disregard any impact of locality, one can allow all-to-all communication between the nodes. Since the communication graph then is a clique, and thus every piece of information is at most one hop away, any complexity can be ascribed to the issue of congestion. In particular, if the network had unlimited bandwidth and memory, any problem could be solved in a single round: every node gathers the whole graph and then computes a solution offline. The *CC (Congested Clique)* and MPC *(Massively Parallel Computation)* model edge and node congestion, respectively, if locality is not a concern.

At the other extreme, the LOCAL model can be seen as the complement of these bandwidth-restricted models. It explores the limits imposed by locality without any interference of congestion. The bandwidth and memory constraints are removed from the picture by allowing messages to be of unbounded size and nodes having unlimited memory. Fast LOCAL algorithms are local algorithms that only need to access close-by information—conversely, lower bounds can be interpreted as the need to look beyond a certain horizon.

If one additionally imposes a bound on the message sizes in the LOCAL model, one arrives at the classic CONGEST model. It is sometimes also called E-CONGEST model, since the capacity of the edges in the communication network is restricted. There is a little less well-known variant of CONGEST with node congestion, called V-CONGEST. These CONGEST models simultaneously consider locality and congestion. On the other hand, if neither locality nor congestion play a role, communication is free; we can think of this case as the classic centralized model where all computation takes place in a single node.

Note that there is a close connection and no clear-cut boundary between these distributed message-passing and *parallel computing* with models like PRAM *(Parallel Random-Access Machine)*. However, most parallel computation models allow shared memory and hence do not rely on message-passing only for communication.

1.1.1 The **LOCAL** Model

The foundations for the systematic study of local algorithms were laid by the seminal works of Linial [171] in 1987 and Naor and Stockmeyer with the pithy title 'What can be computed locally?' [195] in 1993. The significance of locality and locality-sensitivity for distributed computing is reflected by the huge and still rapidly growing body of research and monographs devoted to this theme, such as the famous book called 'Distributed Computing: A Locality-Sensitive Approach' by Peleg [211] whose plea

> *"It is necessary to develop a robust and general methodology for addressing locality in distributed algorithms."*

gets to the heart of the matter.

The LOCAL model, the standard synchronous message-passing model

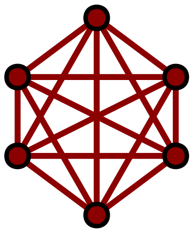
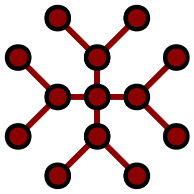
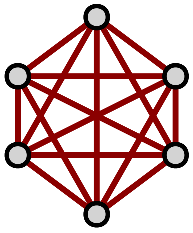
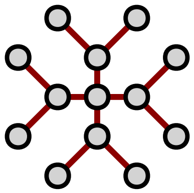
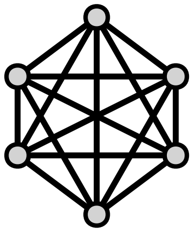
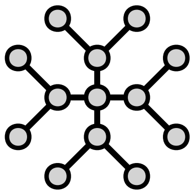
		locality	
		no	yes
congestion	node	 MPC	 V-CONGEST
	edge	 CC	 E-CONGEST
	none	 centralized	 LOCAL

Table 1.2: Different (combinations of) challenges for synchronous message-passing and one representative model each.

of distributed computing, was introduced by Linial in 1987 [171, 172] and named by Peleg [211]. It isolates the pure concept of *locality of an algorithm*: faster LOCAL algorithms are more local. In this model, graph problems are studied on the communication graph—in other words, the input graph instance is equal to the communication graph. The motivation for this is two-fold. On the one hand, as we will see in Section 1.2.1, many problems naturally arising in networks can be phrased as classic graph problems on the communication graph. On the other hand, distributed algorithms can be useful for determining the *locality of a graph problem*. Roughly speaking, the locality of a problem corresponds to its *dependency radius*, thus to the distance up to which nodes' solutions have to depend on each other, and can be bounded by the round complexity of a LOCAL algorithm. The LOCAL model thus allows to study the role of the purely graph-theoretical notion locality, the way distant nodes affect each other, and how far the effect of a node spreads by designing message-passing protocols on the input graph.

More concretely, in the LOCAL model, a problem on an input graph $G = (V, E)$ with $n = |V|$ vertices, $m = |E|$ edges, and maximum degree Δ is studied on a communication graph which is equal to G. We can think of every vertex of the input graph sitting at the respective node of the communication graph. If important for the discussion, we will (try to) distinguish between a vertex in the input graph and node as the computing entity in the communication graph. For the LOCAL model, however, these are tightly coupled. See Figure 1.3. Every node is equipped with a unique $O(\log n)$-bit ID and initially knows about its neighbors, its part of the input, as well as n and Δ.[2] The communication happens in synchronous rounds, where, per round, each node[3] can perform arbitrary computations

[2]For most settings this is without loss of generality, because if n and Δ are not known, it is enough to try exponentially increasing estimates (perform an exponential search on the parameter space) [155].

[3]Sometimes, to simplify the discussion, one describes the behavior of an edge

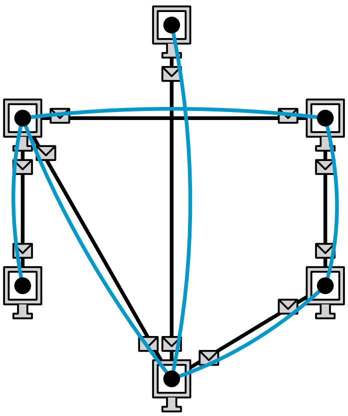


Figure 1.3: In the LOCAL model, communication graph (black) and input graph (blue) coincide. Every vertex of the input graph instance is a computing entity, hence a node in the communication graph.

and send a message of arbitrary[4] size to each of its neighbors. In the end, every node is expected to output its part of the solution, for instance the color of its vertex for a graph coloring problem, or whether its incident edges are part of the matching.

Due to the power of the LOCAL model—thanks to the relaxations of having free offline computation and unbounded message sizes—there is a one-to-one correspondence of the round complexity and the problem's locality: in r rounds, a node can learn the entire

instead of that of a node. One can think of this as having the higher-ID node of the two endpoints taking responsibility for the edge, or the two nodes discussing with each other about what happens to the edge, at the cost of a multiplicative overhead of 2 in the round complexity.

[4]A variant of the model where the messages have to be of bounded size, at most $O(\log n)$ bits, is known as the CONGEST or E-CONGEST model [211]. In fact, some of our algorithms do apply to this setting, too, but it is not the focus of this thesis.

topology of its r-hop neighborhood, but nothing beyond this radius-r-horizon. Only nodes within can affect the information available. Consequently, an r-round algorithm is mapping from a node's r-hop neighborhood to its output [172]. If the number of rounds exceeds the diameter of the graph, every node can learn about the whole graph and solve any problem offline. On the other hand, for certain problems, it is impossible to compute a solution in less rounds. For instance, one cannot decide whether a graph is acyclic without seeing the whole input. We call a problem *global* if any LOCAL algorithm requires $\Omega(n)$ rounds and *local* if it admits an efficient LOCAL solution. In general, the accepted standard for efficiency is poly $\log n$ rounds.

1.1.2 The Congested Clique Model

The CC *(Congested Clique)* model, introduced by Lotker, Patt-Shamir, Pavlov, and Peleg in 2003 [179, 175], can be seen as a complement of LOCAL: it schematically captures (edge) congestion while it disregards any impact of locality. This model plays not only an important role for understanding bandwidth-restricted systems and hence for the design of algorithms that optimally use available bandwidth, its all-to-all communication also is an increasingly common feature in many distributed settings: They appear in cloud computing, overlay networks, P2P (peer-to-peer) networks and high-performance computing.

In the CC model, the communication graph is a clique on n nodes. Every node has a unique ID of $O(\log n)$ bits. The computation proceeds in rounds; in every round, every node can perform arbitrary computations and send a (possibly different[5]) message of $O(\log n)$ bits to each of its $n-1$ neighbors. This bound on the size essentially means that every message can carry a constant number of *words*,

[5] A variant where the same message needs to be sent to all neighbors is known as *Broadcast Congested Clique*

each word for instance describing a node or an edge identifier. The complexity of interest is the number of communication rounds until the result is output, in a distributed manner.

While the model is more general, a main focus lies on graph problems where the input is a graph $G = (V, E)$ on $n = |V|$ vertices, $m = |E|$ edges, and maximum degree Δ where each node (i.e., computing entity) initially receives one vertex of the input graph and eventually has to output its vertex' solution. See Figure 1.4. However, this mapping is not binding; in principle, it is possible to rearrange the vertices in an arbitrary manner during the computation. This is in stark contrast to the LOCAL (and CONGEST) model where the communication and input graph are closely tied together.

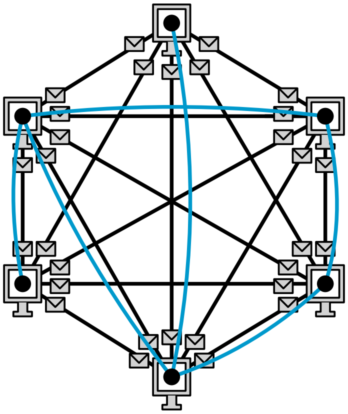


Figure 1.4: In the Congested Clique model, there is a one-to-one correspondence between the nodes in the communication graph and the vertices in the input graph. The edges in the communication graph (black), however, form a clique regardless of the edges in the problem input graph (blue).

In the CC model, $\Theta(n^2)$ messages of size $\Theta(\log n)$ can be exchanged in every round. This massively parallel communication power has been exploited to devise super fast and sometimes even constant-round algorithms [168, 150]. In fact, Drucker et al. [84] showed that substantially super-constant lower bounds on the round complexity in CC imply long-standing open and known to be notoriously hard lower bounds in circuit complexity. On the other hand, any graph problem can be solved in $O(\Delta)$ many CC rounds: each node goes through its vertex' incident edges one by one and sends them to all other nodes. After at most Δ rounds, every node has a full copy of the input graph and hence can compute a solution offline.

1.1.3 The Massively Parallel Computation Model

The ever-increasing amount of data available has caused the resource memory to become a major bottleneck for efficient algorithms. To overcome this obstacle, inspired by the MapReduce paradigm [78], several computation frameworks for performing large-scale computations across multiple machines have been proposed: *Dryad* [144], *Flume* [79], *Spark* [232], *Pregel* [182], and *Hadoop* [228], to name just a few.

The MPC *(Massively Parallel Computation)* model constitutes a clean theoretical abstraction of these frameworks and thus serves as the basis for the systematic study of memory-restricted (that is, node-congested) distributed algorithms. It goes back to works by Karloff et al. [151] and Feldman et al. [92] in 2010, was refined later in a sequence of works [126, 29, 8, 30, 73], and has become tremendously popular over the past decade. Although MPC has had its advent in parallel computing, unlike most parallel models (such as PRAM), it does not feature any shared memory and therefore relies solely on message-passing for communication.

In the MPC model, the distributed network consists of $\mathcal{M}$ nodes, each node being a computing machine with a local memory of size $\mathcal{S}$ and a unique ID of size $O(\log \mathcal{M})$. To clear up the notation and hence to simplify the presentation, the space is specified in number of words, each word consisting of $O(\log \mathcal{M})$ bits. Initially, the input is distributed arbitrarily across the nodes. The computation proceeds in synchronous rounds. In every round, every node can perform arbitrary offline computation based on the data it has stored in its memory and then exchange messages with the other nodes. As in the CC model, the communication graph is fully connected; every node is allowed to send as many messages to as many nodes as it wants, as long as for every node the total size of sent and received messages does not exceed its memory capacity S. This communication step, which is called *shuffling* in the MapReduce terminology, requires a massive data volume (up to $O(\mathcal{M}\mathcal{S})$ words) to be transferred between nodes. It is thus the main goal to keep the number of rounds as small as possible. In the end, the nodes collaboratively have to hold and output a solution.

Many different variants of this model have been studied, for different choices of the parameters $\mathcal{M}$ and $\mathcal{S}$ depending on the size $|\mathcal{I}|$ of the input instance $\mathcal{I}$. Note that $\mathcal{S} \geq |\mathcal{I}|$ leads to a degenerate case that allows for a trivial solution. Indeed, as the data fits into the local memory of a single node, the input can be loaded there, and a solution can be computed offline. Due to the targeted application of MPC in the presence of massive data sets, thus large input instances $\mathcal{I}$, it is often crucial that $\mathcal{S}$ is not only smaller than $|\mathcal{I}|$ but actually substantially sublinear in $|\mathcal{I}|$. On the other hand, the global memory, i.e., the total memory $\mathcal{M}\mathcal{S}$ in the system, has to be at least $|\mathcal{I}|$, so that the input actually fits, but usually is assumed not to be much larger. Summarized, one requires $\mathcal{S} = \widetilde{O}\left(|\mathcal{I}|^{\delta}\right)$ memory on each of the $\mathcal{M} = \widetilde{O}\left(|\mathcal{I}|^{1-\delta}\right)$ nodes, for some $0 < \delta < 1$.[6]

For graph problems, the input $\mathcal{I}$ is a graph $G = (V, E)$ with n vertices and m edges, of total size $|\mathcal{I}| = \widetilde{\Theta}(n + m)$. As opposed to the LOCAL (and somewhat also the CC model), the computation is performed by nodes that are not associated with any particular

[6]Throughout, $\widetilde{}$ in the O-notation is used to hide lower-order terms.

part of G; the vertices of the input graph are distributed arbitrarily across the computing nodes.[7] See Figure 1.5. At the end, each node should know the output of the vertices in its memory.

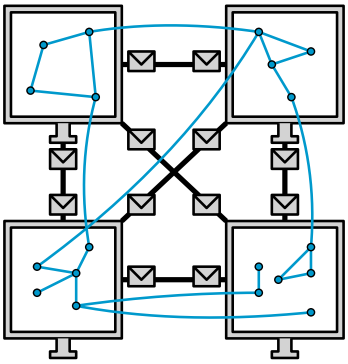


Figure 1.5: In the MPC model, the communication graph is a clique (depicted in black) on the nodes (also called machines). The vertices of the problem input graph G (depicted in blue) are distributed across the nodes arbitrarily.

Note that if $\Delta > \mathcal{S}$, and hence a vertex cannot be stored with all its incident on a single node, one has to introduce some sort of a workaround. One explicit way for many natural problems is to split high-degree vertices into many copies distributed among many nodes. For the communication between the copies, one can imagine a balanced tree of depth $1/\delta$ rooted at one of the copies. Through this tree, the copies can exchange information in $O(1/\delta)$ communication rounds. This issue is usually ignored by making the simplifying assumption that every node has $\mathcal{S} = \Omega(\Delta)$ memory.

[7]This is usually done using a hash function that then can be used to determine which node holds a vertex or a (potential) edge. We assume that the vertices are stored in the nodes in a balanced way, i.e., as long as a single vertex fits onto a single node and the total memory is not exceeded, the underlying system takes care of load balancing.

While many classic parallel (e.g., PRAM) or distributed (e.g., LOCAL) graph algorithms can be directly adopted in the MPC model in the same number of rounds using a standard simulation technique [151, 126], its additional power of free offline computation (compared to PRAM) and of global communication (compared to LOCAL) could potentially be exploited to obtain faster MPC algorithms. Accordingly, the question

> *"Are the MPC parallel round bounds "inherited" from the PRAM model tight? In particular, which problems can be solved in significantly smaller number of MPC rounds than what the [...] PRAM model suggest[s]?"*

of Czumaj et al. [73] summarizes the main goal of the area: to devise algorithms which are substantially faster than their PRAM and LOCAL counterparts. Typically, a graph algorithm in the MPC model is considered efficient if its time complexity is $\operatorname{poly}\log\log n$, which is exponentially faster than the efficiency threshold for the LOCAL model. This seems particularly reasonable in the light of recent developments: Ghaffari, Kuhn, and Uitto [121] showed that certain lower bounds from the LOCAL model can be transferred to (certain variants of) the MPC model in an exponentially scaled down version. Note that no general unconditional super-constant lower bounds are known, and, in fact, expected to be known [215, 193].

1.2 Local Graph Problems

Graph problems, and especially local graph problems, are of central importance for distributed computing, as overviewed next.

1.2.1 Graph Problems and Local Coordination

One of the major challenges faced when devising distributed algorithms is *local coordination* and *symmetry breaking*: nodes with symmetrical views of the network need to take on different roles without explicitly getting assigned one. Problems that are trivial with a bird's-eye view become hard if there is no central control. This is neatly captured by Linial's [173] analogy:

> *"It is very difficult to have many processors perform in concert when there is no conductor around."*

There should be neither a perfect unison nor full disharmony—nodes should follow their own agenda without creating chaos.

There are four classic problems, phrased as graph problems on the communication network, that serve as a main abstraction of symmetry breaking and local coordination: vertex coloring, edge coloring, maximal independent set, and maximal matching. See Figure 1.6 for an overview. Roughly speaking, they can be used to (locally optimally) schedule the nodes' protocols so that no two interdependent computations are executed at the same time. Consequently, algorithms for these problems serve as subroutine in many network algorithms.

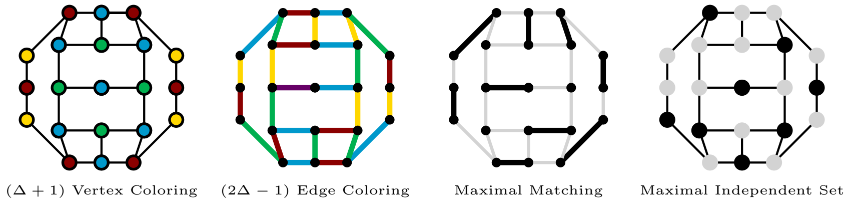


Figure 1.6: The four classic local graph problems that prototypically model symmetry breaking and local coordination.

A q *vertex coloring*—that is, an assignment of one of q colors to nodes so that no two adjacent nodes have the same color—can be seen as a schedule of length q where a node with color i is active in round i and hence no two neighboring nodes are active in the same round. Note that the unique IDs of $O(\log n)$ bits already give rise to such a coloring, but with poly n many colors. The goal is to have a coloring with as few colors as possible, hence as short schedule as possible. The minimum number of colors needed for the guaranteed existence of a proper coloring is $q = \Delta + 1$. In fact, by Brook's theorem [48], for most graphs $q = \Delta$ is sufficient. However, without a global view of the network, this is not always possible to decide. Moreover, with $\Delta + 1$ colors, a coloring can be found by a greedy sequential algorithm that goes through the vertices one by one, assigning each a still available color. In other words, any partial coloring can be extended to a proper coloring—no matter how a vertex' neighbors are colored: since they can block at most Δ colors, there is always at least one available color left—without having to reconsider previous decisions. This is a particularly handy property for distributed computing as it allows to take decision at several places simultaneously, only focusing on immediate correctness (i.e., compatibility with the direct neighborhood), without having to worry about further-reaching interferences.

The generalized problems $(\Delta + 1)$ *list vertex coloring* and $(\mathsf{deg} + 1)$ *list vertex coloring* are frequently studied as well. There, every vertex v is given an individual list $\Psi(v)$ of colors with $|\Psi(v)| = \Delta+1$ or $|\Psi(v)| = d(v) + 1$, respectively, where $d(v)$ is the degree of vertex v. The goal is to find a proper vertex coloring where each vertex v is assigned a color from $\Psi(v)$. The interest in the latter problem mainly stems from the fact that completing a coloring after some vertices have already been assigned a color can be phrased as a $(d + 1)$ list vertex coloring problem.

Similarly to vertex coloring, an *edge coloring*—an assignment of colors to edges such that no two incident edges share their color—serves as a schedule for the coordination of noiseless interactions between nodes so that no node has to interact with multiple neighbors in the same round. By Vizing's theorem [226], Δ or $\Delta + 1$ colors are enough. However, again one usually aims for the greedy (and hence conveniently parallelizable) threshold $2\Delta - 1$ (since an edge can have up to $2\Delta - 2$ incident edges) for the number of colors.

MIS *(Maximal Independent Set)* and MM *(Maximal Matching)*—that is, an inclusion-maximal set of non-adjacent nodes and non-incident edges, respectively—are variants of vertex coloring and edge coloring, respectively, where instead of a full schedule (where all nodes or all pairs of nodes get their turn) only an assignment to a single slot is planned. Again these problems are locally weakened versions: instead of looking for the biggest possible such set (that is, a *maximum* independent set or matching), one is only interested in a locally maximal one. These relaxations are useful especially when the unrelaxed problem is inherently global. Indeed, finding a maximum matching in a communication network that is as simple as a cycle will require a node to learn about the whole graph.

Note that MIS is the hardest of the four problems, as the other three can be reduced to it locally [171]. In particular, MM and $(2\Delta - 1)$ edge coloring are special cases of MIS and $(\Delta + 1)$ vertex coloring, respectively, on the line graph[8] representing the adjacencies. According to Kuhn [158],

> *"MIS computation can be seen as a Drosophila of distributed computing as it prototypically models symmetry breaking."*

This central role is also underpinned by two recent Dijkstra Prizes awarded to papers dedicated to this problem—to Linial [172] in 2013 and to Luby [181] and Alon, Babai, and Itai [5] in 2016—who call it "the crown jewel of distributed computing".

While in the centralized setting, these symmetry breaking problems admit trivial sequential greedy solutions, understanding their com-

[8]The line graph $L(G)$ of a graph $G = (V, E)$ is a graph with vertex set E and an edge between two of its vertices if the corresponding edges are incident.

plexity in the LOCAL model is an open question which goes back to the very beginning of the area [171]; it thus arguably is one of the ultimate goals of distributed computing.

1.2.2 Lovász Local Lemma and LCL Problems

The LLL *(Lovász Local Lemma)* is a beautiful result by Erdős and Lovász in 1975 [88] that has become an indispensable tool for the probabilistic method [7] when proving that certain combinatorial objects exist. It can be seen as a generalization of the following well-known fact: if each of a set of "bad" events has probability less than 1 and all those events are independent, then there is a positive probability that none of the bad events occur. In other words, there exists a configuration to avoid all bad events. The Lovász Local Lemma allows to relax the condition of independence to extend the applicability to events with sparse dependencies.

Although the LLL applies to general probability spaces and general notions of dependency, a simpler setting is used in most applications. In its easiest form, the probability space is defined by a set $\mathcal{X}$ of boolean variables, each $X \in \mathcal{X}$ drawn independently with $\mathsf{Pr}[X = 0] = \mathsf{Pr}[X = 1] = 1/2$. Each bad event $B \in \mathcal{B}$ is an (arbitrary complex) boolean function determined by the variables $\mathsf{vbl}(B) \subseteq \mathcal{X}$. Events B and B' are called dependent, denoted by $B \sim B'$, if $\mathsf{vbl}(B) \cap \mathsf{vbl}(B') \neq \varnothing$. The events are only sparsely dependent, meaning that each $B \in \mathcal{B}$ depends on at most d other events $B' \in \mathcal{B}$ with $B' \neq B$. In other words, the dependency graph $G_{\mathcal{B}} = (\mathcal{B}, \{(B, B') \mid \mathsf{vbl}(B) \cap \mathsf{vbl}(B') \neq \varnothing\})$ that connects any two events which share at least one variable has maximum degree d. See Figure 1.7. We denote the set of events depending on B, thus the neighbors of the vertex B in the dependency graph, by $N(B)$. Every bad event $B \in \mathcal{B}$ occurs with probability at most p, so $p = \max_{B \in \mathcal{B}} \mathsf{Pr}[B]$. The Lovász Local Lemma shows that $\mathsf{Pr}\left[\cap_{B \in \mathcal{B}} \bar{B}\right] > 0$, under the (symmetric) *LLL criterion* that $epd \leq 1$.

Intuitively, if a local union bound is satisfied around each node in the dependency graph, with some slack, then there is a positive probability to avoid all bad events.

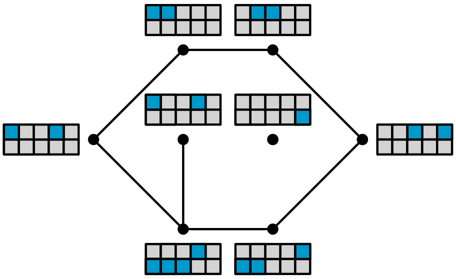


Figure 1.7: The dependency graph of an LLL instance with 8 events and 10 variables. For each event, the set of variables it depends on is given as a bitmap (with blue meaning dependent and grey meaning independent). The maximum degree is $d = 3$. In the constructive LLL problem, the goal is to find an assignment in $\{0, 1\}$ for all the variables so that none of the events is violated.

The constructive version of the purely existential LLL is the algorithmic problem of finding an assignment of values to the variables $X \in \mathcal{X}$ so that all bad events are avoided. Although the LLL itself does not provide an efficient way for finding such a configuration, and that remained open for about 15 years, a number of efficient centralized algorithms have been developed for it, starting with Beck's breakthrough in 1991 [31], through [4, 186, 74, 222, 188], and leading to the elegant algorithm of Moser and Tardos in 2010 [189], who received the Gödel Prize in 2020 for their work.

In the standard distributed formulation of LLL, one considers LOCAL algorithms that work on the dependency graph $G_{\mathcal{B}}$ as communication graph where every event is thought of as a computing entity. Note that one can imagine a few alternative graph formulations, all of which turn out to be essentially equivalent up to an $O(1)$ overhead in the round complexity.

The LOCAL Lovász Local Lemma is an important tool for the design of distributed graph algorithms [67, 213, 87, 55] and has recently gained an extraordinary significance due to the enlightening revelation by Chang and Pettie [61] that LLL is a complete problem for sublogarithmic-time problems. Concretely, they showed that any $o(\log n)$-round randomized algorithm for any LCL problem on bounded-degree graphs can be transformed to an algorithm with complexity $O(T_{\mathsf{LLL}}(n))$. Here, $T_{\mathsf{LLL}}(n)$ denotes the randomized complexity for solving LLL on n-node bounded-degree graphs w.h.p.[9], and LCL stands for *Locally Checkable Labeling* [196], a class of natural local problems whose solution can be verified within a constant radius and hence in $O(1)$ many LOCAL rounds. This class is expressive enough to include all the classic local problems while ruling out inherently global problems (such as leader election or connectivity) and artificially non-local ones (such as finding the maximum ID within radius $\sqrt{n}$). The result by Chang and Pettie thus implies that LLL is important not only for a few special problems, but in fact for essentially *all* sublogarithmic-time distributed problems, and can be seen as the analogue of understanding the classic complexity of NP-complete problems, in that it seeks answers to the question of whether for efficiently checkable problems a solution can also be found efficiently. Due to its remarkable role, Chang and Pettie [61] state that

> *"understanding the distributed complexity of the* LLL *is a significant open problem."*

In contrast to the centralized setting, distributed algorithms for LLL, and the related round complexity, are less well-understood.

[9]Throughout, w.h.p. stands for with high probability and means with probability $1 - n^{-\Omega(1)}$, thus a failure probability that is polynomially small in the number of nodes in the graph.

1.2.3 Local Sampling

The MCMC *(Markov Chain Monte Carlo)* method is a central class of algorithms for *sampling*, that is, for randomly drawing an element from a ground set according to a certain probability distribution. It works by constructing a Markov chain with the targeted sampling distribution as its stationary distribution. Within a number of steps, known as the mixing time, the Markov chain converges; its state then (approximately) follows this distribution. Besides the intrinsic interest of such a general sampling method, in particular for complex distributions where simple sampling techniques fail, the MCMC method gives rise to efficient approximation algorithms in a variety of areas: enumerative combinatorics (due to the fundamental connection between sampling and counting established by Jerrum, Valiant, and Vazirani [148]), simulated annealing [191] in combinatorial optimization, Monte Carlo simulations [185] in statistical physics, and computation of intractable integrals for, among many others, Bayesian inference [10] in machine learning, to mention a few.

The employment of MCMC methods is particularly important when confronted with high-dimensional data where traditional (exact) approaches quickly become intractable. Such data sets are not only increasingly frequent, but also critical for the success of many applications. For instance in machine learning, higher-dimensional models help expressability and hence predictability. It is thus central that MCMC algorithms scale well with increasing dimensions. This is not the case, however, for most centralized methods, as they process and update the variables one by one, that is, a single site per step. To speed up the sampling process, Markov chain updates can be parallelized by spreading the variables across several processors. In other settings, such as distributed machine learning, the (data associated to) variables might already be naturally distributed among several nodes, and the overhead of aggregating them into one, if they fit there in the first place, would be untenable. In either case, to avoid overhead in communication and coordination, local update rules for Markov chains are needed: a node must be able to change the value of its variables without knowing all the values of the variables on other nodes. Yet, the joint distribution over all variables in the system must converge to a certain globally defined distribution.

This local sampling problem was introduced in a recent work by Feng, Sun, and Yin [96], whose title asks "What can be sampled locally?". They study the problem of sampling a proper q-coloring in the LOCAL model: given the communication graph G on n nodes with maximum degree Δ, each node is required to output one of q colors at random such that no two neighboring nodes have the same color and such that the distribution of the joint output is close in terms of total variation distance to the uniform distribution over all proper q-colorings of G. Besides minimizing the round complexity, the objective is to keep the number q of colors as low as possible. The ultimate goal is to achieve a linear speedup in the number of update steps of the Markov chain over centralized algorithms with the same number of colors.

CHAPTER 2

Contributions and Outline

The goal of this thesis is to advance the research in the area of local graph algorithms at the heart of the theory of distributed computing. Besides the theoretical interest of understanding the exact limitations of locality, local approaches for distributed computing offer a wide range of other benefits. They are inevitable for communication-efficient thus fast network algorithms and often are surprisingly simple and easy to implement, hence in particular also do not require memory-intensive or calculation-heavy computations. Moreover, they are fault-tolerant by design: a corrupted node can affect nearby nodes only, therefore its effect cannot percolate too far; and there is no single point of failure or vulnerability. Last but not least, the study of local algorithms often leads to insights in other research areas. For instance, they are of great importance for the design of self-stabilizing distributed algorithms [1] and not seldom give rise to fast dynamic and parallel algorithms in a straightforward manner, since less communication usually implies less coordination. In general, the areas of parallel computing and local algorithms for distributed computing are closely related, and many results apply to both settings [68, 181, 5, 189].

We develop novel algorithmic tools and techniques with provable guarantees that resolve long-standing open questions in the LOCAL model and give rise to fast algorithms in other models of distributed computing (such as CONGEST) and parallel computing (such as PRAM) in Part I.

In Part II, we show that, possibly surprisingly, local approaches are essential even for distributed computing models orthogonal to LOCAL: we adopt local techniques to devise fast distributed algorithms in the all-to-all communication models CC and MPC that considerably improve the state of the art. Interestingly, our methods can also be used for the design of local centralized algorithms [217], which, in turn, often serve as building blocks for faster classic centralized algorithms [192].

We here briefly outline our conceptual contributions; for the precise results and techniques, we refer to the respective chapters.

2.1 Part I: The LOCAL Model

The results presented in Part I of the thesis are based on the following publications.

[99, 100]	*Improved Deterministic Distributed Matching via Rounding*, by Manuela Fischer. *Best Student Paper.*
[103]	*Deterministic Distributed Edge-Coloring via Hypergraph Maximal Matching*, by Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn.

[101] *Sublogarithmic Distributed Algorithms for Lovász Local Lemma with Implications on Complexity Hierarchies*, by Manuela Fischer and Mohsen Ghaffari.

[105, 106] *Tight Analysis of Randomized Greedy MIS*, by Manuela Fischer and Andreas Noever.

[102] *A Simple Parallel and Distributed Sampling Technique: Local Glauber Dynamics*, by Manuela Fischer and Mohsen Ghaffari.

They are concerned with central questions in the area of LOCAL graph algorithms, which can be mostly grouped into three different directions, as we will outline briefly.

Deterministic versus Randomized Algorithms

For several decades, there had been an exponential gap between randomized and deterministic round complexity for many classic local graph problems: most problems can be solved w.h.p. in $\operatorname{poly}\log n$ or even $O(\log n)$ rounds by randomized algorithms [181, 5, 174], while the best deterministic algorithm is based on a generic approach which, until two years ago, took $2^{O(\sqrt{\log n})}$ rounds [207]. Panconesi and Rizzi pointed out in the year 2001 [205] that

> *"while maximal matchings can be computed in polylogarithmic time, in n, in the distributed model [131], it is a decade old open problem whether the same running time is achievable for the remaining 3 structures [maximal independent set, vertex coloring, and edge coloring]."*

Only very recently, Rozhoň and Ghaffari [216] and Ghaffari, Grunau, and Rozhoň [114] in a subsequent work managed to resolve this question by bringing down the bound of network decomposition to $O(\log^5 n)$. Despite this breakthrough, the best randomized algorithms for symmetry breaking and dozens of other classic problems, currently still are significantly (though admittedly not exponentially) faster and considerably simpler and more elegant than their deterministic counterparts.

Understanding whether such a separation is inherent as well as potentially narrowing down this gap are not only interesting from a (complexity-)theoretical point of view, and thus according to Barenboim and Elkin [24] "*perhaps the most fundamental open problem*", but there are also strong practical motivations for developing efficient deterministic algorithms. Besides being indispensable for applications where reproducibility of the computation is critical or where even tiny error probabilities cannot be tolerated[1], faster deterministic algorithms curiously often help to obtain even faster randomized algorithms. The reason for that is as follows. Most of the recent developments in randomized network algorithms are based on the shattering technique (see Section 3.1.5), which randomly breaks down the graph into several small components, typically of exponentially smaller size, and then applies a deterministic algorithm to each of these components separately. In fact, this connection is more general: Chang, Kopelowitz, and Pettie [57] showed that it is impossible to improve the randomized complexities without also improving the deterministic complexity, which makes the shattering technique absolutely essential. It is thus crucial to understand the deterministic complexity of problems, even if one does not mind the use of randomness. This is demonstrated in Chapters 4 to 6 (outlined in Sections 2.1.1 and 2.1.2), where we improve on the state-of-the-art randomized algorithms by devising faster deterministic techniques.

[1]Note that the standard approach of repeating an algorithm until it succeeds is not so easy to adopt in distributed computing, as detecting a global failure requires additional communication.

Deterministic Algorithms for Low-Degree Graphs

A parallel branch of research is concerned with a more local measure of complexity: aiming to keep the dependency on n as low as possible and only characterize the Δ-dependency in the round complexity [142, 127]. Ideally, we would want and likely also expect the dependency radius, and hence the running time, to be independent of the size of the network. For many problems, however, it turns out that it is not possible to have running times purely depending on Δ without any dependence on n, as there is at least a $\Omega(\log^* n)$ lower bound[2] by Linial [172] (deterministic) and Naor [194] (randomized). This is why one main interest lies in getting $T(\Delta) + O(\log^* n)$ running times with at most an $O(\log^* n)$ additive term, for some ideally as small as possible function T. In general, expressing the complexity in terms of both the size n and the maximum degree Δ of the network allows us to distinguish the impact of these two parameters.

Randomized Algorithms for Low-Degree Graphs

There is also an interest in getting more efficient randomized algorithms for low-degree graphs. More concretely, the goal is to get $T(\Delta) + \operatorname{poly}\log\log n$-round algorithms for small functions T. It is no coincidence that the n-dependency is exponentially smaller than in the deterministic round complexities purely expressed in n. As we will see in Section 3.1.5, this is a direct consequence of the shattering technique which randomly shatters the graph into components of size (roughly speaking) $\operatorname{poly}\log n$, on which the deterministic algorithm runs exponentially faster. The first term $T(\Delta)$ comes from the randomized shattering part; the $\operatorname{poly}\log\log n$ from applying an efficient deterministic algorithm to the small components.

[2]More precisely, if IDs consist of $O(\log k)$ bits, or alternatively, if a $\operatorname{poly} k$-coloring of the network is given, the lower bound is $\Omega(\log^* k)$. This allows to go around the $\Omega(\log^* n)$ lower bound of Linial [171].

2.1.1 Deterministic Local Rounding

In 2016, Ghaffari, Kuhn, and Maus [119] formalized the role of randomness for distributed graph algorithms by a *completeness*-type result which showed that, at that time, *rounding* is the only obstacle for efficient deterministic algorithms or, put differently, that deterministically rounding fractional values to integral values while approximately preserving some linear constraints is essentially all that we do need to know how to perform efficiently to get poly $\log n$-round deterministic algorithms for basically all the problems[3].

Although this obstacle has been surmounted by Rozhoň and Ghaffari [216], the problem of rounding still prototypically captures the essence of the difference between randomization and determinism. It illustrates the power of random coins by extracting and pinpointing a single problem whose randomized and deterministic difficulty differ drastically: many variants of rounding admit a trivial 0-round randomized algorithm but cannot be solved efficiently deterministically by any other approach than network decomposition.

One major goal of this thesis is to better understand this discrepancy between randomization and determinism by shedding some light to the problem of deterministic rounding. The key novelty is a simple deterministic local rounding method which transforms fractional solutions of certain linear programs to integral solutions. This is the first such rounding method, to our knowledge.

More specifically, in a first step, as presented in Chapter 4, we provide a technique to efficiently round a fractional matching of a graph (an assignment of a value in $[0, 1]$ to each edge so that for every vertex the values of its incident edges sum up to no more than 1) to an integral one, without changing the overall value of the match-

[3]Stating this result formally and in full generality requires more definitions. We refer to [119] for the precise statement. The significance of rounding is further discussed in [21, 116].

ing by too much. Our deterministic rounding method gives rise to the first improvement on maximal matching and many variants of matching (even some global matching approximation problems) in over 20 years with simpler, faster, and more accurate algorithms.

In a second step in Chapter 5, we generalize our rounding technique from graphs to hypergraphs. In the LOCAL model, when communicating on a hypergraph, per round each node v can send a message on each of its hyperedges, which then gets delivered to all the other endpoints of that hyperedge[4]. For our purposes, hypergraphs are mainly used for formulating the requirements of the problem, and the real communication happens on the base graph where every hyperedge on r nodes is replaced by an r-clique on these nodes.

The problem of matching in hypergraphs is relevant for several reasons. Firstly, it is a natural extension to study classic graph problems in the more general setting of hypergraphs. Secondly, as we will see, local hypergraph matching algorithms serve as a subroutine for several distributed algorithms, such as maximum-weight matching, edge coloring [120], and Nash-Williams decomposition [115]. Thirdly, and probably most importantly in the historic context of our result in [103], hypergraph maximal matching is a means to interpolate between all four classic symmetry breaking problems. In other words, vertex coloring, edge coloring, maximal matching, and maximal independent set on graphs all can be phrased as a maximal matching problem on hypergraphs. Our hypergraph maximal matching algorithm thus leads to numerous improvements over the state of the art. To name just a few, it yields the first efficient (and only one disregarding network decomposition) algorithms for edge coloring on general graphs and maximal independent set on bounded-neighborhood-independence graphs, resolving two longstanding open questions in the affirmative.

[4]The variant of the model with bounded-size messages can be specialized in a few different ways, see e.g. [166].

Notably, these problems are among the very few problems admitting a direct solution not relying on network decomposition. This comes with several advantages: For one, since there is no need to gather and whole graph topologies into single nodes, the algorithms only send small messages and hence directly work in the CONGEST model as well. For another, it leaves room for improvement in the polylogarithmic bound, especially for low-degree graphs. Indeed, for the problem of matching, our algorithm is considerably faster than network decomposition and the to date fastest algorithm; for graphs with sublogarithmic degree, our algorithms are significantly faster than the generic ones based on network decomposition. Furthermore, our approach is completely different and novel; and it demonstrates the power and flexibility of rounding.

We are confident that our results open the road for further progress on deterministic distributed graph algorithms, with clear consequences also on randomized algorithms, and that ideas of deterministic distributed rounding will be of interest well beyond the scope of these results: that they will prove useful to get efficient deterministic—and hence randomized—LOCAL algorithms for an even wider range of problems and, as Barenboim and Elkin [24] suggested, they may serve as a "*good stepping stone*" towards a $\operatorname{poly}\log n$-round deterministic algorithm for MIS that does not rely on network decomposition. In fact, in a recent development, our rounding techniques have been extended to coloring in [118], to give the first efficient $(\Delta + 1)$ vertex coloring algorithm without the use of network decomposition. Moreover, they have given rise to faster algorithms for seemingly unrelated problems such as finding a low outdegree orientation [223].

2.1.2 Lovász Local Lemma and Bootstrapping

Chang and Pettie [61, 53] proved that LLL is a complete problem for LCL problems with sublogarithmic complexity in bounded-degree graphs: they show that when there is an $o(\log n)$-round randomized algorithm $\mathcal{A}$ for an LCL problem $\mathcal{P}$ on a graph G with constant degree, it can be sped up to run in $O(T_{\mathsf{LLL}}(n))$ rounds. Their neat idea can be summarized as follows. They lie to the algorithm $\mathcal{A}$ and say that the network size is some much smaller value $n^* \ll n$. This deceived algorithm $\mathcal{A}$ may have a substantial probability to fail, creating an output that violates the requirements of $\mathcal{P}$ somewhere. However, the probability of failure in each local neighborhood is at most $1/n^*$. Choosing n^* a large enough constant, depending on the complexity of $\mathcal{A}$, the algorithm $\mathcal{A}$ provides an LLL system—where there is one bad event for violation of each local requirement of $\mathcal{P}$—that satisfies the criterion $pd^c < 1$ for some (desirably large) constant $c \geq 1$. By solving this LLL system, one thus obtains a solution for the original problem $\mathcal{P}$ in $O(T_{\mathsf{LLL}}(n))$ time. This is illustrated in Figure 2.1.

Inspired by this idea, in Chapter 6 we improve $T_{\mathsf{LLL}}(n)$ using a bootstrapping approach as follows: we first devise a sublogarithmic-round base LLL algorithm $\mathcal{A}$, and then use the deception technique to speed up $\mathcal{A}$. This works by viewing $\mathcal{A}$ as setting up a new LLL with a much larger exponent in its LLL criterion, hence allowing us get to a much smaller complexity by (recursively) applying the same scheme. We find this recursive application of the idea to speed up the complexity of LLL itself, through increasing the exponent of the corresponding LLL criterion, somewhat amusing.

Many subsequent works are following our ideas. Examples are [55, 56, 53, 47], and [135, 136, 134, 216, 114], and even range as far as descriptive combinatorics [36].

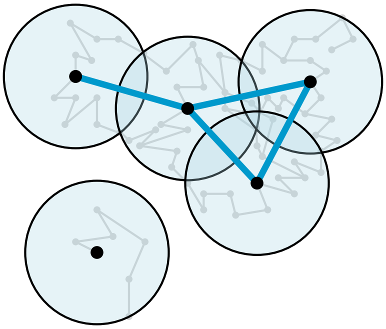


Figure 2.1: An illustration of the deception technique: if the outputs for two nodes of the deceived algorithm depend on each other (because their dependency disks overlap), they are connected by an edge (depicted in blue) in the corresponding LLL problem.

2.1.3 Tight Analysis of Local Greedy Algorithms

Luby's algorithm by Luby [181] and independently Alon, Babai, and Itai [5] is a famous and simple randomized algorithm for the problem of maximal independent set: For $O(\log n)$ iterations, every node picks a random number and all *local minima*—i.e., nodes that have the smallest number among their neighbors—join the independent set and are removed from the graph along with their neighbors. One iteration can be implemented in two rounds of communication: one round to share the random numbers and round to inform neighbors about their decisions (whether to join the MIS). It is a well-known fact that the removal of local minima and their neighbors for random numbers leads to a decrease in the total number of edges by a constant factor (see, e.g., [184] for a simple proof). Since in every iteration new random numbers are generated, repeated application of this argument directly implies an upper bound of $O(\log n)$.

There is another very similar and strikingly simple randomized algorithm, called the local greedy MIS algorithm, that works as follows:

An order of the vertices is chosen uniformly at random. Then, in each iteration, all local minima join the MIS. The only difference to Luby's algorithm is that the randomness is not regenerated in every round, but only generated in the very beginning. This reduces the amount of required communication to a minimum. Indeed, a node initially only needs to inform its neighbors about its position in the random order and then, in the round of its removal from the graph, about its decision. Another nice property of this algorithm is that—once an order is fixed—it always yields the so-called *lexicographically first MIS*, i.e., the same as the *sequential greedy MIS* algorithm that goes through the vertices in this order one by one and adds a vertex to the MIS if none of its neighbors has been added in a previous step. Such determinism can be an important feature of parallel algorithms [40, 38].

These practical advantages are mainly owed to the fact that the same random order is used throughout. This, however, comes with the drawback of complicating the analysis significantly, due to the lack of independence among different iterations. Indeed, while for Luby's algorithm the round complexity was established at $O(\log n)$ almost fourty years ago [180, 5], no similar result is known for the local greedy MIS algorithm.

In Chapter 7, we establish that the local greedy MIS algorithm is as fast as Luby's, while requiring significantly less communication and randomness, confirming a widespread belief. Note that this in particular implies that the trivial algorithm that iteratively lets nodes with locally minimal IDs join the MIS is efficient on average—if the IDs are chosen uniformly at random, $O(\log n)$ rounds are enough. This is significantly faster than the worst case (for instance, a path with monotonically increasing IDs) that requires $\Omega(n)$ rounds.

This observation and our analysis are built upon by several works in various different areas, to obtain improved LOCAL [62], MPC [112], parallel [82, 35, 83, 81, 138], and dynamic [33, 203] algorithms.

2.1.4 Local Sampling of Uniform Colorings

Over the past few years, several methods to parallelize sequential Markov chains have been proposed. Most of them rely on a heavy coordination machinery, are special purpose, and/or do not provide any theoretical guarantees. In the following, we briefly introduce two of the most promising and more generic parallel and distributed sampling techniques, in the context of colorings.

The most natural one follows a standard decentralization approach, also implemented in the *LubyGlauber* algorithm of [96]: an independent set of nodes (e.g., a color class of a proper coloring) simultaneously updates their colors [96], ensuring that no two neighboring nodes change their color at the same time. This approach mainly suffers from the limitation that the number of independent sets needed to cover all nodes might be large, which slows down mixing. In particular, a multiplicative Δ-term in the mixing time seems inevitable [125, 96]. In the worst case of a clique, this approach falls back to sequential sampling, updating one node after the other. Moreover, this method requires an independent set to be computed, which incurs a significant amount of additional communication and coordination.

An orthogonal direction was pursued by [200, 229, 96], where methods are introduced to update the colors of all nodes simultaneously. One example is the *LocalMetropolis* algorithm of [96]. This extreme parallelism, however, comes at a cost of either introducing a bias in the stationary distribution, resulting in a non-uniform coloring [200, 229], or demanding stronger mixing conditions [96].

We aim for the middle ground between these two approaches, motivated by the following observation: we do not need to prevent simultaneous updates of adjacent nodes, only simultaneous *conflicting* updates of adjacent nodes. Indeed, preventing two adjacent nodes in the first place from picking a new color in the same round seems to be way too restrictive, in particular because it is unlikely that both nodes choose the same new color. On the other hand, if all nodes update their colors simultaneously, a node is expected to have a conflict with at least one of its neighbors, which prevents progress. Since a simultaneous update of nodes in an independent set is too slow and of all nodes is too risky, we want an almost independent set (or, in other words, a low-degree graph) to update their colors. This can be achieved by letting each node decide whether it participates in a particular color update step independently with (small) constant probability so that only a small fraction of a node's neighbors is expected to update the color, and hence also, in the worst case, only these can conflict with its update. An example is given in Figure 2.2. As opposed to centralized sampling, where only one

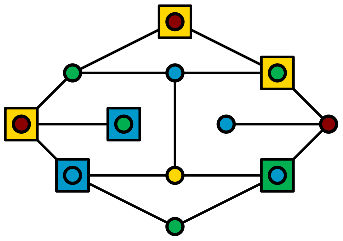


Figure 2.2: A color update: the update of the three vertices on the left are valid; the one on the top (two neighbors proposing the same color) and to the right (a vertex proposing a neighbor's current color) are not.

variable per step updates its value, here the expected number of variables simultaneously updating their value is $\Omega(n)$, resulting in a linear speedup in the number of update steps. This local sampling technique is introduced in Chapter 8.

Our algorithm leads to applications in several areas, such as [37] in the LOCAL setting and [97] for dynamic algorithms.

2.2 Part II: Global Communication Models

In Part II of this thesis we show how local approaches can be used to devise fast algorithms the all-to-all communication models Congested Clique and MPC. It is based on the following publications and manuscripts.

[44, 45]	*Breaking the Linear-Memory Barrier in* MPC*: Fast* MIS *on Trees with Strongly Sublinear Memory*, by Sebastian Brandt, Manuela Fischer, and Jara Uitto. *Best Student Paper.*
[43]	*Matching and* MIS *for Uniformly Sparse Graphs in the Low-Memory* MPC *Model*, by Sebastian Brandt, Manuela Fischer, and Jara Uitto.
[32]	*Massively Parallel Computation of Matching and* MIS *in Sparse Graphs*, by Soheil Behnezhad, Sebastian Brandt, Masha Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M. Karp, and Jara Uitto.
[104]	*Simple Graph Coloring Algorithms for Congested Clique and Massively Parallel Computation*, by Manuela Fischer, Mohsen Ghaffari, and Jara Uitto.
[54]	*The Complexity of* $(\Delta + 1)$*-Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation*, by Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. *Best Student Paper.*

Global communication models and locality-based models (like LOCAL) are contrasting models dealing with opposite challenges. At first sight, thus totally different sets of techniques seem to be needed. However, as we demonstrate in Part II of this thesis, one should not jump to this conclusion too hastily.

We use local techniques to design faster algorithms in global all-to-all communication models. More concretely, we accelerate the simulation of LOCAL algorithms by enhancing them with non-local communication. Our technique is based on the observation (as discussed in more detail in Section 3.2) that models that suffer from congestion often admit simple if not trivial algorithms when the underlying graph is sparse. Intuitively, the reason for that is as follows. All we need to know to predict the output of an r-round LOCAL algorithm for a vertex v is the r-hop neighborhood $N^r(v)$ of v. If the maximum degree of the graph is small enough, $N^r(v)$ can be gathered efficiently into a single node, where the LOCAL algorithm can be run offline. In fact, this often takes $O(\log r)$ rounds, leading to an exponential speedup.

In Section 2.2.1, we show how to take this idea even further to a super-exponential speedup with an additional sparsification of the corresponding LOCAL algorithm. In Section 2.2.2, we discuss the sublinear-memory MPC model, a practically realistic variant of the model. As it turns out, for this model, local simulation techniques are not only helpful but apparently inevitable.

2.2.1 Sparsification of Local Algorithms

Imagine that we have an r-round LOCAL algorithm $\mathcal{A}$ that has not only small locality but also a small *locality volume*, meaning that a vertex' output only depends on a *subset* $N^r_*(v) \subseteq N^r(v)$ of vertices within its r-hop neighborhood $N^r(v)$. To simulate $\mathcal{A}$'s output for v, it thus suffices to gather information from $N^r_*(v)$ only, instead of $N^r(v)$, which is potentially much smaller and hence can be collected much more efficiently.

One common approach is thus to sparsify LOCAL algorithms, i.e., to reduce their locality volume [110, 123]. We exemplify this with our constant-round $(\Delta + 1)$ list vertex coloring algorithm in the CC model presented in Chapter 9: we first sparsify and then efficiently simulate a LOCAL vertex coloring algorithm, leading to a superexponential speedup in the round complexity.

2.2.2 Sublinear-Memory MPC Model

In the study of classic graph problems for MPC, there are three different natural regimes for the memory S in terms of the number of vertices n of the graph: strongly superlinear memory with $S = \widetilde{\Omega}(n^{1+\delta})$; almost-linear memory with $S = \widetilde{\Theta}(n)$; and strongly sublinear memory with $S = \widetilde{O}\left(n^{1-\delta}\right)$, for some $0 < \delta < 1$. Unsatisfactorily, an assumption common to a vast majority of approaches and techniques is to allow essentially linear in n memory per node (for instance, strongly superlinear or mildly sublinear like $n^{1-o(1)}$) memory per node.

Indeed, for instance for the problem of maximal matching, the only subpolylogarithmic algorithm of Lattanzi, Moseley, Suri, and Vassilvitskii [167] requires strongly superlinear memory. When the local memory is restricted to be (nearly) linear, the round complexity of Lattanzi et al.'s algorithm drastically degrades, falling back to the trivial bound attained by the simulation of the $O(\log n)$-round LOCAL (and PRAM) algorithm due to Luby [181] and, independently, Alon, Babai, and Itai [5].

For other classic problems, such as maximal independent set or approximate maximum matching, we have a slightly better understanding. There, the local memory can be reduced to be linear while still having $\operatorname{poly}\log\log n$-round algorithms [12, 73, 112]. Yet, all these algorithms fail to go (substantially) below linear space without an (almost) exponential blow-up in the running time. This is true for a wide variety of problems [214, 52].

Ghaffari et al. [117] aptly sum up this bleak state with *“the technical difficulty (and the round complexity) of the problem increases as the memory per machine decreases”*

and

“The case of the strongly sub-linear memory regime appears to be considerably harder.”

Linear memory is often not only prohibitively large and hence impractical for massive graphs, but also allows an easy or even trivial solution for sparse graphs. Indeed, for graphs with $\widetilde{O}(n)$ edges, this is getting close to the degenerate regime where we can afford to store the whole input graph in a single node. This issue has been artificially circumvented by explicitly restricting the attention to dense graphs with $m = \widetilde{\Omega}(n^{1+\delta})$ edges, as to ensure sublinearity of the total memory in G while still not having to relinquish the nice property that (essentially) all vertices fit into the memory of a single node [151]. Besides being a stretch of the definition, this additionally imposed condition of denseness of the input graph does not seem to be realistic. In fact, as recently also pointed out by [73], most practical large graphs are sparse. It is thus natural to ask whether there is a fundamental reason why the known techniques get stuck at the linear-memory barrier and to what extent linear memory is necessary for efficient algorithms.

To address this question, we focus on the MPC model with sublinear memory. One important aspect of our work is, from the theory perspective, that it breaks this threshold and thereby opens up a whole new unexplored domain of research questions. Moreover, since all common methods—such as filtering [167, 165], (randomized composable) coresets [14, 12], and round compression [14, 11]—seem to hit a boundary at the linear-memory barrier, completely new techniques are required.

In the sublinear-memory setting, one is inevitably confronted with the inherent challenge of locality: In contrast to the MPC model with linear or strongly superlinear memory, in this MPC setting, as the space of a node is strongly sublinear, it will never be able to see a significant fraction of the vertices, regardless of how sparse the graph is. Even though the communication graph does not suffer from locality, the node congestion causes that a single node at once only gets to see a non-significant part of the graph, and there is nothing it can do about it. Note that this is different from CC. There, in principle, a node can see the whole graph—it just needs some time to gather the whole graph due to congestion. In MPC no matter how many rounds are given, it is just utterly impossible to at some point fit the whole graph even if it is incredibly sparse.

We thus need to deal with this intrinsic local view of the nodes. It seems natural to borrow ideas from LOCAL algorithms, which are designed exactly to cope with this locality restriction. A direct simulation, however, in most cases only results in $\log^{\Omega(1)} n$-round algorithms. The problem is that these algorithms do not make use of the additional power of the MPC model—the global all-to-all communication—as the communication is restricted to neighboring vertices.

We introduce a new and strikingly simple technique to cope with this imposed locality: we enhance local-inspired approaches with global communication in order to arrive at efficient algorithms in the world of sublinear-memory MPC which are exponentially faster than their LOCAL counterparts and whose memory requirements are polynomially smaller than their traditional MPC counterparts. Ironically, while Lattanzi et al. [167] observe that approaches to

> *“shoehorn message passing style algorithms into the framework”*

have given rise to (too) slow MPC algorithms in the linear-memory setting, we are convinced that in the presence of strongly sublinear memory, LOCAL (and hence message-passing) inspired techniques are absolutely essential for efficiency.

Our method leads to substantial improvements on the state of the art that are either exponentially faster or require exponentially less memory than their predecessors.

In particular, we study the classic local graph problems MIS, and MM on the special graph families of trees and, more generally, graphs with arboricity $\lambda = \operatorname{poly}\log n$. These sparse graphs are particularly interesting for the following reasons. While trees—and in general, graphs with a linear number of edges—admit a trivial solution in the linear-memory MPC model, this cheat does not work in the sublinear-memory setting. In some sense, sparse graphs are thus the easiest non-trivial case, which makes it the most natural starting point for further studies. We strongly believe that our techniques can be extended to more general graph families. Moreover, arboricity is a well-received measure of sparsity that does not impose strict structural constraints such as planarity, bounds on maximum degree, or the like [23, 28, 89, 70, 128]. The family of graphs with arboricity $\operatorname{poly}\log n$—often called *uniformly sparse* graphs, and also known as *sparse everywhere* graphs—includes but is not restricted to graphs with maximum degree $\operatorname{poly}\log n$, minor-closed graphs (e.g., planar graphs and graphs with bounded treewidth), as well as preferential attachment graphs, and thus arguably contains most sparse graphs of practical relevance [124, 202].

CHAPTER 3

Tools and Techniques

We present some frequently used techniques in two parts. First, in Section 3.1, we explain a variety of local decomposition techniques used to design faster LOCAL algorithms. Second, in Section 3.2, we introduce tools for the efficient simulation of LOCAL algorithms in all-to-all communication models.

3.1 Local Decomposition Techniques

Many problems turn out to become easy if the graph is somewhat small—small in terms of, for instance, number of nodes, diameter, or maximum degree. To exploit this fact, a common strategy is to break the graph into smaller parts and to deal with each part separately (either sequentially or in parallel). Often, there is a tradeoff between how simple each part is, how many parts are needed, and how much time it takes to compute the decomposition. The purpose of this section is to convey the intuition behind these techniques. For a more thorough overview, we refer to [24].

3.1.1 Coloring

Proper Coloring

If a node's decision only depends on neighboring nodes, a proper coloring can be used to decouple these decisions: given a proper coloring of the nodes of G with k colors, one can process G in k phases, in each phase dealing with one color class. For instance, given a k-coloring, one can compute a MIS in k rounds as follows: in round i, every node with color i without a neighbor in the MIS joins the MIS and informs all its neighbors about this decision. Note that a proper coloring can be seen as an extreme case of degree reduction, since every part (i.e., every color class) has maximum degree 0.

Defective Coloring

While a proper vertex coloring makes the computation within each part particularly easy, the number of parts often is prohibitively large. One remedy is to consider a relaxation of proper coloring that allows for a smaller number of colors, at the cost of having some (but not too many) adjacent nodes with the same color. A *defective k-coloring with defect d* is a k-coloring where each color class has degree at most d—it decomposes the graph into k many degree-d graphs. An example of a defective coloring is depicted in Figure 3.1.

Proper edge colorings and defective edge colorings with defect d—where the maximum degree induced by edges of the same color is d—are the analogues for decisions on edges instead of nodes.

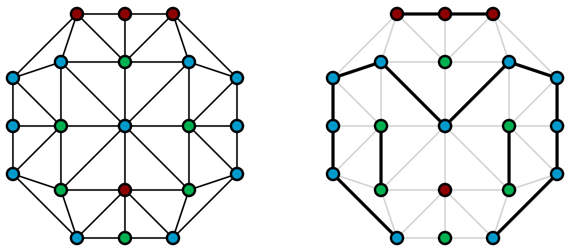


Figure 3.1: A 2-defective 3-coloring and the corresponding decomposition into 3 graphs of degree at most 2.

Distance Coloring and Power Graphs

For problems where more distant nodes' decisions depend on each other, the approach of a proper (or a defective) graph coloring does not work. For instance, when computing a maximal matching, two nodes at distance 2 cannot decide simultaneously about their edges to join the MM: it could lead to a common neighbor of these nodes to have two incident edges in the MM. In general, if nodes' decisions depend on nodes within distance k, to have the decisions decoupled, a coloring is needed where no two nodes within distance k have the same color is needed. In other words, one needs a proper coloring of the power graph $G^k = (V, E^k)$ of G obtained by adding an edge iff two nodes have distance at most k in G. Such a coloring of the power-k graph is called a distance-k coloring. Note that a distance-k coloring is equivalent to a coloring for which all nodes within the k-hop neighborhood of a node have distinct colors. A coloring algorithm—or any algorithm for that matter—on G^k can be executed on G by letting each node in G simulate the behavior of a node in G^k in a coloring algorithm, at the cost of a multiplicative factor of $\Theta(k)$ in the running time. An example of a distance coloring can be found in Figure 3.2. Note that this trick with power graphs is more general, as also nicely exemplified in Section 3.1.4.

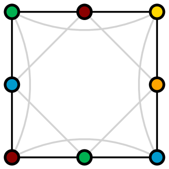


Figure 3.2: A graph G and its distance-2 coloring. The additional edges of G^2 are depicted in grey. Any two nodes of the same color have distance at least 3 in G.

3.1.2 2-Decomposition

A 2-*decomposition* is a simple transformation that decomposes a graph into vertex-disjoint paths and cycles with the same edge set (but a larger vertex set), in zero rounds. It has been used frequently before, for instance by [146, 131, 132], and also gives rise to an almost trivial proof of Petersen's 2-factorization theorem from 1891 [190].

The 2-decomposition graph G' of a graph G is generated as follows. Each node v in G introduces $\lceil d_G(v)/2 \rceil$ copies and arbitrarily splits its incident edges among these copies in such a way that every copy has degree 2, with the possible exception of one copy which has degree 1 if v has odd degree. Notice that the edge sets of G and G' are the same and that G' is simply a set of cycles and paths. See Figure 3.3 for an example. A node v in G then simulates the algorithm for each of its copies in G'.

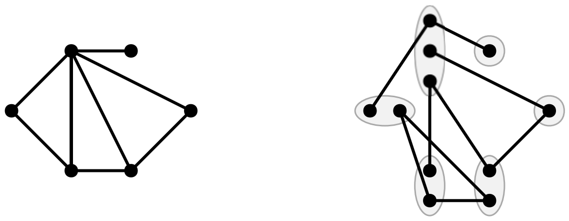


Figure 3.3: A graph and its 2-decomposition.

3.1.3 H-Partition

The *H-partition*, also called Nash-Williams decomposition [199, 197, 198], see Barenboim and Elkin [24, Chapter 5.1], can be used to decompose a graph with small arboricity into parts with small degree. The arboricity λ of a graph is the minimum number of edge-disjoint forests needed to cover the edges of the graph. Put differently, it is the smallest possible maximum outdegree of a node over all possible orientations of the edges. A graph with small arboricity thus can be thought of as a uniformly sparse graph. Note that any graph satisfies $\lambda \leq \Delta/2$.

More concretely, the H-partition distributes the nodes of a graph into layers in a way that in each layer the nodes have small outdegree into subsequent layers. Initially, the idea was to decompose graph with constant arboricity into $O(\log n)$ layers of nodes with constant outdegree, but it easily generalizes to basically any outdegree as follows:

An H-partition with outdegree d, defined for any $d > 2\lambda$, is a partition of the nodes into $\ell = \Theta(\log_{d/\lambda} n)$ layers with the property that a node in layer i has at most d neighbors in the union of the layers $i, \ldots, \ell$. An example can be found in Figure 3.4.

Note that for $d > 2\lambda$ such a partition can be computed easily by the following sequential greedy algorithm, also known as *peeling* algorithm: Iteratively, for $i \geq 1$, put all remaining vertices with remaining degree at most d into layer i, and remove them from the graph. Due to the well-known fact that the average degree of a graph with arboricity λ is at most 2λ, in every iteration a fraction $2\lambda/d$ of the vertices will be removed. Hence, the algorithm terminates after $O(\log_{d/\lambda} n)$ iterations. Similarly, this partition can be computed by a LOCAL algorithm in the same number of rounds.

We next prove some properties of the resulting H-partition that will be useful later.

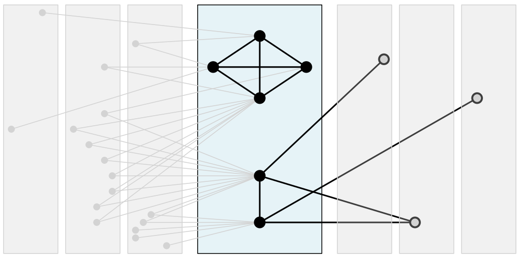


Figure 3.4: A schematic depiction of one level of an H-partition with parameter $d = 3$. Only edges relevant for this level are shown. There can be arbitrarily many incoming edges (light grey) but only at most d outgoing edges (black) to the same or subsequent levels.

Lemma 3.1. *The H-partition with outdegree d, constructed by the greedy peeling algorithm, satisfies the following properties.*

(i) For all $0 \leq i \leq \ell$, the number $\left|\bigcup_{j=i}^{\ell} L_j\right|$ of vertices in layers with index $\geq i$ is at most $n\left(\frac{2\lambda}{d}\right)^{i-1}$. In other words, if we remove all vertices in layer i from the set of vertices in layers $\geq i$, then the number of vertices drops by a factor of $\frac{2\lambda}{d}$, i.e., $\left|\bigcup_{j=i+1}^{\ell} L_j\right| \leq \frac{2\lambda}{d}\left|\bigcup_{j=i}^{\ell} L_j\right|$ for all $0 \leq i \leq \ell$.

(ii) There are at most $\ell = O\left(\log_{\frac{d}{\lambda}} n\right)$ layers.

Proof. We prove (i) by induction, thus assume that there are $n_i \leq n\left(\frac{2\lambda}{d}\right)^{i-1}$ vertices in the graph H_i induced by vertices in layers $\geq i$. Towards a contradiction, suppose that there are $n_{i+1} > n\left(\frac{2\lambda}{d}\right)^{i}$ vertices in layers $\geq i+1$. By construction, all these vertices must have had degree larger than d in H_i, as otherwise they would have been added to layer i. This results in an average degree of more than $\frac{n_{i+1}d}{n_i} = 2\lambda$ in H_i, which contradicts the well-known upper bound of 2λ on the average degree in a graph that has arboricty at most λ. Note that (ii) is a direct consequence of (i). $\square$

3.1.4 Network Decomposition

The network decomposition approach is a classic technique that goes back to the pioneering work by Awerbuch et al. [17, 207] in the very beginning of the area. Roughly speaking, a network decomposition partitions the nodes into a few blocks, each of which is made of a number of low-diameter connected components.

We next give a formal definition, along with an intuitive discussion of the approach, and then provide one specific network decomposition algorithm.

A (C, D) network decomposition is a partition of the nodes V into C node-disjoint blocks V_1, V_2, …, V_C such that the induced subgraph $G[V_i]$ of each block V_i consists of (a number of) connected components of diameter at most D. See Figure 3.5 for an illustration.

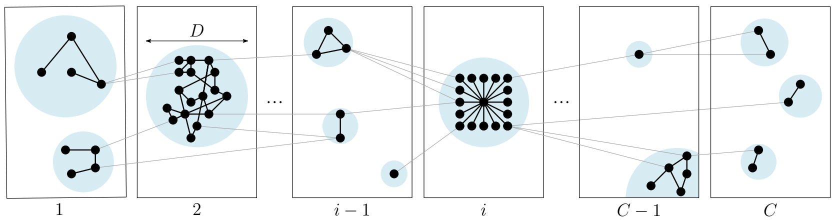


Figure 3.5: A schematic depiction of a (C, D) network decomposition. There are C blocks consisting of several connected components with diameter at most D.

The idea of the network decomposition approach is to decompose the graph into many low-diameter components. Each of these components then can be solved easily, by gathering the whole topology and computing a solution offline. Different components of the same block can be handled simultaneously. Different blocks, however, might depend on each other. The blocks thus have to be processed sequentially, one after the other. Overall, if a (C, D) network decomposition can be computed in $T(C, D)$ rounds, basically all problems can be solved in $O(T(C, D) + C \cdot (D+1))$ rounds, since each of the C blocks takes $O(D+1)$ rounds to gather all diameter-D components.

Network decomposition is a generic approach that provides an easy solution for many problems. On the other hand, the technique is not very practical for real-world applications: it heavily relies on gathering topologies of diameter-D graphs (hence messages of up to $\widetilde{O}(\min\{\Delta^D, n^2\})$ size) and brute-forcing solutions on them: this cheat constitutes a stretch of the definition of distributed computing and abuses the unbounded computing power and message sizes. Ideally, we thus want to design algorithms that do not rely on network decomposition. For many problems, however, it is unfortunately the only efficient approach we know.

An $(O(\log n), O(\log n))$ network decomposition can be computed in $O(\log^2 n)$ randomized rounds by Linial and Saks [174, 86]. Until recently, the best deterministic network decomposition algorithm was with parameters $C, D, T(C, D) = 2^{O(\sqrt{\log n})}$ by Panconesi and Srinivasan [207], improving on $2^{O(\sqrt{\log n \log\log n})}$ by Awerbuch et al. [17]. A very recent breakthrough result by Rozhoň and Ghaffari [216] and a follow-up of Ghaffari, Grunau, and Rozhoň [114] have further decreased C, D, and $T(C, D)$ to $\operatorname{poly}\log n$.

In the following, we present a specific variant of network decomposition that works mainly by putting together some ideas of Awerbuch and Peleg [18], Panconesi and Srinivasan [207], and Awerbuch et al. [16]. We are not aware of this result appearing in prior work. Note that combining it with the recent technique of [216, 114], we can improve the round complexity to $\lambda n^{1/\lambda} \log^5 n$.

Lemma 3.2. *A $(\lambda, n^{1/\lambda} \log n)$ network decomposition can be computed by a deterministic LOCAL algorithm in $\lambda n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds.*

Proof. We first describe a centralized algorithm for computing a $(\lambda, n^{1/\lambda} \log n)$ network decomposition, colloquially referred to as *ball* *carving*. This technique was first presented by Awerbuch and Peleg [18]. Then, we explain how to transform this sequential ball carving process into an efficient deterministic LOCAL algorithm with round complexity $n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$, using another network decomposition algorithm of Panconesi and Srinivasan [207], and an idea of Awerbuch et al. [16].

Centralized Network Decomposition via Ball Carving: We will decompose the graph into node-disjoint blocks A_1 to A_λ, such that in the graph $G[A_i]$, each connected component has diameter at most $n^{1/\lambda}\log n$. We first describe the process of generating the first block A_1. The generation of the next blocks is similar.

We choose an arbitrary node v to be the center of a new ball, and we use $B_r(v)$ denote the set of nodes with distance at most r from v. Let r^* be the smallest $r \geq 0$ for which

$$|B_r(v)| < \left(1 + n^{-1/\lambda}\right)|B_{r-1}(v)|.$$

Observe that since $|B_r(v)| \geq \left(1 + n^{-\frac{1}{\lambda}}\right)^r$ for $r \leq r^*$, it must hold that $r^* \leq n^{1/\lambda}\log n$. We put all the nodes in $B_{r^*-1}(v)$ into A_1, and then delete $B_{r^*}(v)$ from G. That is, we carve and remove a ball of radius r^* around the node v, but then only take the inner part—nodes, that are not on the boundary—of it to A_1. Then, we pick another node in this remainder graph, and perform another ball carving; we repeat this process until no node is left.

Once we are done with defining A_1, we remove all nodes of block A_1 from the graph G, and then work on the remaining graph $G_2 = G \setminus A_1$. Then, we create the new block A_2, by a similar iterative ball-carving process on G_2. More generally, after computing blocks A_1 to A_{i-1}, we move to the graph $G_i = G \setminus \cup_{j=1}^{i-1} A_j$ and compute the block A_i, by a sequential ball-carving process similar to above.

We now argue that λ blocks exhaust the graph. In each iteration i

of computing another block, the size of the remaining graph shrinks such that $|G_i| < n^{-1/\lambda}|G_{i-1}|$. This is because for each carved ball where we include $B_{r^*-1}(v)$ in A_i and then discard the boundary $B_{r^*}(v) \setminus B_{r^*-1}(v)$, leaving them for the next blocks, we have

$$|B_{r^*}(v) \setminus B_{r^*-1}(v)| \leq n^{-\frac{1}{\lambda}}|B_{r^*-1}(v)|.$$

Since $|G_i| < n^{-1/\lambda}|G_{i-1}|$, after λ blocks, the remaining graph $G_{\lambda+1}$ is empty, and the process terminates.

Distributed Network Decomposition via Ball Carving: To compute the desired $(\lambda, n^{1/\lambda}\log n)$ network decomposition, we will simulate the ball carving idea explained above. However, we need to speed up the process, and make it run in $\lambda n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds. For that, we use another network decomposition as a helper tool. In particular, we first compute a $\left(2^{O(\sqrt{\log n})}, 2^{O(\sqrt{\log n})}\right)$ network decomposition of G^d for $d = 2n^{1/\lambda}\log n + 1$, by running the algorithm of Panconesi and Srinivasan [207] on G^d. This takes $d2^{O(\sqrt{\log n})}$ rounds. It partitions the graph G into $\ell = 2^{O(\sqrt{\log n})}$ node-disjoint blocks $G_1, \ldots, G_\ell$ such that for each block G_i, each connected component of G_i has diameter at most $d \cdot 2^{O(\sqrt{\log n})}$ in the graph G, while any two components of G_i are non-adjacent in G^d and thus have distance at least $d+1$ in G.

We now use this network decomposition to compute the desired output $(\lambda, n^{1/\lambda}\log n)$ network decomposition which partitions V into node-disjoint sets A_1, A_2, …, A_λ such that in each subgraph $G[A_i]$ for $i \in \{1, \ldots, \lambda\}$, each connected component has diameter at most $n^{1/\lambda}\log n$. The construction is made of λ epochs, each of which computes one of the blocks A_i, in $n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds. We next discuss the first epoch, which computes the block A_1. The next epochs are similar, and compute the other blocks A_2 to A_λ, each repeating the procedure on the remaining graph.

Each Epoch in the Construction of $\mathbf{A_1}$: The epoch is broken into $\ell = 2^{O(\sqrt{\log n})}$ phases, each of which takes $n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds, hence making for an overall round complexity of $n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ for the epoch. We will simulate the ball-carving process of computing A_1, throughout these phases. During this process, each node is in one of the following three states: some nodes are put in A_1 (these are the inner parts of the carved balls), some nodes are processed and discarded (these are the boundaries of the carved balls), and some nodes are unprocessed.

In phase j, we do as follows: Consider the set of nodes of G_j. Notice that each component of G_j has diameter at most $d \cdot 2^{O(\sqrt{\log n})}$, and moreover, each two components have distance at least $d + 1$. We first make the minimum-ID node of each of these components learn the $(n^{1/\lambda} \log n)$-neighborhood of its component, as well as the status of the nodes in this neighborhood. Notice that this information is within distance at most

$$d \cdot 2^{O(\sqrt{\log n})} + n^{1/\lambda} \log n = n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$$

from that minimum-ID node. This can be done in $n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds. Then, this minimum-ID node simulates the ball-carving process offline, as follows: each time, it picks another unprocessed node in its component, and then carves the ball around it similar to the sequential ball-carving process explained above. Notice that this ball can potentially go out of G_j. However, the ball will grow at most $n^{1/\lambda} \log n$ hops away from its center. Hence, the ball carving processes of two different components of G_j never reach each other, as the components are more than $d > 2n^{1/\lambda} \log n$ hops apart. Also, notice that the minimum-ID node is doing this process offline, once it has gathered the relevant information, and thus this computation does not consume any further communication. Once the node with minimum ID has computed the newly carved A_i-balls centered at the nodes of its component, it informs the related nodes of their status: whether they are in A_1, discarded from the A_1 due to falling on the boundary, or remaining unprocessed. This finishes the description of phase j. We then move to the next phase. $\square$

3.1.5 Shattering

The *shattering technique* is a general framework which has been introduced in the LOCAL model by Barenboim, Elkin, Pettie, and Schneider [27, 28] and goes back to the early nineties where it is a key ingredient in Beck's LLL method [31, 4]. Analogues of it appear in [186, 187, 217, 6]. It has inspired a diverse set of results over the past decade.

The shattering framework consists of two phases. First, in the *shattering phase*, a randomized *shattering algorithm* computes a random partial solution to the problem at hand such that the remainder graph is decomposed (i.e., shattered) into several (disconnected) small (think of size $\operatorname{poly}\log n$) components. Roughly speaking, at each step of the shattering algorithm, one specifies an invariant that all nodes must satisfy in order to continue to participate. Those *bad nodes* that violate the invariant are removed from consideration and taken care of in the second phase.

Then, in the *post-shattering phase*, a solution on these remaining small components formed by bad nodes is found using a deterministic algorithm for graphs of size $\operatorname{poly}\log n$. Note that one cannot apply a conventional randomized algorithm, as its success probability on the exponentially smaller component would only be $1 - (\log n)^{-\Omega(1)}$ instead of $1 - n^{-\Omega(1)}$. This usually gives rise to algorithms with a running time that can be expressed as the sum $\operatorname{poly}\log \Delta + \operatorname{poly}\log\log n$ of two terms, the first coming from the shattering phase, which usually takes $\operatorname{poly}\log \Delta$ rounds to shatter the graph, and the latter being the exponentially scaled down running time of the deterministic algorithm for the problem, which usually is $\operatorname{poly}\log\log n$, as the network decomposition approach gives rise to $\operatorname{poly}\log n$-round deterministic algorithms.

The shattering thus can be seen as a randomized decomposition technique to decompose the graph into smaller components (in terms of number of nodes). An example is given in Figure 3.6.

The shattering lemma, roughly speaking, shows that if each node of the graph remains after the shattering algorithm with some small probability and we have certain independence between these events, the remaining connected components are somewhat small. More concretely, if every node stays with probability $\Delta^{-\Omega(1)}$ and nodes of distance $\Omega(1)$ are independent, then the remainder graph has connected components of size $\operatorname{poly}\Delta \cdot \log n$ with probability $1 - n^{-\Omega(1)}$.

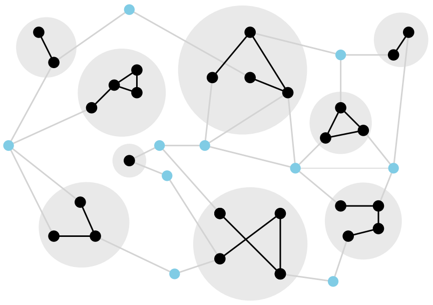


Figure 3.6: A shattered graph. Light blue nodes are nodes that succeeded (hence have committed to a solution). The remaining (black) nodes form small connected components.

As explained by [27, 28], this approach can be used to circumvent the *union bound barrier*, which is a “fundamental barrier in randomized distributed algorithms” [27] that “refers to the limitations attendant to any analysis that employs the union bound to upper bound the global probability of failure” [28]: For a local randomized algorithm, one often has the property that in each round every node tries to commit to a solution and fails to do so with some constant probability p. Thus, all nodes succeed after $O(1/p)$ rounds in expectation. However, to conclude that with probability $1 - n^{-\Omega(1)}$ all nodes succeed, the union bound needs each node to succeed with probability $1 - n^{-\Omega(1)}$, which requires $\Omega(\log_{1/p} n)$ iterations.

The shattering technique remedies this situation as follows. The randomized shattering algorithm with failure probability $1 - \Omega(1)$ at each node is executed for $\operatorname{poly}\log\Delta$ rounds only instead of $\log n$. This will make some of the nodes not succeed. However, not too many. If nodes of sufficiently large distance are independent and every node fails, and hence stays, with probability $\Delta^{-\Omega(1)}$, the size of remaining components can be shown to have size $\operatorname{poly}\Delta \cdot \log n$ with high probability. As pointed out by [109], this can be illustrated by a simple intuition based on Galton-Watson branching processes: assuming the nodes' failures to be independent, the graph is expected to shatter into small pieces as soon as the probability of a node failing goes below $1/\Delta$. Since nodes' failures are not independent, we need some slack of $\Delta^{-\Omega(1)}$ in the failure probability.

We next provide a formal proof of the shattering lemma. The idea origins from [27, 28] and can be found in similar versions in various papers, especially in [109, 111] and also in [115, Appendix A].

Lemma 3.3 (The Shattering Lemma). *Let $G = (V, E)$ be a graph with maximum degree Δ. Consider a process which generates a random subset $B \subseteq V$ such that $Pr[v \in B] \leq \Delta^{-c_1}$, for some constant $c_1 \geq 1$, and such that the random variables $1(v \in B)$ depend only on the randomness of nodes within at most c_2 hops from v, for all $v \in V$, for some constant $c_2 \geq 1$. Then, for any constant $c_3 \geq 1$, we have the following three properties:*

(i) $G[B]$ has size at most $O\big(\log_\Delta n\Delta^{2c_2}\big)$ with probability at least $1 - n^{-c_3}$.

(ii) With probability at least $1 - n^{-c_3}$, each connected component of $G[B]$ admits a $(\lambda, O(\log^{1/\lambda} n \cdot \log^2 \log n))$ network decomposition, for any integer $\lambda \geq 1$, which w.h.p. can be computed in $\lambda \log^{1/\lambda} n \cdot 2^{O(\sqrt{\log\log n})}$ rounds.

(iii) With probability $1 - O(\Delta^{c_2}) \cdot e^{-\Omega(n\Delta^{-c_3})}$*, the number of edges induced by* B *is* $O(n)$*.*

Proof. Consider the graph $H = G^{[2c_2+1, 4c_2+2]}$ which contains an edge between u and v iff their distance in G is between $2c_2 + 1$ and $4c_2 + 2$. We first show the following claim.

Claim 3.4. *Graph* H *has no connected component* U *with* $|U| \geq t := \frac{c_3}{c_4} \log n$ *with probability at least* $1 - n^{-c_3}$*, for some* $c_4 \geq 1$*.*

Proof. The existence of such a connected set U would imply that $H[B]$ contained a tree on t nodes. There are at most 4^t different such (unlabeled) tree topologies, and each can be embedded into H in less than $n \cdot \Delta^{(4c_2+2)t}$ ways. Moreover, the probability that a particular tree occurs in $H[B]$ is at most $\Delta^{-c_1 \cdot t}$, since all the nodes stay in B with probability at most Δ^{-c_1} independently, as they have distance at least $2c_2 + 1$. A union bound over all trees thus lets us conclude that such a set U exists with probability at most $4^t n \Delta^{(4c_2+2)t} \Delta^{-c_1 \cdot t} \leq n^{-c_3}$, for some choice of c_4, which proves the claim. □

We observe that a connected component S in $G[B]$ of size $k\Delta^{2c_2}$ implies the existence of a connected set U in $H[B]$ of size k. To that end, we greedily add nodes from S to U one by one, each time discarding all (at most Δ^{2c_2} many) nodes within $2c_2$ hops of the added vertex. Property (i) thus follows from Claim 3.4.

To prove (ii), we first compute a $(4c_2 + 3, \Theta(\log \log n))$-ruling set on G in $O(\log \log n)$ rounds, w.h.p., using either the algorithm of Schneider et al. [220] and Gfeller and Vicari [108] (see [28, Table IV]) or the algorithm of Ghaffari [111, Lemma 2.2]. This in particular gives us such a ruling set for each connected component C of B (with regard to the distances in G). Recall that an (α, β)-ruling set for C is a set $R_C \subseteq C$ of nodes where each two have distance at least α, while for each vertex in C, there is at least one vertex in

R_C within β hops. Then, each vertex joins the cluster of its nearest ruling set vertex in R_C (breaking ties arbitrarily using IDs). We contract the clusters into super-nodes, and connect two super-nodes if their clusters contain adjacent nodes. The following computations will only take place on the super-nodes; we can simulate one round of this in $O(\log\log n)$ rounds on G, since each of these super-nodes has radius $O(\log\log n)$ in G.

We next show that every component consisting of super-nodes with probability at least $1 - n^{-c_3}$ has at most at most $O(\log_\Delta n)$ super-nodes. For the analysis, we add edges to make R_C connected (if it is not) in H (so that there is a path of length at most $2c_2 + 1$ but no path of length at most $4c_2 + 1$). See [28, Page 19, Steps 3 and 4]. Due to Claim 3.4, the resulting connected set, and hence R_C, has size at most $O(\log_\Delta n)$.

We then compute a $(\lambda, \log^{1/\lambda} n \cdot \log\log n)$ network decomposition on each connected component of this super-graph, independently and all in parallel, using the algorithm described in Lemma 3.2. Notice that when invoking Lemma 3.2, we are now working on graphs of size $O(\log_\Delta n)$, which means the network decomposition that we obtain has parameters $(\lambda, O(\log^{1/\lambda} n \cdot \log\log n))$ and it runs in $\lambda \log^{1/\lambda} n \cdot 2^{O(\sqrt{\log\log n})}$ rounds on the super-graph, and hence also on G. In the end, we extend this decomposition to the original graph on nodes of (this connected component of) B, where each vertex of B belongs to the block of the network decomposition where its contracted cluster is. This increases the radius of each block by the radius of the contracted cluster, which is at most a factor $O(\log\log n)$. This thus leads to a $\big(\lambda, O(\log^{1/\lambda} n \cdot \log^2 \log n)\big)$ network decomposition.

Note that there is one subtlety. The network decomposition algorithm works under the assumption that the graph has size $O(\log n)$, and hence that all the nodes it sees have unique $O(\log\log n)$-bit IDs. This is not the case, however. To achieve that, we will compute a proper coloring in the power graph of the super-graph, with power

$\lambda \log^{1/\lambda} n \cdot 2^{O(\sqrt{\log\log n})}$, which works in $O(\log^* n)$ rounds, and hence will not affect the overall round complexity. We refer to [53, Lemma 1.2] and [28, Remark 3.6] for a more thorough discussion.

Finally, observe that each edge e survives if both its endpoints survive, which happens with probability at most $2\Delta^{-c_1}$. Hence, the expected number of surviving edges is at most $n\Delta^{1-c_1}$. A Chernoff bound with bounded dependence (see, e.g., [212]) concludes the proof of (iii). □

3.2 Local Simulation in All-to-All Models

3.2.1 Lenzen's Routing

One central and frequently used communication primitive for designing CC algorithms is the routing algorithm of Lenzen [169, 168], colloquially known as *Lenzen's routing*. It allows to exchange messages in $O(1)$ rounds, as long as each vertex v is the source and the destination of at most $O(n)$ messages of size $O(\log n)$, or, put differently, $O(n \log n)$ bits of information.

We use Lenzen's routing implicitly in several places, mainly in two different variants.

Sparse Graphs: Whenever a graph has $O(n)$ edges, any problem can be solved easily in $O(1)$ many CC rounds as follows. Every node sends its incident edges to a dedicated node (for instance, the one with the smallest ID) using Lenzen's routing. Note that trivially, no node sends or receives more than $O(n)$ edges, hence more than $O(n)$ messages of size $O(\log n)$. This dedicated node then computes a solution offline and shares it with all other nodes.

Simulation of LOCAL Algorithms: A single-round LOCAL algorithm with messages of size at most $O(\log n)$ (in other words, a single-round CONGEST algorithm) can be implemented in CC in

$O(1)$ rounds: each node sends and receives at most Δ messages, hence at most $O(\Delta \log n) = O(n \log n)$ bits.

Similarly, if $\Delta^r s = O(n \log n)$, an r-round LOCAL algorithm with messages of size at most s can be simulated in CC in $O(1)$ rounds.

3.2.2 Graph Exponentiation

Many LOCAL algorithms can be simulated in the same number of rounds in the CC and MPC models using a standard simulation technique [151, 126]. An interesting question is whether one can use the additional power of global all-to-all communication to (substantially) speed up the corresponding round complexities from LOCAL. One partial answer was given by the very natural and intuitive *graph exponentiation* approach by Lenzen and Wattenhofer [169], which is frequently applied in the design of algorithms in the CC and MPC models. It usually leads to an exponential speedup.

The main idea behind it is as follows: An r-round LOCAL algorithm can be seen as a function that maps a vertex and its r-hop neighborhood to an output for that vertex. Hence, to determine the output of an r-round LOCAL algorithm in MPC or CC for a vertex v, it suffices if a single node knows v's r-hop neighborhood. The goal is to design a technique that allows to learn the 2^i-hop neighborhood of every vertex in i rounds. Suppose that every vertex knows its 2^{i-1}-hop neighborhood in iteration $i-1$. Then, in iteration i, each vertex can inform the vertices in its 2^{i-1}-hop neighborhood of the topology of its 2^{i-1}-hop neighborhood.

The main difficulty is that the amount of information in the r-hop neighborhood of a single vertex can be as high as $\Theta(n^2)$. We need to ensure that the graph is sparse enough so that fast learning of the neighborhood is possible. We briefly discuss the implications for graph exponentiation in CC and MPC, respectively.

Graph Exponentiation in CC

If $\Delta^r = O(n)$, so that every r-hop neighborhood has linear size, it is possible to achieve an exponential speedup in the round complexity compared to that of LOCAL, using Lenzen's routing to gather r-hop neighborhoods.

Graph Exponentiation in Sublinear-Memory MPC

The memory restrictions in the sublinear regime of MPC impose the following two fundamental barriers for this graph exponentiation approach. One needs to carefully design graph exponentiation techniques that do circumvent these difficulties.

Local Memory Barrier: Even for $r = 1$, the r-hop neighborhood of a vertex may have size $\Omega(m)$, exceeding the local memory $\mathcal{S}$ of a node.

Global Memory Barrier: Storing the neighborhood of each vertex on its corresponding node leads to storing overlapping neighborhoods, and hence redundant copies of vertices and edges, and thus a total aggregated memory $\mathcal{MS}$ that is significantly larger than the input size m.

3.2.3 Sparsification and Opportunistic Speedup

The graph exponentiation technique seems to be limited to exponential speedup for CC and MPC compared to LOCAL. We next present a method that potentially gives rise to a super-exponential speedup in CC: under certain conditions, the LOCAL algorithm can be accelerated to run in $O(1)$ rounds.

The output of a node v for a r-round LOCAL algorithm $\mathcal{A}$ may depend on the whole r-hop neighborhood of v, thus on up to Δ^r nodes. To efficiently simulate $\mathcal{A}$ in CC, a strategy is to sparsify the algorithm $\mathcal{A}$ so that the number of nodes a node has to explore to decide its output is significantly smaller than Δ^r. This notion of sparsification of LOCAL algorithms is a key idea behind [110, 123, 71].

We explain how to use this sparsification technique for LOCAL algorithms to obtain constant-round solutions in CC. Lemma 3.5, presented below, summarizes the criteria for this method to work. We note that a somewhat similar idea was at the heart of the $O(1)$-round minimum spanning tree algorithm of [150].

Let ℓ_{in} denote the number of bits needed to represent the random bits and the input for executing $\mathcal{A}$ at a node and let ℓ_{out} denote the number of bits needed to encode the output of $\mathcal{A}$ at a node. We assume that each node v initially knows a set $N_*(v) \subseteq N(v)$ such that throughout the algorithm $\mathcal{A}$, each node v only receives information from nodes in $N_*(v)$. We use Δ_* for the size of the biggest set $N_*(v)$. Note that it is possible that $u \in N_*(v)$ but $v \notin N_*(u)$. In this case, during the execution of $\mathcal{A}$, all messages sent via the edge $\{u, v\}$ are from u to v. We use $N_*^k(v)$ to denote the set of all nodes u such that there is a path $(v = w_0, w_1, \ldots, w_{x-1} = u)$ such that $x \leq k$ and $w_i \in N_*(w_{i-1})$ for each $i \in [1, x-1]$. Intuitively, if $\mathcal{A}$ takes r rounds, then all information needed for vertex $v \in V$ to calculate its output is the IDs and the inputs of all nodes in $N_*^r(v)$.

Lemma 3.5 (Opportunistic Speedup)**.** *An r-round LOCAL algorithm $\mathcal{A}$ can be simulated w.h.p. in $O(1)$ rounds in CC if*

(i) $\Delta_*^r \log\left(\Delta_* + \frac{\ell_{\mathsf{in}}}{\log n}\right) = O(\log n)$,

(ii) $\ell_{\mathsf{in}} = O(n)$, *and*

(iii) $\ell_{\mathsf{out}} = O(\log n)$.

Proof. Assume $\mathcal{A}$ is in the following canonical form. Each node first generates certain amount of local random bits, and then collects all information in its r-neighborhood. This information includes not only the graph topology, but also IDs, inputs, and the random bits of these nodes. After gathering this information, each node computes its output offline based on the gathered information.

Consider the following procedure in CC for simulating $\mathcal{A}$. In the first phase, for each ordered node pair (u, v), with probability p to be determined, u sends all its information to v. The information can be encoded in $\Theta(\Delta_* \log n + \ell_{\mathsf{in}})$ bits. This includes the local input of u, the local random bits needed for u to run $\mathcal{A}$, and the list of IDs in $N_*(u) \cup \{u\}$. In the second phase, for each ordered node pair (u, v), if v has gathered all the required information to calculate the output of $\mathcal{A}$ at u, then v sends the output of $\mathcal{A}$ at u to u.

At first sight, the procedure seems to take $\omega(1)$ rounds. However, if we set $p = \Theta\left((\Delta_* + \ell_{\mathsf{in}}/\log n)^{-1}\right)$, the expected number of $O(\log n)$-bit messages sent from or received by a node is $np \cdot \Theta(\Delta_* + \ell_{\mathsf{in}}/\log n) = O(n)$.

More precisely, let X_u be the number of nodes u sends its local information to in the first phase; similarly, let Y_v be the number of nodes sending their local information to node v. We have $\mathbb{E}[X_u] = np$ for each $u \in V$, and $\mathbb{E}[Y_v] = np$ for each $v \in V$. By a Chernoff bound, when $np = \Omega(\log n)$, with probability $1 - e^{-\Omega(np)} = n^{-\Omega(1)}$, we have $X_u = O(np)$, for each $u \in V$, and $Y_v = O(np)$ for each $v \in V$. That is, the number of $O(\log n)$-bit messages sent from and received by a node w.h.p. is at most $np \cdot \Theta(\Delta_* + \ell_{\mathsf{in}}/\log n) = O(n)$.

We verify that $np = \Omega(\log n)$. Condition (i) implicitly requires $\Delta_* = O(\log n)$, and (ii) ensures $\ell_{\mathsf{in}} = O(n)$. Therefore,

$$np = \Theta\left(\frac{n}{\Delta + \frac{\ell_{\mathsf{in}}}{\log n}}\right) = \Omega(\log n).$$

Thus, we can route all messages in $O(1)$ rounds using Lenzen's routing; the first phase hence can be implemented in $O(1)$ CC rounds.

Condition (iii) guarantees that $\ell_{\mathsf{out}} = O(\log n)$, and so the messages in the second phase can be sent directly in $O(1)$ rounds. What remains to do is to show that w.h.p. for each $u \in V$ there is a node $v \in V$ that receives messages from all nodes in $N_*^r(u)$ during the first phase, and so v is able to calculate the output of u offline.

Let $\mathcal{E}_{u,v}$ be the event that v receives messages from all nodes in $N_*^r(u)$ during the first phase, and define $\mathcal{E}_u$ to be the event that at least one of $\{\mathcal{E}_{u,v} \mid v \in V\}$ occurs. We have $\mathsf{Pr}[\mathcal{E}_{u,v}] \geq p^{\Delta_*^r}$, since $|N_*^r(u)| \leq \Delta_*^r$. Thus, $\mathsf{Pr}[\mathcal{E}_u] \geq 1 - (1 - p^{\Delta_*^r})^n$.

Condition (i) guarantees that $\Delta_*^r \log(\Delta_* + \ell_{\mathsf{in}} / \log n) = O(\log n)$. By setting

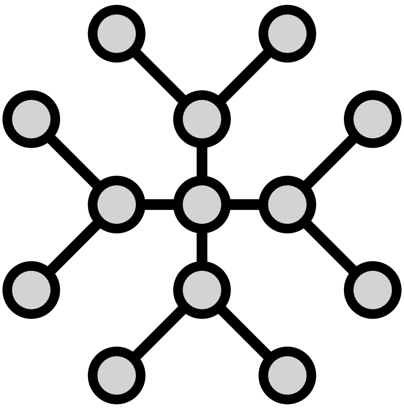
$$p = \frac{\varepsilon}{\Delta_* + \frac{\ell_{\mathsf{in}}}{\log n}}$$

for some sufficiently small constant ε, we have $\Delta_*^r \log p \geq -\frac{1}{2} \log n$. This implies $p^{\Delta_*^r} \geq 1/\sqrt{n}$. Therefore,

$$\mathsf{Pr}[\mathcal{E}_u] \geq 1 - (1 - p^{\Delta_*^r})^n = 1 - e^{-\Omega(\sqrt{n})},$$

which means that the simulation w.h.p. computes the correct output for all nodes. $\square$

Part I



LOCAL Model

CHAPTER 4

Local Rounding for Matching

4.1 Introduction

We present the work from the publications 'Improved Deterministic Distributed Matching via Rounding' [99, 100].

4.1.1 Our Results and Related Work

We illustrate the power as well as the flexibility of deterministic rounding by presenting improved distributed algorithms for a number of well-studied matching problems. Our algorithms are simpler, faster, more accurate, and/or more general than their known counterparts.

Approximate Maximum Matching

Our main ingredient is an approximation algorithm for matching.

Theorem 4.1. *There is an $O\big(\log^2 \Delta \cdot \log \frac{1}{\varepsilon} + \log^* n\big)$-round deterministic LOCAL algorithm that computes a $(2+\varepsilon)$-approximate maximum matching, for any $\varepsilon > 0$.*

For constant $\varepsilon > 0$, this $O(\log^2 \Delta + \log^* n)$-round algorithm is significantly faster than the previously best known deterministic constant approximations, especially in low-degree graphs: the $O(\Delta + \log^* n)$-round 2-approximation of Panconesi and Rizzi [205], the $O(\log^4 n)$-round 2-approximation of Hańćkowiak et al. [132], the $O(\log^4 n)$-round (3/2)-approximation of Czygrinow et al. [77, 76], and its extension [75] which finds a $(1+\varepsilon)$-approximation in $\log^{O(1/\varepsilon)} n$ rounds.

Our $O(\log^2 \Delta + \log^* n)$-round algorithm gets close to the lower bound of $\Omega(\log \Delta / \log\log \Delta + \log^* n)$, due to the celebrated results of Kuhn et al. [162, 163, 164], Linial [171], and Naor [194], that holds for any (randomized) constant approximation of matching.

Maximal Matching

Iterative invocation of our matching approximation algorithm yields a maximal matching.

Theorem 4.2. *There is an $O(\log^2 \Delta \cdot \log n)$-round deterministic LOCAL maximal matching algorithm.*

This is the first improvement in about 20 years over the breakthroughs of Hańćkowiak et al., which presented first an $O(\log^7 n)$- [131] and then an $O(\log^4 n)$-round [132] algorithm for the problem of maximal matching. Moreover, plugging in our improved deterministic algorithm in the randomized maximal matching algorithm of Barenboim et al. [27, 28] improves their round complexity from $O(\log \Delta + \log^4 \log n)$ to $O(\log \Delta + \log^3 \log n)$.

Corollary 4.3. *There is an $O(\log \Delta + \log^3 \log n)$-round* LOCAL *algorithm that w.h.p. computes a maximal matching.*

Almost Maximal Matching

We get a faster algorithm for ε-almost maximal matching, a matching that leaves only ε-fraction of edges among unmatched vertices.

Theorem 4.4. *There is a deterministic $O\big(\log^2 \Delta \cdot \log \frac{1}{\varepsilon} + \log^* n\big)$-round* LOCAL *$\varepsilon$-almost maximal matching algorithm, for any $\varepsilon > 0$.*

This theorem statement is interesting because of two aspects: First, in some practical settings, this almost maximal matching—which practically looks maximal for essentially all vertices—may be as useful as maximal matching, especially since it can be computed much faster. Notice that the complexity grows slowly as a function of ε. Thus, we can set ε quite small. By setting $\varepsilon = \Delta^{-\operatorname{poly}\log \Delta}$, we get an algorithm that, in $O(\operatorname{poly}\log \Delta + \log^* n)$ rounds, produces a matching that seems to be maximal for almost all vertices, even if they look up to their $\operatorname{poly}\log \Delta$-hop neighborhood.

Second, this faster almost maximal matching algorithm sheds some light on the difficulties for maximal matching: Balliu et al. [19] recently proved an $\Omega\left(\min\left\{\Delta, \log n / \log\log n\right\}\right)$-round lower bound for deterministic LOCAL maximal matching algorithms, confirming a conjecture by Göös et al. [127] that "*there should be no $o(\Delta) + O(\log^* n)$ algorithm for computing a maximal matching.*" Since our ε-almost maximal matching algorithm clearly undercuts this bound, the challenge for maximal matching must stem from the fact that *all* matchable edges must be matched.

Edge Dominating Set

As a corollary of the almost maximal matching algorithm of Theorem 4.4, we get a fast algorithm for approximating *minimum edge* *dominating set*, which is the smallest set of edges such that any edge shares at least one endpoint with them.

Corollary 4.5. *There is an $O(\log^2 \Delta \cdot \log \frac{\Delta}{\varepsilon} + \log^* n)$-round deterministic* LOCAL *algorithm for a $(2+\varepsilon)$-approximate minimum edge dominating set, for any $\varepsilon > 0$.*

Previously, the fastest algorithms ran in $O(\Delta + \log^* n)$ rounds [205] or $O(\log^4 n)$ rounds [132], providing 2-approximations. Moreover, Suomela [224] provided roughly 4-approximations in $O(\Delta^2)$ rounds, in a restricted variant of the LOCAL model with port numberings.

Weighted Matching and (Weighted) b-Matching

An interesting aspect of the method we use is its flexibility and generality. In particular, the algorithm of Theorem 4.1 can be easily extended to a $(2+\varepsilon)$-approximation of maximum *weighted* matching, and more interestingly, of maximum weighted *b-matching*. Throughout, W will denote the maximum normalized edge weight.

Theorem 4.6. *There is an $O(\log^2 \Delta \cdot \log \frac{1}{\varepsilon} \cdot \log_{1+\varepsilon} W + \log^* n)$-round deterministic* LOCAL *algorithm for a $(2+\varepsilon)$-approximate maximum weighted matching, or b-matching, for any $\varepsilon > 0$ and $W \geq 2$.*

To the best of our knowledge, this is the first deterministic distributed algorithm for b-matching. Moreover, even in the case of standard matching, it improves on the previously best-known algorithm: a deterministic algorithm for $(6+\varepsilon)$-approximation of maximum weighted matching was provided by Panconesi and Sozio [206], with a round complexity of $O\big(\log^4 n \cdot \log_{1+\varepsilon} W\big)$. However, that deterministic algorithm does not extend to b-matching.

Recently, Ahmadi et al. [2] have slightly improved and generalized our rounding technique to get a $(1+\varepsilon+o(1))$-approximate maximum weighted matching in $O\left(\log(\Delta W)/\varepsilon^2 + \log^2 \Delta/\varepsilon + \log^* n/\varepsilon\right)$.

Further Related Work

Aside from the deterministic algorithms discussed above, there is a long line of research on randomized distributed approximation algorithms for matching: for the unweighted case, [145] provide a 2-approximation in $O(\log n)$ rounds, and [176] a $(1+\varepsilon)$-approximation in $O(\log n)$ rounds for any constant $\varepsilon > 0$. For the weighted case, [227, 178, 176] provide successively improved algorithms, culminating in the $O(\log 1/\varepsilon \cdot \log n)$-round $(2+\varepsilon)$-approximation of [176]. Moreover, [157] present an $O(\log n)$-round randomized algorithm for 2-approximate weighted b-matching.

4.1.2 Notation and Preliminaries

Matching and Fractional Matching: An integral matching M is a subset of E such that $e \cap e' = \varnothing$ for all $e \neq e' \in M$. It can be seen as an assignment of values $x_e \in \{0,1\}$ to edges, where $x_e = 1$ iff $e \in M$, such that $c_v := \sum_{e \in E(v)} x_e \leq 1$ for all $v \in V$. Here, $E(v)$ denotes the set of edges incident to vertex v. When the condition $x_e \in \{0,1\}$ is relaxed to $0 \leq x_e \leq 1$, such an assignment is called a fractional matching. We call a fractional matching 2^{-i}-fractional for an $i \in \mathbb{N}$ if $x_e \in \{0\} \cup \left\{2^{-j} : 0 \leq j \leq i\right\}$. Notice that a 2^{-0}-fractional matching is simply an integral matching. Given a fractional matching, we call a vertex v *half-tight* if $c_v = \sum_{e \in E(v)} x_e > \frac{1}{2}$. Given a fractional b-matching, we call a vertex v *half-tight* if $c_v = \sum_{e \in E(v)} x_e > \frac{b_v}{2}$.

Variants of Matching: An integral matching is called maximal if we cannot add any edge to it without violating the constraints. For $\varepsilon > 0$, we say that $M \subseteq E$ is an ε-almost maximal matching if after removing the edges in and incident to M from G, at most $\varepsilon|E|$ edges remain, thus if $|N^+(M)| \geq (1-\varepsilon)|E|$ for $N^+(M) := \{e \in E \mid \exists e' \in M : e \cap e' \neq \varnothing\}$.

A b-matching for b-values $\{1 \leq b_v \leq d(v) : v \in V\}$ is an assignment of values $x_e \in \{0,1\}$ to edges $e \in E$ such that $\sum_{e \in E(v)} x_e \leq b_v$ for all

$v \in V$. Throughout, $d(v)$ denotes the degree of v in G. A fractional b-matching only requires $0 \leq x_e \leq 1$.

In an unweighted graph, a matching M^* is called maximum if it is a largest matching in terms of cardinality. For any $c \geq 1$, we say that a matching is c-approximate if $c \sum_{e \in E} x_e \geq |M^*|$ for a maximum matching M^*. In a weighted graph where each edge e is assigned a weight $w_e \geq 0$, we say that M^* is a maximum weighted matching if it is a matching with maximum weight $w(M^*) := \sum_{e \in M^*} w_e$. An integral matching M is a c-approximate weighted matching if $c \sum_{e \in M} w_e \geq w(M^*)$.

An edge dominating set is a set $D \subseteq E$ such that for every $e \in E$ there is an $e' \in D$ such that $e \cap e' \neq \varnothing$. A minimum edge dominating set is an edge dominating set of minimum cardinality.

We now state some simple facts about matchings.

Lemma 4.7. *For a maximal matching M and a maximum matching M^*, we have the following two properties:*

(i) $|M| \geq \frac{1}{2\Delta - 1}|E|$, and (ii) $\frac{1}{2}|M^*| \leq |M| \leq |M^*|$.

Proof. For (i), observe that an edge has at most $2\Delta - 2$ neighbors. Hence, per edge that joins a maximal matching, at most $2\Delta - 2$ cannot join. In other words, out of $2\Delta - 1$ edges, at least one will be in a maximal matching. For (ii), we want to see that a maximal matching is a 2-approximation. For every edge $e \in M^*$, we have that either e or at least one of the edges incident to e is in the maximal matching M, due to maximality. On the other hand, every edge $e \in M$ can have at most 2 edges incident to it that are in M^*. $\square$

Lemma 4.8 (Panconesi and Rizzi [205]). *There is an $O(\Delta + \log^* n)$-round deterministic LOCAL algorithm for maximal matching.*

Remark 4.9. *Note that if a q-coloring is provided, the algorithm from Lemma 4.8 runs in $O(\Delta + \log^* q)$ rounds. In particular, for bipartite graphs (where the bipartition is given), the round complexity is simply $O(\Delta)$. This gives rise to the following frequently used trick: If their algorithm is applied repeatedly, by precomputing an $O(\Delta^2)$-coloring using Linial's algorithm [172] in $O(\log^* n)$ rounds, one can replace the $O(\log^* n)$ term in each iteration by $O(\log^* \Delta)$, at the cost of initially spending $O(\log^* n)$ rounds once.*

4.1.3 Overview and Outline

Our main ingredient is a deterministic local rounding technique for matching. To the best of our knowledge, this is the first known deterministic distributed rounding method. Somewhat similar approaches have been studied in [131, 132, 133] that can be interpreted as approximate rounding algorithms, although they are not explicitly phrased in this way. Their key component is, roughly speaking, to decompose edges of any regular graph into two groups, say green and blue, so that almost all vertices see a fair split of their edges into the two colors. This is a special case of rounding for regular graphs.

To present the flavor of our deterministic rounding method, here we overview it in a simple special case: we describe an $O(\log^2 \Delta)$-round algorithm for a constant approximation of the maximum unweighted matching in 2-colored bipartite graphs. The precise algorithm, as well as the extensions to general graphs, better approximations, and more general problems appear later.

In approximating the maximum matching, the main challenge is finding an *integral* matching with such an approximation guarantee; finding a *fractional* matching with such an approximation is trivial.

Fractional Solution

There is a simple $O(\log \Delta)$-round greedy algorithm that works as follows.

Greedy Algorithm: We call a vertex v *half-tight* if its value $c_v := \sum_{e \in E(v)} x_e > 1/2$, where $E(v) := \{e \in E \colon v \in e\}$ denotes the set of edges incident to vertex v. Initially, we set $x_e = 1/\Delta$ for all edges e. Then, for $\log \Delta$ iterations, we freeze all edges with at least one half-tight endpoint and double the value of all non-frozen edges.

Analysis: One can see that this produces a 4-approximation, as the following neat "blaming" argument shows. To that end, we give 1 dollar to each edge e in a maximum matching M^* and ask it to redistribute this money among all edges in such a way that no edge e' receives more than $4x_{e'}$ dollars. Consider a maximum matching M. Every edge $e \in M$ in the maximum matching blames one dollar on a tight endpoint in the fractional matching. Note that there must be at least one, as otherwise the fractional matching could be further increased. Every tight endpoint that has received a dollar now blames it on all its incident edges proportionally to their value, every edge getting at most twice its value. Since an edge has potentially two endpoints distributing blame on it, in total it receives at most a quadruple of its value in dollars. Since overall $|M|$ dollars have been distributed, this proves that M is at most 4 times larger than the fractional matching. This is also illustrated in Figure 4.1.

See [156, 163, 119], which discuss other distributed aspects of linear programs.

Gradual Rounding

We gradually round this fractional matching $x = \{x_e \colon e \in E\}$ bit by bit to an integral matching $x' = \{x'_e \colon e \in E\}$ while ensuring that $\sum_e x'_e \geq (\sum_e x_e)/C$, for some constant C.

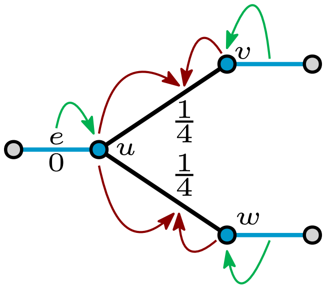


Figure 4.1: An edge e in the maximum matching (drawn in blue) blames its dollar on its tight (blue) endpoint u (depicted as green arrows). This vertex in turn distributes this dollar to its incident edges proportionally to their value: 0 to edge e with value 0 and 50 cents to each of the two edges with value $\frac{1}{4}$ (shown as red arrows). These edges, however, might also receive (no more than) 50 cents from their (tight) endpoints v and w, respectively.

We have $O(\log \Delta)$ rounding phases, each of which takes $O(\log \Delta)$ rounds. In each phase, we get rid of the most-fractional values and thereby move closer to integrality. The initial fractional matching has[1] only values $x_e = 2^{-i}$ for $i \in \{0, \ldots, \lceil \log \Delta \rceil\}$ or $x_e = 0$. In the phase k, we partially round the edge values $x_e = 2^{-i}$ for $i = \lceil \log \Delta \rceil - k + 1$. Some of these edges will be raised to $x_e = 2 \cdot 2^{-i}$, while others are dropped to $x_e = 0$. The choices are made in a way that keeps $\sum_e x_e$ essentially unchanged, as we explain next.

Consider the graph H edge-induced by edges e with value $x_e = 2^{-i}$. For the sake of simplicity, suppose all vertices of H have even degree. Dealing with odd degrees requires some delicate care, but it will not incur a loss worse than an $O(2^{-i})$-fraction of the total value. In this

[1]Any fractional maximum matching can be transformed to this format, with at most a factor 2 loss in the total value: simply round down each value to the next power of 2, and then drop edges with values below $2^{-(\lceil \log \Delta \rceil + 1)}$.

even-degree graph H, we effectively want that for each vertex v of H, half of its edges raise $x_e = 2^{-i}$ to $x_e = 2 \cdot 2^{-i}$ while the others drop it to $x_e = 0$. For that, we generate a degree-2 graph H' by computing the 2-decomposition: Graph H' is simply a set of cycles of even length, as H was bipartite.

In each cycle of H', we would want that the raise and drop of edge weights is alternating. That is, odd-numbered, say, edges are raised to $x_e = 2 \cdot 2^{-i}$ while even-numbered edges are dropped to $x_e = 0$. This would keep x a valid fractional matching—meaning that each vertex v still has $\sum_{e \in E(v)} x_e \leq 1$—because the summation $\sum_{e \in E(v)} x_e$ does not increase, for each vertex v. Furthermore, it would keep the total weight $\sum_e x_e$ unchanged. If the cycle is shorter than length $O(\log \Delta)$, this raise/drop sequence can be identified in $O(\log \Delta)$ rounds. For longer cycles, we cannot compute such a perfect alternation in $O(\log \Delta)$ rounds. However, one can do something that does not lose much[2]: imagine that we chop the longer cycles into edge-disjoint paths of length $\Theta(\log \Delta)$. In each path, we drop the endpoints to $x_e = 0$ while using a perfect alternation inside the path. An example can be found in Figure 4.2.

These border settings mean that we lose $\Theta(1/\log \Delta)$-fraction of the weight. Thus, even over all the $O(\log \Delta)$ iterations, the total loss is only a small constant fraction of the total weight.

4.2 Matching in Bipartite Graphs

The common denominator of our results is a deterministic $O(\log^2 \Delta)$-round algorithm for a constant approximation of the maximum unweighted matching in 2-colored bipartite graphs.

[2]Our algorithm actually does something slightly different, but describing this ideal procedure is easier.

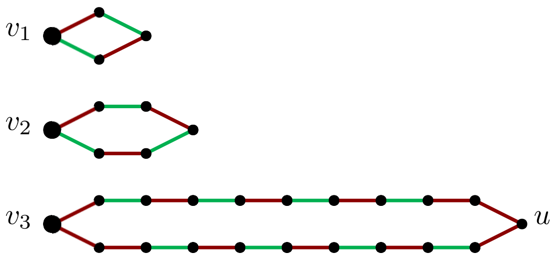


Figure 4.2: Three copies v_1, v_2, v_3 of a vertex v with its even-length cycles in H' and their raise/drop sequence: red means set to 0 and green means increased by a factor 2. The cycles of v_1 and v_2 are short enough to identify a perfect alternation. The cycle of v_3 is too long and has to be chopped at u, say, into two alternating paths with dropped edge values at the border.

Lemma 4.10. *There is an $O(\log^2 \Delta)$-round deterministic LOCAL algorithm for a c-approximate maximum matching in a 2-colored bipartite graph, for some constant c.*

The proof of Lemma 4.10 is split into three parts. In the first step, explained in Section 4.2.1, we compute a $2^{-\lceil \log \Delta \rceil}$-fractional 4-approximate maximum matching in $O(\log \Delta)$ rounds. The second step, which constitutes the heart of our approach and is formalized in our *Rounding Lemma* in Section 4.2.2, is a method to round these fractional values to almost integrality in $O(\log^2 \Delta)$ rounds. In the third step, presented in Section 4.2.3, we resort to a simple constant-round algorithm to transform the almost integral matching that we have found up to this step into an integral matching. As a side remark, we note that we explicitly state some of the constants, for the sake of readability. These constants are not the focus of this work, and we have not tried to optimize them.

4.2.1 Step 1: Fractional Matching

We show that a simple greedy algorithm already leads to a fractional 4-approximate maximum matching.

Lemma 4.11. *There is an $O(\log \Delta)$-round deterministic LOCAL algorithm that computes a $2^{-\lceil \log \Delta \rceil}$-fractional 4-approximate maximum matching.*

Proof. Initially, set $x_e = 2^{-\lceil \log \Delta \rceil}$ for all $e \in E$. Notice that this trivially satisfies the constraints $c_v = \sum_{e \in E(v)} x_e \leq 1$. Then, we iteratively raise the value of all loose edges by a factor 2 until they have at least one half-tight endpoint. This can be done in $O(\log \Delta)$ rounds, since at the latest when the value of an edge is $1/2$, both endpoints would be half-tight. Once there is no loose edge left, we have

$$\sum_{e \in E} x_e = \frac{1}{2} \sum_{v \in V} c_v \geq \frac{1}{2} \sum_{e=\{u,v\} \in M^*} (c_u + c_v) > \frac{|M^*|}{4}$$

for a maximum matching M^*. □

4.2.2 Step 2: Main Rounding

The Rounding Lemma, is a method that gradually turns a 2^{-i}-fractional matching into a 2^{-i+1}-fractional one, bit by bit, for decreasing values of i, while only worsening the approximation ratio by a little.

Lemma 4.12 (Rounding Lemma). *There is an $O\big(\log^2 \Delta\big)$-round deterministic LOCAL algorithm that transforms a $2^{-\lceil \log \Delta \rceil}$-fractional 4-approximate maximum matching in a 2-colored bipartite graph into a 2^{-4}-fractional 14-approximate maximum matching.*

Proof. Iteratively, for $k = 1, \ldots, \lceil \log \Delta \rceil - 4$, in phase k, we get rid of edges e with value $x_e = 2^{-i}$ for $i = \lceil \log \Delta \rceil - k + 1$ by either increasing their values by a factor 2 to $x_e = 2^{-i+1}$ or setting them to $x_e = 0$.

Next, we describe the process for one phase k, thus a fixed i. Let H be the graph induced by the set $E_i := \{e \in E \colon x_e = 2^{-i}\}$ of edges with value 2^{-i} and use H' to denote its 2-decomposition. Notice that H' is a vertex-disjoint union of paths and even-length cycles. Set $\ell = 24 \log \Delta$. We call a path/cycle *short* if it has length at most ℓ, and *long* otherwise. We now process short and long cycles and paths, by distinguishing three cases, as we discuss next. Each of these cases will be done in $O(\log \Delta)$ rounds, which implies that the complexity of one phase is $O(\log \Delta)$. Thus, over all the $O(\log \Delta)$ phases, this rounding algorithm takes $O(\log^2 \Delta)$ rounds.

Case A: Short Cycles

Alternately set the values of the edges to 0 and to 2^{-i+1}. Since the cycle has even length, every vertex has exactly one incident edge whose value is set to 0 and exactly one set to 2^{-i+1}. Hence, the values $c_v = \sum_{e \in E(v)} x_e$ for all vertices v in the cycle remain unaffected by this update. Moreover, the total value of all the edges in the cycle stays the same. See Figure 4.3 for an example.

Case B: Long Cycles and Long Paths

We first orient the edges in a manner that ensures that each maximal directed path has length at least ℓ. This is done in $O(\ell)$ rounds. For that purpose, we start with an arbitrary orientation of the edges. Then, for each $j = 1, \dots, \lceil \log \ell \rceil$, we iteratively merge two (maximal) directed paths of length $< 2^j$ that are directed towards each other by reversing the shorter one, breaking ties arbitrarily. For more details of this orientation step, we refer to [133, Fact 5.2] and to [61, Theorem A.2].

Given this orientation, we determine the new values of x_e as follows.

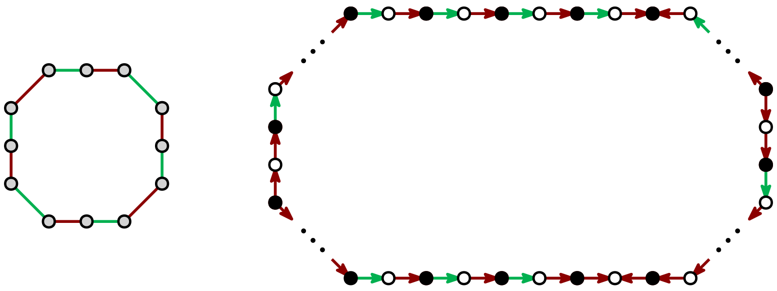


Figure 4.3: The edge values of a short and a long cycle induced by edges in E_i after rounding: green stands for value 2^{-i+1} and red means value 0. In the long cycle, vertices of color 1 and color 2 are depicted as white and black disks, respectively.

Recall that we are given a 2-coloring of vertices. Set the value of all border edges (that is, edges that have an incident edge such that they are either oriented towards each other or away from each other) to 0, increase the value of a non-border edge to 2^{-i+1} if it is oriented towards a vertex of color 1, say, and set it to 0 otherwise. See Figure 4.3 for an example.

Now, we show that this process generates a valid fractional matching while incurring only a small loss in the value. Observe that no constraint is violated, as for each vertex the value of at most one incident edge can be raised to 2^{-i+1} while the other is dropped to 0. Moreover, in each maximal directed path, we can lose at most $3 \cdot 2^{-i}$ in the total sum of edge values. This happens in the case of an odd-length path starting with a vertex of color 2. Hence, we can say that we lose at most a $\frac{3}{\ell}$-fraction of the total sum of the edge values in long cycles and long paths.

Case C: Short Paths

Give the path an arbitrary direction, that is, identify the first and the last vertex. Set the value of the first edge to 2^{-i+1} if the first vertex is loose, and to 0 otherwise. Alternately, starting with value 0 for the second edge, set the value of every even edge to 0 and of every odd edge to 2^{-i+1}. If the last edge should be set to 2^{-i+1} (that is, if the path has odd length) but the last vertex is half-tight, set the value of that last edge to 0 instead. See Figure 4.4 for an example.

We now discuss the validity of the new fractional matching. If a vertex v is in the interior of the path, that is, not one of the endpoints, then v can have at most one of its incident edges increased to 2^{-i+1} while the other one decreases to 0. Hence the summation $c_v = \sum_{e \in E(v)} x_e$ does not increase. If v is the first or last vertex in the path, the value of the edge incident to v is increased only if v was loose, i.e., if $c_v = \sum_{e \in E(v)} x_e \leq \frac{1}{2}$. In this case, we still have $c_v = \sum_{e \in E(v)} x_e \leq 1$ after the increase, as the value of the edge raises by at most a factor 2.

We now argue that the value of the matching has not decreased by too much during this update. For that, we group the edges into blocks of two consecutive edges, starting from the first edge. If the path has odd length, the last block consists of a single edge. It is easy to see that the block value, that is, the sum of the values of its two edges, of every interior (neither first nor last) block is unaffected.

Let v be an endpoint of a path. If v is loose, the value of the block containing v remains unchanged or increases (in the case of an odd-length path ending in v). If v is half-tight, then the value of its block stays the same or decreases by 2^{-i+1}, which is at most a 2^{-i+2}-fraction of the value c_v.

This allows us to bound the loss in terms of these half-tight endpoints. The crucial observation is that every vertex can be endpoint of a short path at most once. This is because, in the 2-decomposition, a vertex can be the endpoint of a path only if it has a degree-1 copy. This happens only if it has odd degree, and in that case, it has exactly one degree-1 copy, hence, also exactly one endpoint of a short path. Therefore, we lose at most a 2^{-i+2}-fraction in $\sum_{v \in V} c_v$ when updating the values in short paths.

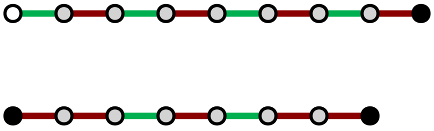


Figure 4.4: The edge values of two short paths induced by edges in E_i after rounding: green edges stands for 2^{-i+1} and red means 0. Half-tight endpoints are depicted as black disks and loose ones as white disks.

Analyzing the Overall Loss Due to Rounding

First, we show that over all the rounding phases, the overall loss is only a constant fraction of the total value $\sum_{e \in E} x_e$.

Let $x_e^{(i)}$ and $c_v^{(i)}$ denote the value of edge e and vertex v, respectively, before eliminating all the edges with value 2^{-i}. Putting together the loss analyses discussed above, we get

$$\begin{aligned}\sum_{e \in E} x_e^{(i-1)} &\geq \sum_{e \in E} x_e^{(i)} - \frac{3}{\ell} \sum_{e \in E} x_e^{(i)} - 2^{-i+2} \sum_{v \in V} c_v^{(i)} \\ &\geq \left(1 - \frac{3}{\ell} - 2^{-i+3}\right) \sum_{e \in E} x_e^{(i)}.\end{aligned}$$

It follows that

$$\begin{aligned}\sum_{e\in E} x_e^{(4)} &\geq \left(\prod_{i=5}^{\lceil\log\Delta\rceil}\left(1-\frac{3}{\ell}-2^{-i+3}\right)\right)\sum_{e\in E} x_e^{(\lceil\log\Delta\rceil)}\\ &\geq \left(\prod_{i=5}^{\lceil\log\Delta\rceil} e^{-2\left(\frac{3}{\ell}+2^{-i+3}\right)}\right)\sum_{e\in E} x_e^{(\lceil\log\Delta\rceil)}\\ &\geq e^{-\frac{1}{4}-16\sum_{i=5}^{\lceil\log\Delta\rceil}2^{-i}}\sum_{e\in E} x_e^{(\lceil\log\Delta\rceil)}\\ &\geq \frac{1}{e^{\frac{5}{4}}}\sum_{e\in E} x_e^{(\lceil\log\Delta\rceil)} \geq \frac{1}{4e^{\frac{5}{4}}}|M^*| \geq \frac{1}{14}|M^*|\end{aligned}$$

for a maximum matching M^*, recalling that we started with a 4-approximate maximum matching. Here, the second inequality holds because $1-y \geq e^{-2y}$ for $y \leq \frac{1}{2}$, and $3/\ell + 2^{-i+3} \leq 1/2$, as $i \geq 5$.

Finally, observe that in all the rounding phases the constraints $c_v = \sum_{e\in E(v)} x_e \leq 1$ are preserved, since the value c_v can increase only if v is loose (and in fact only if the degree-1-copy of v is an endpoint of a short path), which means $c_v \leq 1/2$, and in that case only by at most a factor 2. □

4.2.3 Step 3: Final Rounding

So far, we have an almost integral matching. Next, we round all edges to either 0 or 1, by finding a maximal matching in the subgraph induced by edges with positive value.

Lemma 4.13. *Given a 2^{-4}-fractional 14-approximate maximum matching in a 2-colored bipartite graph, there is deterministic LOCAL algorithm that computes an integral 434-approximate maximum matching in $O(1)$ rounds.*

Proof. In the given 2^{-4}-fractional matching, $x_e \neq 0$ means $x_e \geq 1/16$. Thus, a vertex cannot have more than 16 incident edges with non-zero value in this fractional matching. In this constant-degree subgraph, a maximal matching M can be found in $O(1)$ rounds using the algorithm in Lemma 4.8, recalling that we are given a 2-coloring. We have

$$|M| \geq \frac{|\{e \in E \colon x_e > 0\}|}{31} \geq \frac{1}{31} \sum_{e \in E} x_e$$

by Lemma 4.7 (i). As we started with a 14-approximation, it follows that M is 434-approximate. □

4.3 Matching in General Graphs

We now explain how the approximation algorithm for maximum matchings in 2-colored bipartite graphs can be employed to find approximate maximum matchings in general graphs. The main idea is to transform the given general graph into a bipartite graph with the same edge set in such a way that a matching in this bipartite graph can be easily turned into a matching in the general graph.

4.3.1 Constant-Approximate Maximum Matching

Theorem 4.14. *There is an $O(\log^2 \Delta + \log^* n)$-round deterministic* LOCAL *algorithm for a c-approximate maximum matching, for some constant c.*

Proof. Let $\overrightarrow{E}$ be an arbitrary orientation of the edges E. Split every vertex $v \in V$ into two siblings v_{in} and v_{out}, and add an edge $\{u_{\text{out}}, v_{\text{in}}\}$ to E_B for every oriented edge $(u, v) \in \overrightarrow{E}$. Let $V_{\text{in}} := \{v_{\text{in}} \colon v \in V\}$ be the vertices in color class 1 and $V_{\text{out}} := \{v_{\text{out}} \colon v \in V\}$ the vertices with color 2. By Lemma 4.10, a c-approximate maximum matching M_B in the bipartite graph $B = (V_{\text{in}} \cup V_{\text{out}}, E_B)$ can be computed in $O\big(\log^2 \Delta\big)$ rounds. We now go back to V, that is, we merge v_{in} and v_{out} back to v. This makes the edges of M_B incident to v_{in} or v_{out} now be incident to v, yielding a graph $G' = (V, M_B) \subseteq G$ with maximum degree 2.

We compute a maximal matching M' in G'. Using the algorithm of Lemma 4.8, this can be done in $O(\log^* n)$ rounds. If an poly Δ-coloring of G is provided, which implies a coloring of G' with poly Δ colors, the round complexity of this step is merely $O(\log^* \Delta)$.

It follows from Lemma 4.7 (i) that

$$|M'| \geq \frac{|M_B|}{3} \geq \frac{|M_B^*|}{3c} \geq \frac{|M^*|}{3c}$$

for maximum matchings M_B^* in B and M^* in G, respectively. Thus, M' is a $(3c)$-approximate maximum matching in G. Notice that the last inequality is true since by introducing additional vertices but leaving the edge set unchanged when going from G to B, the maximum matching size cannot decrease. □

4.3.2 $(2 + \varepsilon)$-Approximate Maximum Matching

The approximation ratio of a matching algorithm can be improved from $\Theta(1)$ to $2 + \varepsilon$ easily, by $O\big(\log \frac{1}{\varepsilon}\big)$ repetitions: each time, we apply the algorithm of Lemma 4.14 to the remaining graph, and remove the found matching together with its neighboring edges from the graph.

Proof of Theorem 4.1. Starting with $G_0 = G$, for $i = 0, \ldots, k-1$, where $k = O\big(\log \frac{1}{\varepsilon}\big)$, iteratively compute a c-approximate maximum matching M_i in G_i, using the algorithm of Lemma 4.14. We delete M_i together with its incident edges from the graph, that is, set $G_{i+1} = (V, E(G_i) \setminus N^+(M_i))$.

Now, we argue that the obtained matching $\bigcup_{i=0}^{k-1} M_i$ is $(2 + \varepsilon)$-approximate. To this end, we bound the size of a maximum matching in the remainder graph G_k.

Let M_i^* be a maximum matching in G_i. An inductive argument shows that $|M_i^*| \leq (1-1/c)^i|M^*|$. Indeed, observe

$$\left|M_{i+1}^*\right| \leq |M_i^*| - |M_i| \leq \left(1-\frac{1}{c}\right)|M_i^*| \leq \left(1-\frac{1}{c}\right)^{i+1}|M^*|,$$

where the first inequality holds since otherwise $M_{i+1}^* \cup M_i$ would be a better matching than M_i^* in G_i, contradicting the latter's optimality. For $k = \log_{1-1/c} \frac{\varepsilon}{2(2+\varepsilon)}$, we thus have $|M_k^*| \leq \frac{\varepsilon}{2(2+\varepsilon)}|M^*|$.

As $\bigcup_{i=0}^{k-1} M_i$ is a maximal matching in $G \setminus G_k$ by construction, $\left(\bigcup_{i=0}^{k-1} M_i\right) \cup M_k^*$ is a maximal matching in G. By Lemma 4.7 (ii), this means that

$$\left|\bigcup_{i=0}^{k-1} M_i\right| + |M_k^*| \geq |M^*|/2,$$

hence

$$\left|\bigcup_{i=0}^{k-1} M_i\right| \geq \left(\frac{1}{2} - \frac{\varepsilon}{2(2+\varepsilon)}\right)|M^*| \geq \frac{|M^*|}{2+\varepsilon}.$$

We have $O(\log \frac{1}{\varepsilon})$ iterations, each taking $O(\log^2 \Delta + \log^* n)$ rounds. As observed in Remark 4.9, by precomputing an $O(\Delta^2)$-coloring in $O(\log^* n)$ rounds, the round complexity of each iteration decreases to $O(\log^2 \Delta + \log^* \Delta) = O(\log^2 \Delta)$, overall leading to a round complexity of $O(\log^2 \Delta \cdot \log \frac{1}{\varepsilon} + \log^* n)$. $\square$

Remark 4.15. *The analysis above shows that the matching M computed by the algorithm of Theorem 4.1 is not only $(2+\varepsilon)$-approximate, but also has the property that any matching in the remainder graph (induced by $E \setminus N^+(M)$) can have size at most $\varepsilon|M^*|$ for a maximum matching M^* in G.*

4.3.3 Maximal Matching

If one increases the number of repetitions to $O(\log n)$, the found matching is maximal.

Proof of Theorem 4.2. We apply the c-approximation algorithm of Lemma 4.14 for $k = \log_{1-1/c} \frac{1}{n}$ iterations on the respective remainder graph, as described in the proof of Theorem 4.1. The same analysis (also adopting the notation from there) shows that any matching M_k^* in the remainder graph G_k has $|M_k^*| \leq |M^*|/n < 1$, which means that G_k is an empty graph. But then $\bigcup_{i=1}^{k-1} M_i$ must be maximal. $\square$

4.4 Extensions and Corollaries

4.4.1 Almost Maximal Matching

In Section 4.3 (see Remark 4.15), we have seen how one can obtain a matching that reduces the size of the matching in the remainder graph, that is, the graph after removing the matching and all incident edges, by a constant factor. Intuitively, one would expect that this also reduces the number of remaining edges by a constant factor, which would directly lead to an (almost) maximal matching just by repetitions. However, this is not the case, since not every matched edge removes the same number of edges from the graph, particularly in non-regular graphs. This calls for an approach that weights edges incident to vertices of different degrees differently, which naturally brings into play weighted matchings.

Constant-Approximate Maximum Weighted Matching: We thus first present an algorithm that finds a constant approximation of maximum weighted matching, based on the approximate (unweighted) matching algorithm of Theorem 4.1. Note that this is a faster version of Theorem 4.6. In particular, this algorithm's round complexity does not depend on the maximum weight W.

Lemma 4.16. *There is an $O\big(\log^2 \Delta + \log^* n\big)$-round deterministic* LOCAL *algorithm that computes a* 256*-approximate maximum weighted matching.*

Proof. We assume without loss of generality that the edge weights are normalized, that is, from a set $\{1, \dots, W\}$ for some maximum weight $W \geq 2$. Round the weights w_e for $e \in E$ down to the next power of 8, resulting in weights w'_e. This rounding procedure lets us lose at most a factor 8 in the total weight, as every single edge weight is decreased by at most this. Moreover, it provides us with a decomposition of G into graphs $C_i = (V, E_i)$ with $E_i := \big\{e \in E \colon w'_e = 8^i\big\}$ for $i \in \{0, \dots, \lfloor \log_8 W \rfloor\}$.

In parallel, run the algorithm of Theorem 4.1 with $\varepsilon = 1$ on every C_i to find a 3-approximate maximum matching M_i in C_i in $O(\log^2 \Delta + \log^* n)$ rounds. Observe that the edges in $\bigcup_i M_i$ do not need to form a matching since edges from M_i and M_j for $i \neq j$ can be incident. However, a matching $M \subseteq \bigcup_i M_i$ can be easily obtained by deleting all but the highest-index edge in every such conflict, that is, by removing all edges $e \in M_i$ that have an incident edge $e' \in M_j$ for some $j > i$.

In the following, we argue that the weight of M cannot be too small compared to the weight of $\bigcup_i M_i$ by an argument based on counting in two ways.

Let every edge $e \in (\bigcup_i M_i) \setminus M$ put blame w'_e on an edge in M as follows. Since $e \notin M$, there must be an edge e' incident to e such that $e \in M_i$ and $e' \in M_j$ for some $j > i$. If $e' \in M$, then e blames weight w_e on e'. If $e' \notin M$, then e puts blame w_e on the same edge as e' does.

For an edge $e \in M \cap E_i$ and $j \in [i]$, let n_j denote the maximum number of edges from M_{i-j} that blame e. A simple inductive argument shows that $n_j \leq 2^j$. Indeed, there can be at most two edges from M_{i-1} blaming e, at most one per endpoint of e, and, for $j > 1$, we have

$$n_j \le 2 + \sum_{j'=1}^{j-1} n_{j'} \le 2 + \sum_{j'=1}^{j-1} 2^{j'} = 2^j,$$

since at most two edges in M_{i-j} can be incident to e and at most one further edge can be incident to each edge in $M_{i-j'}$ for $j' < j$, as M_{i-j} is a matching.

Therefore, overall, at most

$$\sum_{j=1}^{i} 2^j 8^{i-j} \le \frac{1}{3} 8^i \le \frac{w'_e}{3}$$

weight is blamed on e. This means that

$$\sum_{e \in (\cup_i M_i) \setminus M} w'_e \le \frac{1}{3} \sum_{e \in M} w'_e,$$

hence

$$\sum_{e \in \cup_i M_i} w'_e \le \frac{4}{3} \sum_{e \in M} w'_e.$$

Taken together, we thus have that

$$\sum_{e \in M^*} w_e \le 8 \sum_{e \in M^*} w'_e \le 24 \sum_{e \in \cup_i M_i} w'_e \le 32 \sum_{e \in M} w'_e \le 256 \sum_{e \in M} w_e$$

for a maximum weighted matching M^*. □

We use our fast weighted maximum approximation algorithm from Lemma 4.16, by assigning appropriately chosen weights, to find a matching that removes a constant fraction of the edges in Lemma 4.17. The main idea is to define the weight of each edge to be the number of its incident edges. This way, an (approximate) maximum weighted matching corresponds to a matching that removes a large number of edges.

Lemma 4.17. *There is an $O\big(\log^2 \Delta + \log^* n\big)$-round deterministic* LOCAL *algorithm for a $\frac{511}{512}$-almost maximal matching.*

Proof. For each edge $e = \{u, v\} \in E$, introduce a weight $w_e = d(u) + d(v) - 1$, and apply the algorithm of Lemma 4.16 to find a 256-approximate maximum weighted matching M in G.

For the weight $w(M^*)$ of a maximum weighted matching M^*, it holds that $w(M^*) \geq |E|$, as the following simple argument based on counting in two ways shows. Let every edge in E put a blame on an edge in M^* that is responsible for its removal from the graph as follows. An edge $e \in M^*$ blames itself. An edge $e \notin M^*$ blames an arbitrary incident edge $e' \in M^*$. Notice that at least one such edge must exist, as otherwise M^* would not even be maximal. In this way, $|E|$ many blames have been put onto edges in M^* such that no edge $e = \{u, v\} \in M^*$ is blamed more than w_e times, as e can be blamed by itself and any incident edge. Therefore, indeed $w(M^*) = \sum_{e \in M^*} w_e \geq |E|$, and, as M is a 256-approximate, it follows that $\sum_{e \in M} w_e \geq \frac{|E|}{256}$.

Now, observe that w_e is twice the number of edges that are deleted when removing e together with its incident edges from G. Since every edge can be incident to at most two matched edges (and thus can be deleted by at most two edges in the matching), in total $|N^+(M)| \geq \frac{1}{2} \sum_{e \in M} w_e \geq \frac{|E|}{512}$ many edges are removed from G when deleting the edges in and incident to M, which proves that M is a $\frac{511}{512}$-almost maximal matching. □

Similarly as in the proof of Theorem 4.1, where we iteratively invoked the constant approximate maximum (unweighted) matching algorithm to gradually decrease the maximum matching size in the remainder graph, we here iteratively apply the constant almost maximal matching algorithm of Lemma 4.17 to successively reduce the number of remaining edges. Via $O\big(\log \frac{1}{\varepsilon}\big)$ repetitions of this, each time removing the found matching and its incident edges, we get an ε-almost maximal matching, thus proving Theorem 4.4.

Proof of Theorem 4.4. Let $G_0 = G$. For $i = 0, \ldots, k = O\big(\log \frac{1}{\varepsilon}\big)$, iteratively apply the algorithm of Lemma 4.17 to G_i to obtain a c-almost maximal matching M_i in G_i. We then remove the found matching as well as its incident edges from the graph, thus set $G_{i+1} = (V, E(G_i) \setminus N^+(M_i))$. It is easy to see that $M := \bigcup_{i=0}^{k-1} M_i$ for $k = \log_c \varepsilon$ is ε-approximate. Indeed, it follows from

$$|E(G_{i+1})| \leq c\,|E(G_i)|$$

that

$$\big|E \setminus N^+(M)\big| = |E(G_k)| \leq c^k |E| \leq \varepsilon |E|.$$

Overall, this takes $O(\log^2 \Delta \cdot \log \frac{1}{\varepsilon} + \log^* n)$ rounds. □

When setting $\varepsilon = 1/n^2$, thus increasing the number of repetitions to $O(\log n)$, we obtain a maximal matching.

Alternative Proof of Theorem 4.2. We invoke the ε-almost maximal matching algorithm of Theorem 4.4 with $\varepsilon = \frac{1}{n^2}$, leading to at most $\frac{1}{n^2}|E| < 1$ remaining edges. □

4.4.2 b-Matching

In this subsection, we explain that only slight changes to the algorithm of Theorem 4.1 are sufficient to make it suitable also for computing approximations of maximum b-matching. To this end, we first introduce an approximation algorithm for maximum b-matching in 2-colored bipartite graphs in Lemma 4.18. Then, we extend this algorithm to work for general graphs, in Lemma 4.22. Finally, in the second part of the proof of Theorem 4.1 presented at the end of this subsection, we show that the approximation ratio can be improved to a value arbitrarily close to 2, simply by repetitions of this constant approximation algorithm.

Lemma 4.18. *There is an $O(\log^2 \Delta)$-round deterministic* LOCAL *algorithm for a c-approximate maximum b-matching in a 2-colored bipartite graph, for some constant c.*

This result is a direct consequence of Lemma 4.19, Lemma 4.20, and Lemma 4.21, which we present next. These lemmas respectively show how a fractional constant approximate b-matching can be found, how this fractional matching can be round to almost integrality, and how these almost integral values can be turned into an integral matching, while only losing a constant fraction of the total value. The proofs are very similar to the ones in Section 4.2, except for the very last step of rounding (Lemma 4.21), which requires one extra step, as we shall discuss.

In the following, we call a vertex $v \in V$ loose if $c_v = \sum_{e \in E(v)} x_e < b_v/2$, and half-tight otherwise. As before, an edge e is called *loose* if either of its endpoints are half-tight.

Lemma 4.19. *There is an $O(\log \Delta)$-round deterministic* LOCAL *algorithm for $2^{-\lceil \log \Delta \rceil}$-fractional 4-approximate maximum b-matching.*

Proof of Lemma 4.19. As in the proof of Lemma 4.11, starting with $x_e = 2^{-\lceil \log \Delta \rceil}$ (and thus $c_v \leq 1 \leq b_v$), the edge values of loose edges are gradually increased until all edges have at least one half-tight endpoint. This takes no more than $O(\log \Delta)$ rounds, since at the latest when the value of an edge incident to v is $b_v/2$, then v becomes half-tight. We employ a simple argument based on counting in two ways to show that this yields a 4-approximation of a maximum b-matching M^*. Let each edge $e \in M^*$ blame one of its half-tight endpoints. In this way, each half-tight vertex v—which by definition has value $c_v = \sum_{e \in E(v)} x_e \geq \frac{b_v}{2}$—is blamed at most b_v times. Let v split this blame uniformly among its incident edges in M^* such that each edge e' is blamed at most twice its value $x_{e'}$. In this way, every edge e' is blamed at most $4x_{e'}$, as it can be blamed by both of its endpoints. It follows that $|M^*| \leq 4 \sum_{e \in E} x_e$. □

Next, we transform this fractional solution into an almost integral solution, which is still a constant approximation.

Lemma 4.20. *There is an $O(\log^2 \Delta)$-round deterministic LOCAL algorithm that transforms a $2^{-\lceil \log \Delta \rceil}$-fractional 4-approximate maximum b-matching in a 2-colored bipartite graph into a 2^{-4}-fractional 14-approximate maximum b-matching.*

Proof of Lemma 4.20. As in the proof of Lemma 4.12, the edges of values 2^{-i} for $i = \lceil \log \Delta \rceil, \ldots, 5$ are eliminated. We derive analogously that the fractional matching obtained at the end is a 14-approximation, observing that changing the condition for half-tightness of a vertex from $c_v \geq \frac{1}{2}$ to $c_v \geq \frac{b_v}{2} \geq \frac{1}{2}$ only helps in the analysis. □

In a final step, the almost integral solution is transformed into an integral one. Notice that for b-matchings, as opposed to standard matchings, the subgraph induced by edges with positive value need not have constant degree. In fact, a vertex $v \in V$ can have up to $16b_v$ incident edges with non-zero value. This prevents us from directly applying the algorithm of Lemma 4.8 to find a maximal matching in the subgraph with non-zero edge values, as this could take $O(\max_v b_v) = O(\Delta)$ rounds.

Lemma 4.21. *Given a 2^{-4}-fractional 14-approximate maximum b-matching in a 2-colored bipartite graph, there is a deterministic LOCAL algorithm that finds an integral 434-approximate maximum b-matching in $O(1)$ rounds.*

Proof. We decompose the edge set induced by edges of positive value in the 2^{-4}-fractional maximum b-matching $\left\{x_e^{(4)} : e \in E\right\}$ into constant-degree subgraphs $C_i = (V, E_i)$, as follows. We make at most b_v copies of vertex v, and we arbitrarily split the edges among these copies in such a way that every copy has degree at most 16. This is done in a manner similar to the 2-decomposition procedure.

In parallel, run the algorithm of Lemma 4.8 on each C_i, in $O(1)$ rounds. This yields a maximal matching M_i for each C_i that trivially, by Lemma 4.7 (i), satisfies the condition $|M_i| \geq |E_i|/31$. Now, let $M := \bigcup_i M_i$. Since each vertex v occurs in at most b_v subgraphs and each M_i is a matching in C_i, vertex v cannot have more than b_v incident edges in M. Thus, indeed, M is a b-matching. Finally, observe that

$$|M| \geq \frac{\left|\left\{e \in E \colon x_e^{(4)} > 0\right\}\right|}{31} \geq \frac{1}{31} \sum_{e \in E} x_e^{(4)},$$

thus that M is 434-approximate. □

A similar argument as in Lemma 4.14 shows that the algorithm for approximate maximum b-matchings in bipartite graphs from Lemma 4.18 can be adapted to work for general graphs.

Lemma 4.22. *There is an $O(\log^2 \Delta + \log^* n)$-round deterministic LOCAL algorithm that computes a c-approximate maximum b-matching, for some constant c.*

Proof. Do the same reduction to a bipartite graph B as in the proof of Lemma 4.14, that is, create an in- and an out-copy of every vertex, and, for an arbitrary orientation of the edges, make each oriented edge incident to the respective copy of the corresponding vertices.

Compute a c-approximate maximum b-matching M_B in B using the algorithm of Lemma 4.18. Merging back the two copies of a vertex into one yields a graph with degree of vertex v bounded by $2b_v$, as v_{in} and v_{out} both can have at most b_v incident edges in M_B. Now, compute a 2-decomposition of this graph. On each component C

with edges $E_C \subseteq M_B$, find a maximal matching M_C in $O(1)$ rounds by the algorithm of Lemma 4.8.

Notice that for each vertex v without a degree-1 copy, its degree is at least halved in $\bigcup_C M_C$ compared to M_B, and thus at most b_v. If a vertex v has a degree-1 copy, then its degree need not be halved. But this can happen only if v's degree in M_B is odd, thus at most $2b_v - 1$. In this case, v has at most $b_v - 1$ degree-2 copies and one degree-1 copy, which means that its degree in $\bigcup_C M_C$ is upper bounded by b_v. We conclude that $\bigcup_C M_C$ is indeed a b-matching.

Moreover, it follows from $|M_C| \geq |E_C|/3$ by Lemma 4.7 (i) that $|\bigcup_C M_C| \geq |M_B|/3 \geq |M_B^*|/(3c) \geq |M^*|/(3c)$ for maximum b-matchings M_B^* in B and M^* in G. Thus, the matching $\bigcup_C M_C$ is $3c$-approximate. □

Proof of Theorem 4.1 for b-matching. Starting with $S_0 = \varnothing$, $G_0 = G$, and $b_v^0 = b_v$ for all $v \in V$, for $i = 0, \dots, k = O\big(\log \frac{1}{\varepsilon}\big)$, iteratively apply the algorithm of Lemma 4.22 to G_i with b-values b_v^i to obtain a c-approximate maximum b-matching M_i in G_i. Update $b_v^{i+1} = b_v^i - d_{M_i}(v)$ and $G_{i+1} = (V, E_{i+1})$ with $E_{i+1} := E_i \setminus \big(M_i \cup \big\{\{u,v\} \in E_i \colon b_v^{i+1} = 0 \text{ or } b_u^{i+1} = 0\big\}\big)$, that is, reduce the b-value of each vertex v by the number $d_{M_i}(v)$ of incident edges in the matching M_i and remove M_i as well as all the edges incident to a vertex with remaining b-value 0 from the graph. The same analysis as in the proof of Theorem 4.1 for standard matchings goes through and concludes the proof. □

Remark 4.23. *The analysis above shows that the b-matching M returned by the b-matching approximation algorithm of Theorem 4.1 is not only $(2+\varepsilon)$-approximate in G, but also has the property that any b-matching in the remainder graph, after removing M and all edges incident to a vertex v with b_v incident edges in M, can have size at most $\varepsilon|M^*|$ for a maximum b-matching M^* in G.*

4.4.3 Weighted Matching and b-Matching

We use the following simple approach to extend our results for matchings and b-matchings from the unweighted to the weighted case: partition the edge set into buckets of edges with almost the same weight, modulo factors of $1+\varepsilon'$ for a small $\varepsilon' > 0$. Iteratively, for decreasing weight buckets, find an approximate maximum (unweighted) matching in each of these buckets and remove this found matching and its incident edges from the graph. As a maximum weighted matching cannot differ by too much from a maximum unweighted matching, if all the weights are roughly the same, this leads to a good approximation. We next make this more precise.

Proof of Theorem 4.6. In the following, we assume without loss of generality that all weights are normalized, thus that they are from the set $\{1, \ldots, W\}$ for some $W \geq 2$. We present the proof for weighted b-matching; the result for weighted standard matching follows directly.

Let $\varepsilon' = \frac{\varepsilon}{8}$ and $k = \lceil \log_{1+\varepsilon'} W \rceil$. We decompose G into $k+1$ many graphs $C_i = (V, E_i)$ for $i \in \{0, \ldots, k\}$ where we define $E_i := \bigl\{e \in E \colon (1+\varepsilon')^{k-i} \leq w_e < (1+\varepsilon')^{k+1-i}\bigr\}$. For $i = 0, \ldots, k$, we iteratively apply the b-matching algorithm of Theorem 4.1 to the graph $C_i' := (V, E_i')$, where the b-values are $b_v^i := b_v - \sum_{j=0}^{i-1} d_{M_j}(v)$, and

$$E_i' := E_i \setminus \left(\bigcup_{j=0}^{i-1} M_j \cup \bigl\{\{u,v\} \in E \colon b_u^i = 0 \text{ or } b_v^i = 0\bigr\} \right),$$

is the graph induced by edges in bucket i after removing the previously found b-matchings M_j for $0 \leq j < i$ and the edges having an endpoint with remaining b-value 0. This results in a b-matching M_i in C_i' in $O\bigl(\log^2 \Delta \cdot \log 1/\varepsilon\bigr)$ rounds. Here, $d_{M_j}(v)$ denotes the number of edges in M_j incident to v. Let

$$R_i := E_i \setminus \left(\bigcup_{j=0}^{i} M_j \cup \left\{ \{u,v\} \in E \colon b_u^{i+1} = 0 \text{ or } b_v^{i+1} = 0 \right\} \right)$$

be the set of edges in E_i that are not in conflict with $\bigcup_{j=0}^{i} M_j$, thus are not part of this matching and also do not have too many incident edges in it.

Let $M := \bigcup_{i=0}^{k} M_j$ be the union of the found matchings and use M^* to denote a maximum weighted b-matching in G. We show that $w(M^*) \leq (2+\varepsilon)w(M)$ by an argument based on counting in two ways. In particular, we make each edge $e \in M^*$ put a total blame of w_e on edges in M such that every edge $e' \in M$ receives at most $(2+\varepsilon)w_{e'}$ blame, as follows.

If $e \in M$, then it puts blame w_e on e. If $e \notin M$, there are two cases. Let i be such that $e \in E_i$. In case 1, if e has an endpoint, say v, with $b_v^{i+1} = 0$, and thus has b_v many edges in $\bigcup_{j=0}^{i} M_j$ incident to v, it puts blame $\frac{w_e}{b_v}$ on each of these edges. In case 2, if both endpoints have remaining b-value at least 1 after removing $\bigcup_{j=0}^{i} M_j$ from the graph, thus, if $e \in R_i$, it splits the blame w_e uniformly among all the edges in M_i.

We first bound the blame that is put on an edge $e' \in M_i$ in case 2. Observe that, in case 2, only edges $e \in R_i \cap M^*$ can blame e'. For these edges e it holds that $w_e \leq (1+\varepsilon')w_{e'}$. As observed in Remark 4.15, $|M_i| \geq \frac{1}{2+\varepsilon'}|M_i^*|$ and any matching in R_i (thus, in particular $M^* \cap R_i$) has size at most $\varepsilon'|M_i^*|$ for a maximum unweighted matching M_i^* in C_i'. As the blame of edges in $M^* \cap R_i$ is split uniformly among the edges in M_i, edge e' receives at most $\varepsilon'(2+\varepsilon')(1+\varepsilon')w_{e'}$ blame in case 2.

For case 1, observe that $e' = \{u,v\}$ has at most b_u edges e incident to u and at most b_v edges incident to v in M^* that could blame $\frac{w_e}{b_u}$

and $\frac{w_e}{b_v}$, respectively, weight on e'. Since these edges need to come from an E_j for $j \geq i$, they satisfy $w_e \leq (1+\varepsilon')w_{e'}$. Hence, at most $2(1+\varepsilon')w_{e'}$ blame is put on e' in case 1.

Summarized, e' is blamed at most

$$2\big(1+\varepsilon'\big)w_{e'} + \varepsilon'(2+\varepsilon')(1+\varepsilon')w_{e'} \leq (2+\varepsilon)w_{e'}$$

weight, observing that e' can either be in M^* and then blame itself, or not be in M^* and then receive (at most two) blames of case 1. It follows that $\sum_{e\in M^*} w_e \leq (2+\varepsilon)\sum_{e\in M} w_e$.

By precomputing an $O(\Delta^2)$-coloring with Linial's algorithm [172], we get the desired round complexity. □

4.4.4 Edge Dominating Set

Since any maximal matching is an edge dominating set, an almost maximal matching can easily be turned into an edge dominating set: additionally to the edges in the almost maximal matching, add all the remaining (at most $\varepsilon'|E|$ many) edges to the edge dominating set. When ε' is small enough, the obtained edge dominating set is a good approximation to the minimum edge dominating set. We next make this relation more precise.

Proof of Corollary 4.5. Apply the algorithm of Theorem 4.4 with $\varepsilon' = \varepsilon/(4\Delta)$, say, to find an ε'-almost maximal matching M in G. It is easy to see that $D = M \cup (E \setminus N^+(M))$ is an edge dominating set. Moreover, due to the fact that a minimum maximal matching is a minimum edge dominating set (see e.g. [230]) and since maximal matchings can differ by at most a factor 2 from each other,

$$\begin{aligned}(2+\varepsilon)|D^*| \geq \left(1+\frac{\varepsilon}{2}\right)|M| &\geq |M| + \frac{\varepsilon}{2(2\Delta-1)}|E| \\ &> |M| + \varepsilon'|E| \geq |D|\end{aligned}$$

by Lemma 4.7 (ii), also using Lemma 4.7 (i). □

CHAPTER 5

Local Rounding for Hypergraph Matching

5.1 Introduction

In this chapter—based on the publication 'Deterministic Distributed Edge-Coloring via Hypergraph Maximal Matching' [103]—we present a drastic generalization of the rounding technique for matching on graphs from Chapter 4 to matching on hypergraphs.

5.1.1 Our Results and Related Work

Hypergraph Maximal Matching

Our main technical contribution is an efficient deterministic algorithm for maximal matching in low-rank hypergraphs.

Theorem 5.1. *There is a deterministic* LOCAL *algorithm that computes a maximal matching in* $O(r^5 \log^{6+\log r} \Delta \cdot \log n)$ *rounds*[1]*, in any* n*-vertex rank-*r *hypergraph with maximum degree* Δ*.*

In subsequent works, this round complexity has been improved to $O\big(r^2 \log(n\Delta) \cdot \log n \cdot \log^4 \Delta\big)$ by Ghaffari, Harris, and Kuhn [115] and to $\widetilde{O}\big(r^2 \log n \cdot \log \Delta + r \log^2 \Delta\big)$ by Harris [137].

Note that besides being a natural extensions of maximal matchings in graphs—also supplying an alternative poly $\log n$-round deterministic algorithm for graph maximal matching—our result allows us to obtain answers and improvements for many other problems.

Unified Formulation of Symmetry Breaking Problems

One first important observation is that hypergraph maximal matching serves as a unification of all four classic symmetry breaking problems: maximal matching, $(2\Delta - 1)$ edge coloring, $(\Delta + 1)$ vertex coloring, and MIS. We can cast each of these problems as a maximal matching problem on hypergraphs of some rank r which depends on the problem and increases as we move from maximal matching to maximal independent set. In other words, with the hypergraph maximal matching problem we obtain a smooth interpolation between maximal matching in graphs with $r = 2$, then $(2\Delta - 1)$ edge coloring with $r = 3$, followed by $(\Delta + 1)$ vertex coloring with $r = \Delta$, and maximal independent set with $r = \Delta$ in graphs.

Edge Coloring: We next present a reduction from $(2\Delta - 1)$ edge coloring on a graph G to maximal matching in rank-3 hypergraphs. A similar reduction can be used for list edge coloring, as formalized in Lemma 5.3. We note that these reductions are inspired by the well-known reduction of Luby from $(\Delta + 1)$ vertex coloring to MIS [181, 171].

[1]Throughout, unless stated otherwise, all logarithms are to base 2. Moreover, we stress that one does not need a ceiling sign for $\log r$ in this complexity bound.

Lemma 5.2. *Given a deterministic* LOCAL *algorithm* $\mathcal{A}$ *that computes a maximal matching in* N*-vertex hypergraphs of rank* 3 *and maximum degree* d *in* $T(N, d)$ *rounds, there is a deterministic* LOCAL *algorithm* $\mathcal{B}$ *that computes a* $(2\Delta - 1)$ *edge coloring of any* n*-vertex graph* $G = (V, E)$ *with maximum degree* Δ *in at most* $T(3n\Delta, 2\Delta - 1)$ *rounds.*

Proof. To edge-color $G = (V, E)$ with $2\Delta - 1$ colors, we generate a hypergraph H as follows: Take $2\Delta - 1$ copies of G, one for each color. For each edge $e \in E$, let e_1 to $e_{2\Delta-1}$ denote its copies. For each $e \in E$, add one extra vertex w_e to H and change all edge copies e_1 to $e_{2\Delta-1}$ to 3-hyperedges by adding w_e to them. Algorithm $\mathcal{B}$ runs the maximal matching algorithm $\mathcal{A}$ on H. Then, for each $e \in E$, if the copy e_i of e is in the computed maximal matching, $\mathcal{B}$ colors e with color i. One can see that each G-edge e must have exactly one copy e_i in the maximal matching, which we then interpret as its color. Indeed, e cannot have more than one copy in the matching, since all these edges are connected to w_e, and the matching would not be maximal if none of the copies would be picked. $\square$

List Edge Coloring: This can be extended easily to list edge coloring. In the list edge coloring problem, each edge e must choose its color from an arbitrary given list $\Psi(e)$ of colors with $|\Psi(e)| = d(e) + 1$, where $d(e)$ denotes the number of edges adjacent to e.

Lemma 5.3. *Given a deterministic* LOCAL *algorithm* $\mathcal{A}$ *that computes a maximal matching in* N*-vertex hypergraphs of rank* 3 *and maximum degree* d *in* $T(N, d)$ *rounds, there is a deterministic* LOCAL *algorithm* $\mathcal{B}$ *that solves list edge coloring of any* n*-vertex graph* $G = (V, E)$ *with maximum degree* Δ *in at most* $T(2n\Delta^2, 2\Delta - 1)$ *rounds.*

Proof. To edge color $G = (V, E)$, we generate a hypergraph H: For each edge $e \in E$ with color list $\Psi(e)$, we take $|\Psi(e)|$ copies of

$e = \{v, u\}$ as follows: For each color $i \in \Psi(e)$, we take one copy e_i of e which is put incident to copies v_i and u_i of v and u. Thus, if two adjacent edges $e = \{v, u\}$ and $e' = \{v, u'\}$ have a common color $i \in \Psi(e) \cap \Psi(e')$, then their i^{th} copies e_i and e'_i will be present and will both be incident to v_i. Notice that for each vertex v, at most $2\Delta^2$ copies of it will be used because for each of the edges e incident to v, at most $|\Psi(e)| < 2\Delta$ additional copies of v are added.

Algorithm $\mathcal{B}$ runs the maximal matching algorithm $\mathcal{A}$ on H, and then, for each edge $e \in E$, if the copy e_i of e is in the computed maximal matching, $\mathcal{B}$ colors e with color i. One can verify that each G-edge e has exactly one copy e_i in the maximal matching, and thus we get a list edge coloring. □

Deterministic Edge Coloring

Our reduction of edge coloring to rank-3 hypergraph maximal matching combined with our hypergraph maximal matching algorithm leads to the first poly $\log n$-round edge coloring algorithm, resolving a decades old problem.

Theorem 5.4. *There is a deterministic LOCAL algorithm that computes a* $(2\Delta - 1)$ *(list) edge coloring in* $O(\log^8 \Delta \cdot \log n)$ *rounds.*

This is the first poly $\log n$-round and to date only non-network-decomposition-based deterministic algorithm for edge coloring. The previously best known round complexity was $2^{O(\sqrt{\log n})}$, by a classic network decomposition of Panconesi and Srinivasan [207], which itself improved on an $2^{O(\sqrt{\log n \cdot \log\log n})}$-round algorithm of Awerbuch et al. [17]. The novel network decomposition algorithm of [216, 114] can solve edge coloring in $O(\log^5 n)$ rounds. Note that for a wide range of Δ, our result is significantly faster than the generic network decomposition approach.

The hypergraph maximal matching algorithms by Ghaffari, Harris, and Kuhn [115] and by Harris [137] yield algorithms for edge coloring in $O(\log^4 \Delta \cdot \log^2 n)$ and $\widetilde{O}(\log^2 \Delta \cdot \log n)$ rounds, respectively.

For low-degree graphs, the approach by Kuhn [160] gives an algorithm in $2^{O(\sqrt{\log \Delta})} + O(\log^* n)$ rounds, improving over the bound of $O(\sqrt{\Delta \log \Delta} \cdot \log^* \Delta + \log^* n)$ [107, 25]. Very recently, this was further improved to $2^{O(\log^2 \log \Delta)} + \log^* n$ by [20]. Note that the only known lower bound is $\Omega(\log^* n)$ by Linial [172].

Randomized Edge Coloring

Our deterministic list edge coloring of Theorem 5.4, in combination with some randomized edge coloring algorithms of [87, 27], also improves the complexity of randomized algorithms for $(2\Delta - 1)$ edge coloring, making it the first among the four classic problems whose randomized complexity falls down to $\operatorname{poly}\log\log n$.

Corollary 5.5. *There is a randomized* LOCAL *algorithm that w.h.p. computes a* $(2\Delta - 1)$ *edge coloring in* $O(\log^9 \log n)$ *rounds.*

The previous (worst-case) complexity for randomized $(2\Delta - 1)$ edge coloring was $2^{O(\sqrt{\log \log n})}$ rounds, due to Elkin, Pettie, and Su [87]. By improving this, we widen the provable gap between the complexity of $(2\Delta - 1)$ edge coloring, which is now in $\operatorname{poly}(\log \log n)$ rounds, and the complexity of maximal matching, which has a lower bound of $\Omega(\log n / \log \log n)$ rounds [164, 19]. The only known lower bound for edge coloring is $\Omega(\log^* n)$ due to Linial [172] and Naor [194].

Our result recently got improved by Ghaffari, Harris, and Kuhn [115] to $O\big(\log^6 \log n\big)$, by Harris [137] to $\widetilde{O}(\log^3 \log n)$, by Rozhoň and Ghaffari [216] to $O(\log^7 \log n)$, and eventually by Ghaffari, Grunau, and Rozhoň [114] to $O(\log^5 \log n)$.

Edge Coloring in Sparse Graphs

A work of Barenboim, Elkin, and Maimon [26] presents an efficient deterministic algorithm for computing a $(\Delta + o(\Delta))$ edge coloring in graphs with arboricity $\lambda \leq \Delta^{1-\delta}$, for some constant $\delta > 0$. A simple combination of our list edge coloring algorithm of Theorem 5.4 with H-partition [24, Chapter 5.1] significantly extends their result.

Corollary 5.6. *There is a deterministic* LOCAL *algorithm that computes an edge coloring with* $\Delta+(2+\varepsilon)\lambda-1$ *colors on any* n*-vertex graph with maximum degree* Δ *and arboricity* λ *in* $O(\frac{1}{\varepsilon}\log^8 \Delta \cdot \log^2 n)$ *rounds.*

Notice that any graph has arboricity $\lambda \leq \Delta/2$. The above corollary shows that we start seeing savings in the number of colors as soon as the arboricity goes slightly below this upper bound. For instance, for $\lambda < \Delta(1-\varepsilon)/2$, we already get colorings with less than $2\Delta - 2$ colors.

Recently, Kuhn [160] devised an algorithm that finds a coloring with $(2 + o(1))\lambda$ colors in $2^{O(\sqrt{\log \lambda})} \log^2 n$.

MIS in Graphs with Bounded Neighborhood-Independence

To translate MIS on a graph G to maximal matching on a hypergraph H, one can consider the following reduction: View each G-edge as one H-vertex and each G-vertex v as one H-edge on the H-vertices corresponding to the G-edges incident to v. However, unfortunately, in this naïve formulation, the rank becomes Δ. As such, we do not obtain any improvement on the known algorithms for these problems, in the general case. It remains an intriguing open question whether any alternative formulation, perhaps in combination with other ideas, can help. However, using some more involved ideas, we obtain improvements for some special cases, which lead to answers for a few other open problems.

We extend our algorithm to the problem of computing an MIS in graphs with *neighborhood independence* bounded by an integer r, i.e., where the number of mutually non-adjacent neighbors of each vertex is at most r. Notice that maximal matching in graphs is the same as MIS in the corresponding line graph, which is a graph of neighborhood independence $r = 2$. It is not clear how to extend the methods of [131, 132] to MIS in such graphs[2], even for $r = 2$. As an open question alluding to this point, and as "*a good stepping stone towards the MIS problem in general graphs*", Barenboim and Elkin asked in their book [24]:

> **Open Problem 11.5 [24]:**
> *"Devise or rule out a deterministic polylogarithmic algorithm for the MIS problem in graphs with neighborhood independence bounded by 2."*

Our method for hypergraph maximal matching in Theorem 5.1 generalizes to MIS in graphs with bounded neighborhood independence, as we state formally next, hence positively answers this question.

Theorem 5.7. *There is a deterministic LOCAL algorithm that computes a maximal independent set in $O(r^5 \log^{6+\log r} \Delta \cdot \log n)$ rounds, in any n-vertex graph with maximum degree Δ and neighborhood independence bounded by r.*

Moreover, as Luby's reduction of $(\Delta+1)$ vertex coloring to MIS [181, 171] increases the neighborhood independence by at most 1, we also get efficient algorithms for $(\Delta + 1)$ vertex coloring in graphs with bounded neighborhood independence.

Corollary 5.8. *There is a deterministic LOCAL algorithm that computes a $(\Delta + 1)$ vertex coloring in $O(r^5 \log^{6+\log(r+1)} \Delta \cdot \log n)$*

[2]For a more strict definition of bounded neighborhood, MIS turns out to be much easier. See [161, 221].

rounds, in any n-vertex graph with maximum degree Δ and neighborhood independence bounded by r. Moreover, the same algorithm solves list vertex coloring, where each vertex $v \in V$ must get a color from an arbitrary given list $\Psi(v)$ of colors with $|\Psi(v)| \geq d(v) + 1$.

Note that a recent coloring algorithm of Kuhn [160] has round complexity $2^{O(\sqrt{\log r \cdot \log \Delta})} + O(\log^* n)$ if the colors come from a color space of size poly Δ.

Augmenting Paths

Another family of improvements comes for graph problems in which the main technical challenge is to find a maximal set of disjoint augmenting paths of short length ℓ. These problems can be phrased as maximal matching in hypergraphs with rank $r = \Theta(\ell)$, essentially by viewing each augmenting path as one hyperedge on its elements (depending on the required disjointness).

Maximum Matching Approximation: By integrating our hypergraph maximal matching into the framework of Hopcroft and Karp [143], we can compute a $(1+\varepsilon)$-approximation of maximum matching in graphs in $(\log \Delta/\varepsilon)^{O(\log(1/\varepsilon))}$ rounds. For that, we mainly need to find maximal sets of vertex-disjoint augmenting paths of length at most $\ell = O(1/\varepsilon)$.

Theorem 5.9. *There is a deterministic* LOCAL *algorithm that computes a $(1+\varepsilon)$-approximation of maximum matching for any $0 < \varepsilon \leq 1$ in* $\text{poly}\,\frac{1}{\varepsilon} \cdot O\left(\left(\frac{1}{\varepsilon}\log\Delta\right)^{7+\log\frac{1}{\varepsilon}}\right)$ *rounds.*

This is faster than the previously best known deterministic algorithm for $(1+\varepsilon)$-approximation in bipartite graphs, which required $\log^{O(1/\varepsilon)} n$ rounds [75]. We remark that an $O(\log n/\varepsilon^3)$-round randomized $(1+\varepsilon)$-approximation algorithm was presented by Lotker et al. [176], mainly by computing this maximal set of vertex-disjoint augmenting paths using Luby's randomized MIS algorithm [181, 5].

In subsequent works, our result got improved by Ghaffari, Harris, and Kuhn [115] to $O(\log^5 \Delta \log^2 n/\varepsilon^9)$ and by Harris [137] to $\widetilde{O}\big(\log^2 \Delta/\varepsilon^4 + \log^* n/\varepsilon\big)$. The latter also gives a randomized algorithm in $\widetilde{O}\big(\log \Delta/\varepsilon^3 + \log\log n/\varepsilon^3 + \log^2 \log n/\varepsilon^2\big)$ rounds.

Low-Outdegree Orientation and Forest Decomposition: Integrating our hypergraph maximal matching into the low-outdegree orientation framework of Ghaffari and Su [122], we can compute orientations with outdegree at most $\lceil(1+\varepsilon)\lambda\rceil$, for any $0 < \varepsilon < 1$, in graphs with arboricity λ. For that, we mainly need to find maximal sets of disjoint augmenting paths of length $\ell = O(\log n/\varepsilon)$. This low-outdegree orientation directly implies a decomposition into $\lceil(1+\varepsilon)\lambda\rceil$ edge-disjoint pseudo-forests.

Theorem 5.10. *There is a deterministic LOCAL algorithm that computes an orientation with maximum outdegree at most $\lceil\lambda(1+\varepsilon)\rceil$ in $2^{O\left(\log^2\left(\log \frac{n}{\varepsilon}\right)\right)}$ rounds, for any $\varepsilon > 0$, in any n-vertex graph with arboricity at most λ.*

For constant ε and even $\varepsilon = \log^{-O(1)} n$, the resulting round complexity is quasi-polylogarithmic—that is, $2^{O(\log^2 \log n)}$. Although this is not a polylogarithmic complexity, it gets close and it is almost exponentially faster than the previously best known $2^{O(\sqrt{\log n})}$ deterministic algorithm [122, 207]. This improvement can be viewed as partial solution for Open Problem 11.10 of Barenboim and Elkin [24] which asks for an efficient deterministic distributed algorithm for decomposing the graph into less than 2λ forests.

The subsequent works by Ghaffari, Harris, and Kuhn [115] and by Harris [137] improved our result to $O(\log^5 \Delta \cdot \log^{10} n/\varepsilon^9)$ and to $\widetilde{O}(\log^6 n/\varepsilon^4)$ (as well as to $\widetilde{O}(\log^3 n/\varepsilon^3)$ randomized), respectively. Recently, based on our rounding technique, Su and Vu [223] provided an $\widetilde{O}(\log^2 n/\varepsilon^2)$-round deterministic algorithm.

5.1.2 Notation and Preliminaries

Hypergraph Matching: A hypergraph $H = (V, E)$ is said to have rank r when each hyperedge $e \in E \subseteq V$ contains at most $|e| \leq r$ vertices. A matching in a hypergraph is a set M of hyperedges, no two of which share an endpoint: i.e., for all $e, f \in M$, we have that $e \cap f = \varnothing$.

Fractional Matching: A fractional matching of H is an assignment of values in $0 \leq x_e \leq 1$ to edges $e \in E$ such that for each vertex $v \in V$, we have $\sum_{e \in E(v)} x_e \leq 1$. Here, $E(v) := \{e \in E \colon v \in e\}$ is the set of edges incident to v. We call a fractional matching $(1/d)$-fractional if we have $x_e \geq 1/d$ or $x_e = 0$ for each $e \in E$.

Half-tight Vertices: For fractional matching, we say that a vertex v is *half-tight* if $\sum_{e \in E(v)} x_e \geq \frac{1}{2}$.

Bounded Neighborhood Independence: For an integer $r \geq 1$, we say that a graph $G = (V, E)$ has neighborhood independence at most r if for every vertex $v \in V$, the graph $G[N(v)]$ induced by the set $N(v)$ of neighbors of v has independence number at most r.

We note that the line graph of a hypergraph H of rank r has neighborhood independence at most r. This is because for each hyperedge e of H all incident hyperedges f share at least one vertex with e and because all the hyperedges sharing a vertex of H form a clique in the line graph of H. Hence, the neighborhood of each edge can be covered by at most r cliques in the line graph.

Recurrence Relation: Finally, we provide the solution of a recurrence relation that will be useful later to bound the number of levels in our recursive algorithm.

Lemma 5.11. *Let $r \geq 2$ and $\Delta \geq 2$ be two parameters and let $\alpha \geq 1$ and $c > 0$ be two given constants. Further, let $R(L)$ be a function* *that is defined for $L \geq 1$ by the following recurrence relation:*

$$R(L) \;:=\; \begin{cases} cr^2 + c\log\Delta, & \text{if } L \leq 4, \\ \alpha r R(\sqrt{2L}) + cr, & \text{otherwise.} \end{cases} \tag{5.1}$$

Then we have $R(L) = O\big(r^2 + (\log L)^{\log_2 \alpha + \log_2 r}(r^2 + \log\Delta)\big)$.

Proof. For all $x \geq 1$, we define a non-negative integer t_x as

$$t_x := \min\left\{t \in \mathbb{N}_0 \,:\, \left(\frac{x}{2}\right)^{2^{-t}} \leq 2\right\}.$$

We prove that for all $L \geq 1$, we have

$$\begin{aligned} R(L) &\leq (\alpha r)^{t_L}(cr^2 + c\log\Delta) + cr\sum_{i=0}^{t_L-1}(\alpha r)^i \\ &< 2(\alpha r)^{t_L}(cr^2 + c\log\Delta), \end{aligned} \tag{5.2}$$

where the last inequality follows from $\alpha r \geq 2$. For $x \geq 1$, we have $t_x \leq \max\{0, \log\log x\}$ and thus the claim of the lemma directly follows from (5.2).

To prove (5.2), first note that for $L \leq 4$, we have $t_L \geq 0$ and because $\alpha r \geq 1$, we thus have $R(L) \leq cr^2$ as required by (5.1). For $L > 4$, we prove (5.2) by induction. More formally, for each $L > 4$, we show that there is a finite sequence $L = L_k > L_{k-1} > \cdots > L_0$ such that $L_0 \leq 4$ and such that for each $i \in \{1, \ldots, k\}$, (5.1) implies that if (5.2) holds for L_{i-1}, it also holds for L_i.

Let us therefore assume that $L_k = L > 4$. For $i \geq 1$, we define $L_{i-1} := \sqrt{2L_i}$. First note that because for $x > 4$, $\sqrt{2x} \leq x/\sqrt{2}$ and thus we reach a value smaller than 4 in a bounded number of steps. For every $i \geq 1$ such that $L_i > 4$, we have

$$t_{L_{i-1}} = \min\left\{t \in \mathbb{N}_0 \,:\, \left(\sqrt{\frac{L_i}{2}}\right)^{2^{-t}} = \left(\frac{L_i}{2}\right)^{2^{-(t+1)}} \leq 2\right\} \;=\; t_{L_i} - 1.$$

From (5.1), for $L_i > 4$, we therefore have

$$\begin{aligned} R(L_i) &\leq \alpha r R(L_{i-1}) + cr \\ &\leq \alpha r \left((\alpha r)^{t_{L_i}-1}(cr^2 + c\log\Delta) + cr \sum_{j=0}^{t_{L_i}-2} (\alpha r)^j \right) + cr \\ &= (\alpha r)^{t_{L_i}}(cr^2 + \log\Delta) + cr \sum_{j=0}^{t_{L_i}-1} (\alpha r)^j . \end{aligned}$$

This proves (5.2), and thus concludes the proof. $\square$

5.1.3 Overview and Outline

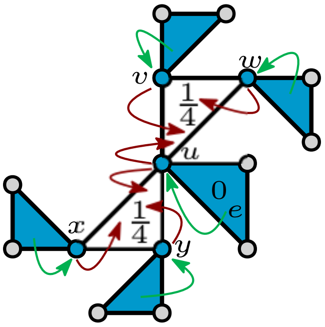


Figure 5.1: Every hyperedge of the maximum matching (drawn in blue) in the rank-3 hypergraph blames 1 dollar on one of its tight (blue) endpoints (depicted as green arrows). Vertex u distributes this dollar it has received from e to each of its incident hyperedges proportionally to their value: 0 to e with value 0 and 50 cents to each of two of its hyperedges with value $\frac{1}{4}$ (shown as red arrows). These hyperedges might receive at most 50 cents from each of their other endpoints v and w and x and y, respectively.

As for the case of graphs, finding a fractional matching with a good approximation ratio is easy—the challenge again lies in finding a good integral matching.

Fractional Hyergraph Matching

A simple greedy algorithm finds a $(2r)$-approximation in $O(\log \Delta)$ rounds: Starting with edge values $x_e = 2^{-\lfloor \log \Delta \rfloor}$ for all e, for $O(\log \Delta)$ rounds, we freeze edges with half-tight endpoints and double the values of all other edges. See Figure 5.1 for an illustration of why this gives the desired approximation.

Challenges for Rounding in Hypergraphs

The core part of the rounding for graphs (i.e., hypergraphs with rank $r = 2$) is that it can be done essentially with no loss: in each iteration we can move a factor 2 closer to integrality while decreasing the matching size only negligibly, by a factor $(1 - \frac{\varepsilon}{\Theta(\log \Delta)})$. This whole methodology of rounding without more than a factor $o(1)$ loss in the size seems to be quite limited, and it certainly gets stuck at rank $r = 2$. These matching rounding methods [99, 131, 132] decompose the edges of the graph into bipartite low-diameter degree-2 graphs (i.e., short even-length cycles)—aside from a smaller portion of some not-so-nice parts, which are handled separately—and then 2-color edges of each short cycle so that each vertex has half of its edges in each color. Then, in rounding, one color is raised by a factor 2 while the other is dropped to zero. Unfortunately, this type of locally-balanced splittings of edges did not seem within reach for hypergraphs. We refer to [115] for a thorough discussion and a solution to this problem.

When trying to deterministically round fractional matchings in hypergraphs, we face essentially two challenges: (1) It is not clear how to efficiently perform any slight rounding—e.g., rounding all fractional values so that the minimum moves from at least $1/d$ to at least $2/d$ without violating the constraints—without a considerable loss in the matching size. (2) An even more crucial issue comes from the need to do many levels of rounding. Even once we have an efficient solution for a single iteration of rounding, which moves, say, a constant factor closer to integrality, a factor-$\Theta(r)$ reduction of the matching size seems inevitable. However, if we do this repeatedly, and our matching size drops by a factor $\Theta(r)$ in each rounding iteration, the matching size becomes too small. Notice that we need about $O(\log \Delta)$ levels of factor-2 roundings. If we decrease by a factor $\Omega(r)$ per iteration, we are left with a matching of size a factor $1/r^{\Theta(\log \Delta)} = 1/\operatorname{poly}(\Delta)$ of the maximum matching, which is essentially useless.

Our Rounding Method for Matchings in Hypergraphs

Solution for Challenge (1): We devise a rounding procedure for hypergraph matchings which rounds the fractional matching by a factor L—i.e., raises the fractional values by factor L—while reducing the matching size only by a factor $\Theta(r)$. On a high level, this rounding is by recursion on L. The base level of the recursion is an algorithm that rounds the fractional matching by a constant factor, for $L = O(1)$, with only a factor $\Theta(r)$ decrease of the matching size. This part is somewhat simpler and is performed efficiently using defective coloring results of [159]. This is a solution for the first challenge above.

Solution for Challenge (2): To overcome the second challenge, our method interleaves some iterations of *rounding* with *refilling* the fractional matching. In particular, suppose that we would like to do a factor-L rounding of a given fractional matching x, thus producing an output fractional matching y with fractional values raised by a factor L compared to x. We do this in $\Theta(r)$ iterations, using a number of factor-$\sqrt{2L}$ rounding procedures. Concretely, per iteration, we first '*remove*' the current output fractional matching y from the input fractional matching x, in a sense to be made precise, and then we apply two successive factor-($\sqrt{2L}$) rounding operations on the leftover fractional matching. This creates a fractional matching which is rounded by a factor of $2L$, but may be a factor $1/\Theta(r^2)$ smaller than x. We add (a half of) this to the current fractional matching y, in a sense to be made precise. The removal and also the addition are done carefully, so as to ensure that the size of the output fractional matching grows by about a factor $(1/\Theta(r^2))$ of the size of x while the fractionality is by a factor L better than the one of x. After $\Theta(r)$ such iterations, we get that the output fractional matching is a $\Theta(r)$-approximation of the input.

5.2 Hypergraph Maximal Matching

In this section, we present our hypergraph maximal matching algorithm, thus proving Theorem 5.1.

The core of our rank-r hypergraph maximal matching algorithm is an algorithm that computes a matching whose size is within a factor $O(r^3)$ of the maximum matching.

Lemma 5.12. *There is a deterministic* LOCAL *algorithm that given an $O(r^2\Delta^2)$ edge coloring computes a $(32r^3)$-approximate matching in $O\big(r^2 \log^{6+\log r} \Delta\big)$ rounds.*

This was slightly improved in [137] to $\widetilde{O}(r \log \Delta + \log^2 \Delta + \log^* n)$.

Over the next two subsections, we discuss the matching approximation procedure of Lemma 5.12. We note that finding a *fractional* matching with size close to the maximum matching is straightforward, as we soon overview in Section 5.2.1. The challenge is in finding an integral matching with the same guarantee. In other words, the core technical component of our method is an algorithm for *rounding* fractional hypergraph matchings to integral matchings, without losing much in the size. In particular, we present our deterministic rounding technique for hypergraph matchings in Section 5.2.2.

5.2.1 Fractional Matching Approximation

In the following, we present a simple $O(\log \Delta)$-round algorithm that computes a $(2r)$-approximate fractional matching.

Greedy Fractional Matching Algorithm: Initially, we set $x_e = \frac{1}{\Delta}$ for all edges e. This obviously is a valid fractional matching. Then, for $\log \Delta$ iterations, in each iteration, we freeze all the edges that have at least one half-tight vertex and then raise the value of all unfrozen edges by a factor 2.

This way, we always keep a valid fractional matching, since only values of edges incident to non-half-tight vertices are increased. Moreover, within $O(\log \Delta)$ iterations all edges will be frozen. We next show that this property already implies an approximation ratio $2r$.

Lemma 5.13. *The greedy algorithm described above computes a $(2r)$-approximate fractional matching. Moreover, any (fractional) matching x with the property that each edge has at least one half-tight endpoint is a $(2r)$-approximation.*

Proof. We show that x must have size at least a factor $(1/(2r))$ of a maximum matching M^* employing an argument based on counting in two ways. To that end, we give 1 dollar to each edge $e \in M^*$ and ask it to redistribute this money among edges in such a way that no edge e' receives more than $2rx_{e'}$ dollars. This can be achieved as follows. Each edge $e \in M^*$ asks a half-tight vertex, say $v \in e$, to distribute e's dollar on e's behalf. Vertex v does so by splitting this money among its incident edges $e' \in E(v)$ proportionally to the edge values $x_{e'}$. In this way, every edge $e' \in E(v)$ receives no more than $2x_{e'}$ dollars from v. This is because v does not receive more than 1 dollar, as it is half-tight and it cannot have more than one incident edge in M^*. Since an edge can receive money only from its vertices, every edge e' receives at most $2rx_{e'}$ dollars in total. $\square$

5.2.2 Rounding Overview

Our method for rounding fractional matchings is recursive, and parametrized mainly by a parameter L which captures the extent of the performed rounding. In simple words, given a fractional matching $x \in [0,1]^{|E|}$, the algorithm $\mathsf{round}(x, L)$ rounds x by a factor L. That is, if in the input fractional matching x the smallest (non-zero) value is $1/d$, then in the output fractional matching the smallest (non-zero) value is at least L/d. On the other hand, the guarantee is that the output fractional matching has size at least a factor $(1/(4r))$ of the input fractional matching. The functionality of this rounding method is abstracted by the following definition.

Definition 5.14 (Factor-L Rounding $\mathsf{round}(x, L)$)**.** *The factor-L rounding method $\mathsf{round}(x, L)$ turns a $(1/d)$-fractional matching $x \in [0,1]^{|E|}$ into an (L/d)-fractional matching $y \in [0,1]^{|E|}$ such that $\sum_{e\in E} y_e \geq \frac{1}{4r}\sum_{e\in E} x_e$.*

Remark 5.15. *The method requires some condition on the values of L and d. Since L/d refers to the fractionality, the statement is meaningful only when $L \leq d$. Due to some small technicalities, we will perform the recursive parts of rounding only for values of L that satisfy a slightly stronger condition of $L \log^2 L \leq d$. For the remaining cases, we resort to our basic rounding.*

We explain our rounding method in two main parts. The first part, explained in Section 5.2.3, is a procedure that we use as the base case, to round the matching by a constant factor $L = O(1)$ in $O(r^2 + \log \Delta)$ rounds. The second part, discussed in Section 5.2.4, is the recursive step which explains how our factor-L rounding works by making a few calls to factor-$\sqrt{2L}$ rounding procedures, and a few smaller steps. Finally, in Section 5.2.5, we combine these rounding procedures with the previously seen algorithm of Section 5.2.1 for fractional matchings to obtain our matching approximation procedure of Lemma 5.12.

5.2.3 Basic Rounding

In this subsection, we explain our base case rounding procedure for small rounding parameters, i.e., $L = O(1)$. Throughout, we will assume that the base hypergraph already has an $O(r^2\Delta^2)$ edge coloring, which can be computed easily using Linial's algorithm [171], in $O(\log^* n)$ rounds.

Lemma 5.16 (Basic Rounding)**.** *There is an $O(L^2r^2+\log\Delta)$-round deterministic LOCAL algorithm that transforms a $(1/d)$-fractional matching x into an (L/d)-fractional matching y with $\sum_{e\in E} y_e \geq \frac{1}{2r}\sum_{e\in E} x_e$, for any $L \leq d$.*

Algorithm Outline and Intuitive Discussions

Let E_x be the set of all edges e for which $x_e > 0$, and let $H_x = (V, E_x)$ be the subgraph of H with this edge set. Notice that H_x has degree at most d, because x is a $(1/d)$-fractional matching. Our goal is to compute a fractional matching y, supported on the edge set E_x, such that for each edge $e \in E_x$, at least one of its endpoints $v \in e$ is half-tight in y, meaning that $\sum_{e'\in E_x(v)} y_{e'} \geq 1/2$. One can easily see that such a fractional matching is a $(2r)$-approximation of x, i.e., $\sum_{e\in E} y_e \geq \frac{1}{2r}\sum_{e\in E} x_e$. Thus, the goal is to find a fractional matching y such that for each edge $e \in E_x$, at least one of its endpoints is *half-tight* in y. Furthermore, we want y to be (L/d)-fractional, meaning that all the non-zero y_e-values must be greater than or equal to L/d.

If we had no concern for the time complexity, we could go through the color classes of edges one by one, each time setting $y_e = 1$ for all edges of that color, and then removing edges of E_x that have half-tight vertices. This would ensure that, at the end, all edges in E_x have at least one half-tight endpoint. However, this would require time proportional to the number of colors. Even if we were given an ideal edge coloring for free, that would be $\Omega(d)$ rounds, which is too slow for us.

To speed up the process, we use a relaxed notion of edge coloring, namely *defective edge coloring*, which allows us to have much less colors, while each color class has a bounded number of edges incident to each vertex, say k. Now, we cannot raise the y_e-values of all the edges of the same color at the same time to $y_e = 1$, because that would be too fast and could violate the condition $\sum_{e' \in E_x(v)} y_{e'} \leq 1$. However, we can raise each of these edge values to say $y_e = \frac{1}{2k}$ and still be sure that the summation $\sum_{e' \in E_x(v)} y_{e'}$ for each vertex does not increase faster than an additive $1/2$. That is because there are only k edges incident to each vertex, per color class. If we freeze and remove all edges that now have one half-tight vertex, these fractional value raises would never violate the condition $\sum_{e' \in E_x(v)} y_{e'} \leq 1$, thus always lead to a valid fractional matching.

To materialize the above intuitive approach, we first compute a *defective edge coloring* with $O\big(L^2\Delta^2\big)$ colors and defect $k = d/(2L)$. Then, we go through the colors, one by one, applying the above fractional-value increases. This ensures that all the non-zero fractional values y_e are at least $\frac{1}{2k} \geq \frac{L}{d}$. At the very end, we perform $O(\log d/L)$ doubling steps to ensure that each edge has at least one half-tight endpoint. We next explain the steps of this algorithm, and then provide the related analysis.

Step 1: Defective Edge Coloring

We compute a defective edge coloring of H_x with $O(L^2r^2)$ colors and defect—that is, maximum degree induced by edges of the same color—at most $d/(2L)$, as follows. Let $F = (V_F, E_F)$ be the line graph of H_x, that is, the graph which has a vertex $v_e \in V_F$ for every edge $e \in E_x$ and an edge $\{v_e, v_{e'}\} \in E_F$ if e and e' are incident, thus $e \cap e' \neq \varnothing$. Note that F has maximum degree at most rd, since H_x's maximum degree is bounded by d. With the defective coloring algorithm of Kuhn [159], we can compute a $(d/(2L) - 1)$-defective vertex coloring of F with

$$O\left(\left(\frac{rd}{\frac{d}{2L} - 1}\right)^2\right) = O\big(L^2r^2\big)$$

colors[3]. Exploiting the given $O\big(r^2\Delta^2\big)$ edge coloring of H, and thus H_x, which is an $O\big(r^2\Delta^2\big)$ vertex coloring of the line graph F, we can make this algorithm run in $O(\log^*(r\Delta))$ rounds. The vertex coloring of the line graph with defect $d/(2L) - 1$ is an edge coloring of H_x where every edge has at most $d/(2L) - 1$ incident edges of the same color, resulting in at most $d/(2L)$ many edges of the same color incident to each vertex.

Step 2: Fractional Matching via Defective Coloring

We process the colors of the $d/(2L)$-defect defective coloring one by one, in $O\big(L^2r^2\big)$ iterations. In the i^{th} iteration, for each non-frozen edge e with color i, we raise y_e from $y_e = 0$ to $y_e = L/d$. Then for each vertex v that is already half-tight, meaning that $\sum_{e \in E_x(v)} y_e \geq 1/2$, we freeze all the edges incident to v. This means the fractional

[3]If we happen to have $d/(2L) \leq 1$, then a $(d/(2L) - 1)$-defective coloring becomes a degenerate case of the definition, as $(d/(2L) - 1) \leq 0$, and then by convention, this simply means proper coloring. In that case the algorithm of Kuhn [159] provides a proper coloring with $O(d^2r^2) = O(L^2r^2)$ colors.

value of these edges will not be raised in the future. Notice that since we raise values only incident to vertices that are not already half-tight, and as for each such vertex the summation goes up by at most $\frac{d}{2L} \cdot \frac{L}{d} = 1/2$, the vector y always remains a fractional matching, meaning that we always have $\sum_{e \in E_x(v)} y_e \leq 1$ for each vertex v.

At the very end, once we are done with processing all colors, some edges in E_x may remain without any half-tight endpoint. Though any such edge e would itself have $y_e = L/d$. We perform $\log(d/L)$ iterations of doubling, where in each iteration, we double all the fractional values y_e for all edges that do not have a half-tight endpoint. At the end, we are ensured that each edge has at least one half-tight endpoint, and moreover, each non-zero fractional value y_e is at least L/d.

Lemma 5.17. *The above algorithm computes an (L/d)-fractional matching y such that $\sum_{e \in E} y_e \geq \frac{1}{2r} \sum_{e \in E} x_e$, in $O(L^2r^2 + \log(d/L) + \log^*(r\Delta)) = O(L^2r^2 + \log\Delta)$ rounds.*

Proof. The round complexity comes from $O(\log^*(r\Delta))$ rounds spent for computing the defective edge coloring, $O(L^2r^2)$ rounds for processing the colors of the defective coloring one by one, and then $O(\log(d/L))$ rounds for the final doubling steps.

It is clear by construction that the computed vector y is a fractional matching, because we always have $\sum_{e \in E_x(v)} y_e \leq 1$, and that it is (L/d)-fractional, because the smallest non-zero y_e value that we use is L/d. What remains to be proved is that $\sum_{e \in E} y_e \geq \frac{1}{2r} \sum_{e \in E} x_e$. For that, we use the property that the fractional matching y that we compute is such that for each $e \in E_x$, at least one of the vertices $v \in e$ must be half-tight, meaning that $\sum_{e \in E(v)} y_e \geq 1/2$. We use this property to argue that the fractional matching y has size at least a factor $(1/(2r))$ of x. This is done via a blaming argument along the same lines as the proof of Lemma 5.13. We let every edge $e \in E_x$ put x_e dollars on edges $e' \in E_y$ as follows. Each edge e

passes its x_e dollars to one of its half-tight vertices $v \in e$. Then, the half-tight vertex v distributes these x_e dollars among all its incident edges $e' \in E_x(v)$ proportionally to the values $y_{e'}$. As x is a fractional matching, in this way, v cannot receive more than 1 dollar in total from its incident edges in E_x. Therefore, and since v is half-tight, no edge e' incident to v receives more than $2y_{e'}$ dollars from $v \in e'$. In total, an edge $e' \in E_y$ can receive at most $2ry_{e'}$ dollars from edges in E_x, at most $2y_{e'}$ from each of its endpoints. Therefore, $\sum_{e \in E} x_e \leq 2r \sum_{e \in E} y_e$. □

5.2.4 Recursive Rounding

We explain a recursive method $\mathsf{round}(x, L)$ that takes any $(1/d)$-fractional matching x and computes an (L/d)-fractional matching y such that $\sum_{e \in E} y_e \geq \frac{1}{4r} \sum_{e \in E} x_e$. This procedure will be applied when L is greater than some fixed constant. The procedure works mainly by a number of recursive calls to factor-$\sqrt{2L}$ rounding procedures, and a few additional steps.

Lemma 5.18 (Recursive Rounding)**.** *There is a deterministic* LOCAL *algorithm that turns a $(1/d)$-fractional matching x into an (L/d)-fractional matching y with $\sum_{e \in E} y_e \geq \frac{1}{4r} \sum_{e \in E} x_e$, for any L such that $L \log^2 L \leq d$, in $O\big((r^2 + \log \Delta) \log^{5+\log r} L\big)$ rounds.*

The Recursive Rounding Algorithm

The method $\mathsf{round}(x, L)$ consists of $16r$ iterations. Initially, we set $y_e = 0$ for all edges. Then, in 16 iterations, we gradually grow y while keeping it (L/d)-fractional.

The process in each iteration is as follows:

Step 1: We first generate a fractional matching z by initially setting it equal to x, and then removing from it each edge e that is incident to a at least one half-tight vertex of y. In other words, for each vertex v such that $\sum_{e \in E(v)} y_e \geq 1/2$, we set $z_e = 0$ for all e with $v \in e$; for all other edges, we set $z_e = x_e$.

Step 2: We perform $\mathsf{round}(z, \sqrt{2L})$, producing some intermediate $(\sqrt{2L}/d)$-fractional matching z', and then call $\mathsf{round}(z', \sqrt{2L})$. This creates a $(2L/d)$-fractional matching z'' whose size is at least a factor $\frac{1}{4r} \cdot \frac{1}{4r} = \frac{1}{16r^2}$ of the size of z.

Step 3: We divide the values of this fractional matching z'' by a factor 2, creating an (L/d)-fractional matching, and we add the result to y. Thus, we effectively update $y \leftarrow y + z''/2$.

Remark 5.19. *Recall the promise from Remark 5.15 that we will apply the rounding method only for values such that $L \log^2 L \leq d$. The main reason for this stronger condition, compared to the more natural condition of $L \leq d$, is the factor 2 that we have in the recursive rounding call. For instance, the matching z'' is a $(2L/d)$-fractional matching and thus, for this to be meaningful, we need $2L \leq d$. However, with the stronger condition that $L \log^2 L \leq d$, we can say that the promise is satisfied throughout the recursive calls. For instance, in the second call to $\mathsf{round}(z', \sqrt{2L})$, the new condition would be $\sqrt{2L}(\log \sqrt{2L})^2 \leq d/\sqrt{2L}$, which is readily satisfied given that $L \log^2 L \leq d$ and $L \geq 8$.*

Analysis of the Recursive Rounding

We next provide the related analysis. In particular, Lemma 5.20 proves that the generated fractional matching y is valid, Lemma 5.21 proves that it is a good approximation of x, and Lemma 5.22 analyzes the running time of this recursive procedure.

Lemma 5.20. *The fractional matching y is valid, meaning that $\sum_{e \in E(v)} y_e \leq 1$ for all vertices v.*

Proof. We show by induction on i that the fractional matching y in iteration i does not violate the constraints $\sum_{e\in E(v)} y_e \le 1$ for all v. At the beginning, the condition is trivially satisfied. If v is half-tight at the beginning of an iteration, then $z_e = 0$ and hence $z''_e = 0$ for all $e \in E(v)$, thus no value is added to $\sum_{e\in E(v)} y_e$ in this iteration. If v is not half-tight at the beginning of an iteration, we add at most half of a fractional matching to edges incident to v, thus at most a value $1/2$ to the summation $\sum_{e\in E(v)} y_e$. More formally, we have $\sum_{e\in E(v)} z''_e \le 1$, thus $\sum_{e\in E(v)} z''_e/2 \le \frac{1}{2}$, which results in a new value of at most $\sum_{e\in E(v)}(y_e + z''_e/2) \le 1$. □

Lemma 5.21. *We have* $\sum_{e\in E} y_e \ge \frac{1}{4r}\sum_{e\in E} x_e$ *at the end of* $16r$ *iterations.*

Proof. Consider one iteration and suppose $\sum_{e\in E} y_e < \frac{1}{4r}\sum_{e\in E} x_e$. We first show that then $\sum_{e\in E} z_e \ge \frac{1}{2}\sum_{e\in E} x_e$ by a blaming argument along the same lines as the proof of Lemma 5.13. For that, we let every edge $e \in E$ which is incident to a half-tight vertex in y put x_e dollars on edges in a manner that each edge e' receives at most $2ry_{e'}$ dollars. This can be done by sending those x_e dollars of e to (one of) its y-half-tight vertex v, and then letting v distribute these x_e dollars among its incident edges $e' \in E(v)$ proportionally to the values $y_{e'}$. Since x is a matching, each vertex v in total receives at most 1 dollar from its incident edges. Then, since v is y-half-tight, it can distribute this dollar among its incident edges such that no edge e' receives more than $2y_{e'}$ dollars from one of its endpoints v. Now, an edge $e' \in E$ can possibly receive $2y_{e'}$ dollars from each of its (half-tight) endpoint vertices, thus in total at most $2ry_{e'}$ dollars. Therefore, indeed

$$\sum_{e\in E\colon\, \exists v\in e\colon\, \sum_{e'\in E(v)} y_{e'} \ge 1/2} x_e \le 2r\sum_{e\in E} y_e \le \frac{2r}{4r}\sum_{e\in E} x_e = \frac{1}{2}\sum_{e\in E} x_e.$$

It follows that if $\sum_{e\in E} y_e < \frac{1}{4r}\sum_{e\in E} x_e$ holds, then $\sum_{e\in E} z_e \ge$

$\frac{1}{2}\sum_{e\in E} x_e$. Thus, in each such iteration, the matching y grows by at least

$$\sum_{e\in E}\frac{z''_e}{2} \geq \frac{1}{2}\cdot\frac{1}{16r^2}\sum_{e\in E} z_e \geq \frac{1}{2}\cdot\frac{1}{16r^2}\cdot\frac{1}{2}\sum_{e\in E} x_e.$$

Therefore we have $\sum_{e\in E} y_e \geq \frac{1}{4r}\sum_{e\in E} x_e$ after at most $16r$ iterations. □

Lemma 5.22. *It takes* $O((r^2+\log\Delta)\log^{5+\log r} L)$ *rounds to run the method* $\mathsf{round}(x, L)$.[4]

Proof. The complexity $R(L)$ of the rounding method $\mathsf{round}(x, L)$ follows the recursive inequality

$$R(L) \leq 16r(R(\sqrt{2L}) + R(\sqrt{2L}) + O(1)).$$

Furthermore, we have the base case solution of $R(L) = O(L^2r^2 + \log\Delta)$ for $L = O(1)$. The claim can now be proved by an induction on L, as formalized in Lemma 5.11. Here, instead of the formal calculations, we mention an intuitive explanation: the complexity gets multiplied by roughly $32r$ as we move from L to $\sqrt{2L}$. There are roughly $\log\log L$ such moves, and hence the complexity gets multiplied by $(32r)^{\log\log L} < \log^{5+\log r} L$ until we reach the base case of $L = O(1)$, where the base complexity is $O(r^2+\log\Delta)$ by Lemma 5.16. □

5.2.5 Maximum and Maximal Matching

Approximate Maximum Matching

We now use our rounding procedure to find the $(32r^3)$-approximate matching of Lemma 5.12 in $O\big(r^2\log^{6+\log r}\Delta\big)$ rounds.

[4]We remark that we have not tried to optimize the constant that appears in the exponent of the round complexity. This constant mainly comes from the constant in the number of iterations in our recursive rounding, which is currently set to $16r$, for simplicity.

Proof of Lemma 5.12. First, we compute a $(1/\Delta)$-fractional $(2r)$-approximate matching x in $O(\log \Delta)$ rounds, by the greedy algorithm described in Section 5.2.1. Then, we apply the recursive rounding from Lemma 5.18 for $L = \Delta/\log^2 \Delta$. This produces a $(1/\log^2 \Delta)$-fractional matching x' whose size is a factor $(1/(8r^2))$ of the maximum fractional matching of the hypergraph, and takes $O\big(\log^{5+\log r} \Delta(r^2 + \log \Delta)\big)$ rounds. To finish up the rounding, we apply the basic rounding of Lemma 5.16 for $L = \log^2 \Delta$, which runs in $O(r^2 \log^4 \Delta)$ and produces a 1-fractional, i.e., integral, matching x'' whose size is at least a factor $(1/(4r))$ of the size of x'. Hence, the final produced integral matching is a $(32r^3)$-approximation of the maximum matching. $\square$

Maximal Matching

Once given such an approximation algorithm, we can easily find a maximal matching via iterative applications of this matching approximation, by repeatedly applying this matching approximation procedure to the remainder hypergraph for $O(r^3 \log n)$ iterations, each time adding the found matching to the output matching, and then removing the found matching and its incident edges from the hypergraph.

Proof of Theorem 5.1. First, we pre-compute an $O(r^2\Delta^2)$ edge coloring of H in $O(\log^* n)$ rounds by Linial's algorithm [171]. Then, iteratively, we apply the maximum matching approximation procedure of Lemma 5.12 to the remaining hypergraph. We add the found matching M to the matching that we will output at the end, and remove M along with its incident edges from the hypergraph.

In each iteration, the size of the maximum matching of the remaining hypergraph goes down to at least a factor of $1 - 1/(32r^3)$ of the previous size. This is because otherwise we could combine the matching computed so far with the maximum matching in the remainder hypergraph to obtain a matching larger than the maximum matching in H. After $O(r^3 \log n)$ repetitions, the remaining maximum matching size is 0, which means the remaining hypergraph is empty. Hence, we have found a maximal matching in $O\left(\log^* n + r^3 \log n \left(r^2 \log^{6+\log r} \Delta\right)\right) = O(r^5 \log^{6+\log r} \Delta \cdot \log n)$ rounds. □

5.3 Implications and Corollaries

5.3.1 Edge Coloring

We use our hypergraph maximal matching algorithm to prove our edge coloring results.

Deterministic List Edge Coloring

First, we prove our deterministic (list) edge coloring result.

Proof of Theorem 5.4. Follows directly from Lemma 5.3 and Theorem 5.1. □

Remark 5.23. *We note that Lemma 5.3, and hence also Theorem 5.4, can be easily extended from graphs to hypergraphs. In particular, list edge coloring of hypergraphs of rank r can be reduced to maximal matching in hypergraphs of rank $r+1$. Thus, we can obtain a deterministic list edge coloring algorithm for hypergraphs of rank r with round complexity $O(r^5 \log^{6+\log(r+1)} \Delta \log n)$ rounds.*

Radomized List Edge Coloring

As stated before, the deterministic list edge coloring algorithm of Theorem 5.4, in combination with known randomized algorithms of Elkin, Pettie, and Su [87] as well as Johansson [149], leads to a poly $\log \log n$-round randomized algorithm for $(2\Delta-1)$ edge coloring.

Proof of Corollary 5.5. For $\Delta = \Omega(\log^2 n)$, we run the algorithm of Elkin, Pettie, and Su [87] for $((1 + \varepsilon)\Delta)$ edge coloring, which takes $O\Big(\log^* \Delta + \frac{\log n}{\Delta^{1-o(1)}}\Big) = O(\log^* \Delta)$ rounds. For $\Delta = o(\log^2 n)$, we first apply the simple randomized coloring algorithm of Johansson [149] for $O(\log \Delta + \log\log n) = O(\log\log n)$ rounds. In particular, in each iteration, every remaining edge e independently picks a color q_e from its remaining palette uniformly at random. If there is no incident edge that picked the same color q_e, then the edge e is colored with this color q_e and removed from the graph. Moreover, the color q_e gets deleted from the palettes of every incident edge. As proved in e.g. [27], after $O(\log \Delta + \log\log n)$ rounds, this procedure leaves us with a graph where each connected component of remaining edges has size at most $N = \text{poly}\log n$. On these components, we then run the list edge coloring algorithm of Theorem 5.4 to complete the partial coloring. This takes at most $O(\log^8 N) = O(\log^8 \log n)$ rounds. Hence, including the $O(\log\log n)$ initial rounds, the overall complexity is $O(\log^8 \log n)$ rounds. □

Edge Coloring in Sparse Graphs

Finally, we prove our edge coloring result for graphs with low arboricity.

Proof of Corollary 5.6. First, we compute an H-partition [24, Chapter 5.1] in $O(\log n/\varepsilon)$ rounds. This decomposes V into disjoint vertex sets $H_1, H_2, \ldots, H_\ell$, for $\ell = O(\log n/\varepsilon)$, with the property that each vertex in H_i has degree at most $(2 + \varepsilon)\lambda$ in the graph $G[\cup_{j=i}^{\ell} H_j]$. To compute this decomposition, one just needs to iteratively peel vertices of degree at most $(2 + \varepsilon)\lambda$ from the remaining graph.

Having this partition, we compute a $(\Delta + (2+\varepsilon)\lambda - 1)$ edge coloring by gradually moving backwards in this partition, from H_ℓ towards H_1. Each step is as follows. Suppose we already have a coloring of edges of $G[\cup_{j=i+1}^{\ell} H_j]$. We now introduce the vertices of H_i and also their edges whose other endpoint is in $\cup_{j=i}^{\ell} H_j$. Each such edge e has at most $(2+\varepsilon)\lambda - 1$ other incident edges on the side of its H_i-endpoint and at most $\Delta - 1$ other incident edges on the other endpoint. If we take away the colors of $\{1, 2, \ldots, \Delta + (2+\varepsilon)\lambda - 1\}$ that are already used by neighboring edges e' whose both endpoints are in $\cup_{j=i+1}^{\ell} H_j$, the edge e would still have at least $d_e + 1$ remaining colors in its palette, where d_e is the number of edges in $G[\cup_{j=i}^{\ell} H_j]$ incident on e that remain uncolored. Hence, we can color all these edges by applying the list edge coloring algorithm of Theorem 5.4, in $O(\log^8 \Delta \log n)$ rounds. This is the round complexity needed for coloring new edges after introducing each layer H_i. Hence, the overall complexity until we go through all the ℓ layers and finish the edge coloring of $G = G[\cup_{j=1}^{\ell} H_j]$ is $\ell O(\log^7 \Delta \log n) = O(\frac{1}{\varepsilon} \log^7 \Delta \log^2 n)$. $\square$

5.3.2 Approximate Maximum Matching in Graphs

Proof of Section 5.1.1. We first discuss an algorithm with complexity poly $\frac{1}{\varepsilon} \cdot O\big((\frac{1}{\varepsilon} \log \Delta)^{6+\log 1/\varepsilon} \log n\big)$, and then explain how a small change improves the complexity to poly $\frac{1}{\varepsilon} \cdot O\big((\frac{1}{\varepsilon} \log \Delta)^{7+\log 1/\varepsilon}\big)$.

We follow a well-known approach of Hopcroft and Karp [143] of increasing the size of the matching using short augmenting paths. Given a matching M, an augmenting path P with respect to M is a path that starts with an unmatched vertex, then alternates between non-matching and matching edges, and ends in an unmatched vertex. Augmenting the matching M with this path P means replacing the matching edges in $P \cap M$ with the edges $P \setminus M$. Notice that the result is a matching, with one more edge.

The approximation algorithm variant of Hopcroft and Karp [143] works as follows: For each $\ell = 1$ to $2(1/\varepsilon) - 1$, we find a maximal set of vertex-disjoint augmenting paths of length ℓ, and we augment them all. Hopcroft and Karp [143] show that this produces a

$(1+\varepsilon)$-approximation of maximum matching. See also [177], where they use the same method to obtain a $O(\log n/\varepsilon^3)$-round randomized distributed algorithm for $(1+\varepsilon)$-approximation of maximum matching, using the help of the $O(\log n)$ round randomized MIS algorithm of Luby [181].

What remains to be discussed is how do we compute a maximal set of vertex-disjoint augmenting paths of a given length $\ell \leq 2(1/\varepsilon)-1$. This can be easily formulated as a hypergraph maximal matching for a hypergraph of rank at most $1/\varepsilon+1$: create a hypergraph H by including one vertex for each unmatched vertex and also one vertex for each matching edge. Then, each augmenting path is simply a hyperedge made of its elements, i.e., its unmatched vertices and its matching edges. This hypergraph has rank at most $1/\varepsilon+1$, maximum degree at most $\Delta^{2(1/\varepsilon)}$, and the number of its vertices is no more than n. Moreover, a single round of communication on this hypergraph can be simulated in $O(1/\varepsilon)$ rounds of the base graph, simply because each hyperedge spans a path of length at most $O(1/\varepsilon)$. Hence, we can directly apply Theorem 5.1 to compute a maximal matching of it, i.e., a maximal set of vertex-disjoint augmenting paths. This runs in $O\big(\frac{1}{\varepsilon^6}(\frac{2}{\varepsilon}\log\Delta)^{6+\log(1/\varepsilon+1)}\log n\big)$ rounds. This is the complexity of the algorithm for each one value of $\ell \in [1, 2/\varepsilon-1]$. Thus, the overall complexity is at most $O\big(\frac{1}{\varepsilon^7}(\frac{2}{\varepsilon}\log\Delta)^{6+\log(1/\varepsilon+1)}\log n\big)$.

We now want to remove the factor $\log n$ from the complexity. In the above algorithm, we compute a maximal set of disjoint augmenting paths, and this precise maximality necessitates the factor $\log n$ (in our approach). However, we do not need such a precise maximality. It suffices if the set of disjoint augmenting paths is almost maximal, in particular in the sense that the fraction of the remaining augmenting paths is less than $\mathrm{poly}(\varepsilon\Delta^{-1/\varepsilon})$, say. Then, even if we permanently remove all vertices that have such a remaining augmenting path, we lose only a negligible factor $\mathrm{poly}(\varepsilon\Delta)$ of the matching, which at the end only changes our approximation ratio to $1+2\varepsilon$. To compute such an almost maximal set of disjoint augmenting paths, instead of $O(r^3 \log n)$ iterations in the proof of Theorem 5.1, it suffices to have $O\Big(r^3 \log\Big(\text{poly}\, \frac{\Delta^{1/\varepsilon}}{\varepsilon}\Big)\Big)$ iterations. This brings down the overall complexity to $\text{poly}\, \frac{1}{\varepsilon} \cdot O\big(\big(\frac{1}{\varepsilon} \log \Delta\big)^{7+\log 1/\varepsilon}\big)$. □

5.3.3 Orientations with Small Outdegree

We show how to use our hypergraph maximal matching algorithm to obtain orientations with small outdegree.

Proof of Theorem 5.10. We closely follow the approach of Ghaffari and Su [122], which iteratively improves the orientation, i.e., reduces its maximum outdegree, using suitably defined augmenting paths. They developed this approach and used it along with Luby's randomized MIS algorithm [181] to obtain a polylogarithmic round randomized algorithm for finding an orientation with outdegree at most $\lceil \lambda(1+\varepsilon) \rceil$. We show how to turn that algorithm into a quasi-polylogarithmic round deterministic algorithm, mainly by replacing their MIS module with our hypergraph maximal matching algorithm.

Let $D = \lceil \lambda(1+\varepsilon) \rceil$. Given an arbitrary orientation, we call a path P an augmenting path for this orientation if P is a directed path that starts in a vertex with outdegree at least $D+1$ and ends in a vertex with outdegree at most $D-1$. Augmenting this path means reversing the direction of all of its edges. Notice that this would improve the orientation, as it would decrease the outdegree of one of the vertices whose outdegree is above the budget D, without creating a new such vertex.

Let G_0 be the graph with our initial arbitrary orientation. Define G'_0 to be a directed graph obtained by adding a source vertex s and a sink vertex t to G_0. Then, we add $d^{\mathsf{out}}_{G_0}(u) - D$ edges from s to every vertex u with outdegree at least $D+1$, and $D - d^{\mathsf{out}}_{G_0}(u)$ edges from every vertex u with outdegree at most $D-1$ to t. We will improve the orientation gradually, in $\ell = O(\log n/\varepsilon)$ iterations. In the i^{th} iteration, we find a maximal set of edge-disjoint augmenting paths of length $3+i$ from s to t in G'_i, and then we reverse all these augmenting paths. The resulting graph is called G'_{i+1}.

Ghaffari and Su [122, Lemma D.6] showed that in this manner, each time the length of the augmenting path increases by at least an additive 1. Moreover, they showed that at the end of the process, no augmenting paths of length at most $\ell = O(\log n/\varepsilon)$ remains. They used this to prove that there must be no vertex of outdegree $D+1$ left, at the end of the process, as any such vertex would imply the existence of an augmenting path of length at most $\ell = O(\log n/\varepsilon)$ [122, Lemma D.9].

The only algorithmic piece that remains to be explained is how we compute a maximal set of edge-disjoint augmenting paths of length at most $3+i < \ell$, in a given orientation. Ghaffari and Su[122, Theorem D.4] solved this part using Luby's randomized MIS algorithm [181]. We instead use our hypergraph maximal matching algorithm. In particular, we view each edge as one vertex of our hypergraph, and each augmenting path of length at most $3+i < \ell$ as one hyperedge of our hypergraph. Then, we invoke Theorem 5.1, which provides us with a maximal set of edge-disjoint augmenting paths. The round complexity of the process is at most $\operatorname{poly} \ell \cdot \log^{\log \frac{\log n}{\varepsilon} + O(1)} \Delta \cdot \log n$, where the first term ℓ is because simulating each hyperedge needs ℓ rounds, and the second factor ℓ comes from the fact that the degree of the hypergraph may be as large as Δ^{ℓ}, which means the related logarithm is at most $\ell \log \Delta$. This is the complexity for each iteration. Since the algorithm has ℓ iterations, each time working on an incremented augmenting-path length, the overall complexity is at most $\operatorname{poly} \ell \cdot \log^{\log \frac{\log n}{\varepsilon} + O(1)} \Delta \cdot \log n$. This is no more than $2^{O(\log^2 \frac{\log n}{\varepsilon})}$ rounds, which is quasi-polylogarithmic in n for most ε-values of interest, e.g., $\varepsilon = \Omega(1/\operatorname{poly}\log n)$. $\square$

5.4 Extension to MIS and Coloring

We will now generalize the hypergraph maximal matching algorithm of Section 5.2 to computing maximal independent sets and $(\Delta + 1)$ vertex colorings of graphs of *bounded neighborhood independence*, thus proving Theorem 5.7 and Corollary 5.8.

Despite the fact that graphs of neighborhood independence r are significantly more general than line graphs of rank-r hypergraphs, we show that our techniques for computing a maximal matching in rank-r hypergraphs can be generalized to computing an MIS in graphs of neighborhood independence r, and this, in fact, even with exactly the same asymptotic dependency on r and Δ.

Intuitive Discussion

When moving from matchings in hypergraphs to independent sets in graphs of neighborhood independence at most r, it is not directly clear how to define a fractional solution of an MIS in such graphs. While in the case of hypergraph matchings, the natural LP relaxation leads to fractional solutions that are within a small factor of a maximum matching, it is not as straightforward to model fractional versions of independent sets in graphs of bounded neighborhood independence in a meaningful way. Note that for example even for $r = O(1)$, the integrality gap of the natural LP relaxation of the maximum independent set problem might be as large as $\Omega(\Delta)$.

However, any MIS is within a factor r of a maximum independent set, and this can in fact be generalized to maximal fractional solutions of the following kind. We start by setting the fractional values of all vertices to 0 and then, we iteratively increment the value of some vertices. As long as right after incrementing the value of a vertex v the total value in the 1-neighborhood of v does not exceed 1, the total value of the resulting fractional solution is guaranteed to be within a factor r of a maximum independent set. We call such a fractional solution a *greedy packing* and show that our rounding scheme for hypergraph matching can be adapted to greedy packings of graphs of bounded neighborhood independence.

Integral greedy packings are exactly independent sets. Thus, integral greedy packings of the line graph of a hypergraph H correspond to matchings of H. However, we note that a fractional greedy packing of the line graph of H is not the same as a fractional matching of H. We believe that this stresses the robustness of our approach. For example, when running the MIS algorithm for graphs of bounded neighborhood independence on the line graph of a bounded rank hypergraph H, we get a slightly different but equally efficient algorithm for computing a maximal matching of H.

Overview and Outline

In the following, we study special fractional solutions x that assign a non-negative value $x_v \geq 0$ to each vertex $v \in V$ of a graph $G = (V, E)$ and allow to approximate maximum and maximal independent sets in G if G is a graph of bounded neighborhood independence. For convenience, we first introduce some notation. Recall that given a graph $G = (V, E)$ and a vertex $v \in V$, we use $N(v)$ to denote the set of neighbors of v. Further, we define

$$N^+(v) := \{v\} \cup N(v)$$

to denote the set of vertices in the 1-neighborhood of v. Moreover, for vertex vector x assigning values x_v to every vertex $v \in V$, for each vertex $v \in V$, we define

$$\Sigma_x(v) := \sum_{u \in N^+(v)} x_u$$

to be the local sum of the values x_u in the 1-neighborhood of v. As a fractional relaxation of the independent set of a graph G, we define a *greedy packing* as follows.

Definition 5.24 (Greedy Packing)**.** *For a graph $G = (V, E)$, a vertex vector x assigning a non-negative value $x_v \geq 0$ to each vertex $v \in V$ is called a* greedy packing *if there exists a global order $\prec$ on the vertices V such that*

$$\forall v \in V : \quad x_v + \sum_{u \in N(v) : u \prec v} x_u \leq 1.$$

Hence, in a greedy packing, the values x_v can be assigned to the vertices in some order such that for all vertices $v \in V$, when vertex v gets assigned value x_v, the sum of the values in v's 1-neighborhood is bounded by 1.

Analogously to the matching algorithm in Section 5.2, the key part is a recursive algorithm that finds and independent set that is an approximation of a maximum independent set in graphs of bounded neighborhood independence. We formally prove the following result.

Lemma 5.25. *In graphs of neighborhood independence at most r, there is a deterministic LOCAL algorithm that computes a $(32r^3)$-approximate independent set in $O\big(r^2 \log^{6+\log r} \Delta\big)$ rounds, given an $O(\Delta^2)$ vertex coloring.*

We first show that in graphs of bounded neighborhood independence for any such greedy packing x, the local sum $\Sigma_x(v)$ is bounded for all vertices.

Lemma 5.26. *Let $G = (V, E)$ be a graph with neighborhood independence at most r and assume that we are given a greedy packing x. Then, for all $v \in V$, we have $\Sigma_x(v) \leq r$.*

Proof. Consider an arbitrary vertex $v \in V$ and let G_v be the subgraph of G induced by the vertices in $N^+(v)$. Let $\prec$ be the global order on V which is defined by Definition 5.24 because x is a greedy packing. Assume that the vertices in $N^+(v)$ are named $u_0, \ldots, u_{d(v)}$

such that for all $0 \leq i < d(v)$, $u_i \succ u_{i+1}$. We construct an MIS S of G_v by processing the vertices in $N^+(v)$ in the order $u_0, u_1, \dots, u_{d(v)}$, always adding the current vertex u_i to S if no neighbor of u_i has already been added to S. In this way, every vertex $u_i \in N^+(v) \setminus S$ has an MIS neighbor $u_j \in N^+(v)$ for which $j < i$ and thus $u_i \prec u_j$. We charge the value x_{u_i} of every vertex $u_i \in N^+(v) \setminus S$ to some MIS neighbor u_j for which $j < i$. In addition, the value x_{u_j} of each MIS vertex $u_j \in S$ is charged to the vertex itself. For each MIS vertex $u_j \in S$, let X_{u_j} be the total value charged to u_j. We can upper bound X_{u_j} as follows:

$$X_{u_j} \leq \sum_{u_i \in N^+(u_j) \cap N^+(v) : i > j} x_{u_i} \leq x_{u_j} + \sum_{w \in N(u_j) : w \prec u_j} x_w \leq 1.$$

The last inequality follows because x is a greedy packing with respect to the global order $\prec$. The claim of the lemma now follows because G has neighborhood independence bounded by r, and thus the MIS S can contain at most $|S| \leq r$ vertices. $\square$

We will show how to recursively compute a large greedy packing. Before doing this, we first prove some useful simple properties of greedy packings.

Lemma 5.27. *Given a global order $\prec$ on the vertices V and a greedy packing x with respect to the order $\prec$. Then, the following statements hold:*

(1) Let $v \in V$ be vertex for which $\Sigma_x(v) \leq 1$ and let y be a vertex vector such that $y_u = x_u$ for all $u \neq v$ and such that $y_v \leq x_v + 1 - \Sigma_x(v)$. Then y is also a greedy packing.

(2) Let $U \subseteq V$ be a subset of the vertices and let y be a vertex vector such that $y_v = x_v$ for all vertices $v \in V \setminus U$ and such that $y_u \geq x_u$ for all $u \in U$. If $\Sigma_y(u) \leq 1$ for all vertices $u \in U$, y is also a greedy packing.

(3) Let $U \subseteq V$ be a set of vertices u for which $\Sigma_x(u) \leq 1/2$ and consider a vertex vector y such that $y_v = x_v$ for all $v \notin U$ and such that $y_u = 2x_u$ for all $u \in U$. Then y is also a greedy packing.

Proof. We first prove claim (1). Consider a global order $\prec_0$ that is obtained from $\prec$ by moving vertex v to the very end of the order (without changing the relative order of any of the other vertices). We claim that y is a greedy packing with respect to the global order $\prec_0$. For all vertices $u \neq v$, the condition of Definition 5.24 follows because x is a greedy packing with respect to the global order $\prec$. For vertex v, the condition follows because $\Sigma_y(v) = \Sigma_x(v) + y_v - x_v \leq 1$.

Claim (2) follows from claim (1) by sequentially processing the vertices in U. We start with vector x and when processing vertex u, we replace the current value x_u of vertex u by y_u. Because for all vertices $y_v \geq x_v$, for each of the intermediate vectors z, we have $\Sigma_z(u) \leq 1$ for all $u \in U$. The conditions for claim (1) are therefore satisfied for each vertex $u \in U$.

Finally, to prove claim (3), observe that because we have $\Sigma_x(u) \leq 1/2$ for all vertices $u \in U$, the local sum for the vertices in u is still bounded by 1 even if we double the values of all vertices $v \in V$. Claim (3) therefore follows as a special case of claim (2). □

In order to recursively compute a large greedy packing, we will need to be able to add a new greedy packing to an existing one. The next lemma shows in which way this can be done. In the following, given a real-valued non-negative vertex vector x and a parameter $c > 0$, we say that a vertex $v \in V$ is c-tight if $\Sigma_x(v) \geq c$.

Lemma 5.28. *Let $G = (V, E)$ be a graph and let x be a greedy packing of G. Further, let $F \subseteq V$ be the vertices of G that are not $1/2$-tight with respect to x and let y be a greedy packing of G*

for which $y_v > 0$ only for $v \in F$. Then, the fractional assignment $z := x + y/2$ is a greedy packing of G.

Proof. Assume that x is a greedy packing of G with respect to to the global order $\prec_x$ on V and that y is a greedy packing of G with respect to the global order $\prec_y$ on F. Note that for all vertices $v \in F$, we have $\Sigma_x(v) < 1/2$. Therefore for the greedy packing x the condition of Definition 5.24 is satisfied for the vertices in F for every choice of the global order $\prec_x$. Without loss of generality, we can therefore assume that $\prec_x$ first orders all vertices in $V \setminus F$ and it then orders the vertices in F in an arbitrary way. We can thus define a global order $\prec$ on the vertices V as a combination of $\prec_x$ and $\prec_y$ in an obvious way. The order $\prec$ first orders the vertices in $V \setminus F$ in the same order as $\prec_x$ and it then orders the vertices in F in the same order as $\prec_y$. We show that $z = x + y/2$ is a greed packing of G with respect to the global order $\prec$. For each vertex $v \in V \setminus F$, the condition of Definition 5.24 is satisfied because x is a greedy packing. For a vertex $v \in F$, we have

$$z_v + \sum_{u \in N(v): u \prec v} z_u = \frac{y_v}{2} + \sum_{u \in N(v) \cap F: u \prec_y v} \frac{y_u}{2} + x_v + \sum_{u \in N(v): u \prec_x v} x_u$$
$$< \frac{1}{2} + \frac{1}{2} = 1,$$

and therefore the claim of the lemma follows. □

As in the case of computing matchings in hypergraphs, our goal is to start with a large fractional greedy packing and to gradually round the fractional solution to an integer one of approximately the same size. For a given parameter $\delta > 0$, a non-negative real-valued vertex vector x is called δ-fractional if for every vertex $v \in V$, either $x_v = 0$ or $x_v \geq \delta$. Given a parameter $L > 1$, we show how to recursively turn a δ-fractional greedy packing into an $(L\delta)$-fractional greedy packing of a similar size. The proof follows the same basic structure as the rounding for fractional hypergraph matchings in

Sections 5.2.3 and 5.2.4. The following lemma provides a way to upper bound the size of a greedy packing in terms of another greedy packing. We will use it to compare the size of a computed $(L\delta)$-fractional greedy packing to the existing δ-fractional greedy packing.

Lemma 5.29. *Let x and y be two greedy packings of a n-vertex graph with neighborhood independence at most r. Further, let $U \subseteq V$ be the set of vertices of V for which $\Sigma_y(v) \geq 1/2$. We have*

$$\sum_{v \in V} y_v \geq \frac{1}{2r} \sum_{v \in U} x_v.$$

Proof. To prove the lemma, we use a blaming argument. Let V_y be the set of vertices for which $y_v > 0$. We distribute all the x_v-values of vertices in U among the vertices in V_y. That is, for each vertex $v \in V_y$, we define a variable α_v such that $\sum_{v \in V_y} \alpha_v = \sum_{v \in U} x_v$. More concretely, we define the values α_v for each vertex $v \in V_y$ as follows:

$$\alpha_v := \sum_{u \in N^+(v) \cap U} x_u \cdot \frac{y_v}{\Sigma_y(u)} \leq \sum_{u \in N^+(v) \cap U} x_u \cdot 2y_v = 2y_v \Sigma_x(v). \quad (5.3)$$

Hence, every vertex $u \in U$ distributes its value x_u among the neighboring vertices in $v \in V_y$ proportionally to the values y_v. Because x is a greedy packing of G, Lemma 5.26 implies that $\Sigma_x(v) \leq r$ for all $v \in V$. Together with (5.3), we thus get $\alpha_v \leq 2ry_v$ for all $v \in V_y$ and the claim of the lemma follows. □

5.4.1 Basic Rounding of Greedy Packings

Lemma 5.30 (Basic Rounding of Greedy Packings)**.** *Assume that a parameter $L > 1$, an integer $d \geq L$, and a $(1/d)$-fractional greedy packing x as well as an $O(\Delta^2)$ vertex coloring of an n-vertex graph with neighborhood independence at most r are given. Then there is* *an $O\big((rL)^2 + \log^* \Delta + \log d\big)$-round deterministic* LOCAL *algorithm that computes an (L/d)-fractional greedy packing y for which $y_v > 0$ only if $x_v > 0$ and such that y is of size $\sum_{v \in V} y_v \geq \frac{1}{2r} \sum_{v \in V} x_v$.*

Proof. Let V_x be the set of vertices $v \in V$ for which $x_v > 0$ and let $G_x = G[V_x]$ be the subgraph of G induced by V_x. Note that because x is a greedy packing, Lemma 5.26 implies that for every vertex $v \in V$, $\Sigma_x(v) \leq r$ and since x is $(1/d)$-fractional, this implies that G_x has maximum degree at most $r \cdot d$. We compute the (L/d)-fractional greedy packing y in two steps. In a first step, we compute an arbitrary (L/d)-fractional greedy packing z of G by assigning value $z_v = L/d$ to a subset of the vertices $v \in V_x$. In the second step, we obtain y from z by iteratively doubling the value of each vertex that is not $(1/2)$-tight at most $O(\log d)$ times.

For the first step, we apply the deterministic defective coloring algorithm of Kuhn [159]. For a C vertex colored graph G of maximum degree Δ and a parameter $p \geq 1$, the algorithm allows to compute a p-defective $O((\Delta/p)^2)$-coloring of G in time $O(\log^* C)$. That is, the algorithm assigns one of $O((\Delta/p)^2)$ colors to each vertex of G such that the subgraph induced by each of the colors has maximum degree at most p. We apply the defective coloring algorithm of [159] to the graph G_x with parameter $p = d/(2L)$. Because we are given an $O(\Delta^2)$-coloring of G (and thus also of G_x), the time for computing this defective coloring is $O(\log^* \Delta)$ and because the maximum degree of G_x is at most dr, the number of colors of the defective coloring is at most $O\big((rL)^2\big)$.

We now compute an initial (L/d)-fractional greedy packing z as follows. For all vertices $v \in V \setminus V_x$, we set $z_v = 0$. For the vertices in V_x, we iterate through the $O((rL)^2)$ colors of the defective coloring of G_x and process all vertices of the same color in parallel. At the beginning, we set $z_v = 0$ for all $v \in V_x$. When processing the vertices of colors c, for each vertex $v \in V_x$ of color c, we set $z_v = L/d$ if and only if $\Sigma_z(v) \leq 1/2$. Because each vertex of color c has at most $d/(2L)$ neighbors of color c, this implies that even after this step, $\Sigma_z(v) \leq 1$ for all vertices of color c. Claim (2) of Lemma 5.27 therefore implies that throughout this process, vector z remains a valid greedy packing. Because at the end all non-zero values of z are equal to L/d, clearly, z is (L/d)-fractional. Note also that for all vertices $v \in V_x$ for which $z_v = 0$, we have $\Sigma_z(v) \geq 1/2$.

To obtain the greedy packing y from z we first set $y = z$ and we then proceed in synchronous rounds. Let $V_y \subseteq V_x$ be the set of vertices for which $z_v > 0$. In each round, each vertex $v \in V_y$ for which $\Sigma_y(v) \leq 1/2$ doubles its value y_v. The process stops when $\Sigma_y(v) \geq 1/2$ for all vertices $v \in V_y$. Because $y_v \geq 1/2$ implies that $\Sigma_y(v) \geq 1/2$, this happens after at most $\log(d/L) \leq \log d$ rounds. Claim (3) of Lemma 5.27 implies that the vector y remains a valid greedy packing throughout this process.

We therefore obtain an (L/d)-fractional greedy packing y where for each vertex $v \in V_x$, we have $\Sigma_y(v) \geq 1/2$. Lemma 5.29 then shows that that $\sum_{v\in V} y_v \geq 1/(2r)\sum_{v\in V} x_v$, which concludes the proof. □

5.4.2 Recursive Rounding of Greedy Packings

We next explain a recursive method $\mathsf{round}(x, L)$ that given a δ-fractional greedy packing x of a graph $G = (V, E)$ of neighborhood independence $\leq r$ computes an $(L\delta)$-fractional greedy packing y of G of size $\sum_{v\in V} y_v \geq \frac{1}{4r}\sum_{v\in V} x_v$ and such that $y_v > 0$ only if $x_v > 0$. The method $\mathsf{round}(x, L)$ runs in $16r$ phases. At the beginning, $y = 0$ and in each phase, some values of y are increased. As soon as for some vertex $v \in V$, $\Sigma_y(v) \geq 1/2$, the value y_v is not increased any further. In each phase, the method therefore first defines a δ-fractional greedy packing z which is identical to x on all vertices v for which $\Sigma_y(v) < 1/2$ and which is 0 on all other vertices. On this vector z, the method is called recursively with parameter $\sqrt{2L}$, resulting in a $(\sqrt{2L}\delta)$-fractional greedy packing z'.

Afterwards, the method is again called recursively with parameter $\sqrt{2L}$ on the vector z', resulting in a $(2L\delta)$-fractional greedy packing z''. Finally, the vector y is updated by adding $z''/2$ to it.

The following lemma shows that the algorithm $\mathsf{round}(x, L)$ computes an $(L\delta)$-fractional greedy packing of size within a factor $4r$ of the size of x.

Lemma 5.31. *Assume that we are given parameters $0 < \delta < 1$ and $L < 1/(2\delta)$, and a δ-fractional greedy packing x of a n-vertex graph with neighborhood independence at most r. Then the method $\mathsf{round}(x, L)$ computes an $(L\delta)$-fractional greedy packing y for which $y_v > 0$ only if $x_v > 0$ and such that*

$$\sum_{v \in V} y_v \geq \frac{1}{4r} \sum_{v \in V} x_v. \tag{5.4}$$

Proof. Note that for $L \leq 4$, the algorithm directly applies the basic rounding algorithm of Lemma 5.30 and the claims of the lemma thus directly hold by applying Lemma 5.30. Let us therefore assume that $L > 4$ and let us therefore (inductively) also assume that the recursive calls to $\mathsf{round}(z, \sqrt{2L})$ and $\mathsf{round}(z', \sqrt{2L})$ satisfy the claims of the lemma.

We first show that y is an $(L\delta)$-fractional greedy packing of G and that $y_v > 0$ only if $x_v > 0$. Note that $z_v > 0$ only if $x_v > 0$ and we have $z'_v > 0$ only if $z_v > 0$ and $z''_v > 0$ only if $z'_v > 0$ because that is guaranteed by the recursive calls to $\mathsf{round}(z, \sqrt{2L})$ and $\mathsf{round}(z', \sqrt{2L})$. Because y_v is only increased for vertices $v \in V$ for which $z''_v > 0$, we therefore have $y_v > 0$ only if $x_v > 0$ throughout the algorithm. To see that y is $(L\delta)$-fractional, note that because x is δ-fractional, the recursive calls to $\mathsf{round}(z, \sqrt{2L})$ and $\mathsf{round}(z', \sqrt{2L})$ guarantee that z' is $(\sqrt{2L}\delta)$-fractional and z'' is $(2L\delta)$-fractional. We update y by adding $z''/2$ and thus an $(L\delta)$-fractional vector to it. Thus, y is $(L\delta)$-fractional at all times during the execution of the method $\mathsf{round}(\cdot,\cdot)$. We prove that y at all times is a greedy packing by induction on the number of phases. Clearly at the beginning when $y_e = 0$ for all e, y is a greedy packing. Also, whenever, y is updated, we add $z''/2$ to it. Note that because z'' is the result of the call to $\mathsf{round}(z', \sqrt{2L})$, z'' is a greedy packing. Further, $z''_v > 0$ only where $z_v > 0$ and thus only for vertices v where $\Sigma_y(v) < 1/2$ at the beginning of the respective phase. It therefore follows directly from Lemma 5.28 that $y + z''/2$ is a greedy packing of G.

It thus remains to show (5.4). As long as (5.4) does not hold, at the beginning of each of the $16r$ phases, we have $\sum_{v\in V} x_v > 4r \sum_{v\in V} y_v$. Let V_z be the set of vertices for which $\Sigma_y(v) < 1/2$ and $\bar{V_z} := V \setminus V_z$ be the set of vertices for which $\Sigma_y(v) \geq 1/2$. From Lemma 5.29, we get that $\sum_{v\in\bar{V_z}} x_v \leq 2r \sum_{v\in V} y_v$ and we thus have $\sum_{v\in V} z_v = \sum_{v\in V_z} x_v > \frac{1}{2}\sum_{v\in V} x_v$. From the guarantees of the recursive calls to $\mathsf{round}(z, \sqrt{2L})$ and $\mathsf{round}(z', \sqrt{2L})$, the size of $z''/2$ that we add to y is thus

$$\frac{1}{2}\sum_{v\in V} z''_v \;\geq\; \frac{1}{8r}\sum_{v\in V} z'_v \;\geq\; \frac{1}{32r^2}\sum_{v\in V} z_v \;\geq\; \frac{1}{64r^2}\sum_{v\in V} x_v.$$

After $16r$ phases, we therefore have $\sum_{v\in V} y_v \geq \frac{16r}{64r^2}\sum_{v\in V} x_v$. □

Lemma 5.32. *It takes $O\big((r^2 + \log\Delta)\log^{5+\log r} L\big)$ rounds to run the method* $\mathsf{round}(x, L)$*.*

Proof. The proof is identical to the proof of Lemma 5.22, the analogous result in the analysis for hypergraph maximal matching. The round complexity $R(L)$ of $\mathsf{round}(x, L)$ follows the recursive inequality $R(L) \leq 16r(R(\sqrt{2L}) + R(\sqrt{2L}) + O(1))$. Furthermore, we have the base case solution of $R(L) = O(L^2r^2 + \log\Delta)$ for $L = O(1)$. The claim can now be proved by an induction on L, as formalized in Lemma 5.11. □

5.4.3 Maximum and Maximal Independent Set

Approximate Maximum Independent Set

We now use our rounding procedure to find the approximate independent set of Lemma 5.25.

Proof of Lemma 5.25. Let S^* be some maximum independent set of the given graph $G = (V, E)$ with neighborhood independence $\leq r$. We first compute a $(1/\Delta)$-fractional greedy packing x of size at least $\frac{1}{2r} \cdot |S^*|$. To compute x, we initially set $x_v = 1/\Delta$ for all vertices $v \in V$. As this guarantees that $\Sigma_x(v) \leq 1$ for all $v \in V$, this initial vector x clearly is a greedy packing. Now, we proceed in $\log \Delta$ synchronous rounds, where in each round, all vertices $v \in V$ for which $\Sigma_x(v) < 1/2$ double their value x_v. Claim (3) of Lemma 5.27 implies that the vector x remains a greedy packing throughout this process. Further, after at most $\log \Delta$ doubling steps, we certainly have $\Sigma_x(v) \geq 1/2$ for all vertices $v \in V$. Lemma 5.29 therefore implies that $\sum_{v \in V} x_v \geq \frac{1}{2r} \sum_{v \in V} y_v$ for every greedy packing y of G. The claim that $\sum_{v \in V} x_v \geq |S^*|/(2r)$ now follows because for any independent S set of G, setting $y_v = 1$ for $v \in S$ and $y_v = 0$ otherwise results in a greedy packing y.

Given the greedy packing x, we now apply the recursive rounding from Lemma 5.31 for $L = \Delta / \log^2 \Delta$. This produces a $(1/\log^2 \Delta)$-fractional greedy packing x', in $O\big(\log^{5+\log r} \Delta (r^2 + \log \Delta)\big)$ rounds, whose size is a factor $(1/(8r^2))$ of $|S^*|$. To finish up the rounding, we apply the basic rounding of Lemma 5.30 for $L = \log^2 \Delta$, which runs in $O(r^2 \log^2 \Delta)$ and produces a 1-fractional, i.e., integral, greedy packing x'' of size is at least a $(1/(4r))$ times the size of x'. Hence, the final produced integral greedy packing is a $(32r^3)$-approximation of the maximum independent set S^*. □

Maximal Independent Set

We compute a maximal independent set via iterative applications of this independent set approximation.

Proof of Theorem 5.7. First, we pre-compute an $O(\Delta^2)$ vertex coloring of G in $O(\log^* n)$ rounds by Linial's algorithm [171]. Then, iteratively, we apply the maximum independent set approximation procedure of Lemma 5.25 to the remaining graph. We add the found independent set S to the independent set that we will output at the end, and remove S along with its neighbors from the graph.

In each iteration, the size of the maximum independent set of the remaining graph goes down to at least a factor of $1 - 1/(32r^3)$. This is because otherwise we could combine the independent set computed so far with the maximum independent set in the remaining graph to obtain an independent set larger than the maximum independent set in G. After at most $O(r^3 \log n)$ repetitions, the remaining independent set size is 0, which means the remaining graph is empty. Hence, we have found a maximal independent set in $O\left(\log^* n + r^3 \log n \left(r^2 \log^{6+\log r} \Delta\right)\right) = O(r^5 \log^{6+\log r} \Delta \cdot \log n)$ rounds. $\square$

CHAPTER 6

Local Algorithms for the Lovász Local Lemma

6.1 Introduction

In this section, we present our work in the publication 'Sublogarithmic Distributed Algorithms for Lovász Local Lemma' [101], where we devise a fast randomized LOCAL algorithm for the constructive Lovász Local Lemma problem.

6.1.1 Our Results and Related Work

Despite its centrality for distributed computing, the state of the art for distributed algorithms for the LLL problem is rather bleak. In their celebrated work, Moser and Tardos [189] provide a randomized parallel algorithm, which can be transformed into a $O(\log^2 n)$-round LOCAL algorithm in a straight-forward manner. Chung, Pettie, and

Su [67] presented an $O(\log n \cdot \log^2 d)$-round algorithm, which was later improved slightly to $O(\log n \cdot \log d)$ [109]. Perhaps more importantly, under a modestly stronger[1] criterion that $epd^2 < 1$, which is satisfied in most of the standard applications, they gave an $O(\log n)$-round algorithm [67]. On the other hand, Brandt et al. [46] showed a lower bound of $\Omega(\log_d \log n)$ rounds, which holds even if a much less permissive LLL criterion of $p \cdot 2^d < 1$ is satisfied. Even under this exponentially stronger criterion, the best known upper bound had changed only slightly to $O(\log n / \log\log n)$ [67]. Although a wide gap between the best upper and lower bound had persisted, Chang and Pettie [61] conjecture the latter to be tight:

> **Conjecture by Chang and Pettie [61]:**
> *"There exists a sufficiently large constant c such that the distributed LLL problem can be solved in $O(\log\log n)$ time on bounded degree graphs, under the symmetric LLL criterion $pd^c < 1$."*

Randomized LLL Algorithm

Making the first step of progress towards this conjecture, and providing a significant improvement on the algorithm of Chung et al. [67], we prove $T_{\mathsf{LLL}}(n) = 2^{O(\sqrt{\log\log n})}$ under the symmetric polynomial LLL criterion that $p(ed)^{32} < 1$.[2]

Theorem 6.1. *There is a $2^{\operatorname{poly} d + O(\sqrt{\log\log n})}$-round LOCAL algorithm that w.h.p. solves the LLL problem under a polynomial LLL criterion $p(ed)^{32} < 1$. In particular, for LLL problems with $d = O(\log^{1/5} \log n)$, we get a $2^{O(\sqrt{\log\log n})}$-round algorithm.*

[1]We say that a criterion is stronger if it is harder to satisfy, so less permissive, and the corresponding theorem weaker.

[2]We remark that we did not try to optimize these constants, and, for that matter, any constant in this thesis.

This improves on the $O(\log n)$-round algorithm of Chung et al. [67]. Our method provides some further supporting evidence for the conjecture of Chang and Pettie [60, 61] that the Lovász Local Lemma can be solved in $O(\log\log n)$ rounds for $d = O(1)$. In particular, if we combine our method with the recent network decomposition algorithm [216] and the derandomization technique [115], we arrive at an $O(d^2) + \operatorname{poly}\log\log n$ round complexity, thus $\operatorname{poly}\log\log n$ for constant-degree LLLs, getting close to the bound conjectured by Chang and Pettie [61] as well as the lower bound $\Omega(\log\log n)$ [46] for $p2^d \leq 1$.

We note that even under a significantly stronger exponential LLL criterion, formally requiring $4epd^4 \cdot 2^d < 1$, the best known round complexity before our result was $O(\log n/\log\log n)$ [67]. Interestingly, for $p \cdot 2^d < 1$ under some additional conditions, Brandt et al. [47] provided an $O(d^2 + \log^* n)$-round deterministic algorithm, which is tight in bounded-degree graphs due to the $\Omega(\log^* n)$ lower bound by Chung, Pettie, and Su [67].

Gap in the Randomized Distributed Complexity Hierarchy

Putting Theorem 6.1 with [60, Theorem 6], we get the following automatic speedup result:

Corollary 6.2. *Let $\mathcal{A}$ be a randomized LOCAL algorithm that solves an LCL problem $\mathcal{P}$ on bounded-degree graphs w.h.p. in $o(\log n)$ rounds. Then it is possible to transform $\mathcal{A}$ into a new randomized LOCAL algorithm $\mathcal{A}'$ that solves $\mathcal{P}$ w.h.p. in $2^{O(\sqrt{\log\log n})}$ rounds.*

Using a similar method, as well as our deterministic LLL algorithm from Theorem 6.10, we obtain the following corollary, the proof of which appears in Section 6.3.3. It shows that any $o(\log\log n)$-round randomized algorithm for an LCL problem on bounded-degree graphs can be improved to a deterministic $O(\log^* n)$-round LOCAL algorithm. This result seems to be implicit in the recent work of

Chang, Kopelowitz, and Pettie [57], though with a quite different proof. It can be derived from [57, Corollary 3] and [57, Theorem 3].

Corollary 6.3. *Let $\mathcal{A}$ be a randomized* LOCAL *algorithm that solves some* LCL *problem $\mathcal{P}$ on bounded-degree graphs w.h.p. in $o(\log\log n)$ rounds. Then, it is possible to transform $\mathcal{A}$ into a new deterministic* LOCAL *algorithm $\mathcal{A}'$ that solves $\mathcal{P}$ in $O(\log^* n)$ rounds.*

Algorithms for Various Graph Coloring Problems

For several distributed graph problems on bounded-degree graphs, we can immediately get faster algorithms by applying our LLL algorithm. However, there are two quantifiers which appear to limit its applicability: (1) it needs a stronger form of the LLL criterion, concretely requiring $p(ed)^{32} < 1$ instead of $epd \leq 1$; (2) it applies to graphs with degree $d = O(\log^{1/5}\log n)$. We explain how to overcome these two limitations in most of the LLL-based problems studied by Chung, Pettie, and Su [67]. Regarding limitation (1), we show that even though in many coloring problems the direct LLL formulation would not satisfy the polynomial criterion $p(ed)^{32} < 1$, we can still solve the problem, through a number of iterations of partial colorings, each of which satisfies this stronger LLL criterion. Regarding limitation (2), we explain that in most of these coloring problems, the first step of our LLL algorithm, which is its only part that relies on bounded degrees, can be replaced by a faster randomized step, suited for that coloring. The end results of our method contain algorithms with round complexity $2^{O(\sqrt{\log\log n})}$ (and combined with [115, 216] in poly $\log\log n$ rounds) for a number of coloring problems including defective coloring, frugal coloring, and list vertex-coloring, substantially improving on the corresponding $O(\log n)$-round algorithms of Chung, Pettie, and Su [67].

One key ingredient for these coloring algorithms is another result obtained via our speedup method, targeting higher-degree graphs.

Lemma 6.4. *Let $\mathcal{A}$ be a randomized LOCAL algorithm that solves some LCL problem $\mathcal{P}$ on n-vertex graphs with maximum degree $d = 2^{O(\log^{1/4}\log n)}$ in $O(\log^{1/4} n)$ rounds. Then, it is possible to transform $\mathcal{A}$ into a new randomized LOCAL algorithm $\mathcal{A}'$ that solves $\mathcal{P}$ w.h.p. in $2^{O(\sqrt{\log\log n})}$ rounds.*

The proof is deferred to Section 6.3.2.

We next present the different results for several variants of coloring problems.

Defective Coloring: An f-defective coloring is a (not necessarily proper) coloring of vertices, where each vertex has at most f neighbors with the same color. In other words, in an f-defective coloring, each color class induces a subgraph with maximum degree f. Chung, Pettie, and Su [67] gave an $O(\log n)$-round distributed algorithm for computing an f-defective coloring with $O(\Delta/f)$ colors. We here improve this complexity to $2^{O\left(\sqrt{\log\log n}\right)}$ rounds.

Theorem 6.5. *There is a $2^{O\left(\sqrt{\log\log n}\right)}$-round randomized distributed algorithm that w.h.p. computes an f-defective $O(\Delta/f)$-coloring in an n-vertex graph with maximum degree Δ, for any integer $f \geq 0$.*

Frugal Coloring: An *f-frugal coloring* is a proper coloring in which no color appears more than f times in the neighborhood of any vertex. We improve the complexity of f-frugal $O(\Delta^{1+1/f})$-coloring from $O(\log n)$ by Chung, Pettie, and Su [67] to $2^{O(\sqrt{\log\log n})}$.

Theorem 6.6. *There is a $2^{O(\sqrt{\log\log n})}$-round randomized distributed algorithm for a f-frugal $(120\Delta^{1+1/f})$ coloring[3] in a graph with n vertices and maximum degree Δ w.h.p. for any integer $f \geq 1$.*

List Vertex Coloring: A list vertex coloring assigns each vertex v a color from its *color list* L_v such that no two neighboring vertices

[3]We remark that we have not tried to optimize this constant 120.

choose the same color. The color lists satisfy the following properties: (1) $|L_v| \geq L$ for all $v \in V$. We emphasize that the list size L may be much smaller than the degree Δ. (2) For each $v \in V$ and each color $q \in L_v$, the set $N_q(v) = \{u \,|\, u \in N(v) \text{ and } q \in L_u\}$ of neighbors u of v that also have color q in their color list L_u has size $|N_q(v)| \leq L/C$, for a given large constant $C > 2e$.

Chung, Pettie, and Su [67] gave an $O(\log n)$-round randomized distributed algorithm for list vertex coloring with $C \geq 2e + \varepsilon$. We here improve this complexity to $2^{O(\sqrt{\log\log n})}$ rounds, for a sufficiently large constant C which we have not tried to optimize.

Theorem 6.7. *There is a $2^{O(\sqrt{\log\log n})}$-round randomized LOCAL algorithm that w.h.p. computes a list vertex coloring in an n-vertex graph where each list L_v has size L and for each color $q \in L_v$, we have we have $|N_q(v)| \leq L/C$, for some sufficiently large constant $C > 2e$.*

6.1.2 Overview and Outline

Our sublogarithmic-time LLL algorithm of Theorem 6.1—which solves LLL in $2^{O(\text{poly}\, d + \sqrt{\log\log n})}$ rounds, given the condition that $p(ed)^{32} < 1$—is developed in two stages: a base algorithm that runs in $O(d^2) + \log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})}$ rounds under the LLL criterion $p(ed)^{4\beta} < 1$; and a bootstrapping part to boost the running time of the base algorithm to $2^{O(\sqrt{\log\log n})}$ for $d = O(\log^{1/5}\log n)$.

Stage 1: Base LLL Algorithm

In the first stage we run a randomzied sublogarithmic-round algorithm based on the shattering technique. Its proof can be found in Section 6.2. The main idea is to color the square of the dependency graph and go through the color classes one by one.

Theorem 6.8. *For any integer $\beta \geq 8$, there is a randomized distributed algorithm w.h.p. solving the* LLL *problem under the symmetric criterion $p(ed)^{4\beta} < 1$, in $O(d^2) + \beta \log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})}$ rounds.*

In the main regime of interest, the best LLL criterion exponent that we will assume is $\beta = O(1)$, and thus this $(\beta \log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})})$-round algorithm, on its own, would not get us to our target complexity of $2^{O(\sqrt{\log\log n})}$, although still being an improvement on the $O(\log n)$-round algorithm of [67].

Stage 2: Bootstrapping of Base LLL Algorithm

In the second stage—presented in Section 6.3—we boost the round complexity of our base algorithm to $2^{O(\operatorname{poly} d + \sqrt{\log\log n})}$ with the help of a *bootstrapping* approach where the LLL algorithm generates a new LLL instance with amplified slack in the LLL criterion.

6.2 Base LLL Algorithm

6.2.1 Overview and Outline

Our base algorithm follows the shattering technique and consequently consists of two parts: a randomized shattering and a deterministic post-shattering.

Randomized Shattering Algorithm

Our randomized shattering algorithm performs a partial sampling in the LLL space—i.e., finds an assignment for some of the variables of the LLL—in a manner that shatters the graph so that the leftover graph induced by events with still unset variables is somewhat small.

Partial Assignment: We use $\mathcal{X}^* \subseteq \mathcal{X}$ for the variables already set by the shattering algorithm; $\mathcal{X}' := \mathcal{X} \setminus \mathcal{X}^*$ for the still unset variables; and $\mathcal{B}'$ for the events that still have at least one unset variable, thus $\mathsf{vbl}(B) \cap \mathcal{X}' \neq \varnothing$ for all $B \in \mathcal{B}'$. Moreover, $G_{\mathcal{B}'}[\mathcal{X}']$ stands for the graph induced by events in $\mathcal{B}'$ and variables in $\mathcal{X}'$, where events $B, B' \in \mathcal{B}'$ are connected if $\mathsf{vbl}(B) \cap \mathsf{vbl}(B') \cap \mathcal{X}' \neq \varnothing$.

Lemma 6.9 (LLL Shattering)**.** *There is a randomized LOCAL $O(d^2 + \log^* n)$-round algorithm that w.h.p. computes a partial assignment of values to variables in $\mathcal{X}^*$ of an LLL satisfying $p(ed)^{4\beta} < 1$, for any integer $\beta \geq 8$, such that*

(i) $\Pr[B \mid \mathcal{X}^*] \leq \sqrt{p}$ for all $B \in \mathcal{B}$, and

(ii) each connected component of the square $G^2_{\mathcal{B}}[\mathcal{X}']$ of the leftover graph w.h.p. admits a $(\beta, O(\log^{1/\beta} n \cdot \log^2 \log n))$ network decomposition, which can be computed in $\beta \log^{1/\beta} n \cdot 2^{O(\sqrt{\log \log n})}$ rounds.

This algorithm essentially works by going over the $O(d^2)$ color classes of $G^2_{\mathcal{B}}$ one by one, setting the variables of the corresponding events carefully as to ensure to not make a bad event too likely. The distance coloring (coloring of $G^2_{\mathcal{B}}$ as opposed to $G_{\mathcal{B}}$) is needed to ensure that for no event more than one incident event is setting its variables at the same time. A precise algortihm description as well as the proof can be found Section 6.2.2.

The two properties of Lemma 6.9 will allow us to invoke the deterministic LLL algorithm that we present later in Section 6.2.3 on the components of events with unset variables. In particular, (i) means that the bad events $\mathcal{B}$ form another LLL problem on the variables that remain unset, where each new bad event has probability at most $\sqrt{p}$. Furthermore, (ii) ensures that the components are small enough to make the deterministic algorithm efficient.

Deterministic LLL Algorithm

A key ingredient in developing our randomized LLL algorithm from Theorem 6.1 is a deterministic distributed algorithm for LLL, which we present in Section 6.2.3.

Theorem 6.10 (Deterministic LLL Algorithm). *For any integer $\beta \geq 1$, any n-vertex LOCAL LLL problem can be solved deterministically in $\beta n^{1/\beta} \cdot 2^{O(\sqrt{\log n})}$ rounds, under the symmetric LLL criterion $p(ed)^{\beta} < 1$. If the algorithm is provided a (β, γ) network decomposition of the square graph $G_{\mathcal{B}}^2$, then the LLL algorithm runs in just $O(\beta(\gamma + 1))$ rounds.*

Our deterministic algorithm can be used as post-shattering algorithm to complete the partial assignment given by the randomized shattering algorithm from Lemma 6.9. To the best of our knowledge, this is the first non-trivial deterministic distributed LLL algorithm. In fact, we believe that a conceivable future improvement of our LLL algorithm may need to improve this deterministic LLL algorithm—ideally to complexity $O(\log n)$, matching the deterministic lower bound of $\Omega(\log n)$ by [57]—for proving the $T_{\mathsf{LLL}}(n) = O(\log \log n)$ conjecture of Chang and Pettie [60].

Our algorithm makes use of network decompositions, and, in particular, makes a black-box invocation to the algorithm stated in Lemma 3.2 for computing a $(\beta, n^{1/\beta} \log n)$ network decomposition. The running time of our deterministic LLL algorithm hence directly depends on the network decomposition it works with. In particular, if we plug in the novel $(\operatorname{poly}\log n, \operatorname{poly}\log n)$ network decomposition algorithm in $\operatorname{poly}\log n$ rounds by [216, 114], using the methods of Lemma 3.2, we can obtain a $(\beta, n^{1/\beta} \log n)$-network decomposition in $n^{1/\beta} \operatorname{poly}\log n$ rounds.

6.2.2 Randomized LLL Shattering Algorithm

We now explain the randomized component of our LLL algorithm for bounded-degree graphs, which performs a partial sampling in the LLL space, thus setting some of the variables, in a manner that guarantees the following two properties needed for Lemma 6.9: (i) the conditional probabilities of the bad events, conditioned on the already set variables, satisfy a polynomial LLL criterion, (ii) the connected components of the events on variables that remain unset are small (e.g., for bounded-degree graphs, they have size at most $O(\log n)$), with high probability.

Our partial sampling is inspired by a centralized LLL algorithm of Molloy and Reed [186] and Pach and Tardos [204]. See also [115, Algorithm 1] who wrote a section about our algorithm (even with pseudo-code).

Proof of Lemma 6.9. We first compute a (d^2+1) coloring of the square graph $G^2_{\mathcal{B}}$ on the events, which can be done even deterministically in $\widetilde{O}(d) + O(\log^* n)$ rounds [107]. Suppose $\mathcal{B}_i$ is the set of events colored with color i, for $i \in \{1, \ldots, d^2+1\}$. We process the color classes one by one.

Initially, all the variables are unset and non-frozen. For each color $i \in \{1, \ldots, d^2+1\}$, and for each node $B \in \mathcal{B}_i$ in parallel, we make node B sample values for its non-frozen and yet unset variables offline, one by one, independently and uniformly at random. Notice that since we are using a coloring of $G^2_{\mathcal{B}}$, for each color i, each event $A \in \mathcal{B}$ shares variables with at most one event $B \in \mathcal{B}_i$. Hence, during this iteration, at most one node B is sampling variables of event A. Each time, when node B is choosing a value for a variable $v \in \mathsf{vbl}(B)$, it checks whether this setting makes one of the events $A \in \mathcal{B}$ involving variable v dangerous. We call an event A *dangerous* if $\mathsf{Pr}[A \mid \mathcal{V}^*_A] \geq \sqrt{p}$, where $\mathcal{V}^*_A$ denotes the already set variables of A up to this point in the sampling process. If the recently set variable v

leads to a dangerous event A, then we undo this variable assignment to v, and *freeze* variable v as well as all the remaining variables of event A. We will not assign any value to these frozen variables in the remainder of the randomized sampling process. We have two key observations regarding this process:

Observation 6.11. *At the end of each iteration, for each event $A \in \mathcal{B}$, its conditional probability $\mathsf{Pr}[A \mid \mathcal{V}_A^*]$, conditioned on the already made assignments $\mathcal{V}_A^*$, is at most $\sqrt{p} < 1/(ed)^{2\beta}$.*

This immediately follows by the design of our sampling algorithm: Towards a contradiction, assume that $\mathsf{Pr}[A \mid \mathcal{V}_A^*] > \sqrt{p}$. Then there is a round in which an event B samples a variable $\mathsf{vbl}(A)$ such that the probability exceeds $\sqrt{p}$ (for the first time). In that case, B reverts this choice—thus decreasing A's probability below $\sqrt{p}$—and freezes all the variables in $\mathsf{vbl}(A)$. It is thus not possible for any other event to change A's and hence A's probability after that round.

Observation 6.12. *For each event $B \in \mathcal{B}$, the probability of B having at least one unset variable is at most $(d+1)\sqrt{p}$. Furthermore, this is independent of events that are further than 2 hops from B.*

The reason for this is as follows. For each $B \in \mathcal{B}$, the probability that B ever becomes dangerous is at most $\sqrt{p}$. This is because otherwise the total probability of B happening would exceed $\sqrt{p}$. Now, an event $B \in \mathcal{B}$ can have frozen variables only if at least one of its neighboring events A, or event B itself, becomes dangerous at some point during the process. Since B has at most d neighboring events, by a union bound, the latter has probability at most $(d + 1)\sqrt{p}$.

Observation 6.11 directly implies property (i) of Lemma 6.9. We use Observation 6.12 to conclude that the events with at least one unset variable comprise small connected components: In particular, we apply our Shattering Lemma (Lemma 3.3) to $G_{\mathcal{B}}^2$ with the random partial setting process generating a set $\mathcal{B}' \subseteq \mathcal{B}$ of the events that have at least one variable unset. By Observation 6.12, each event remains with probability at most

$$(d+1)\sqrt{p} \leq (d+1)e^{-2\beta}d^{-2\beta} \leq d^{-15},$$

hence we can set $c_1 \leftarrow 15$. These events depend only on events within at most 1 hop in $G_{\mathcal{B}}$, thus $c_2 \leftarrow 2$ hops in $G^2_{\mathcal{B}}$. Lemma 3.3 (iii) shows that w.h.p. property (ii) of Lemma 6.9 holds. □

6.2.3 Deterministic LLL Algorithm

Proof of Theorem 6.10. We first compute a $(\beta, n^{1/\beta}\log n)$ network decomposition of $G^2_{\mathcal{B}}$, which decomposes its vertices into β disjoint blocks $\mathcal{B}_1, \ldots, \mathcal{B}_\beta$, such that each connected component of $G^2_{\mathcal{B}}[\mathcal{B}_i]$ has diameter at most $n^{1/\beta}\log n$. This decomposition can be computed in $\beta n^{1/\beta} \cdot 2^{O(\sqrt{\log n})}$ rounds, using Lemma 3.2. The rest of the proof is described assuming this $(\beta, n^{1/\beta}\log n)$ network decomposition and works in $O(\beta n^{1/\beta}\log n)$ rounds; one can easily see that given a (β, γ) network decomposition $G^2_{\mathcal{B}}$, the algorithm would work instead in $O(\beta(\gamma+1))$ rounds.

Iteratively for $i = 1, \ldots, \beta$, we assign values to all variables of events in $\mathcal{B}_i$ that have remained unset. The values are chosen is such a way that, after i steps, the conditional probability of *any* event in $\mathcal{B}$, conditioned on all the assignments in variables of events in $\bigcup_{j=1}^{i} \mathcal{B}_j$, is at most $p(ed)^i < 1$. Once $i = \beta$, since the conditional failure probability is $p(ed)^\beta < 1$ but all the variables are already assigned, we know that none of the events occurs.

The base case $i = 0$ is trivial. In the following, we explain how to set the values for variables involved in events of $\mathcal{B}_i$ in $n^{1/\beta} \cdot \log n$ rounds. Let $\mathcal{X}_i$ be the set of variables in events of $\mathcal{B}_i$ that remain with no assigned value. We form a new LLL problem, as follows: For each bad event $A \in \mathcal{B}$, we introduce an event $B_{A,i}$ on the space of values of $\mathcal{X}_i$. This is the event that the values of $\mathcal{X}_i$ get chosen such that the conditional probability of the event A, conditioned on the variables in $\bigcup_{j=1}^{i} \mathcal{X}_j$, is larger than $p(ed)^i$. Notice that

$$\mathsf{Pr}\left[B_{A,i} \mid \bigcup_{j=1}^{i-1} \mathcal{X}_j\right] \leq \frac{p(ed)^{i-1}}{p(ed)^i} = \frac{1}{ed}.$$

Moreover, each event $B_{A,i}$ depends on at most d other events $B_{A',i}$. Hence, the family of events $B_{A,i}$ on the variable set $\mathcal{X}_i$ satisfies the conditions of the tight (symmetric) LLL. Therefore, by the Lovász Local Lemma, we know that there exists an assignment to variables of $\mathcal{X}_i$ which makes no event $B_{A,i}$ happen. That is, an assignment such that the conditional probability of each event A, conditioned on the assignments in $\bigcup_{j=1}^{i} \mathcal{X}_j$, is bounded by at most $p(ed)^i$.

Given the existence, we find such an assignment in $n^{1/\beta} \cdot \log n$ rounds, as follows: each component of $G^2_{\mathcal{B}}[\mathcal{B}_i]$ first gathers the whole topology of this component (as well as its incident events and the current assignments to any of their variables), in $n^{1/\beta} \log n$ rounds. Then, it decides about an assignment for its own variables in $\mathcal{X}_i$, by brute-forcing all possibilities. Different components can decide independently as there is no event that shares variables with two of them, since they are non-adjacent in $G^2_{\mathcal{B}}$. □

6.2.4 Wrap-Up: The Base LLL Algorithm

Combining the two parts gives us the randomized LLL algorithm of Theorem 6.8.

Proof of Theorem 6.8. We first run the randomized shattering algorithm of Lemma 6.9 for computing a partial setting of the variables, in $O(d^2 + \log^* n)$ rounds. Then, by Lemma 6.9 (i), the remaining events $\mathcal{B}'$ (those which have at least one unset variable) form a new LLL system on the unset variables, where each bad event has probability at most $\sqrt{p}$.

Moreover, by Lemma 6.9 (ii), each connected component of the square graph $G^2_{\mathcal{B}}[\mathcal{B}']$ of these remaining events $\mathcal{B}'$ admits the computation of a $(\beta, O(\log^{1/\beta} n \cdot \log^2 \log n))$ network decomposition in $\beta \log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})}$ rounds. From now on, we handle the remaining events in different connected components of $G^2_{\mathcal{B}}[\mathcal{B}']$ independently.

Since $\sqrt{p}(ed)^\beta < 1$, we can now invoke the deterministic LLL algorithm of Theorem 6.10 on top of the network decomposition of each component. Our deterministic LLL then runs in $\beta \log^{1/\beta} n \cdot \log^2 \log n$ additional rounds, and finds assignments for these remaining variables, without any of the events occurring, hence solving the overall LLL problem. The overall round complexity is $O(d^2) + \beta \log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})}$. □

6.3 Bootstrapping

In this section, we show how to use bootstrapping to speed up our base LLL algorithm, proving Theorem 6.1 for bounded-degree graphs and Lemma 6.4 for higher-degree graphs, as well as our automatic speedup result from Corollary 6.3.

6.3.1 Bounded-Degree LLL Algorithm

Proof of Theorem 6.1. In Theorem 6.8, we saw an algorithm $\mathcal{A}$ that solves any n-event LLL under the criterion $p(ed)^{32} < 1$ in $T_{n,d} = O(d^2 + \log^{1/4} n)$ rounds. We now explain how to bootstrap this algorithm to run in $2^{O(\sqrt{\log\log n})}$ rounds, on bounded-degree graphs.

Inspired by the idea of Chang and Pettie [60], we will lie to $\mathcal{A}$ and say that the LLL graph has $n^* \ll n$ vertices, for a value of n^* to be fixed later. Then, $\mathcal{A}_{n^*}$ runs in $T_{n^*,d} = O(d^2 + \log^{1/4} n^*)$ rounds. In this algorithm, the probability of any local failure (i.e., a bad event of LLL happening) is at most $1/n^*$. We can view this as a new system of bad events which satisfies a much stronger LLL criterion. In particular, we consider each of the previous bad LLL events as a bad event of the new LLL system, on the space of the random values used by $\mathcal{A}_{n^*}$, but now we connect two bad events if their distance is at most $2T_{n^*,d}+1$. Notice that if two events are not connected in this new LLL, then in algorithm $A_{n^*,d}$, they depend on disjoint sets of random variables and thus they are independent.

The degree of the new LLL system is

$$d' = d^{2T_{n^*,d}+1} = d^{O(d^2+\log^{1/4} n^*)}.$$

On the other hand, the probability of the bad events of the new system is at most $p' = 1/n^*$. Hence, the polynomial LLL criterion is satisfied with exponent

$$\beta' = \frac{\log_d n^*}{O(d^2+\log^{1/4} n^*)}.$$

We choose $n^* = \log n$, which, for $d = O((\log\log n)^{1/5})$, means $\beta' = \Omega(\sqrt{\log\log n})$. Hence, this new LLL system can be solved using the LLL algorithm of Theorem 6.8 in time

$$\begin{aligned}
&(d')^2 + \beta' \log^{1/\beta'} n \cdot 2^{O(\sqrt{\log\log n})} \\
&= d^{O(d^2+(\log\log n)^{1/4})} + \sqrt{\log\log n}\cdot(\log n)^{1/\Omega(\sqrt{\log\log n})}\cdot 2^{O(\sqrt{\log\log n})} \\
&= 2^{O(\sqrt{\log\log n})}.
\end{aligned}$$

We should note that these are rounds on the new LLL system, but each of them can be performed in

$$2T_{n^*,d}+1 = O(d^2+\log^{1/4} n^*) = O(\sqrt{\log\log n})$$

rounds on the original graph. Hence, the overall complexity is still $2^{O(\sqrt{\log\log n})}$. □

6.3.2 Higher-Degree LCL Algorithms

Similar ideas as in Theorem 6.1 lead to the proof of Lemma 6.4.

Proof of Lemma 6.4. Consider the randomized algorithm $\mathcal{A}_{n^*}$ that solves some LCL problem $\mathcal{P}$ on n^*-vertex graphs with complexity $O(\log^{1/4} n^*)$. We now bootstrap $\mathcal{A}_{n^*}$ using an approach similar to the proof of Theorem 6.1: In particular, we shall run $\mathcal{A}_{n^*}$ on our full graph of n vertices, where we set $n^* = \log n$, while $\mathcal{A}_{n^*}$ is still told that the network size is n^*. This runs in $O(\log^{1/4} \log n)$ rounds. The probability of each local bad event—i.e., a violation of one of the conditions of $\mathcal{P}$—is at most $p' = 1/n^* = 1/\log n$. On the other hand, each two of these local bad events that are further than $O(\log^{1/4} \log n)$ hops apart rely on disjoint random variables in the execution of $\mathcal{A}_{n^*}$. Hence, this new LLL system has dependency degree at most $d' = d^{O(\log^{1/4} \log n)} = 2^{O(\sqrt{\log \log n})}$. Thus, this system satisfies the polynomial LLL criterion with a value of $\beta = \Theta(\sqrt{\log \log n})$, because $p'(ed')^{\beta} < 1$. Therefore, we can solve it using the algorithm of Theorem 6.8 in $O((d')^2) + \beta(\log n)^{1/\beta} \cdot 2^{O(\sqrt{\log \log n})} = 2^{O(\sqrt{\log \log n})}$ rounds of the new LLL system. Each of these rounds can be performed in $O(\log^{1/4} n^*) = O(\log^{1/4} \log n)$ rounds of the base graph, and thus the overall round complexity of the new algorithm $\mathcal{A}'_n$ is $2^{O(\sqrt{\log \log n})}$. $\square$

6.3.3 Automatic Speedup and Derandomization

We finally provide an alternative proof for the result in [57] that $o(\log \log n)$-round algorithms can be sped up to a deterministic $O(\log^* n)$-round algorithm.

Proof of Corollary 6.3. Consider the randomized algorithm $\mathcal{A}_{n^*}$ that solves the LCL problem $\mathcal{P}$ on n^*-vertex graphs in $o(\log \log n^*)$ rounds. We bootstrap $\mathcal{A}_{n^*}$ using an approach similar to the proof of Theorem 6.1. In particular, we shall run $\mathcal{A}_{n^*}$ on our full graph of n vertices, while $\mathcal{A}_{n^*}$ is still told that the network size is n^*, for a sufficiently large constant value of n^*. This runs in $T = o(\log\log n^*)$ rounds. The probability of each local bad event—i.e., a violation of one of the conditions of $\mathcal{P}$—is at most $p' = 1/n^*$. On the other hand, each two of these local bad events that are further than $2T+1 = o(\log\log n^*)$ hops apart rely on disjoint random variables in the execution of $\mathcal{A}_{n^*}$. Hence, this new LLL system has dependency degree at most $d' = d^{2T+1}$. Thus, this system satisfies the polynomial LLL criterion with $\beta \geq (d')^2 + 1$. That is because

$$\begin{aligned} p(ed')^{d'+1} &\leq \frac{1}{n^*}\left(ed^{o(\log\log n^*)}\right)^{d^{o(\log\log n^*)}+1} \\ &< \frac{1}{n^*}\left(\sqrt{\log n^*}\right)^{\sqrt{\log n^*}} < 1, \end{aligned}$$

where the penultimate inequality uses that $d = O(1)$.

On the other hand, we can easily compute a $((d')^2+1, 0)$ network decomposition of the square graph $G^2_{\mathcal{B}}$ of this new LLL's dependency graph, simply by taking a $((d')^2+1)$ coloring of it. Notice that this coloring can be computed in $O(\log^* n)$ time, using the deterministic distributed coloring algorithm [107]. Then, we apply the deterministic LLL algorithm of Theorem 6.10 on top of this network decomposition, with $\beta = (d')^2 + 1$ and $\gamma = 0$. The algorithm of Theorem 6.10 then runs in $O((d')^2)$ rounds, and solves this LLL, hence providing a solution for the LCL problem $\mathcal{P}$. Overall, we get a deterministic algorithm with complexity $O(\log^* n)$ for solving the LCL problem $\mathcal{P}$ on bounded-degree graphs. □

6.4 Defective Coloring

In this section, we present our defective coloring algorithm using our LLL result in several steps.

Direct LLL Formulation: Chung, Pettie, and Su [67] give a formulation of f-defective $\lceil 2\Delta/f \rceil$ coloring as LLL as follows. Each vertex picks a color uniformly at random. For each vertex v, there is a bad event D_v that v has more than f neighbors assigned the same color as v. The probability of a neighbor u having the same color as v is $f/(2\Delta)$. Hence, the expected number of neighbors of v with the same color as v is at most $f/2$. By a Chernoff bound, the probability of v having more than f neighbors with the same color is at most $e^{-f/6}$. Moreover, the dependency degree between the bad events D_v is $d \leq \Delta^2$. Therefore, $p(ed)^{32} \leq e^{-f/6+32+64\log\Delta} < 1$ for $f = \Omega(\log\Delta)$.

We are unable to directly apply our LLL algorithm of Theorem 6.1 to this formulation, because (1) for $f = o(\log\Delta)$, this LLL formulation does not satisfy the polynomial criterion $p(ed)^{32} < 1$, (2) even if this criterion is satisfied, the dependency degree d may be larger than what Theorem 6.1 can handle.

Iterative LLL Formulation using Bucketing: Instead of directly finding an f-defective $O(\Delta/f)$ coloring with one LLL problem—i.e., a partition of G into $O(\Delta/f)$ buckets with maximum degree f each—we gradually approach this goal by iteratively partitioning the graph into buckets, until they have maximum degree f. In other words, we slow down the process of partitioning. We gradually decrease the degree, moving from maximum degree x to $\log^5 x$ in one iteration. We can see each of these bucketing steps—that is, the partitioning into subgraphs—as a partial coloring, which fixes some bits of the final color. Each of these slower partitioning steps can be formulated as an LLL. The function $x \mapsto \log^5 x$ is chosen large enough for the corresponding LLL to satisfy the polynomial criterion, and small enough so that decreasing the degree from Δ to f does not take too many iterations, namely $O(\log^* \Delta)$ iterations only.

Outline: We first formulate the bucketing as an LLL problem satisfying the polynomial LLL criterion and present ways for solving this LLL for different ranges of Δ in Section 6.4.1. Then, we explain how iterated application of solving these bucketing LLLs leads to a partition of the graph into $O(\Delta/f)$ many degree-f buckets, solving the defective coloring problem, in Section 6.4.2.

6.4.1 Bucketing

LLL Formulation of Bucketing

One Iteration of Bucketing: In one bucketing step, we would like to partition our graph with degree Δ into roughly Δ/Δ' buckets, each with maximum degree Δ', for a $\Delta' = \Omega(\log^5 \Delta)$. Notice that we can achieve the defective coloring of Theorem 6.5, by repeating this bucketing procedure, iteratively. See its proof in Section 6.4.2 for details of iterative bucketing. Each iteration of bucketing can be formulated as an LLL as follows.

LLL Formulation of One Iteration of Bucketing: Let $k = (1+\varepsilon)\Delta/\Delta'$ for $\varepsilon = \log^2 \Delta/\sqrt{\Delta'}$. We consider the random variables assigning each vertex a bucket number in $[k]$. Then, we introduce a bad event D_v for vertex v if more than Δ' neighbors of v are assigned the same number as v. In expectation, the number of neighbors of a vertex in the same bucket is at most $\Delta'/(1+\varepsilon)$. By a Chernoff bound, the probability of having more than Δ' neighbors in the same bucket is at most $p = e^{-\Omega(\varepsilon^2\Delta')} = e^{-\Omega(\log^4 \Delta)}$. Moreover, the dependency degree between these bad events is $d \leq \Delta^2$. Hence, this LLL satisfies the polynomial criterion.

Solving the Bucketing LLL

Bucketing for Low-Degree Graphs: If $\Delta = O(\log^{1/10} \log n)$, then $d = O(\log^{1/5} \log n)$, and thus we can directly apply the LLL algorithm of Theorem 6.1 to compute such a bucketing in $2^{O(\sqrt{\log \log n})}$ rounds.

Bucketing for Higher-Degree Graphs: For larger values of

Δ, however, we cannot apply Theorem 6.1 directly. The following lemma discusses how we handle this range by sacrificing a factor 2 in the number of buckets. In a nutshell, the idea is to just perform one sampling step of bucketing, and then to deal with vertices with too large degree separately, by setting up another bucketing LLL. While the first LLL on the whole graph could not be solved directly, the second LLL is formulated only for a small subset of vertices, which allows an efficient solution. Because of the two trials of solving an LLL, we lose a factor 2 in the total number of buckets.

Lemma 6.13. *For $\Delta = \Omega(\log^{1/10} \log n)$, there is a $2^{O(\sqrt{\log \log n})}$-round randomized LOCAL algorithm that w.h.p. computes a bucketing into $2k$ buckets with maximum degree Δ' each, for parameters $\Delta' = \Omega(\log^5 \Delta)$, $\varepsilon = \log \Delta / \sqrt{\Delta'}$, and $k = (1+\varepsilon)\Delta/\Delta'$.*

Proof. We break the $\Delta = \Omega(\log^{1/10} \log n)$ range into two sub-ranges, based on whether $\Delta = 2^{\Omega(\log^{1/4} \log n)}$ or not. We present the proof for each of these cases separately.

Case 1, $\Delta = 2^{\Omega(\log^{1/4} \log n)}$: We assign each vertex to one of the first k buckets uniformly at random. Then, for each vertex that has more than Δ' neighbors in its bucket, we remove it from its bucket and put it into B. Note that even though we might have removed a vertex from its bucket (and added it to B) in this way, it still counts as neighbor for other vertices in its bucket. By construction, each of the k buckets has degree at most Δ'. However, we have a set B of vertices not assigned to any bucket. We now show how to perform another bucketing step of B into k additional buckets, by setting up and solving another bucketing LLL.

By the above observations, a vertex is put in B with probability at most $e^{-\Omega(\log^2 \Delta)}$. Moreover, the events $1(v \in B)$ depend only on the 1-hop neighborhood. By Lemma 3.3 (iii), we can compute a $(\sqrt{\log \log n}, 2^{O(\sqrt{\log \log n})} \log^2 \log n)$ network decomposition of (each connected component of) $G[B]$ in $2^{O(\sqrt{\log\log n})}$ rounds, setting $\beta = \sqrt{\log\log n}$. We now set up a bucketing LLL (in fact, one for each of the connected components of $G[B]$), and invoke the deterministic algorithm of Theorem 6.10 on top of this network decomposition. Notice that the criterion $p(ed)^{\beta} < 1$ is satisfied for $\beta = \sqrt{\log\log n}$, because $p = \Delta^{-\Omega(\log^3 \Delta)}$ and $\Delta = 2^{\Omega(\log^{1/4}\log n)}$. Hence, the algorithm of Theorem 6.10 runs in $2^{O(\sqrt{\log\log n})}$ rounds, and computes a bucketing for all the vertices in $G[B]$ into k additional buckets.

Case 2, $\Delta \in [\Omega(\log^{1/10}\log n), 2^{O(\log^{1/4}\log n)}]$**:** We first devise a randomized algorithm $\mathcal{A}_{n^*}$ with complexity $O(\log^{1/4} n^*)$ that computes a bucketing of any n^*-vertex graph into $2k$ buckets, each with maximum degree Δ'. Then, we bootstrap this algorithm, using Lemma 6.4, to turn it into another bucketing algorithm $\mathcal{A}'_n$ that runs in $2^{O(\sqrt{\log\log n})}$ on any n-vertex graph with $\Delta = 2^{O(\log^{1/4}\log n)}$.

Algorithm $\mathcal{A}_{n^*}$ performs a simple bucketing by putting each vertex in one of the first k buckets, chosen uniformly at random. For each vertex that has more than Δ' neighbors in its bucket, we remove it from its bucket and put it into a set B of bad vertices. By Lemma 3.3 (iii), we can compute an $(8, O(\log^{1/4} n^*))$ network decomposition of each connected component of the subgraph induced by B, in $O(\log^{1/4} n^*)$ rounds. We can then invoke the deterministic LLL algorithm of Theorem 6.10 to compute a bucketing of these bad vertices of B into the second k buckets, in no more than $O(\log^{1/4} n^*)$ rounds, by setting $\beta = 8$ in Theorem 6.10. This completes the description of the bucketing algorithm $\mathcal{A}_{n^*}$ with complexity $O(\log^{1/4} n^*)$.

We now bootstrap $\mathcal{A}_{n^*}$ using Lemma 6.4 to obtain a bucketing algorithm $\mathcal{A}'_n$ that runs in $2^{O(\sqrt{\log\log n})}$ rounds, on any n-vertex graph with maximum degree at most $2^{\Theta(\log^{1/4}\log n)}$. $\square$

6.4.2 Defective Coloring using Bucketing

Now we show how to use (iterated) bucketing to arrive at a defective coloring.

Proof of Theorem 6.5. If $f \geq \Delta$, any assignment of colors to vertices is an f-defective coloring. If $f = O(1)$, a proper $O(\Delta)$ coloring is a $O(\Delta/f)$ coloring with defect $0 \leq f$. In this case, we can find such a coloring by running the algorithm of Barenboim et. al. [27] in $2^{O(\sqrt{\log\log n})}$ rounds. We thus assume in the following that $f < \Delta$ and $f = \omega(1)$.

Parameters: Let $\Delta_0 = \Delta$, and for $i \geq 1$, set $\Delta_i = \log^5 \Delta_{i-1}$, $\varepsilon_i = \log^2 \Delta_{i-1}/\sqrt{\Delta_i} = \log^2 \Delta_{i-1}/\log^{2.5} \Delta_{i-1} = \log^{-0.5} \Delta_{i-1}$, and $k_i = (1+\varepsilon_i)\Delta_{i-1}/\Delta_i$. Moreover, let t be such that $\Delta_t = \omega(f)$ and $\log^5 \Delta_t = O(f)$, and set $\Delta_{t+1} = f$, $\varepsilon_{t+1} = \log^2 \Delta_t/\sqrt{f}$, as well as $k_{t+1} = (1+\varepsilon_{t+1})\Delta_t/f$.

Algorithm: We run the basic bucketing algorithm of Lemma 6.13 with $\Delta' \leftarrow \Delta_1$, $\varepsilon \leftarrow \varepsilon_1$, $k \leftarrow k_1$ for at most 3 iterations (in each iteration, the algorithm is applied to each of the buckets from the previous iteration), until we have reached $\Delta_i = O(\log^{1/10} \log n)$. Then we switch to the direct LLL algorithm of Theorem 6.1, run for bucketing, and perform it until $i = t$. For $i = t+1$, we apply the LLL algorithm of Theorem 6.1 one last time with $\Delta' \leftarrow \Delta_{t+1} = f$.

Analysis: We first observe that $t = O(\log^* \Delta)$. The overall running time thus is $O(t) \cdot 2^{O(\sqrt{\log\log n})} = 2^{O(\sqrt{\log\log n})}$, since each of the $t+1$ bucketing iterations takes at most $2^{O(\sqrt{\log\log n})}$ rounds by Lemma 6.13 and Theorem 6.1. Notice that all the buckets on the same level (that is, in the same iteration) are treated independently, in parallel.

After these $t+1$ iterations, the resulting f-defective coloring has at most

$$2k_1 \cdot 2k_2 \cdot 2k_3 \prod_{i=4}^{t+1} k_t \leq 8 \prod_{i=1}^{t+1} (1+\varepsilon_i) \frac{\Delta_{i-1}}{\Delta_i}$$
$$\leq 8(1+O(\varepsilon_{t+1}))\frac{\Delta}{f} = 8\Big(1+O(f^{-1/10})\Big)\frac{\Delta}{f}$$

colors, using that ε_i is exponentially increasing in i as well as that $\log^2 \Delta_t = O(f^{2/5})$. □

6.5 Frugal Coloring

Direct LLL Formulation of Frugal Coloring: Molloy and Reed in their famous book [187, Theorem 19.3] formulated frugal coloring as an LLL problem in the following way: Each vertex picks a color uniformly at random. There are two types of bad events: On the one hand, we have the *properness* condition, i.e., a bad event $M_{u,v}$, for each $\{u,v\} \in E$, which happens if u and v have the same color. On the other hand, the *frugality* condition — requiring that no vertex has more than f neighbors of the same color. That is, we have one bad event $F_{u_1,\ldots,u_{f+1}}$ for each set $u_1,\ldots,u_{f+1} \in N(v)$ of vertices in the neighborhood of some vertex v, which happens if all these vertices $u_1,\ldots,u_{f+1}$ are assigned the same color. For palettes of size C, the probability of a bad event is at most $1/C$ and $1/C^f$ for type 1 and type 2, respectively. Each event depends on at most $(f+1)\Delta$ type 1 and at most $(f+1)\Delta\binom{\Delta}{f}$ type 2 events.

Iterated LLL Formulation of Frugal Coloring: While the above formulation is enough to satisfy the asymmetric tight LLL criterion for $C = O(\Delta^{1+1/f})$, it does not satisfy the (symmetric) polynomial LLL. Therefore, the algorithm of Theorem 6.1 is not directly applicable. We show how to break down the frugal coloring problem into a sequence of few partial coloring problems, coloring only some of the vertices that have remained uncolored, each of them satisfying the polynomial LLL criterion.

Outline: In Section 6.5.1, we formalize our notion of partial frugal colorings and present a method for sampling them. Then, in Section 6.5.2, we show how to use this sampling to formulate the problem of finding a partial frugal coloring guaranteeing progress (to be made precise) as a polynomial LLL and how to solve it. In Section 6.5.3, we explain how —after several iterations of setting up and solving these progress-guaranteeing LLLs, gradually extending the partial frugal coloring—we can set up and solve one final polynomial LLL for completing the partial coloring, also based on the sampling method presented in Section 6.5.1.

6.5.1 Sampling a Partial Frugal Coloring

Partial Frugal Coloring: A partial f-frugal coloring of $G = (V, E)$ is an assignment of colors to a subset $V^* \subseteq V$ such that it is proper in $G[V^*]$ and no vertex in V has more than f neighbors with the same color. In other words, it is a f-frugal coloring of $G[V^*]$ with the additional condition that no vertex in $V' := V \setminus V^*$ has more than f neighbors in V^* with the same color.

Base-Graph Degree: A partial coloring naturally splits the base graph G into two parts: a graph $G[V^*]$ induced by colored vertices and a graph $G[V']$ induced by uncolored vertices. However, the problem of extending or completing a partial frugal coloring does not only depend on $G[V']$, but also on the base graph G. That is why we introduce the notion of base-graph degree, a property of the uncolored set V' with respect to the base graph G.

Given a partial coloring, we call the number $d_{V'}(v)$ of neighboring uncolored vertices of a vertex $v \in V$ its *base-graph degree* into the uncolored set V', and we call the maximum base-graph degree Δ' of a vertex $v \in V$ into V' the base-graph degree of V'. Moreover, we use $N_{V'}(v)$ to denote v's neighbors in V'.

In the following, we show how one can sample a partial frugal coloring, thus randomly assign some of the vertices in a set V' of uncolored vertices a color. The main idea of our sampling process is to pick a color uniformly at random, and then discard it if this choice would lead to a violation (in terms of properness and frugality). In order to increase the chances of a vertex being colored, instead of just sampling one color, each vertex v samples x different colors from x different palettes at the same time, for some parameter $x \geq 1$, and then picks the first color that does not lead to a violation. If v has no such violation-free among its x choices, then v remains uncolored.

The next lemma analyzes the probability of two kinds of events: Event $\mathcal{E}_1$ that a vertex is uncolored. This event is important if we aim to color all the vertices in V'. Event $\mathcal{E}_2$ that the base-graph degree of a vertex into the set of uncolored vertices in V' is too large. This event is important if we do not aim at a full coloring of the vertices in V', but want to ensure that we make enough progress in decreasing the base-graph degree of the uncolored set.

Lemma 6.14. *Let $V' \subseteq V$ be an uncolored set with base-graph degree Δ', $f \in [\Delta]$, and $x \geq 1$. Then there is an $O(1)$-round randomized* LOCAL *algorithm that computes a partial f-frugal $(20x\Delta'\Delta^{1/f})$ coloring of some of the vertices in V' such that*

(i) the probability that a vertex in V' is uncolored is at most 10^{-x},

(ii) the probability $\mathsf{Pr}[d_{V'}(v) > 5^{-x}\Delta']$ that the base-graph degree of a vertex $v \in V$ is larger than $5^{-x}\Delta'$ is at most $e^{-\Omega(5^{-x}\Delta')}$.

Proof of Lemma 6.14. We let $C := 20\Delta'\Delta^{1/f}$. Every vertex $v \in V'$ picks x tentative colors $c_j(v)$ for $j \in [x]$ uniformly at random from x disjoint palettes, each of size C. Then vertex v gets permanently colored with the first color that does not lead to a conflict, if there is at least one such color.

For the sake of analysis, we think of these x samplings happening sequentially in steps $1 \leq j \leq x$. We introduce two type of events. An event

$$M_j(v) := \{\exists u \in N_{V'}(v) \colon c_j(u) = c_j(v)\}$$

if v has a monochromatic incident edge in step j, in other words, if there is a vertex in V' adjacent to v that picks the same color as v in step j. An event

$$F_j(v) := \big\{\exists u \in N(v), u_1, \ldots, u_f \in N_{V' \setminus \{v\}}(u) \\ : c_j(u_i) = c_j(v) \text{ for all } i \in [f]\big\}$$

if v is involved in a non-frugal neighborhood of some vertex $u \in V$ with respect to its own j^{th} tentative color choice. With $U_0 = V'$, we let M_j and F_j denote the set of vertices $v \in U_{j-1}$ for which $M_j(v)$ and $F_j(v)$, respectively, holds. We discard the tentative colors of all vertices in $U_j := M_j \cup F_j$ and make the tentative color of vertices in $K_j := U_{j-1} \setminus U_j$ permanent.

We note that even though a vertex $v \in V'$ might have a permanent color, and thus be part of K_j for some j, it still participates in the color sampling for $j' > j$, and can lead to conflicts with other vertices in $u \in U_{j'}$, forcing u to uncolor itself, while v stays colored permanently.

Validity of the Coloring: Let $K := \bigcup_{j=1}^{x} K_j$ be the set of vertices that succeed in finding a permanent color, and let $U := U_x = V' \setminus K$ denote the set of all vertices that remain uncolored. It is easy to see that the coloring of the vertices in K uses at most xC colors and is a partial f-frugal coloring.

Properties (i) and (ii): We bound the probability of a single vertex in V' being uncolored in a single step, show that every vertex has many different colors in its neighborhood in each step, and then conclude that the number of uncolored neighbors of a vertex must decrease in every step by arguing about each color separately. Finally, we combine these results about a single step to derive a bound on the number of uncolored neighbors of a vertex after all x trials.

Probability of Being Uncolored: For $j \in [x]$ and $v \in U_{j-1}$, we have

$$\begin{aligned}\mathsf{Pr}[v \in U_j] \leq \mathsf{Pr}[M_j(v)] + \mathsf{Pr}[F_j(v)] &\leq \frac{\Delta'}{C} + \Delta\binom{\Delta' - 1}{f}\frac{1}{C^f}\\ &\leq \frac{\Delta'}{C} + \frac{\Delta(\Delta')^f}{C^f} \leq 1/10.\end{aligned}$$

Moreover, $\mathsf{Pr}[v \in U] \leq (1/10)^x$, as different steps use different palettes. This proves (i).

Number of Different Colors in Neighborhood in One Step: We show that with large probability a vertex $v \in V$ has many different tentative colors in the neighborhood $N_{U_{j-1}}(v)$ of size d. For an arbitrary ordering $u_1, \ldots, u_d$ of the neighbors of v, we introduce random variables X_i which indicate whether vertex u_i has a color different from all the colors of vertices $u_1, \ldots, u_{i-1}$ in step j. Then $X := \sum_{i=1}^{d} X_i$ is the total number of different colors in the neighborhood $N_{U_{j-1}}(v)$ of v. We have

$$\mathbb{E}[X] \geq d - \frac{1}{C}\sum_{i=1}^{d}(i-1) \geq \left(1 - \frac{1}{2 \cdot 20}\right)d.$$

A Chernoff bound, applicable due to the X_is being negatively correlated, yields

$$\mathsf{Pr}\left[X < \left(1 - \frac{1}{20}\right)d\right] \leq \mathsf{Pr}\left[X \leq \left(1 - \frac{1}{40}\right)\mathbb{E}[X]\right] \leq e^{-\frac{d}{5000}}.$$

Degree in Uncolored Graph in One Step: We first observe that the events $u \in U_j$ and $w \in U_j$ for vertices with $c_j(u) \neq c_j(w)$

are negatively correlated (conditioned on their colors). Intuitively speaking, when there is a conflict with one color, it is unlikelier or as unlikely that there is a conflict with another color. For vertices of the same color, however, these events might be positively correlated.

For the moment, we suppose that $v \in V$ has t many different tentative colors in its neighborhood in U_{j-1} of size $d_{U_{j-1}}(v) = d$ and let $u_1, \ldots, u_t \in U_{j-1}$ be vertices having the respective colors. By the above observations, each of the events $u_i \in U_j$ has probability at most $1/10$ and these events are negatively correlated. It follows by a Chernoff bound that the probability that more than $(3t)/20$ of these vertices are uncolored is at most $e^{-t/120}$.

If a vertex v has more than $d/5$ uncolored vertices in the neighborhood $N_{U_{j-1}}(v)$, this means that is has less than $(1-1/5)\,d$ colored vertices in the neighborhood $N_{U_{j-1}}(v)$. That in particular implies that it has less than $(1-1/5)\,d$ colored vertices among $u_1, \ldots, u_t$, which means more than $t-(1-1/5)d$ uncolored among $u_1, \ldots, u_t$. Thus, since $t-(1-1/5)d \geq (3d)/20$ for $t \geq (1-1/20)\,d$, we have that $\mathsf{Pr}\left[d_{U_{j-1}}(v) > d/5\right]$ is equal to

$$\begin{aligned}
&\sum_{t=1}^{d} \mathsf{Pr}\left[d_{U_{j-1}}(v) > \frac{d}{5} \mid X = t\right] \mathsf{Pr}[X=t] \\
&\leq e^{-\frac{d}{5000}} + \sum_{t=\left(1-\frac{1}{20}\right)d}^{d} \mathsf{Pr}\left[d_{U_{j-1}}(v) > d/5 \mid X = t\right] \\
&\leq e^{-\frac{d}{5000}} + \sum_{t=\left(1-\frac{1}{20}\right)d}^{d} e^{-\frac{t}{120}} \leq e^{-\frac{d}{5000}} + \frac{d}{20} e^{-\frac{\left(1-\frac{1}{20}\right)d}{120}} \\
&\leq e^{-\frac{d}{10000}}.
\end{aligned}$$

Degree in Uncolored Graph After x Trials: It follows by a union bound over all x steps that

$$\mathsf{Pr}\left[d_{U_c}(v) > 5^{-x}\Delta'\right] \leq \sum_{i=1}^{x} e^{-\frac{1}{10000}5^{-i}\Delta'} = e^{-\Omega(5^{-x}\Delta')},$$

where the last step comes from the fact that $e^{-5^{-i}}$ is (doubly-) exponentially increasing in i. $\square$

6.5.2 Iterated Partial Frugal Coloring

In the following, we first show how a progress-guaranteeing partial coloring—that is, a coloring that decreases the base-graph degree of every vertex quickly enough—can be found based on the sampling process presented in Section 6.5.1. Then, we prove that by iterating this algorithm for $O(\log^* \Delta)$ repetitions, using different palettes in each iteration, the base-graph degree reduces to $O(\sqrt{\Delta})$.

In one iteration, given a set V' of uncolored vertices, we want to color a subset $V^* \subseteq V'$ such that the uncolored vertices $V'' := V' \setminus V^*$ have a base-graph degree Δ'' that is sufficiently smaller than the base-graph degree Δ' of V'. Note that the sampling of Section 6.5.1 only provides us with a partial coloring where every vertex is likely to have a decrease in the base-graph degree. Here, however, we want to enforce that for every vertex in V there is such a decrease. To this end, we set up an LLL as follows.

LLL Formulation for Progress-Guaranteeing Coloring: Performing the sampling of Lemma 6.14, we have a bad event D_v for every vertex $v \in V$ that its base-graph degree into V'' is larger than $\Delta'' = 5^{-x}\Delta'$. By Lemma 6.14 (ii), we know that the probability of D_v is at most $e^{-\Omega(5^{-x}\Delta')}$. Moreover, the dependency degree is at most $d \leq \Delta^2$. This LLL thus satisfies the polynomial criterion.

However, as d might be large, we cannot directly apply the LLL algorithm of Theorem 6.1. In the following, we present an alternative way of finding a partial coloring ensuring a drop in the base-graph degree of every vertex. In a nutshell, the idea is to just perform one sampling step of a partial frugal coloring, as described in Section 6.5.1, and then deal with vertices associated with bad events (to be made precise) separately, by setting up another progress-guaranteeing LLL. While the first LLL on the whole graph could not be solved directly, the second LLL is formulated only for a "small" subset of vertices, which allows an efficient solution. Because of the two trials of solving an LLL, we lose a factor 2 in the total number of colors.

Lemma 6.15. *Given a partial f-frugal coloring with uncolored set V' with base-graph degree Δ' and a parameter $x \geq 1$ such that $5^{-x}\Delta' = \Omega(\sqrt{\Delta})$, there is a $2^{O(\sqrt{\log\log n})}$-round randomized distributed algorithm that computes a partial f-frugal $(40x\Delta'\Delta^{1/f})$ coloring such that the uncolored set has base-graph degree at most $\Delta'' = 5^{-x}\Delta'$.*

Proof. We handle the problem in four cases depending on the range of Δ.

Case 1, $\Delta = \omega(\log^2 n)$: We perform a sampling as described in Lemma 6.14. Then, by Lemma 6.14 (ii), the probability of a vertex having more than $\Delta'' = 5^{-x}\Delta'$ uncolored neighbors is at most $p = e^{-\omega(\log n)}$. A simple union bound over all vertices shows that, with high probability, there is no such vertex whose base degree remains high.

Case 2, $\Delta \in [\omega(\log^2 \log n), O(\log^2 n)]$: We perform a sampling as described in Lemma 6.14. This gives us a partial f-frugal coloring with , assigning colors to a subset $W \subseteq V'$. Let $D \subseteq V$ denote the set of vertices in V that have degree larger than Δ'' into the uncolored set $U = V' \backslash W$, and let $B = N(D) \cap U$ be all the neighbors in U of such vertices in D. We will show how to partially color (some) vertices in B with additional $20x\Delta'\Delta^{1/f}$ colors such that the base-graph degree into the uncolored set B' in B drops to Δ''. This is a partial f-frugal $(40x\Delta'\Delta^{1/f})$ coloring such that no vertex in V has more than Δ'' neighbors in the uncolored set $(U \setminus B) \cup B'$, as desired.

Consider the square graph $G^2[B]$, i.e., the graph on vertices of B where each two B-vertices whose G-distance is at most 2 are connected. By Lemma 6.14 (ii), the probability of a vertex u being in D is $e^{-\Omega(5^{-x}\Delta')}$. Hence, the probability of v being in B is at most $\Delta e^{-\Omega(5^{-x}\Delta')} = e^{-\Omega(\sqrt{\Delta})}$, by a union bound over all neighbors $u \in N(v)$ of v, due to the assumption that $5^{-x}\Delta' = \Omega(\sqrt{\Delta})$. Moreover, only the events $u \in B$ for vertices with distance at most 3 in G, and hence distance at most 6 in G^2, are dependent. Lemma 3.3 (iii) thus shows that the connected components of $G^2[B]$ w.h.p. admit a $(\beta, O(\log^{1/\beta}\log^2\log n))$ network decomposition, which we can compute in $\beta\log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})}$ rounds.

We now set up a polynomial LLL for a progress-guaranteeing partial frugal coloring on the (uncolored) set B (with all the base graph vertices V), as discussed in Section 6.5.2. In fact, we set up one independent such LLL for each component of $G^2[B]$. Notice that the colorings of different components do not interfere with each other, as each V-vertex has neighbors in at most one of these components. This LLL satisfies the stronger condition $p(ed)^{4\beta}$ of Theorem 6.10 for $\beta = \sqrt{\log\log n}$. We can thus apply the deterministic algorithm of Theorem 6.10 on top of this network decomposition, which runs in $O(\beta\log^{1/\beta}\log^2\log n)$ rounds. Overall, this takes $\beta\log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})} + O(\beta\log^{1/\beta}\log^2\log n) = 2^{O(\sqrt{\log\log n})}$ rounds, and gives us a partial coloring of vertices in B such that the base-graph degree to the vertices that remain uncolored has dropped to Δ''.

Case 3, $\Delta \in [\omega(\log^{1/5}\log n), O(\log^2\log n)]$: We first devise an algorithm $\mathcal{A}_{n^*}$ that performs the desired partial frugal coloring in $O(\log^{1/4} n^*)$ rounds, in n^*-vertex graphs. Then we use Lemma 6.4 to speed up this partial frugal coloring algorithm to run in $2^{O(\sqrt{\log\log n})}$ rounds.

The algorithm $\mathcal{A}_{n^*}$ is similar to the process we described in case 2: it first performs a sampling according to Lemma 6.14, then defines bad vertices B for this sampling similar to before. The only difference is that, when solving the LLL of each of the components of $G^2[B]$, we may not have the desired polynomial LLL criterion satisfied for $\beta = \Omega(\sqrt{\log\log n^*})$. Still, the condition is satisfied for any desirable large constant $\beta = O(1)$. Hence, the deterministic algorithm of Theorem 6.10 solves these remaining components in no more than $O(\log^{1/4} n^*)$, with probability $1 - 1/\operatorname{poly}(n^*)$. This is the desired algorithm $\mathcal{A}_{n^*}$ for partial frugal coloring.

Now, we invoke Lemma 6.4 to speed up this partial frugal coloring algorithm $\mathcal{A}_{n^*}$ to run in $2^{O(\sqrt{\log\log n})}$ rounds, with high probability, on any n-vertex graph with maximum degree at most $\Delta = O(\log^2\log n)$. Notice that we are able to do this because the maximum degree $\Delta = O(\log^2\log n)$ is (even far) below the requirement $\Delta \leq 2^{\Theta(\log^{1/4}\log n)}$ of Lemma 6.4. Thus, we get an algorithm A_n for partial frugal coloring, that w.h.p. in $2^{O(\sqrt{\log\log n})}$ rounds colors a subset of V' such that the base-degree to the vertices that remain uncolored is at most Δ''.

Case 4, $\Delta \in O(\log^{1/5}\log n)$: Here, we can directly apply the LLL algorithm of Theorem 6.1 to the progress-guaranteeing LLL. This w.h.p. yields a drop in the degree to Δ'', in $2^{O(\sqrt{\log\log n})}$ rounds. □

The next lemma describes how through iterated application of finding partial colorings, as supplied by Lemma 6.15, the base-graph degree of the uncolored set decreases to $O(\sqrt{\Delta})$ after $O(\log^*\Delta)$ rounds and using at most $O(\Delta^{1+1/f})$ colors.

Lemma 6.16. *There is a $2^{O(\sqrt{\log\log n})}$-round randomized algorithm that computes a partial f-frugal $(80\Delta^{1+1/f})$ coloring such that the* *uncolored set V' has base-graph degree $O(\sqrt{\Delta})$.*

Proof of Lemma 6.16. We first set the parameters, then formalize the exact meaning of iterating the sampling process of Lemma 6.15, and finally analyze the number of colors as well as rounds used.

Parameters: We let $x_0 = 1$, $x_{i+1} := (5/4)^{x_i}$, $\Delta_0 = \Delta'$, and $\Delta_{i+1} = 5^{-x_i} \cdot \Delta_i$ for $0 \le i \le t$ for a $t = O(\log^* \Delta)$ such that $\Delta_{t+1} = O(\sqrt{\Delta})$ for the first time (that is, $\Delta_t = \omega(\sqrt{\Delta})$).

Iterated Sampling: Iteratively, for $0 \le i \le t$, we apply the sampling algorithm of Lemma 6.15 with $V' \mapsto V_i$, $\Delta' \mapsto \Delta_i$, and $x \mapsto x_i$ to obtain a partial f-frugal coloring with $C_i := 40 \cdot x_i \cdot \Delta_i \cdot \Delta^{\frac{1}{f}}$ many new colors, leaving a set $V_{i+1} \subseteq V_i$ with base-graph degree at most Δ_{i+1} uncolored.

Analysis: Intuitively, as we know that the number of colors needed to find a f-frugal coloring decreases with Δ_i, we can afford to use more and more disjoint palettes when Δ_i is small. For $x_i\Delta_i = \Delta$, this would mean that we use the same number of colors in each iteration i. However, this would lead to a total number of $40t\Delta^{1+1/f}$ colors over all the t iterations. Instead, we ensure that $x_{i+1}\Delta_{i+1}$ is at least a constant factor smaller than $x_i\Delta_i$, which guarantees that the total number of colors used behaves like a geometric series. Indeed, $\sum_{i=0}^{t} 40x_i\Delta_i\Delta^{1/f} = 40\Delta^{1+\frac{1}{f}} \sum_{i=0}^{t} 2^{-\sum_{j=0}^{i} x_j} \le 40\Delta^{1+\frac{1}{f}} \sum_{i=0}^{t} 2^{-i} \le 80\Delta^{1+\frac{1}{f}}$.

By Lemma 6.15, each of the $O(\log^* \Delta)$ iterations takes $2^{O(\sqrt{\log\log n})}$ rounds, hence likewise the overall complexity. □

6.5.3 Completing a Partial Frugal Coloring

In this section, we describe how, once the base-graph degree is $O(\sqrt{\Delta})$, all the remaining uncolored vertices can be colored, hence completing the partial frugal coloring. We first give a general formulation for the completion of partial frugal colorings.

LLL Formulation for Completion of Partial Frugal Coloring: Performing the sampling of Lemma 6.14, we have a bad event U_v for every vertex $v \in V$ that it is uncolored. By Lemma 6.14 (i), the probability of U_v is at most 10^{-x}. Moreover, the dependency degree d is at most Δ^2. This LLL satisfies the polynomial criterion if $x = \Omega(\log \Delta)$.

In the following lemma, we show to solve this LLL. The idea is to first perform one sampling step (of Lemma 6.14), which shatters the graph into small components of uncolored vertices, then to set up an LLL for completing the partial coloring, and finally to solve it by employing our deterministic LLL algorithm, on each of the components.

Lemma 6.17. *Given a partial f-frugal coloring and a set V' of uncolored vertices with base degree $\Delta' = O(\sqrt{\Delta})$, there is a $2^{O(\sqrt{\log \log n})}$-round randomized algorithm that completes this f-frugal coloring, by assigning colors to all vertices in V', using $40\Delta^{1+1/f}$ additional colors.*

Proof of Lemma 6.17. We handle the problem in four cases, depending on the range of the values of Δ, similar to the proof of Lemma 6.15.

Case 1, $\Delta = \omega(\log^2 n)$: We perform one sampling step as described in Lemma 6.14 with $x = \Delta/\Delta' = \Omega(\sqrt{\Delta})$. Then, by Lemma 6.14 (ii), the probability of a vertex remaining uncolored is at most $p = e^{-\omega(\log n)}$. A simple union bound over all vertices shows that, with high probability, no vertex remains uncolored.

Case 2, $\Delta \in [\omega(\log^2 \log n), O(\log^2 n)]$: We perform one sampling step as described in Lemma 6.14 with $x = \Delta/\Delta' = \Omega(\sqrt{\Delta})$. This gives a partial f-frugal coloring with $20x\Delta'\Delta^{1/f} = 20\Delta^{1+1/f}$ colors. Let $V'' \subseteq V'$ be the set of vertices that remain uncolored. The probability of a vertex being in V'' is at most $10^{-x} = e^{-\Omega(\sqrt{\Delta})}$, by Lemma 6.14 (i). Moreover, uncoloring for vertices with distance at least 3 in G, and hence at least 5 in G^2, are independent. Thus, Lemma 3.3 (iii) implies that the connected components of $G^2[V'']$ w.h.p. admit a $(\beta, O(\log^{1/\beta} n \log^2 \log n))$ network decomposition, which can be computed in $\beta \log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})}$ rounds.

We set up LLLs with $x = \Delta/\Delta'$ for completing a partial frugal coloring, one for each connected component of $G^2[V'']$. Notice that the colorings of different components do not interfere with each other, as each V-vertex has neighbors in at most one of these components. Setting $\beta = \sqrt{\log\log n}$, we have $p(ed)^\beta = (10)^{-\sqrt{\Delta}}(O(\Delta))^{2\sqrt{\log\log n}} < 1$. Thus, the even stronger condition of Theorem 6.10 for $\beta = \sqrt{\log\log n}$ is satisfied, which lets us find a solution for the LLL, and hence a completion of the f-frugal coloring, in additional $\beta \cdot \log^{1/\beta} n \cdot \log^2 \log n$ rounds, on each of the connected components of $G^2[V'']$ in parallel. Overall, this takes $\beta \log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})} + \beta \log^{1/\beta} n \log^2 \log n = 2^{O(\sqrt{\log\log n})}$ rounds.

Case 3, $\Delta \in [\omega(\log^{1/5} \log n), O(\log^2 \log n)]$: We first devise an algorithm $\mathcal{A}_{n^*}$ that completes the given partial frugal coloring in $\Theta(\log^{1/4} n^*)$ rounds, in n^*-vertex graphs. Then, we use Lemma 6.4 to speed up this coloring completion algorithm to run in $2^{O(\sqrt{\log\log n})}$ rounds.

The algorithm $\mathcal{A}_{n^*}$ is similar to the process we described in case 2: it first performs a sampling according to Lemma 6.14 with $x = \Delta/\Delta' = \Omega(\sqrt{\Delta})$, then defines bad vertices B for vertices that remain uncolored, and handles each of connected components of $G^2[B]$ with a new LLL. The only difference is that, when solving the LLL of each of the components of $G^2[B]$, we may not have the desired polynomial LLL criterion satisfied for $\beta = \Omega(\sqrt{\log\log n^*})$. Still, the condition is satisfied for any desirably large constant β. Hence, the deterministic algorithm of Theorem 6.10 can solve these remaining components in at most $\Theta(\log^{1/4} n^*)$ rounds, with local correctness probability at least $1 - 1/\operatorname{poly} n^*$. This is the desired algorithm $\mathcal{A}_{n^*}$ for the completion of the coloring.

Now, we invoke Lemma 6.4 to speed up this frugal coloring completion algorithm $\mathcal{A}_{n^*}$ to run in $2^{O(\sqrt{\log \log n})}$ rounds, with high probability, on any n-vertex graph with maximum degree at most $\Delta = O(\log^2 \log n)$. Notice that we are able to do this because the maximum degree $\Delta = O(\log^2 \log n)$ is (even far) below the requirement $\Delta = 2^{\Theta(\log^{1/4} \log n)}$ of Lemma 6.4. Thus, we get an algorithm A_n for completing the partial frugal coloring that, in $2^{O(\sqrt{\log \log n})}$ rounds, colors all the remaining uncolored vertices, with high probability.

Case 4, $\Delta \in O(\log^{1/5} \log n)$: Here, we can directly apply the LLL algorithm of Theorem 6.1 to the LLL for completing a partial frugal coloring. This gives us an algorithm that w.h.p. completes the given partial frugal coloring in $2^{O(\sqrt{\log \log n})}$ rounds. □

A wrap-up of these results about iterated partial colorings and completing a partial coloring immediately leads to a proof of Theorem 6.6.

Proof of Theorem 6.6. We first apply the iterated coloring algorithm of Lemma 6.16 with $80\Delta^{1+1/f}$ colors, in $2^{O(\sqrt{\log \log n})}$ rounds. Then, we run the algorithms of Lemma 6.17 to complete this partial coloring with $40\Delta^{1+1/f}$ additional colors, in $2^{O(\sqrt{\log \log n})}$ rounds. This yields a f-frugal $(120\Delta^{1+1/f})$ coloring, in $2^{O(\sqrt{\log \log n})}$ rounds. □

6.6 List Vertex Coloring

We remark that essentially without loss of generality, we can focus on the regime where $L = O(\log^2 n)$. This is because if $L = \Omega(\log^2 n)$, we can make each vertex choose $\log^2 n$ colors in its list at random to retain, forming a new color list L'_v with $|L'_v| = \log^2 n$. Then, with high probability, we have the following property: for each vertex v

and each color $q \in L'_v$, the number of neighbors u of v that have $q \in L'_u$ is at most $(1 + o(1)) \log^2 n/C$.

Direct LLL Formulation of List Vertex Coloring: Suppose each vertex picks a color uniformly at random. Define a bad event $E_{u,v,q}$ for each edge $\{u, v\}$ and color q if its endpoints choose the same color q. The probability of each such event is at most $p = (\frac{1}{L})^2$. The dependency degree between these events is at most $d = 2L \cdot L/C$. This is because the event $E_{u,v,q}$ has dependency with the events of at most L colors from each endpoint u or v, and at most L/C edges incident on that endpoint for each of these L colors. Hence, if $C > 2e$, the LLL criterion $epd \leq 1$ is satisfied.

Shortcomings of the Direct LLL Formulation: As before, we face two issues in applying the LLL algorithm Theorem 6.1: (1) the above formulation does not satisfy the polynomial LLL criterion, (2) the dependency degree d may be above what Theorem 6.1 can handle.

Iterated LLL Formulation of List Vertex Coloring via Pruning: In the following, we explain how through a sequence of gradual pruning of color lists, we can get to our target coloring. A pruning step can be formulated as an LLL which satisfies the polynomial LLL criterion. We also explain how to perform each of these pruning steps, albeit the fact that the related (strengthened) LLL does not have bounded degrees.

6.6.1 Pruning

A Factor-2 Pruning of Color Lists: We would like to narrow down the color lists and their conflict sizes by roughly a factor 2. More concretely, we would like that each vertex v keeps a subset $L'_v \subseteq L_v$ such that $|L'_v| \geq \frac{|L_v|}{2}(1 - \frac{1}{\log^2 L})$, and moreover, for each color $q \in L'_v$, the number of neighbors u of v that have $q \in L'_u$ is at most $(1 + \frac{1}{\log^2 L})\frac{L}{2C}$.

By repeating this factor-2 pruning for roughly $\log L/C = O(\log\log n)$ iterations, we get to a setting where each vertex has a color list of size at least $C/2$, none of which are kept by any neighbor. Then, each vertex can pick any of these colors as its final color.

LLL for Factor-2 Pruning of Color Lists: Let each vertex v keep each of its colors in L_v with probability $1/2$, forming its new list L'_v. We have two types of bad events, first that $|L'_v| \leq \frac{|L_v|}{2}(1-\frac{1}{\log^2 L})$, and second that for a color $q \in L'_v$, the number of neighbors u of v that have $q \in L'_u$ is more than $(1+\frac{1}{\log^2 L})\frac{L}{2C}$. By Hoeffding bound, the probability of each of these bad events is at most $p = e^{-\Theta(\sqrt{L}/\log^2 L)}$. Each event depends on at most $d = O(L^2/C)$ many others. Thus, this LLL satisfies the polynomial LLL criterion.

For $L = O((\log\log n)^{1/10})$, we can directly apply the LLL algorithm of Theorem 6.1 to solve the above factor-2 pruning in $2^{O(\sqrt{\log\log n})}$ rounds. Next, we explain how we solve the other cases of this pruning LLL in $2^{O(\sqrt{\log\log n})}$ rounds.

Solving the Factor-2 Pruning LLL

For $L = O((\log\log n)^{1/10})$, we can directly apply the LLL algorithm of Theorem 6.1 to solve the above factor-2 pruning in $2^{O(\sqrt{\log\log n})}$ rounds. In the following, we discuss how we handle the remaining case $L \in [\Omega(\log^{1/10}\log n), O(\log^2 n)]$. We break this range into two cases, depending on whether $L \geq \Omega(\log^4 \log n)$ or not. A key part in both will be a somewhat gradual sampling of which colors to retain in the list. To perform that sampling with an appropriate speed (to be made precise), we will use defective colorings, as we discuss next.

Color-Choice Graph: Consider a graph H where we include one vertex (v, q) for each color $q \in L_v$ of each vertex v. Two vertices (v, q) and (u, q') are connected if and only if either (1) $v = u$, or (2) v and u are adjacent and $q = q'$.

Defective Coloring of the Color-Choice Graph: Notice that the color-choice graph H has maximum degree at most $L + L/C \leq 2L$. We compute a defective coloring χ of H with defect $f = L/(2\log^2 L)$ and $O((\frac{2L}{f})^2) = O(\log^4 L)$ colors, in $O(\log^* n)$ rounds, using the deterministic algorithm of Kuhn [159]. We use this defective coloring mainly to schedule which colors (v, q) are sampled to be kept in the list.

Sampling the Colors in $O(\log^4 L)$ Phases: We have $K_0 \log^4 L$ phases, for some constant K_0, one per color class of the schedule-color χ. During each phase i, we sample each of the colors $(v, q) \in H$ that has χ-color i, with probability $1/2$, for inclusion in L'_v. At the end of the phase, we check two properties, and potentially freeze some of the unset color-choices in H, meaning that we will not sample these, and we defer the decision on them to some later process. This freezing is done as follows: If for a vertex v, we had $z_v \geq L/(16K_0 \log^6 L)$ many of its colors (v, q) that were sampled in this phase, but less than $z_v/2 - L/(16K_0 \log^6 L)$ of them turned out to be included in L'_v, then we freeze vertex v and all of its unsampled colors (v, q'). Moreover, if for a vertex v and a color $q \in L_v$, in this phase we sampled at least $z_{v,q} \geq L/(16K_0 C \log^6 L)$ of colors (u, q) in neighboring vertices of v, but more than $z_{v,q}/2 + L/(16K_0 C \log^6 L)$ of them turned out to be included in their respective lists L'_u, then we freeze all unsampled colors (u, q'), for any q', in neighbors u of v. At the end of all the phases, if a vertex v has less than $L/(2\log^2 L)$ frozen colors (v, q), we discard all of these colors and none of them will be included in L'_v.

Lemma 6.18. *For any (integer) $\beta \geq 1$, each connected component of the graph H^2 induced by frozen colors w.h.p. allows us to compute a $(\beta, O(\log^{1/\beta} n \cdot \log^2 \log n))$ network decomposition in $\beta \log^{1/\beta} n \cdot 2^{O(\sqrt{\log\log n})}$ rounds.*

Proof. Follows from Lemma 3.3 (iii) and the observation that the probability of each color getting frozen is at most $\exp(-\widetilde{\Omega}(\sqrt{L}))$, and the freezing of colors $(v,q) \in H$ that are more than 5 hops apart in H depend on disjoint random bits. □

6.6.2 Completing the Pruning

A New LLL for Completion of the Pruning: Consider the set of frozen colors, and the following new LLL for determining the inclusion of each of these frozen colors in their respective pruned lists, each included with probability 1/2. We have two bad events: (1) $\mathcal{E}_v$ if for a vertex v which has $f_v \geq L/(2\log^2 L)$ frozen colors (v,q), less than $f_v/2 - L/(2\log^2 L)$ of these colors get chosen for inclusion in L'_v, (2) $\mathcal{E}_{v,q}$ if for a vertex v and a color $q \in L_v$ which has $f_{v,q}$ frozen colors (u,q) in neighboring vertices u of v, more than $f_{v,q}/2 + L/(2C\log^2 L)$ of these frozen colors get chosen for inclusion in their respective pruned lists L'_u.

Observation 6.19. *If we find a fixing for the frozen colors without allowing any of the bad events in the completion LLL to happen, then the overall lists L'_v satisfy the requirements of factor-2 pruning. Moreover, in this completion LLL, each bad event has probability at most $p = \exp(-\widetilde{\Omega}(\sqrt{L}))$ and they have dependency $d = O(L^2)$.*

Lemma 6.20. *If $L \geq (\log\log n)^4$, then we can solve the completion LLL on each of the connected components of H^2 on frozen colors in $2^{O(\sqrt{\log\log n})}$ rounds.*

Proof. For $L \geq (\log\log n)^4$, the new LLL that we set for completing the pruning satisfies $p(ed)^\beta < 1$ for $\beta = \Omega(\sqrt{\log\log n})$, as it had per-event probability $p = \exp(-\widetilde{\Omega}(\sqrt{L}))$ and dependency degree $d = O(L^2)$. Hence, we can apply the deterministic algorithm of Theorem 6.10, on top of the network decomposition supplied by Lemma 6.18 for each of the connected components of H^2 on the frozen colors, both with parameter $\beta = \Omega(\sqrt{\log\log n})$. Thus, we get an algorithm for completing the sampling in $2^{O(\sqrt{\log\log n})}$ rounds. □

Lemma 6.21. *If $L = O(\log^4 \log n)$, we can solve the factor-2 list pruning* LLL *in $2^{O(\sqrt{\log\log n})}$ rounds.*

Proof. We first devise an algorithm $\mathcal{A}_{n^*}$ which solves the factor-2 list pruning problem in $O(\log^{1/4} n^*)$ rounds on any n^*-vertex graph, with probability $1 - 1/n^*$. Then, we use Lemma 6.4 to speed up this algorithm to solve n-vertex list prunings in $2^{O(\sqrt{\log\log n})}$.

Base Algorithm $\mathcal{A}_{n^*}$: As mentioned above, when we target correctness probability $1 - 1/n^*$, we can without loss of generality assume that $L = O(\log^2 n^*)$. Then, we first perform the $O(\log^4 L)$ rounds of partial sampling of the pruning, as explained above (by going through a defective coloring, and then sampling each of its colors, one by one). We then are left with a number of connected components of the color-choice graph H^2 on frozen colors, and new completion LLL for each of them, as described above. Each of these new LLLs satisfies the polynomial LLL criterion $p(ed)^\beta < 1$ for $\beta = \omega(1)$, because it has per-event probability $p = \exp(-\widetilde{\Omega}(\sqrt{L}))$ and dependency degree $d = O(L^2)$. Hence, we are able to deterministically solve these completion LLLs in $\beta(\log n^*)^{1/\beta} \cdot 2^{O(\sqrt{\log\log n^*})}$ rounds, using Theorem 6.10. Therefore, the overall complexity is at most $O(\log^4 L) + O(\log^{1/4} n^*) = O(\log^{1/4} n^*)$.

Speedup: Now, we can apply Lemma 6.4 to speed up this algorithm $\mathcal{A}_{n^*}$. In particular, the procedure of the proof of Lemma 6.4 will set $n^* = \log n$ and then, it will run A_{n^*} on the n-vertex graph, hence forming a new LLL that satisfies a much better exponent of the polynomial LLL criterion, concretely $\beta = \Omega(\log\log n)$. See Lemma 6.4 for details. As a result of solving that LLL, we get an algorithm $\mathcal{A}'_n$ that performs the factor-2 list pruning in $2^{O(\sqrt{\log\log n})}$ rounds, on any n-vertex graph. We note that we have $\leq (\log\log n)^4$ and thus the dependency degree in the pruning LLL is $d = O(L^2) \ll 2^{O(\log^{1/4}\log n)}$, which satisfies the requirement of Lemma 6.4. □

CHAPTER 7

Tight Analysis of Local Greedy Algorithms

7.1 Introduction

We study the round complexity of the well-studied local greedy MIS algorithm, based on the publications 'Tight Analysis of Randomized Greedy MIS' [105, 106].

7.1.1 Our Result and Related Work

Randomized Local Greedy MIS

Our main result is a tight analysis of the local greedy MIS algorithm, settling its round complexity. This algorithm works as follows: for a random assignment of values to nodes, in every iteration every remaining node with the smallest value among its remaining neighbors joins the MIS and informs all its neighbors about this decision. These nodes and all their neighbors delete themselves from the graph for the subsequent rounds.

Theorem 7.1. *The local greedy MIS algorithm w.h.p. terminates in $O(\log n)$ rounds.*

This improves on the $O(\log^2 n)$-analysis by Blelloch, Fineman, and Shun [39] almost ten years ago. Only for Erdős-Rényi random graphs, an (in expectation) upper bound of $O(\log n)$ was known, due to a result by Calkin and Frieze [50], resolving the conjecture of Coppersmith, Raghavan, and Tompa [69] who themselves arrived at $O\left(\log^2 n / \log\log n\right)$.

A result by Calkin, Frieze, and Kučera [51] proves that Theorem 7.1 is aymptotically best possible. We also provide an alternative short proof of the lower bound in Section 7.3, resulting in a tight analysis of the round complexity.

Theorem 7.2. *There exists a graph on which the local greedy MIS algorithm w.h.p. takes $\Omega(\log n)$ rounds to terminate.*

Local Symmetry Breaking

Due to the well-known reductions of maximal matching and $(\Delta+1)$ vertex coloring to MIS due to [181, 171], our analysis in Theorem 7.1 directly applies to the randomized local greedy algorithms for maximal matching and vertex coloring.

Local Greedy MM: The randomized local greedy maximal matching algorithm works as follows: A random order of the edges is chosen. Then, in each round, all locally minimal edges are removed from the graph along with all their incident edges.

Corollary 7.3. *The randomized local greedy maximal matching algorithm w.h.p. terminates in* $O(\log n)$ *rounds.*

Proof. For a graph $G = (V, E)$, the line graph $L = (E, F)$ is defined to be a graph with vertex set E and an edge $\{e, e'\} \in F$ iff $e \cap e' \neq \varnothing$. Running the randomized local greedy MIS algorithm on the line graph L corresponds to running the randomized greedy maximal matching algorithm on G. □

Local Greedy $(\Delta + 1)$ **Vertex Coloring:** The randomized local greedy $(\Delta+1)$ vertex coloring algorithm works as follows: A random order of the vertex-color pairs $V \times [\Delta + 1]$ is chosen. Then, in each round, all locally minimal pairs (v, c) are removed along with all (v', c') such that either $v' = v$ or $\{v, v'\} \in E$ and $c' = c$. Vertex v is assigned color c.

Corollary 7.4. *The randomized local greedy* $(\Delta+1)$ *vertex coloring algorithm, as defined above,*[1] *w.h.p. terminates in* $O(\log n)$ *rounds.*

Proof. Luby [180, 171] presented the following reduction from $(\Delta + 1)$ vertex coloring in a graph G to MIS in a graph H: to construct H, take $\Delta+1$ copies of G and add a clique among all $\Delta+1$ copies of the same vertex, for all vertices in G. It is easy to observe that a MIS in H corresponds to a proper $(\Delta+1)$ vertex coloring of G, when we assign vertex v the color i iff the i^{th} copy of v is in the MIS. Indeed, due to maximality, every vertex in G is assigned at least one color, and because of the added cliques and the independence of the MIS, at most one. Moreover, having a copy of G for every color guarantees that all the edges must be proper (due to independence). □

Local Greedy $(2\Delta - 1)$ **Edge Coloring:** The randomized local greedy $(2\Delta-1)$ edge coloring algorithm works as follows: A random

[1]This is *not* the greedy coloring algorithm where the largest available color is picked greedily.

order of the edge-color pairs $E \times [2\Delta - 1]$ is chosen. Then, in each round, all locally minimal pairs (e, c) are removed along with all (e', c') such that either $e' = e$ or $e \cap e' \neq \varnothing$ and $c' = c$. Edge e is assigned color c.

Corollary 7.5. *The randomized local greedy* $(2\Delta - 1)$ *edge coloring algorithm, as defined above, w.h.p. terminates in* $O(\log n)$ *rounds.*

Proof. This directly follows from the application of the algorithm from Corollary 7.4 on the line graph. □

Correlation Clustering

Correlation clustering has the goal to partition vertices into clusters so that the number of miss-classified edges—that is, edges with its two endpoints in two different clusters or non-edges with endpoints in the same cluster—is minimized. More formally, we are given a complete graph on n vertices where each edge is either labeled $+$ or $-$, indicating that the corresponding vertices should be in the same or in different clusters, respectively. The goal is to group the vertices into (an arbitrary number of) clusters so that the number of $-$ edges within clusters and $+$ edges crossing clusters is minimized [22]. Ailon, Charikar, and Newman [3] showed that the randomized local greedy MIS algorithm, called *CC-Pivot* in their paper, provides a 3-approximation for correlation clustering when each non-MIS vertex is clustered with its inhibitor, that is, its lowest-rank neighbor in the MIS. Moreover, [65] argues how an iteration of this (or a similar) algorithm can be implemented in $O(1)$ rounds of MapReduce or in $O(1)$ passes of the streaming model.

Corollary 7.6. *A 3-approximation for correlation clustering w.h.p. can be computed in* $O(\log n)$ *rounds in the PRAM, LOCAL, as well as MapReduce model, and in* $O(\log n)$ *passes in the streaming model.*

7.1.2 Overview and Outline of Our Analysis

As opposed to Luby's algorithm where new random numbers are chosen in every step, in the local greedy MIS algorithm the order is kept fixed between iterations. This has the effect that the order of the remaining vertices at each iteration are not uniformly distributed, which complicates the analysis significantly. To overcome the problem of dependencies among different iterations, Blelloch, Fineman, and Shun [39]—inspired by an approach of [69] and [50]—divide the algorithm into several phases. In each phase they only expose a *prefix* of the remaining order and run the greedy algorithm on these vertices only (whilst still deleting a vertex in the suffix if it is adjacent to a vertex added to the MIS). This way, in each phase the order among the unprocessed (but possibly already deleted) vertices in the suffix remains random, leading to a sequence of independent subproblems.

Blelloch, Fineman, and Shun [39] exploit this independence to argue that after having processed a prefix of length $\Omega(t \cdot \log n)$, the maximum remaining degree (among the not yet deleted vertices) in the suffix w.h.p. is $d = \frac{n}{t}$. This is because in every step[2] in which a vertex v has more than d neighbors, the probability that one of these is chosen to be exposed next (which causes the deletion of v) is at least $\frac{d}{n}$. In the end of the phase, the probability of v not being deleted is at most $\left(1 - \frac{d}{n}\right)^{\Omega(t \log n)} = n^{-\Omega(1)}$. A union bound over all vertices concludes the argument. By doubling the parameter t after each phase they ensure that after $O(\log n)$ phases the whole graph has been processed. Inside each phase, they use the maximum degree to bound the round complexity by the length of a longest monotonically increasing path[3], which is $O(\log n)$.

[2]For the sake of analysis, we think of the prefix being processed sequentially. This does not change the outcome.

[3]A monotonically increasing path with respect to an order is a path along which the order ranks are increasing.

The main shortcoming of Blelloch et al.'s approach is that it relies heavily on the property that in each phase the remaining degree of *all* vertices *with high probability* falls below a certain value. This imposed union bound unavoidably stretches each prefix by a factor of $\log n$. We will circumvent this problem using the following core ideas.

(1) Instead of bounding the degree of *all* vertices in the graph, we will consider a fixed set of $O(\log n)$ *positions*, that is, indices in $\{1, \ldots, n\}$, and only analyze the degree of vertices assigned to these positions in the random order.

(2) Instead of using one prefix for all these vertices *simultaneously*, we will essentially have *one distinct "prefix"* for each position, that is, one distinct set of vertices which we will use to argue about the degree drop of the vertex assigned to this position. This will preserve independence among positions, and hence spare us the need of a union bound.

(3) Instead of bounding the probability of a long *monotonically increasing path* of vertices based on the *with high probability upper bound* on the degree, we will restrict our attention to the much stronger concept of so-called *dependency paths*[4]. Roughly speaking, a dependency path is a monotonically increasing path which alternates between vertices in the MIS and vertices not in the MIS, and the predecessor of a vertex v not in the MIS is the first vertex that knocked v out. These additional properties of dependency paths allow us to execute a more nuanced analysis of their occurrence probability. In particular, we will not need to argue that the degrees of our vertices drop according to some process with high probability. It suffices to show that for any degree of this vertex its chances of being part of a dependency path are low enough.

[4]Note that this is not the same as *dependence path* in [39].

Proof Outline: Summarized, our method—comprising all the aforementioned ideas—can be outlined as follows. We will show that the round complexity is bounded by the largest dependency path length. It is then enough to show that w.h.p. there cannot be a dependency path of length $L = \Omega(\log n)$. In a first step, we will analyze the probability that a fixed set P of positions forms a dependency path. To this end, we assign each position a position set (which will play the role of a "prefix" for this position's vertex and thus serve the purpose of controlling its degree) of a certain size and at a certain place, both carefully chosen depending on the position. Then we will argue for each position[5] that its probability of being part of and continuing the dependency path is not too high, based on the randomness of its associated position set. Being careful about only exposing positions that have not already been exposed for other positions, we will be able to combine these probabilities to obtain a bound on the probability that P forms a dependency path. This is schematically illustrated in Figure 7.1. Finally, we union bound over all choices for P.

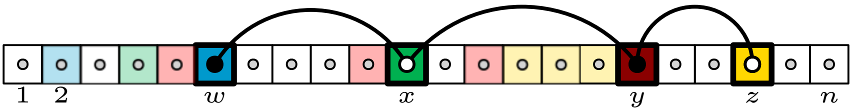


Figure 7.1: For positions $w, x, y, z \in P$, we want to bound the probability that the vertices assigned to these positions form a dependency chain alternating between vertices in the MIS (black) and vertices not in the MIS (white). To do so, we put aside a disjoint set of positions (i.e., a prefix) for every position in P (depicted with the lighter version of their position's color) which we use to bound the degree of the corresponding vertex assigned to a position in P.

[5] In fact, to get a strong enough bound, we will have to argue not only about one position but about two positions simultaneously.

7.1.3 Notation and Preliminaries

We use $[n] := \{1, \ldots, n\}$ and $[x, y] := \{x, \ldots, y\}$. For two sets $X, Y \subseteq \mathbb{N}$, we write $X < Y$ if $\max X < \min Y$. We use $X[i]$ for the i^{th} element in the set X (we think of all sets as ordered) and $X[I]$ the (ordered) set of the elements in X at positions $I \subseteq [|X|]$. For an order $\pi\colon [n] \to V$, we say that vertex $v \in V$ has position i if $\pi(i) = v$, use $\pi(I) := \bigcup_{i \in I} \pi(i)$, and write $\pi(I) = \pi'(I)$ when $\pi(i) = \pi'(i)$ for all $i \in I$. We say that we *expose* a position i when we fix the vertex $\pi(i)$ in a random order π. Moreover, we say a vertex v is exposed if there is a position i such that i is exposed and $\pi(i) = v$. If a set I of positions is already exposed, this means that the considered probabilities are all conditioned on $\pi(I)$. We use subscript I to indicate that the probability is over the randomness in the positions I. For a graph $G = (V, E)$, we use $E(X, Y)$ to denote the set of edges in E between X and Y, for $X, Y \subseteq V$, and write $e(X, Y) = |E(X, Y)|$. Moreover, we let $N(v) := \{u \in V \colon \{u, v\} \in E\}$ denote the neighborhood of vertex $v \in V$.

Framework

We introduce the concept of *dependency paths*, and show that this notion is closely related to the round complexity of the local greedy MIS algorithm.

Dependency Path: For a fixed permutation $\pi\colon [n] \to V$, let $V^* \subseteq V$ denote the MIS generated by the (sequential) greedy algorithm that processes the vertices in the order $(\pi(1), \ldots, \pi(n))$. For every vertex v not in V^*, we use $\mathsf{inhib}(v)$ to denote the neighbor of v in V^* of minimum position, that is, setting $\mathsf{inhib}(v) := \arg\min\{\pi^{-1}(u) \colon u \in N(v) \cap V^*\}$, and call it v's *inhibitor*.

Definition 7.7. *A sequence* $1 \le p_1 < \cdots < p_{2l+1} \le n$ *of positions forms a* dependency path *of length* $2l + 1$ *for* $l \ge 0$ *if* *(i)* $(\pi(p_1), \ldots, \pi(p_{2l+1}))$ *is a path in* G*,*

(ii) $\{\pi(p_k) \mid k \text{ is odd}\} \subseteq V^*$*,*

(iii) $\{\pi(p_k) \mid k \text{ is even}\} \subseteq V \setminus V^*$*,*

(iv) $\pi(p_{k-1}) = \mathsf{inhib}(\pi(p_k))$ *for even* k*.*

We write $p_1 \sim \cdots \sim p_{2l+1}$*.*

Connection to Local Algorithm: In the following, we establish a connection between the round complexity of the local algorithm and the *dependency length*, defined as the length of the longest dependency path in a graph.

Lemma 7.8. *If the dependency length is* $2l + 1$*, the local algorithm takes at most* $l + 1$ *rounds.*

Proof. Consider a slowed-down version of the local algorithm in which the deletion of a vertex $v \notin V^*$ is delayed until the round in which $\mathsf{inhib}(v)$ enters the independent set. That is, even if a neighbor u of v enters the independent set, v is not deleted from the graph unless $u = \mathsf{inhib}(v)$. This algorithm takes at least as many rounds as the original local algorithm, as the time in which a vertex is processed can only be delayed. Furthermore the slowed-down version produces the same independent set as in the original algorithm.

We show by induction that every vertex entering the independent set in round i in this modified local algorithm must be the last vertex of a dependency path of length $2(i-1)+1$. The base case $i = 1$ is immediate. If a vertex w enters V^* in round $i+1$, then there is a neighbor v of w with $\pi(v) < \pi(w)$ that was deleted from the graph (but not added to the MIS) in round i, as otherwise w could have been added to V^* in an earlier round. This means that $\mathsf{inhib}(v)$ was added to V^* in round i. By the induction hypothesis, $\mathsf{inhib}(v)$ is the last vertex of a dependency path of length $2(i-1)+1$, and it is easy to check that v and w extend this dependency path. □

7.2 Upper Bound

7.2.1 Proof Outline

Bound on Dependency Length

The number of rounds taken by the randomized local greedy MIS algorithm in the beginning and in the end—that is, for vertices with positions in $[1, \Theta(\log n)]$ and $[\beta n, n]$, for a constant $\beta \in (0,1)$ which is given by Lemma 7.10 below—can be handled easily, as we will discuss in the proof of Theorem 7.1 in Section 7.2.3. We thus focus on the technically more interesting range of positions in $[\Theta(\log n), \beta n]$ here. By Lemma 7.8, we know that the dependency length constitutes an upper bound on the round complexity of the randomized local greedy MIS algorithm. It is thus enough to show the following.

Theorem 7.9. *W.h.p. there is no dependency path of length* $L = \Omega(\log n)$ *in the interval* $[\beta n]$.

Probability of Continuing a Dependency Path

One main part of this theorem's proof, which is deferred to Section 7.2.3, is to argue that the probability that a fixed set $\{p_1, \ldots, p_L\}$ of positions form a dependency path is low. This, in turn, is proved by bounding the probability that—when already having exposed p_{k-1}—the positions p_k and p_{k+1} continue a dependency path, thus $p_{k-1} \sim p_k \sim p_{k+1}$, for all segments (p_{k-1}, p_k, p_{k+1}) for even $k \in [L]$. By being careful about dependencies among segments, thus in particular about which randomness to expose when, these probabilities for the segments then can be combined to a bound for the probability of $p_1 \sim \cdots \sim p_L$. We will achieve this by assigning disjoint position sets P_i to positions p_i and exposing, roughly speaking, only P_{k-1}, P_k, and $\{p_k, p_{k+1}\}$ when analyzing the probability of the segment (p_{k-1}, p_k, p_{k+1}). More formally, this reads as follows.

Lemma 7.10. *There exist absolute constants $\beta, \varepsilon \in (0, 0.01)$ such that the following holds. Fix three positions $p_1, p_2, p_3 \in [\beta n]$ and two disjoint sets $P_1, P_2 \subseteq [\beta n]$ of equal size $l := |P_1| = |P_2| \geq 10000$ which satisfy $P_1 < P_2 < p_1 < p_2 < p_3$, and let $t_2 := \max P_2$. Consider $S \subseteq [\beta n] \setminus (P_1 \cup P_2 \cup \{p_2, p_3\})$ with $p_1 \in S$. Suppose that the positions in S have already been exposed. Then the probability that p_1, p_2, p_3 can still form a dependency path when additionally exposing $S_2 \setminus S$ for $S_2 := S \cup [t_2] \cup \{p_2, p_3\}$ is $\Pr_{S_2}[p_1 \sim p_2 \sim p_3 \mid \pi(S)] \leq \frac{1-\varepsilon}{(el)^2}$.*

The proof of this lemma is the most technical part and can be found in Section 7.2.2.

7.2.2 Probability of Continuing a Dependency Path

Proof of Lemma 7.10. As we will see next, the probability of $p_1 \sim p_2 \sim p_3$ can be bounded by considering an execution of the sequential greedy MIS algorithm on a random ordering.

Connection to Sequential Algorithm

We will work with the following sequential algorithm. Initially, in step $t = 1$, all vertices are called *alive.* Then, in each step $t \in [n]$, position t is exposed (if not already exposed) and vertex $\pi(t)$ processed as follows. If $\pi(t)$ is alive in step t, then $\pi(t)$ is added to the MIS, and $\pi(t)$ as well as all its neighbors are called *dead* (i.e., not alive) for all steps $t' > t$ after t. If $\pi(t)$ is dead in step t, then we proceed to the next step. We say that a vertex *dies* in step t if it is alive in step t and dead in step $t + 1$, and say that it is dead *after* t.

Let $t_1 = \max P_1$ and $t_2 = \max P_2$. Consider the events

$\mathcal{E}_1$: $\pi(p_1), \pi(p_2)$ are neighbors and alive in step $t_1 + 1 \leq p_1$, and

$\mathcal{E}_2$: $\pi(p_2), \pi(p_3)$ are neighbors and alive in step $t_2 + 1 \leq p_1$, and $\pi(p_1), \pi(p_3)$ are not neighbors.

Observe that these two events are necessary for $p_1 \sim p_2 \sim p_3$. Indeed for both $\pi(p_1)$ and $\pi(p_3)$ to enter the independent set they must not share an edge and must be alive in step $t_2 + 1 \leq p_1 < p_3$. Furthermore by definition of a dependency path $\pi(p_2)$ dies in step p_1 and therefore must be alive in step $t_1 + 1$ as well.

In the following, we call an alive and unexposed (with respect to a step and a set of exposed positions) vertex *active*, and let the *active degree* of a vertex be its number of active neighbors.

Proof Sketch

By the above observation it is enough to bound the probability that during the execution of the sequential algorithm up to step t_2 the events $\mathcal{E}_1$ and $\mathcal{E}_2$ occur. First, we will investigate how the active degree of the vertex $v := \pi(p_1)$ evolves and thereby affects the probability of $\mathcal{E}_1$ when the sequential algorithm runs through the position set P_1 up to step t_1. The main observation is that on the one hand, if the active degree d of v in step t_1 is low, then it is unlikely that $\pi(p_2)$ is an active neighbor of v (this happens with probability $\approx \frac{d}{n}$).[6] On the other hand, if the active degree of v is high in step t_1 (and thus during all steps P_1) then v is likely to die because one of its active neighbors enters the MIS (v stays alive with probability $\approx \left(1 - \frac{d}{n}\right)^l$). The combined probability (over $P_1 \cup \{p_2\}$) that v stays alive and one of its active neighbors is selected for

[6]Note that since there are always at least $(1-\beta)n$ unexposed positions the probability that one of d alive vertices is exposed in the next step is always $\Theta\left(\frac{d}{n}\right)$.

$\pi(p_2)$ is $\approx \frac{d}{n}\left(1-\frac{d}{n}\right)^l$. Therefore the probability of $\mathcal{E}_1$ is maximized for $d \approx \frac{n}{l}$, yielding a value $\frac{1}{el}$ for one additional position, and hence $\frac{1}{(el)^2}$ for two positions, which falls just short of our desired bound.

In that seemingly bad case, however, we will argue that $\pi(p_2)$ is likely to have a low active degree, which in turn will make $\mathcal{E}_2$ improbable (by employing a similar argument for $v' := \pi(p_2)$ when running the algorithm through P_2 up to step t_2, also exposing $\pi(p_3)$). Taken together, $\mathcal{E}_1$ *and* $\mathcal{E}_2$ will have low probability in all cases, i.e., for any active degree d of v.

See Figure 7.2 for an illustration.

Formal Proof

We may assume without loss of generality that $[t_2] \setminus (P_1 \cup P_2) \subseteq S$. If not, we expose the missing positions and add them to S. Let $t_0 = \min P_1$ and $S_1 = S \cup P_1 \cup \{p_2\}$. Suppose that S is exposed and that the sequential algorithm has run up to step $t_0 - 1$. We then run the sequential algorithm through the steps $[t_0, t_1] \supseteq P_1$. For $i \in \{0, \dots, l-1\}$, let N_i^1 denote the set of active neighbors of v in step $P_1[i+1]$, and N_l^1 the set of active neighbors of v after step $t_1 = P_1[l]$. We use d_i for the corresponding degrees, let N_i^2 denote the set of active neighbors of (vertices in) N_i^1 without N_i^1, and set $E_i := E(N_i^1, N_i^2)$ and $e_i := |E_i|$, in the respective step.

We now analyze the probability of $\mathcal{E}_1$ by distinguishing several cases based on the set Ω of all vertex sequences $\omega \colon P_1 \to V \setminus \pi(S)$ that can be encountered when exposing P_1. We restrict our attention to $\omega \in \Omega^* := \Omega \setminus \Omega_0$ for $\Omega_0 := \{\omega \in \Omega \colon v \text{ dead in steps after } t_1 \text{ under } \omega\}$, as otherwise $\mathcal{E}_1$ is impossible.

We introduce the following auxiliary algorithm. Let $\mathcal{A}$ be the algorithm that works exactly as the sequential greedy algorithm, except that it picks a vertex for $P_1[i]$ uniformly at random from $V \setminus$

$\big(\pi(S) \cup N^1_{i-1}\big)$ instead of from $V \setminus \pi(S)$. For $\omega \notin \Omega_0$ this set never becomes empty. If $V \setminus \big(\pi(S) \cup N^1_{i-1}\big) = \varnothing$ at some step then define $\mathcal{A}$ to pick vertices in some arbitrary way. We use $\mathsf{Pr}_{\mathcal{A}}[\omega \mid \pi(S)]$ to denote the probability of the algorithm $\mathcal{A}$ picking the sequence $\omega \in \Omega^*$ when exposing P_1, conditioned on $\pi(S)$. Note that

$$\mathsf{Pr}_{\mathcal{A}}[\omega \mid \pi(S)] = \prod_{i=0}^{l-1} \left(\frac{1}{n - |S| - i - d_i} \right)$$

and therefore

$$\begin{aligned} \mathsf{Pr}_{S_1}\left[\pi(P_1) = \omega(P_1) \mid \pi(S)\right] &= \prod_{i=0}^{l-1} \left(\frac{1}{n - |S| - i} \right) \\ &= \mathsf{Pr}_{\mathcal{A}}[\omega \mid \pi(S)] \prod_{i=0}^{l-1} \left(1 - \frac{d_i}{n - |S| - i} \right). \end{aligned}$$

Moreover, the probability of having an active neighbor of v at p_2 under ω is at most $\frac{d_l}{n-|S|-l}$. Thus,

$$\begin{aligned} &\mathsf{Pr}_{S_1}\left[\mathcal{E}_1 \text{ and } \pi(P_1) = \omega(P_1) \mid \pi(S)\right] \\ &\leq \frac{d_l}{n - |S| - l} \mathsf{Pr}_{\mathcal{A}}\left[\omega \mid \pi(S)\right] \exp\left(- \sum_{i=0}^{l-1} \frac{d_i}{n} \right). \end{aligned} \tag{7.1}$$

Since the d_i are decreasing (in i) this term is maximized for $d_0 = \cdots = d_l = \frac{n}{l}$, yielding an upper bound of

$$\begin{aligned} &\mathsf{Pr}_{\mathcal{A}}\left[\omega \mid \pi(S)\right] \frac{n}{el(n - |S| - l)} \\ &\leq \mathsf{Pr}_{\mathcal{A}}\left[\omega \mid \pi(S)\right] \frac{1}{el(1 - \beta)}. \end{aligned}$$

Summing up over $\omega \in \Omega^*$ results in $\mathsf{Pr}_{S_1}[\mathcal{E}_1 \mid \pi(S)] \leq \frac{1}{el(1-\beta)}$. Note that the same upper bound of $\frac{1}{el(1-\beta)}$ can be obtained for $\mathsf{Pr}_{S_2}[\mathcal{E}_2 \mid$

$\pi(S_1)]$, where $S_2 = S_1 \cup P_2 \cup \{p_3\}$, by repeating the argument while exposing P_2. Thus we obtain our first bound of $\left(\frac{1}{el(1-\beta)}\right)^2$.

The next step is to improve it to the claimed bound of $\frac{1-\varepsilon}{(el)^2}$. To that end, we distinguish three different cases for ω, mainly based on the active degree $d_{\lfloor \delta l \rfloor}$ of v after $\lfloor \delta l \rfloor$ steps under ω, for $\delta = 0.01$, say. Note that our assumption of $l \geq 10000$ ensures that $\lfloor \delta l \rfloor \geq 0.99\delta l$.

Large Deviation from Degree $\approx \frac{n}{l}$:
$\Omega_1 = \{\omega \in \Omega^* \colon d_{\lfloor \delta l \rfloor} > 1.1\frac{n}{l} \text{ or } d_l < 0.8\frac{n}{l} \text{ under } \omega\}$

Easy calculations show that for $\omega \in \Omega_1$, when the degree deviates a lot from the optimizer $\frac{n}{l}$, then the upper bound on (7.1) can be improved by a small constant factor $(1 - \varepsilon_1)$.

Degree $\approx \frac{n}{l}$ and Few Edges:

$\Omega_2 = \{\omega \in \Omega^* \colon d_{\lfloor \delta l \rfloor} \leq 1.1\frac{n}{l}, d_l \geq 0.8\frac{n}{l}, e_l \leq 0.6\frac{n^2}{l^2} \text{ under } \omega\}$

For $\omega \in \Omega_2$, we will argue that because e_l is small, the active degree of $\pi(p_2)$ in steps $> t_1$ is likely to be low. In that case, $\mathcal{E}_2$ will have a low probability by an argument analogous to the one in the case $\omega \in \Omega_1$ where the active degree of v deviates from $\frac{n}{l}$:

There can be at most $0.9375d_l$ vertices in N_l^1 with degree into N_l^2 larger than $0.8\frac{n}{l}$ after step t_1. All other at least $0.0625d_l$ many vertices in N_l^1 thus have degree into N_l^2 at most $0.8\frac{n}{l}$. For $\pi(p_2)$ such that this is the case, analogously[7] to the case $\omega \in \Omega_1$ from before, we improve upon the trivial bound by a constant factor $(1 - \varepsilon_1)$. For $\pi(p_2)$ with larger degree we obtain the trivial bound of $\Pr_{\mathcal{A}'}[\omega' \mid \pi(S_1)]\frac{1}{el(1-\beta)}$. Thus, on average we improve by some factor $(1 - \varepsilon_2)$.

[7]The main observations needed for adapting the proof is that we can ignore all sequences for P_2 under which a vertex in N_l^1 or N_l^2 is exposed, since then either v or v' would die, and that $\pi(p_3) \in N_l^2 \setminus N_l^1$ is necessary for $\mathcal{E}_2$.

Degree $\approx \frac{n}{l}$ and Many Edges:
$\Omega_3 = \{\omega \in \Omega^* \colon d_{\lfloor \delta l \rfloor} \leq 1.1\frac{n}{l}, d_l \geq 0.8\frac{n}{l}, e_l > 0.6\frac{n^2}{l^2} \text{ under } \omega\}$

We will argue that $\sum_{\omega \in \Omega_3} \mathsf{Pr}_{\mathcal{A}}[\omega \mid \pi(S)]$ is bounded away from 1 by a constant. Intuitively speaking, the more edges there are, the likelier it is that vertices in N_i^1 die. It is thus not too likely to have only a small drop in the active degree over all $(1-\delta)l$ steps if in every step the number of edges in E_i is large.

Run $\mathcal{A}$ for $\lfloor \delta l \rfloor$ many steps and suppose that $d_{\lfloor \delta l \rfloor} \leq 1.1\frac{n}{l}$. We will bound the probability that $d_l \geq 0.8\frac{n}{l}$ and $e_l > 0.6\frac{n^2}{l^2}$ if we continue to run $\mathcal{A}$.

If $e_i \geq 0.6\frac{n^2}{l^2}$ then let $M_i \subseteq E_i$ be a subset of exactly $\lceil 0.6\frac{n^2}{l^2} \rceil$ edges, and let X_{i+1} denote the number of vertices in N_i^1 which are connected through M_i with the vertex selected in step $i+1$ scaled down by $\gamma := \left(n - |S \cup N_i^1| - i\right)/n$. To shorten notation, we use $\xi_i := \pi\left(S \cup P_1\left([i]\right)\right)$. Then, for (say) $\beta \leq 0.01$,

$$\mathbb{E}_{\mathcal{A}}\left[X_{i+1} \mid \xi_i\right] = \frac{\gamma |M_i|}{n - |S \cup N_i^1| - i} = \frac{|M_i|}{n} = \frac{\lceil 0.6\frac{n^2}{l^2} \rceil}{n}.$$

If $e_i \leq 0.6\frac{n^2}{l^2}$ then define X_i to be a constant random variable (with the same expectation). Note that these random variables are uncorrelated. Indeed since X_i is fully determined by ξ_i while (for $j > i$) $\mathbb{E}_{\mathcal{A}}[X_j \mid \xi_i]$ does not depend on ξ_i we have:

$$\begin{aligned}
\mathbb{E}_{\mathcal{A}}[X_i X_j] &= \sum_{\xi_i} \mathbb{E}_{\mathcal{A}}[X_i X_j \mid \xi_i] \mathsf{Pr}_{\mathcal{A}}[\xi_i] \\
&= \sum_{\xi_i} \mathbb{E}_{\mathcal{A}}[X_i \mid \xi_i] \mathbb{E}_{\mathcal{A}}[X_j \mid \xi_i] \mathsf{Pr}_{\mathcal{A}}[\xi_i] = \mathbb{E}_{\mathcal{A}}[X_i]\mathbb{E}_{\mathcal{A}}[X_j].
\end{aligned}$$

Now let $X := \sum_{i=\lfloor \delta l \rfloor}^{l} X_i$ (note that if $e_l > 0.6\frac{n^2}{l^2}$ then this is a lower bound on $d_{\lfloor \delta l \rfloor} - d_l$) with expectation $\mathbb{E}_{\mathcal{A}}[X \mid \xi_{\lfloor \delta l \rfloor}] \geq 0.5\frac{n}{l}$

and variance

$$\mathsf{Var}_{\mathcal{A}}[X_{i+1} \mid \xi_{\lfloor \delta l \rfloor}] \le \mathbb{E}_{\mathcal{A}}[X_{i+1}^2 \mid \xi_{\lfloor \delta l \rfloor}] \le d_{\lfloor \delta l \rfloor} \mathbb{E}_{\mathcal{A}}[X_{i+1} \mid \xi_{\lfloor \delta l \rfloor}] \le \frac{n^2}{l^3},$$

where we have used $X_{i+1} \le d_{\lfloor \delta l \rfloor} \le 1.1\frac{n}{l}$.

Since the X_i are uncorrelated, the variance of the sum is at most $\mathsf{Var}_{\mathcal{A}}[X \mid \xi_{\lfloor \delta l \rfloor}] \le \frac{n^2}{l^2}$. Hence, by *Cantelli*'s inequality (the one-sided version of *Chebyshev*'s inequality),

$$\begin{aligned} \mathsf{Pr}_{\mathcal{A}}\left[X \ge 0.4\frac{n}{l} \mid \xi_{\lfloor \delta l \rfloor}\right] &\ge \mathsf{Pr}_{\mathcal{A}}\left[X - \mathbb{E}_{\mathcal{A}}[X \mid \xi_{\lfloor \delta l \rfloor}] \ge -0.1\frac{n}{l} \mid \xi_{\lfloor \delta l \rfloor}\right] \\ &\ge 1 - \frac{\mathsf{Var}_{\mathcal{A}}[X \mid \xi_{\lfloor \delta l \rfloor}]}{0.001\frac{n^2}{l^2} + \mathsf{Var}_{\mathcal{A}}[X \mid \xi_{\lfloor \delta l \rfloor}]} \ge C > 0, \end{aligned}$$

for some constant C.

Thus, if we have $d_{\lfloor \delta l \rfloor} \le 1.1\frac{n}{l}$ then with probability at least C either $e_l \le 0.6\frac{n^2}{l^2}$ or the active degree drops by at least $0.4\frac{n}{l}$. Both cases are excluded from Ω_3. Therefore $\sum_{\omega \in \Omega_3} \mathsf{Pr}_{\mathcal{A}}[\omega \mid \pi(S)] \le 1 - C$.

Wrap-Up: Combining these three cases, we obtain

$$\begin{aligned} &\sum_{\omega \in \Omega} \mathsf{Pr}_{S_2}[\mathcal{E}_1 \text{ and } \mathcal{E}_2 \text{ and } \pi(P_1) = \omega(P_1) \mid \pi(S)] \\ &\le \sum_{\omega \in \Omega_1 \cup \Omega_2} \mathsf{Pr}_{\mathcal{A}}[\omega \mid \pi(S)] \frac{1 - \min\{\varepsilon_1, \varepsilon_2\}}{((1-\beta)el)^2} \\ &\quad + \sum_{\omega \in \Omega_3} \mathsf{Pr}_{\mathcal{A}}(\omega \mid \pi(S)) \frac{1}{((1-\beta)el)^2}, \end{aligned}$$

which is at most $\frac{1-\varepsilon}{(el)^2}$ for some absolute small constants $\varepsilon, \beta > 0$, since $\sum_{\omega \in \Omega_3} \mathsf{Pr}_{\mathcal{A}}[\omega \mid \pi(S)] \le 1 - C$. $\square$

7.2.3 Bound on Dependency Length

We will bound the probability that there exists a dependency path of length $\Omega(\log n)$ in the interval $[A \log n, \beta n]$ for some large A, thereby proving Theorem 7.9.

Proof of Theorem 7.9. We first upper bound the probability that a fixed set P of $L = \Theta(\log n)$ (for odd L) positions forms a dependency path by iteratively applying Lemma 7.10. We then take a union bound over all possible choices for P.

Probability for Fixed Positions

Let $I = [1, \ldots, n]$ and let A denote a large constant which we will fix later. For a fixed set $P = \{p_1, \ldots, p_L\} \subseteq [A \log n, n]$ of $L = \sqrt{A} \log n$ positions, we will choose position sets P_k to associate with each p_k which satisfy the conditions

(i) $P_1 < \cdots < P_L$,

(ii) $P_k < p_k$ for all $k \in [L]$, and additionally

(iii) $P_k < p_{k-1}$ and $|P_k| = |P_{k-1}|$ for all even $k \in [L]$.

This will ensure the applicability of Lemma 7.10 to (p_{k-1}, p_k, p_{k+1}) for even $k \in [L]$.

We break the interval $[n]$ into exponentially growing sub-intervals $I_i := \big[(1+\gamma)^i, (1+\gamma)^{i+1}\big)$ for some small enough constant $0 < \gamma < 1$. Let $s_i := |I_i \cap (P \setminus \{p_L\})|$ denote the number of positions in $P \setminus \{p_L\}$ that intersect I_i. Observe that $s_i = 0$ for all $i + 1 \leq \log_{1+\gamma}(A \log n)$. For convenience, we use $p_{i,j}$ to refer to p_k if p_k is the j^{th} position among $I_i \cap P$.

If $s_i > 0$ then we split $I_{i-1} \setminus P$ into $s_i + 1$ position sets $I_{i-1,1} < \ldots <$

I_{i-1,s_i+1}, each of size

$$l_i := \left\lfloor \frac{|I_{i-1} \setminus P|}{s_i + 1} \right\rfloor = \left\lfloor \frac{\lfloor \gamma(1+\gamma)^{i-1} \rfloor - |P \cap I_{i-1}|}{s_i + 1} \right\rfloor .$$

Note that these subsets are not necessarily contiguous and do not contain any of the positions $p_k \in P$. We claim that $l_i \geq \gamma(1+\gamma)^{i-2}/(s_i+1) \geq 10000$. Indeed for s_i to be non-zero i must satisfy $(1+\gamma)^{i+1} \geq A \log n = \sqrt{A}|P|$. Thus the first inequality holds for $A(\gamma)$ large enough. Secondly $s_i \leq |P|$ thus the second inequality holds for A large enough as well.

Next we assign each position $p_k = p_{i,j} \in P$ a position set

$$P_k = I(p_k) = I(p_{i,j}) := \begin{cases} I_{i-1,j}, & \text{if } k \text{ is odd or } j \neq 1, \\ I_{i-h_i-1, s_{(i-h_i)}+1}, & \text{if } k \text{ is even, } j = 1, \end{cases}$$

where $h_i := i - \max\{j \mid j < i \colon s_j \neq 0\}$ is the distance $i - j$ of I_i to the next smaller interval I_j which contains at least one position from P.[8] In words, we assign the j^{th} position $p_{i,j}$ in the interval I_i the j^{th} position set $I_{i-1,j}$ in the previous interval I_{i-1}, except for positions p_k for which k is even and p_k is the first position in some interval I_i. Since in that case p_{k-1} is in the interval I_{i-h_i}, and thus is assigned a position set $I_{i-h_i-1,s_{(i-h_i)}}$, we have to assign p_k the position set $I_{i-h_i-1,s_{(i-h_i)}+1}$ in order to not violate condition (iii).

Let $S_0 = \{p_1\}$, and for every even $k \in [L]$, let $S_k = S_{k-2} \cup [\max P_k] \cup \{p_k, p_{k+1}\}$. Then, iteratively for every even $k \in [L]$, apply Lemma 7.10 with $(p_1, p_2, p_3) \leftarrow (p_{k-1}, p_k, p_{k+1})$, $(P_1, P_2) \leftarrow (P_{k-1}, P_k)$, $l \leftarrow |P_{k-1}| = |P_k|$, $S \leftarrow S_{k-2}$. Observe that indeed $(P_{k-1} \cup P_k) \cap S_{k-2} = \varnothing$, $p_{k-1} \in S_{k-2}$, and $S_k = S_{k-2} \cup [\max P_k] \cup \{p_k, p_{k+1}\}$, which—together with properties (i)–(iii)—then makes

[8]Note that, since we are interested in h_i only for positions $p_k = p_{i,1}$, where k is even. Thus in particular not for p_1, there will actually always be a $j < i$ with $s_j \neq 0$.

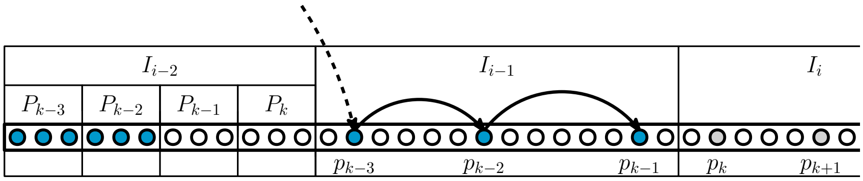


Figure 7.2: The situation before applying Lemma 7.10 to (p_{k-1}, p_k, p_{k+1}). The set $I_{i-2} \setminus P$ has been split into $s_{i-1} + 1 = 4$ parts. The first three parts are associated with the points in $P \cap I_{i-1}$. Since k is even and p_k is the first position in the interval I_i, its associated set is $P_k = I_{i-h_i-1, s_{i-h_i}+1} = I_{i-2,4}$ (and not $I_{i-1,1}$). This ensures that $|P_{k-1}| = |P_k|$. The positions before $P_{k-1} = I_{i-2,3}$ as well as p_{k-3}, p_{k-2}, p_{k-1} have already been exposed. Invoking Lemma 7.10 will additionally expose P_{k-1}, P_k, p_k and p_{k+1}.

the lemma applicable. Thus for every even k we obtain a bound of $\frac{1-\varepsilon}{e^2|P_{k-1}||P_k|}$ on the probability that p_{k-1}, p_k, and p_{k+1} form a dependency path when exposing S_k, conditioned on already having exposed S_{k-2}. Thus, P forms a dependency path with probability at most

$$\prod_{k \in [L]\colon k \text{ even}} \Pr_{S_k}\left[p_{k-1} \sim p_k \sim p_{k+1} \mid \pi(S_{k-2})\right]$$

$$\leq \prod_{k \in [L]\colon k \text{ even}} \frac{1-\varepsilon}{e^2|P_{k-1}||P_k|} = \prod_{i\colon s_i \neq 0} \prod_{j=1}^{s_i} \frac{\sqrt{1-\varepsilon}}{e|I(p_{i,j})|}.$$

Let $\tilde{I} = \{i\colon s_i \neq 0 \wedge p_{i,1} = p_k \text{ for some even } k\}$. Then for $i \in \tilde{I}$

$$\begin{aligned}\frac{l_i}{|I(p_{i,1})|} = \frac{l_i}{l_{i-h_i}} &\leq (1+\gamma)^{h_i} \frac{s_{i-h_i}+1}{s_i+1}(1+\gamma) \\ &\leq (1+\gamma)^{h_i}(s_{i-h_i}+1).\end{aligned}$$

Since $\sum_{i \in \tilde{I}} h_i \leq \log_{1+\gamma} n$ and because the mapping $f\colon \tilde{I} \to \{i\colon s_i \geq 1\}$ that maps $f(i) = i - h_i$ is an injection, we can bound the product by

$$\prod_{i\in\tilde{I}} \frac{l_i}{|I(p_{i,1})|} \leq \prod_{i\in\tilde{I}} (1+\gamma)^{h_i}(s_{i-h_i}+1) \leq n \prod_{i\colon s_i\geq 1} (s_i+1)$$
$$\leq n\left(\frac{2L}{\log_{1+\gamma}(n)}\right)^{\log_{1+\gamma}(n)}.$$

Finally by definition $|I(p_{i,j})| = l_i$ whenever $i \notin \tilde{I}$ or $j > 1$. Therefore we obtain

$$\prod_{i\colon s_i\neq 0}\prod_{j=1}^{s_i} \frac{\sqrt{1-\varepsilon}}{e|I(p_{i,j})|} \leq n\left(\frac{2L}{\log_{1+\gamma}(n)}\right)^{\log_{1+\gamma}(n)} \prod_{i\colon s_i\neq 0}\left(\frac{\sqrt{1-\varepsilon}}{el_i}\right)^{s_i}.$$

Union Bound Over All Positions

For fixed values of $\{s_i\}$ there are at most

$$n\prod_i \binom{\gamma(1+\gamma)^i}{s_i} \leq n \prod_{i\colon s_i\neq 0}\left(\frac{e\gamma(1+\gamma)^i}{s_i}\right)^{s_i}$$
$$= n \prod_{i\colon s_i\neq 0}\left(\left(\frac{s_i+1}{s_i}\right)\left(\frac{e\gamma(1+\gamma)^i}{s_i+1}\right)\right)^{s_i}$$
$$\leq n \prod_{i\colon s_i\neq 0} e\left(el_i(1+\gamma)^2\right)^{s_i}$$
$$\leq n^{1+\frac{1}{\log(1+\gamma)}} \prod_{i\colon s_i\neq 0}\left(el_i\,(1+\gamma)^2\right)^{s_i}$$

choices for positions P with $|I_i\cap(P\setminus\{p_L\})| = s_i$ (counting n choices for p_L), using $\left(\frac{s_i+1}{s_i}\right)^{s_i} \leq e$ in the second inequality and $|\{i\colon s_i \neq 0\}| \leq \log_{1+\gamma} n$ in the third inequality.

Therefore, by a union bound, the probability of having a dependency path of length L for prescribed $\{s_i\}$ is at most

$$n^{2+\frac{1}{\log(1+\gamma)}} \left(\frac{2L}{\log_{1+\gamma} n}\right)^{\log_{1+\gamma} n} \left(\sqrt{1-\varepsilon}(1+\gamma)^2\right)^L .$$

Since we can assign values to $\{s_i\}$ in at most

$$\binom{L+\log_{1+\gamma} n}{\log_{1+\gamma} n} \leq \left(e\left(\frac{L}{\log_{1+\gamma} n}+1\right)\right)^{\log_{1+\gamma} n}$$

ways, it follows that the probability of a dependency path of length L is at most

$$n^{2+\frac{1}{\log(1+\gamma)}} \cdot \left(\frac{2L}{\log_{1+\gamma} n} e \left(\frac{L}{\log_{1+\gamma} n}+1\right)\right)^{\log_{1+\gamma} n} \cdot \left(\sqrt{1-\varepsilon}(1+\gamma)^2\right)^L ,$$

which is $n^{-\Omega(1)}$ for a constant γ small enough depending on ε, and $L = \sqrt{A} \log n$ large enough depending on γ. □

We now use the bound on the dependency length from the previous result to prove our main result in Theorem 7.1.

Proof. By Theorem 7.9, w.h.p. the length of any dependency path in the interval $[\beta n]$ is $O(\log n)$, and by Lemma 7.8, this thus bounds the number of rounds needed by the local algorithm to process this interval by $O(\log n)$.

Suffix $[\beta n, n]$**:** For the suffix $[\beta n, n]$, it can be shown that w.h.p. after processing the first βn positions the maximum degree among all remaining vertices is at most $d = O(\log n)$. See e.g. Lemma 3.1 in [39]. Since each possible path of length L is monotonically increasing with probability $1/L! \leq (e/L)^L$, the probability of having such a path in the suffix is at most $n\,(ed/L)^L$, which is $n^{-\Omega(1)}$ for $L = \Omega(\log n)$ large enough. Finally, observe that for the parallel algorithm to take L rounds there indeed must be a monotonically increasing path of length L. □

7.3 Lower Bound

Proof of Theorem 7.2. Consider a graph consisting of $\sqrt{n}$ connected components, each connected component made of $l+1$ layers for $l := \frac{\log n}{5}$ as follows. The i^{th} layer for $i \in [0, l]$ is a clique on 2^i vertices, and there is a full bipartite graph between any two consecutive layers. The remaining vertices not part of a layer are just isolated.

A path of length l is strictly increasing if the i^{th} vertex on the path for $0 \leq i \leq l$ is in layer $l - i$ and the of the path vertices are sorted (in the random order). We will argue that the probability that a connected component contains such a strictly increasing path is at least $n^{-0.05}$.

Consider the layers $U_0, \ldots, U_l$ of one connected component U and a random order. The probability that U_l contains the minimum among $\bigcup_{i=0}^{l} U_i$ is at least $\frac{|U_l|}{|U|} \geq \frac{1}{4}$. Then, conditioned on the previous event, the probability of U_{l-1} containing the minimum among $\bigcup_{i=0}^{l-1} U_i$ is at least $\frac{|U_{l-1}|}{|\bigcup_{i=0}^{l-1} U_i|} \geq \frac{1}{4}$. Continuing this argument, combining the conditional probabilities, we get a lower bound of $\left(\frac{1}{4}\right)^l = n^{-0.05}$ on the probability that there is a strictly increasing path in U. Since all the $\sqrt{n}$ connected components are independent, the probability of no such component containing a strictly increasing path is at most $\left(1 - n^{-0.05}\right)^{\sqrt{n}} \ll n^{-\Omega(1)}$. Thus, with high probability, the considered graph contains such a path.

Finally, observe that the local greedy MIS algorithm will take at least $(l+1)/2$ rounds until it has processed such a strictly increasing path, since the algorithm processes only 2 layers in each round. $\square$

CHAPTER 8

Local Sampling

8.1 Introduction

We study local sampling of proper vertex coloring, based on the publication ‘A Simple Parallel and Distributed Sampling Technique: Local Glauber Dynamics’ [102].

8.1.1 Our Results and Related Work

The centralized Glauber dynamics for vertex coloring [147, 218] works as follows: in every step, a random vertex picks a random color and updates its color if the new color does not lead to a conflict. This Markov chain takes $n \log n$ steps to mix. We present a strikingly simple local sampling technique, called Local Glauber Dynamics, that fully parallelizes this centralized Glauber dynamics.

Theorem 8.1 (Local Glauber Dynamics)**.** *A uniform proper q-coloring can be sampled within total variation distance $\varepsilon > 0$ in $O\left(\log \frac{n}{\varepsilon}\right)$ rounds, where $q = \alpha\Delta$ for any $\alpha > 2$.*

This improves on the *LubyGlauber* algorithm of Feng, Sun, and Yin [96], which needs $O(\Delta \log n)$ rounds, and their *LocalMetropolis* algorithm, which converges in $O(\log n)$ rounds but requires a considerably stronger condition of $\alpha > 2 + \sqrt{2}$. They state that

> *"We also believe that the $2 + \sqrt{2}$ threshold is of certain significance to this [LocalMetropolis] chain as the Dobrushin's condition to the Glauber dynamics."*

implying that this value is a barrier for their approach. This is also justified by their supposedly easiest special case of a tree that leads to the same threshold. Our result gets rid of the additional $\sqrt{2}$ while not incurring any loss in the round complexity, with a considerably easier and more natural update rule. Not only our proof is simpler and shorter, also our algorithm is asymptotically best possible, as there is an $\Omega\left(\log \frac{n}{\varepsilon}\right)$ lower bound [129, 96, 130] due to the exponential correlation between variables.

The threshold of $\alpha > 2$ corresponds to Dobrushin's condition, thus almost matches the threshold of the centralized Glauber dynamics [147, 218] at $2\Delta + 1$. In other words, we present a technique that fully parallelizes the Glauber dynamics, speeding up the mixing time from poly n steps to $O(\log n)$ rounds. Besides its many practical ramifications, especially on the area of distributed machine learning, this moves us a step closer towards an answer to the question of what can be sampled locally and gives us a theoretical insight about the locality of problems.

In terms of number of colors needed, Dobrushin's condition can be undercut: Vigoda [225] and three recent works [64, 80, 63] showed that, when resorting to a different highly non-local Markov chain,

$\alpha < \frac{11}{6}$ is enough. This gives rise to the question whether efficient distributed algorithms intrinsically need to be stuck at Dobrushin's condition, which would imply that this bound is inherent to the locality of the sampling process, or whether our threshold is an artifact of our possibly suboptimal dynamics.

Note that independently Feng, Hayes, and Yin [93] arrived at the same result with a slightly different sampling algorithm. They further extended their work in [98, 94, 95].

8.1.2 Notation and Preliminaries

Markov Chain: We consider a Markov chain $X = \left(X^{(t)}\right)_{t \geq 0}$, where $X^{(t)} = (X_v^{(t)})_{v \in V} \in [q]^V$ is the coloring of the graph in round t. We will omit the round index, and use $X = (X_v)_{v \in V} \in [q]^V$ for the coloring at time t and $X' = (X'_v)_{v \in V} \in [q]^V$ for the coloring at time $t + 1$, for a $t \geq 0$, instead.

Mixing Time: For a Markov chain $\left(X^{(t)}\right)_{t \geq 0}$ with stationary distribution μ, let $\pi_\sigma^{(t)}$ denote the distribution of the random coloring $X^{(t)}$ of the chain at time $t \geq 0$, conditioned on $X^{(0)} = \sigma$. The mixing time $\tau_{\text{mix}}(\varepsilon) = \max_{\sigma \in \Omega} \min \left\{ t \geq 0 \colon \mathsf{d_{TV}} \left(\pi_\sigma^{(t)}, \mu \right) \right\}$ is defined to be the minimum number of rounds needed so that the Markov chain is ε-close (in terms of total variation distance) to its stationary distribution μ, regardless of $X^{(0)}$. The total variation distance between two distributions μ, ν over Ω is defined as $\mathsf{d_{TV}}(\mu, \nu) = \sum_{\sigma \in \Omega} \frac{1}{2} \left| \mu(\sigma) - \nu(\sigma) \right|$.

Path Coupling: The *Path Coupling Lemma* by Bubley and Dyer [49, Theorem 1] (also see [96, Lemma 4.3]) gives rise to a particularly easy way of designing couplings. In a simplified version, it says that it is enough to define the coupling of a Markov chain only for pairs of colorings that are adjacent, that is, differ at exactly one vertex. The expected number of differing vertices after one coupling step then can be used to bound the mixing time of the Markov chain.

Lemma 8.2 (Path Coupling [49], simplified). *For $\sigma, \sigma' \in [q]^V$, let $\phi(\sigma, \sigma') := |\{v \in V : \sigma_v \neq \sigma'_v\}|$. If there is a coupling $(X, Y) \to (X', Y')$ of the Markov chain, defined only for (X, Y) with $\phi(X, Y) = 1$, that satisfies $\mathbb{E}[\phi(X', Y') \mid X, Y] \leq 1 - \delta$ for some $0 < \delta < 1$, then $\tau_{\mathsf{mix}}(\varepsilon) = O\left(\frac{1}{\delta} \log \frac{n}{\varepsilon}\right)$.*

8.2 Local Glauber Dynamics

We propose the following generic sampling method, which we call *Local Glauber Dynamics*: In every step, every variable independently marks itself at random with a certain (low) probability. If it is marked, it samples a proposal at random and checks with its neighbors whether the proposal leads to a conflict with their current state or their new proposals (if any). If there is a conflict, the variable rolls back and stays with its current state, otherwise the state is updated.

More concretely, we define a transition from $X = (X_v)_{v \in V}$ to $X' = (X'_v)_{v \in V}$ in one round as follows. Every node $v \in V$ marks itself independently with probability $0 < \gamma < 1$. If it is marked, it proposes a new color $c_v \in [q]$ uniformly at random, independently from all the other nodes. If this proposed color does not lead to a conflict with the current and the proposed colors of any neighbor, that is, $c_v \notin \bigcup_{u \in N(v)} \{X_u, c_u\}$ and $c_u \notin \{X_v, c_v\}$ for any $u \in N(v)$[1], then v accepts color c_v, thus sets $X'_v = c_v$. Otherwise, v keeps its current color, that is, sets $X'_v = X_v$. Note that the condition $c_v \notin \bigcup_{u \in N(v)} \{X_u, c_u\}$ is necessary to guarantee reversibility of the Markov chain.

[1]To simplify notation, we assume that $c_u = X_u$ in case u is not marked.

8.2.1 Stationary Distribution

The Local Glauber dynamics is ergodic: it is aperiodic, as there is always a positive probability of not changing any of the colors, and irreducible, since any (proper) coloring can be reached from any coloring. Moreover, the chain might possibly start from an improper coloring, but it will never move from a proper to an improper coloring, that is, it is absorbing to proper colorings. It is easy to verify that this Local Glauber dynamics, due to its symmetric update rule, satisfies the detailed balance equation for the uniform distribution, meaning that the transition from X to X' has the same probability as a transition from X' to X for proper colorings. The chain thus is reversible and has the uniform distribution over all proper colorings as unique stationary distribution.

8.2.2 Mixing Time

Informally speaking, the Path Coupling Lemma says that if for all X and Y which differ in one vertex, we can define a coupling $(X, Y) \to (X', Y')$ in such a way that the expected number of vertices at which X' and Y' differ is bounded away from 1 from above, then the chain converges quickly. In Section 8.2.3, we formally describe such a path coupling, in Section 8.2.3, we list necessary (but not necessarily sufficient) conditions for a vertex to have two different colors after one coupling step, which is then used in Section 8.2.3 to bound the expected number of differing vertices by $1-\delta$ for some constant $0 < \delta < 1$, depending on α. Application of Lemma 8.2 then concludes the proof of Theorem 8.1.

8.2.3 Description of Path Coupling

We look at two colorings X and Y that differ at a vertex $v_0 \in V$ only. That is, $g = X_{v_0} \neq Y_{v_0} = b$, for some $g \neq b \in [q]$, which we will naturally refer to as green and blue, respectively, and $X_v = Y_v$

for all $v \neq v_0 \in V$. In the following, we explain how every node $v \in V$ comes up with a pair (c_v^X, c_v^Y) of new proposals, which then will be accepted or rejected based on the Local Glauber dynamics rules.

Marking: In both chains, every node $v \in V$ is marked independently with probability γ, using the same randomness in both chains. In the following, we restrict our attention to marked nodes only; non-marked nodes are thought of proposing their current color as new color, i.e., $c_v^X = X_v$ and $c_v^Y = Y_v$.

Consistent, Mirrored, and Flipped Proposals: We introduce two possible ways of how proposals for a node v can be sampled: *consistently* and *mirroredly.* For the consistent proposals, both chains propose the same randomly chosen color, that is, $c_v^X = c_v^Y = c$ for a u.a.r. $c \in [q]$. For the mirrored proposals, both chains assign the same random proposal if it is neither green nor blue, and a *flipped* proposal (i.e., green to one and blue to the other chain) otherwise. More formally, $c_v^X = c$ and $c_v^Y = \bar{c}$ if $c \in \{g, b\}$ and $\bar{c}$ the element in $\{g, b\} \setminus \{c\}$, and $c_v^X = c_v^Y = c$ if $c \notin \{g, b\}$, for a u.a.r. $c \in [q]$. We say that v has *flipped* proposals if $c_v^X \neq c_v^Y$. Note that we say mirrored proposal to refer to the process of sampling mirroredly, and we say flipped if, as a result of sampling mirroredly, a node proposes different colors in the two chains.

Breadth-First Assignment of Proposals: Let

$$B = \{v \in V \setminus \{v_0\} \colon X_v \in \{g, b\}\} \subseteq V \setminus \{v_0\}$$

be the set of vertices $v \neq v_0$ with current color green or blue, as well as

$$K = \left(\bigcup_{v \in B} N^+(v) \right) \setminus \{v_0\}$$

its inclusive neighborhood, without v_0, where $N^+(v) := N(v) \cup \{v\}$. We ignore this set K for the moment, and focus on the set $S \subseteq V \setminus K$

of marked vertices that are not adjacent to a vertex with color green or blue (except for possibly v_0). Informally speaking, we will go through these vertices in a breadth-first manner, with increasing distance $d \geq 0$ to vertex v_0, and fix their proposals layer by layer, but defer the assignment of vertices not (yet) adjacent to a vertex with flipped proposals, as follows. We repeatedly add all (still remaining) vertices that have a vertex in the last layer with flipped proposals to a new layer, and sample their proposals mirroredly, thus perform a breadth-first assignment on vertices with flipped proposals only. All remaining vertices sample their proposals consistently. Note that this in particular guarantees that a vertex is sampled consistently only if it not adjacent to a vertex with flipped proposals.

More formally, this can be described as follows. We define $M^0 = F^0 = \{v_0\}$, even if v_0 is not marked. For vertex v_0, if marked, the proposals are sampled consistently. For $d \geq 1$ and $v \in M^d$, the proposals are sampled mirroredly. For the subsequent layer, we restrict the attention to (new) neighbors of vertices in M^d with flipped proposals only, i.e., consider

$$M^{d+1} = N\left(F^d\right) \setminus \bigcup_{i=0}^{d} M^d$$

for

$$F^d = \{v \in M^d \colon c_v^X \neq c_v^Y\}.$$

For all remaining (marked) vertices, that is, vertices in $S \setminus M$ and vertices in K, proposals are sampled consistently. Here, $M := \bigcup_{d \geq 0} M_d$. See Figure 8.1 for an illustration of this breadth-first-based approach.

Accept Proposals: The proposals $(c_v^X)_{v \in V}$ and $(c_v^Y)_{v \in V}$ in the chains X and Y are accepted or rejected based on the Local Glauber dynamics rules, leading to colorings $X', Y' \in [q]^V$.

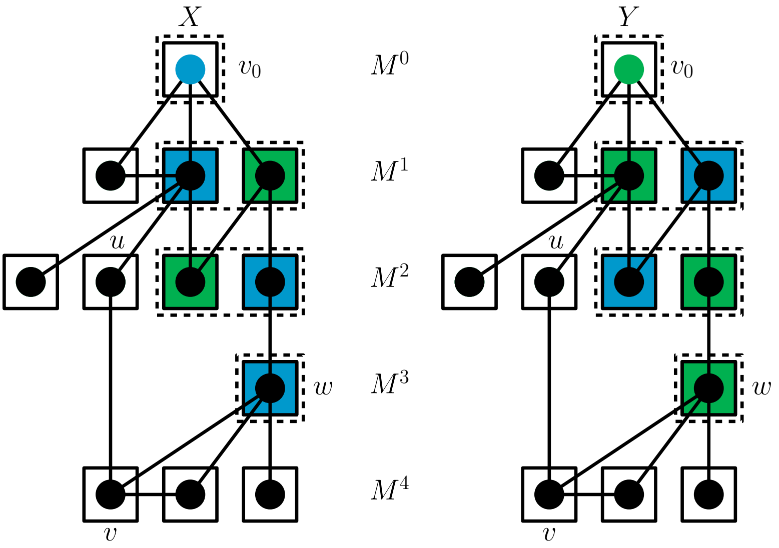


Figure 8.1: The breadth-first layers M^d for $d \geq 0$ of two chains that differ at $v_0 \in M^0$. The disk color corresponds to the vertex' current color, where black means any color except green and blue. The color of the box around a vertex shows this vertex' proposed color, where white stands for any color (possibly also green or blue, but consistent). Dashed boxes indicate the sets F^d of vertices with flipped proposals. Note that vertex v appears in layer 4 even though it has distance 3 to v_0. This is because we perform the breadth-first assignment only on vertices with flipped proposals. v's neighbor u does not have flipped proposals, thus is in $M^2 \setminus F^2$, which means that u's neighbors are not added to the next layer. Only v's neighbor $w \in F^3$ leads to v being added to M^4.

Properties of the Coupling

The main observation is the following. If we ignore vertices with current colors green and blue for the moment, one can argue that X' and Y' can only differ at a vertex different from v_0 if its proposals are flipped. Flipped proposals, however, can only arise when the proposals are sampled mirroredly, which happens only if there is a vertex in the preceding layer with flipped proposals (due to the breadth-first order in which we assign the proposals). A vertex thus can lead to an inconsistency only if there is path in G from v_0 to this vertex consisting of vertices with flipped proposals, called a *flip path.*

We will next make this intuition with the flip paths more precise, in two parts: for vertices in S (that sample their proposals mirroredly if adjacent to a vertex with flipped proposals) in Lemma 8.3 and for vertices in K (that always sample their proposals consistently) in Lemma 8.4. See Figure 8.2 for an illustration of these two cases.

Lemma 8.3. *If X' and Y' differ at $v \neq v_0 \in S$, there is a flip path $(v_0, \dots, v_\ell = v) \in F^0 \times \dots \times F^\ell$ of length $\ell \geq 1$ in G, with the additional property that the proposal of v is the opposite of the last color (in green and blue) seen on this path, in both chains. More formally, $c_Y = c_v^X \neq c_v^Y = c_X$, where $c_X = c_{v_{\ell-1}}^X$ and $c_Y = c_{v_{\ell-1}}^Y$ if $\ell > 1$, and $c_X = X_{v_0}$ and $c_Y = Y_{v_0}$ if $\ell = 1$.*

Proof. We first argue that v's proposals must be flipped and accepted in both chains. Trivially, acceptance of a consistent proposal in both chains or rejection in both chains leads to $X'_v = Y'_v$. Moreover, observe that flipped proposals are, by construction, either accepted in both or rejected in both chains, as flipping changes the role of green and blue, but not the overall behavior. Indeed, suppose, without loss of generality, that $c_v^X = c \in \{g, b\}$ is rejected by X. Thus, in particular, v has a neighbor u with current color or proposal c in X. As we are restricting our attention to the set S

which does not have any adjacent vertex with current color green or blue, except for v_0, either $u = v_0$ or u proposes c. So u either must have different current colors (if $u = v_0$) or have mirrored proposals (if $v \in F^d$, then $u \in M^{d'}$ for some $d' \leq d+1$, because at the latest v's flipped proposal leads to u being added to the subsequent layer, by how we assign the proposals in breadth-first manner) and hence flipped proposals. Thus, v's proposal $\overline{c}$ in Y will be rejected by Y, since either $u = v_0 \in N(v)$ has color $\overline{c}$ or $u \in N(v)$ proposes $\overline{c}$.

It thus remains to rule out the case of consistent proposals that are accepted in one and rejected in the other chain. Towards a contradiction, suppose that v proposes the same color c_v in both chains, and that it is accepted in one and rejected in the other. Since $X_v = Y_v$ and $c_v^X = c_v^Y$, this can happen only if v is adjacent either to v_0 or to at least one vertex with flipped proposals, as otherwise all proposals and all current colors in v's inclusive neighborhood would be the same, leading to the same behavior in both chains. In both cases, $v \in M^d$ for some $d \geq 1$, which means that its proposals are sampled mirroredly. Hence, $c_v \notin \{g, b\}$, as otherwise the proposals would be flipped. Now, since neither v's current color nor v's proposals is green or blue, and neighbors of v can differ in their colors or proposals only if green or blue is involved, the proposals are either accepted or rejected in both chains. It follows that indeed only vertices in S with flipped proposals that are accepted in both chains can have different colors in X' and Y'.

By construction of the layers, and since $v \in F^\ell$ for some $\ell \geq 1$, there must exist a sequence of vertices $v_1 \in F^1, \ldots, v_{\ell-1} \in F^{\ell-1}$ connecting v_0 to v in G: a flip path of length ℓ. Moreover, the proposal is accepted in a chain only if the proposed color is the opposite of the color (green or blue) that is seen on the path (either as proposal if $\ell > 1$, or as current color of v_0 if $\ell = 1$). $\square$

Lemma 8.4. *If X' and Y' differ at $v \neq v_0 \in K$, there is a path*

$(v_0, \dots, v_\ell = v) \in F^0 \times \cdots \times F^{\ell-1} \times K$ of length $\ell \geq 1$ in G, called almost flip path, with the additional property that the proposal of v is either green or blue, that is, $c_v = c_v^X = c_v^Y \in \{g, b\}$.

Proof. Since, by definition of the coupling, $v \in K$ samples its proposals consistently, X' and Y' can only differ at $v \neq v_0$ if the proposal is accepted in one and rejected in the other chain. This can happen only if v is adjacent to either v_0 or to at least one vertex with flipped proposals. Otherwise, all proposals and all current colors in v's inclusive neighborhood would be the same, leading to the same behavior. Hence, v is adjacent to some $u \in F^d$ for some $d \geq 0$. By construction of the layers, there must exist a sequence of vertices $v_1 \in F^1, \dots, v_{\ell-1} = u \in F^{\ell-1}$ connecting v_0 to v in G: an almost flip path of length $\ell = d+1$. Note that, in particular, because neighbors of vertices in B are by definition sampled consistently (as they are in K), and a vertex at the end of an almost flip path has a neighbor with flipped proposals, this last vertex on an almost flip path must be in $K \setminus B$.

The proposal c_v is accepted in one and rejected in the other chain only if $c_v \in \{g, b\}$. In that case, the chain with the same color on the end of the path will reject, the other will (possibly) accept. □

Bounding the Expected Number of Differing Vertices

We show that $\mathbb{E}[\phi(X', Y') \mid X, Y] \leq 1 - \delta$ for some $0 < \delta < 1$, by bounding the expectations $\mathbb{E}[\sum_{v \neq v_0 \in V} 1\left(X'_v \neq Y'_v\right) \mid X, Y]$ and $\mathbb{E}[1\left(X'_{v_0} \neq Y'_{v_0}\right) \mid X, Y]$ separately. We will see that, as δ tends to 0, both terms can be bounded by $\approx \frac{1}{\alpha}$, leading to an expected number of roughly $\frac{2}{\alpha}$, which is strictly less than 1 for $\alpha > 2$.

Vertices $v \neq v_0$: Section 8.2.3, or more precisely, Lemmas 8.3 and 8.4, show that the number of vertices (different from v_0) that have different colors in X' and Y' can be bounded by the number of (almost) flip paths with an additional property. We will next see

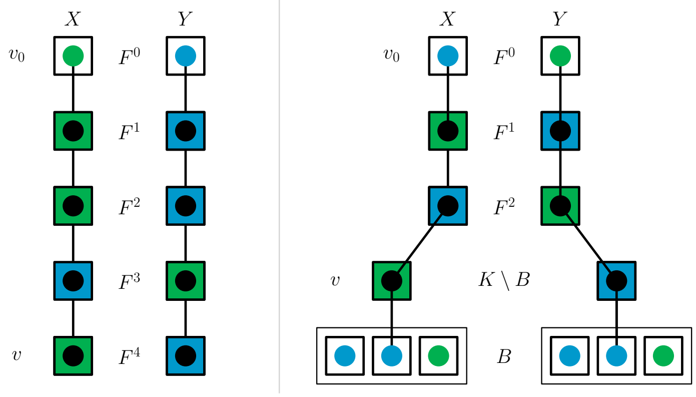


Figure 8.2: The disk color corresponds to the vertex' current color, where black means any color except green and blue. The color of the box around a vertex indicates this vertex' proposed color, where white means any color (also green and blue, but consistent). A flip path on the left: v's flipped proposals are accepted in both chains, yielding $X'_v = g$ and $Y'_v = b$. An almost flip path on the right: $v \in K \setminus B$ samples its proposals consistently. In chain X, the proposal g will be accepted, in chain Y, it will be rejected, leading to $X'_v = g \neq Y'_v = Y_v$.

that the expected number of such (almost) flip paths can be expressed as a geometric series summing over the depths of the layers.

There are at most Δ^ℓ paths $(v_0, \ldots, v_\ell)$ of length ℓ in G. Moreover, each such path has probability $(2\gamma/q)^\ell$ of being a flip or almost flip path with the mentioned additional property, since all intermediate vertices $v_1, \ldots, v_{\ell-1}$ need to mark themselves and to propose one arbitrary color in $\{g, b\}$, and v_ℓ needs to mark itself and to propose the color in $\{g, b\}$ as specified in Lemmas 8.3 and 8.4, respectively. Note that a path in G can either be a flip path or an almost flip path, but never both. Moreover, observe that vertex v_0 does not need to be marked. We get

$$\begin{aligned}\mathbb{E}\left[\sum_{v\neq v_0\in V} 1(X'_v \neq Y'_v) \mid X, Y\right] &\leq \sum_{\ell=1}^{\infty} \Delta^{\ell}\left(\frac{2\gamma}{q}\right)^{\ell} \\ &= \sum_{\ell=1}^{\infty}\left(\frac{2\gamma\Delta}{q}\right)^{\ell} \leq \frac{\frac{2\gamma\Delta}{q}}{1-\frac{2\gamma\Delta}{q}}.\end{aligned} \tag{8.1}$$

Vertex v_0: Chains X' and Y' can agree at vertex v_0 only if at least one the proposals is accepted. For that, v_0 needs to be marked and its proposal $c_{v_0} = c^X_{v_0} = c^Y_{v_0}$ needs to be different from all the at most Δ current colors of its neighbors, that is, $c_v \notin \bigcup_{v\in N(v_0)}\{X_v\}$, which happens with probability at least $\gamma\,(1-\Delta/q)$. Moreover, the proposals of v_0's neighbors (if marked) need to avoid at most three colors in $\{c_{v_0}, g, b\}$, possibly less, which happens with probability at least $1-3\gamma/q$. We thus get

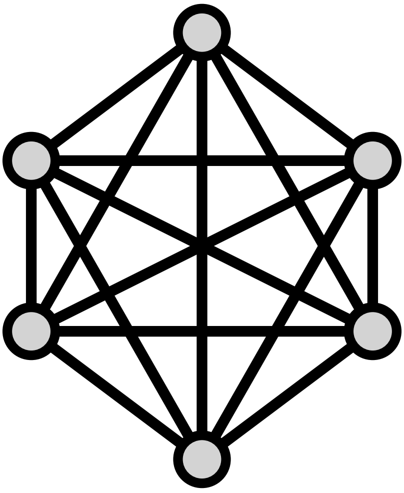
$$\mathbb{E}\left[1\left(X'_{v_0} \neq Y'_{v_0}\right)\right] \leq 1-\gamma\left(1-\frac{\Delta}{q}\right)\left(1-\frac{3\gamma}{q}\right)^{\Delta}. \tag{8.2}$$

Wrap-Up: Overall, combining (8.1) and (8.2), we get

$$\begin{aligned}\mathbb{E}[\phi(X', Y') \mid X, Y] &\leq 1-\gamma\left(1-\frac{1}{\alpha}\right)e^{-\frac{6\gamma}{\alpha}} + \frac{\frac{2\gamma}{\alpha}}{1-\frac{2\gamma}{\alpha}} \\ &= 1-\gamma e^{-\frac{6\gamma}{\alpha}}\left(1-\frac{1}{\alpha}\left(1+\frac{2e^{\frac{6\gamma}{\alpha}}}{1-\frac{2\gamma}{\alpha}}\right)\right).\end{aligned}$$

For $\alpha > 2$ and $\gamma := \gamma(\alpha)$ small enough, this is strictly bounded away from 1 from above, where the hidden constant depends on α (but not on Δ or n).

Part II



All-to-All Communication Models

CHAPTER 9

Vertex Coloring in CC and MPC

9.1 Introduction

We study the problem of vertex coloring in Congested Clique and Massively Parallel Computation. Our results are based on the the manuscript 'Simple Graph Coloring Algorithms for Congested Clique and Massively Parallel Computation' [104] as well as the publication 'The Complexity of $(\Delta + 1)$-Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation' [54]. We mostly focus on conveying the intuition for the CC result; the MPC result works by only slightly adapting the algorithm and parameters. Note that while the result for the Centralized Local Computation model is omitted, as this model is not the focus of the thesis, it follows in a straightforward manner from our analysis.

9.1.1 Our Results and Related Work

$(\Delta+1)$ List Vertex Coloring in Constant CC Rounds

We present an $O(1)$-round randomized algorithm for $(\Delta+1)$ list vertex coloring in the CC model, settling the asymptotic complexity of this problem.

Theorem 9.1. *There is an $O(1)$-round CC algorithm that w.h.p. solves the $(\Delta+1)$ list vertex coloring problem.*

The proof can be found in Section 9.4.

This improves on the $O(\log^* \Delta)$-round randomized algorithms of Parter and Su [210] as well as on the $O(\log\log\Delta\cdot\log^*\Delta)$-round randomized algorithm of Parter [209]. Hegeman and Pemmaraju [141] gave faster algorithms for the significantly more relaxed problem of $O(\Delta)$ coloring: they run in $O(1)$ rounds if $\Delta = \Omega(\log^4 n)$ and in $O(\log\log n)$ rounds otherwise.

Very recently, Czumaj et al. [72] provided a deterministic constant-round $(\Delta+1)$ list vertex coloring algorithm in CC, based on a similar technique.

$(\Delta+1)$ List Vertex Coloring in Sublinear-Memory MPC

We present a randomized $(\Delta+1)$ list vertex coloring algorithm with round complexity $O(\sqrt{\log\log n})$ in the MPC model with strongly sublinear memor.

Theorem 9.2. *There is an $O(\sqrt{\log\log n})$-round sublinear-memory MPC algorithm that w.h.p. solves the $(\Delta+1)$ list vertex coloring problem, using $\mathcal{S} = O(n^\delta)$ memory per node and total memory $\mathcal{MS} = \widetilde{O}(m)$, for any $0 < \delta < 1$.*

The proof can be found in Section 9.5. Notably, this is the first coloring algorithm with sublogarithmic round complexity in the sublinear-memory regime of MPC. For the quasilinear memory regime of MPC, an $O(1)$-round algorithm was given recently by Assadi et al. [13] and also by Czumaj et al. [72]. These algorithms, however, as well as the CC algorithms discussed above, only work in the linear-memory regime.

If we combine our algorithm with the recent LOCAL network decomposition algorithm of [216, 114], we get a round complexity of $O(\log \log \log n)$, which matches a conditional lower bound due to Ghaffari, Kuhn, and Uitto [121].

9.1.2 Overview and Outline

We convey the intuition of our method in two parts corresponding to the two main ingredients of our algorithm: (1) a simple graph partitioning method for coloring which allows us to break a graph coloring instance into many smaller graph coloring instances; and (2) a sparsification of the LOCAL $(\Delta + 1)$ vertex coloring algorithm of Chang, Li, and Pettie [58, 59] that allows us to efficiently implement it in the CC model super-exponentially faster, using our opportunistic speedup technique.

Graph Partitioning

In Section 9.2, we provide a simple random partitioning that significantly simplifies and extends the one in the previous CC algorithms [209, 210]. The main change will be that, besides partitioning the vertices randomly, we also partition the colors randomly. In particular, this new procedure partitions the vertices and colors in a way that allows us to easily apply the LOCAL algorithm CLP of Chang, Li, and Pettie [58, 59] in a black-box manner. Concretely, our partitioning breaks the graph as well as the respective palettes randomly into many subgraphs $B_1, \ldots, B_k$ of maximum degree $O(\sqrt{n})$ and size $O(\sqrt{n})$, while ensuring that each vertex in these subgraphs receives a random part of its palette with size close to the maximum degree of the subgraph. The palettes for each part are disjoint, which allows us to color all parts in parallel. There will be one leftover subgraph L with maximum degree $\widetilde{O}(\Delta^{3/4})$, as well as sufficiently large remaining palettes for each vertex in this leftover subgraph.

Application in CC: Since each subgraph has $O(n)$ edges, all of $B_1, \ldots, B_k$ can be colored in parallel in $O(1)$ rounds, using Lenzen's routing. The leftover part L is handled by recursion. We show that when $\Delta > \log^{4.1} n$, we are done after $O(1)$ levels of recursion. This graph partitioning is illustrated in Figure 9.1.

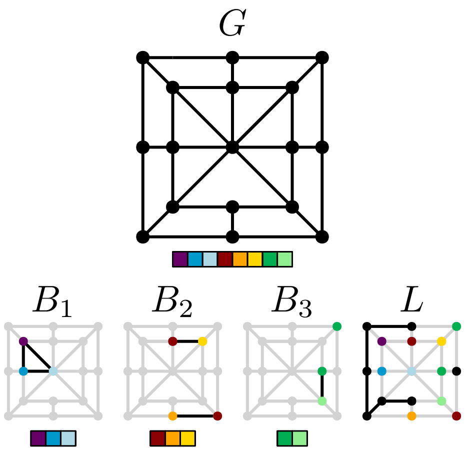


Figure 9.1: The graph partitioning in CC. The graph is partitioned into parts B_1, B_2, B_3 and a leftover graph L. The palettes are randomly distributed among the parts. Each of those parts can be colored immediately, in parallel. The leftover graph with the remaining colors (not leading to a conflict with the colorings of B_1, B_2, and B_3) defines a new list vertex coloring instance, on which we recurse.

Application in Sublinear-Memory MPC: We perform recursive calls not only on L but also on $B_1, \ldots, B_k$. See Figure 9.2 for an illustration. After $O(1/\delta)$ levels of recursion, the maximum degree can be made $O(n^\delta)$, which enables us to run the CLP algorithm on these parts with an exponential speedup using the graph exponentiation method. We note that the previous partitioning approaches [209, 210] are unable to reduce the maximum degree to below $\sqrt{n}$; this is a significant limitation that our partitioning overcomes.

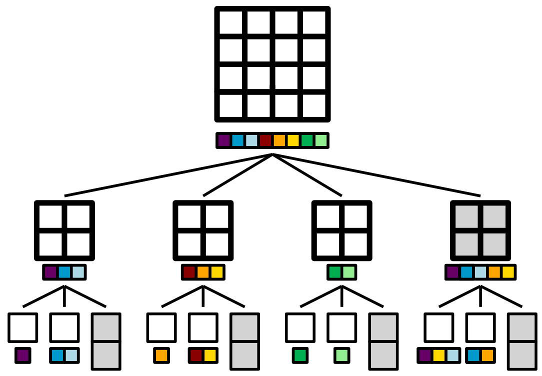


Figure 9.2: The graph partitioning in MPC. The graph is partitioned into several parts and a leftover graph (depicted in grey). The palettes are randomly distributed among the parts. Each of those parts can be colored recursively, in parallel. The leftover graph with the remaining colors (not leading to a conflict with the colorings of the parts) defines a new list-coloring instance, on which we recurse.

Sparsification

For low-degree graphs, the above-explained graph partitioning leads to $\omega(1)$ recursion levels in CC. To get to an $O(1)$ round complexity, additional ideas are required, as we will outline next.

As explained in Section 9.3, we aim to sparsify the algorithm of CLP [58, 59] so that the number of nodes a node has to explore to decide its output is sufficiently small. This allows us to efficiently simulate it in CC with our opportunistic speedup lemma in $O(1)$ rounds.

The shattering phase of the CLP algorithm [58, 59] consists of three parts: (1) initial coloring, (2) dense coloring, and (3) color bidding. Parts (1) and (2) take $O(1)$ rounds[1]; Part (3) takes $O(\log^* \Delta)$ rounds. The post-shattering phase turns out to be easily solvable in CC, as the remaining graph after shattering is sufficiently sparse.

We thus only need to sparsify the color bidding part of the CLP algorithm, which we do as follows. We let each node v sample $\operatorname{poly}\log \Delta$ colors from its palette at the beginning of this procedure, and we show that with probability $1 - \Delta^{-\Omega(1)}$, these colors are enough for v to correctly execute the algorithm. Based on the sampled colors, we can do an $O(1)$-round preprocessing step to let each node v identify a subset $N_*(v) \subseteq N(v)$ of neighbors of size $\Delta_* = \operatorname{poly}\log \Delta$, and v only needs to receive messages from neighbors in $N_*(v)$ in the subsequent steps of the algorithm. For the case $\Delta = \operatorname{poly}\log n$, the parameters $r = O(\log^* \Delta)$ and $\Delta_* = \operatorname{poly}\log \Delta = \operatorname{poly}\log\log n$ satisfy the condition for our opportunistic speedup lemma in Lemma 3.5, so that the color bidding, hence the shattering phase, and thus also the CLP algorithm can be simulated in $O(1)$ rounds in CC.

Notice that the recent work by Assadi et al. [13] on $(\Delta + 1)$ vertex coloring in the linear-memory MPC setting is also based on some form of palette sparsification, as follows. They showed that if each vertex samples $O(\log n)$ colors uniformly at random, then w.h.p. the graph still admits a proper coloring using the sampled colors. Since one only needs to consider the edges $\{u, v\}$ where u and v share a

[1]In preliminary versions of [58], dense coloring takes $O(\log^* \Delta)$ rounds. This time complexity has been improved to $O(1)$ in a revised full version of [58, 59].

sampled color, this effectively reduces the degree to $O(\log^2 n)$. For an MPC algorithm with $\widetilde{\Theta}(n)$ memory per node, the entire sparsified graph can be sent to a single node, and a coloring can be computed offline using any algorithm (e.g., brute-forcing). This sparsification is not applicable in our setting. In particular, in our sparsified CLP algorithm, we need to ensure that the coloring can be computed by a LOCAL algorithm with a small locality volume; this is because the final coloring is constructed distributedly via the opportunistic speedup lemma, as opposed to offline.

Roadmap

In Section 9.2 and Section 9.3, we introduce our two main ingredients graph partitioning and sparsification, respectively. In Section 9.4, we use them to devise our CC algorithm; in Section 9.5, we show how graph partitioning can be used to arrive at our MPC algorithm.

9.1.3 Notation and Preliminaries

Neighborhoods: If there is an edge orientation, $N^{\mathsf{out}}(v)$ refers to the set of out-neighbors of v and use $d^{\mathsf{out}}(v) := |N^{\mathsf{out}}(v)|$. We write $N^k(v) = \{u \in V \mid \mathsf{dist}(u,v) \leq k\}$. We use subscript to indicate the graph or vertex set under consideration, e.g., $N_G(v)$, $N_S(v)$, $N_G^{\mathsf{out}}(v)$, or $N_S^{\mathsf{out}}(v)$, if it is not clear from the context.

Available and Excess Colors: In the course of our algorithms, we sometimes slightly abuse notation to use $\Psi(v)$ not only for the initial color list of v but also to denote the set of *available colors* of v at any point in time. i.e., the subset of $\Psi(v)$ that excludes the colors already taken by v's colored neighbors in $N(v)$. The number of *excess colors* at a vertex is the number of available colors minus the number of uncolored neighbors. Moreover, we assume without loss of generality that each color can be represented using $O(\log n)$ bits. If not, we w.h.p. can hash the colors down to this magnitude.

9.2 Graph Partitioning

In this section, we describe our graph partitioning algorithm, which is the first new technical ingredient in our results. This ingredient on its own leads to our CC result for graphs with $\Delta = \log^{4+\Omega(1)} n$ and to our MPC result.

9.2.1 Overview and Intuitive Discussion

We begin with a high-level overview of our approach and the proof that the $(\Delta + 1)$ list vertex coloring problem can be solved in $O(1)$ rounds in CC when $\Delta = \Omega(\log^{4+\Omega(1)})$. We compare our algorithm's main part with that of the previous CC algorithm of Parter [209] in $O(\log\log\Delta \cdot \log^* \Delta)$ rounds and of Parter and Su [210] in $O(\log^* \Delta)$ rounds as well as the LOCAL algorithm of Chang, Li, and Pettie [58], which we will refer to as CLP.

What's Borrowed from CLP [58]

Most prior works on distributed coloring focus on the LOCAL model. The current state-of-the-art randomized upper bound for the $(\Delta+1)$ list vertex coloring problem is $O(\log^* \Delta)+O(\mathsf{Det}_d(\operatorname{poly}\log n))$ of [58] (which builds upon the techniques of [139]), where $\mathsf{Det}_d(n')$ is the deterministic complexity of $(d + 1)$ list vertex coloring on n'-vertex graphs. In the $(d + 1)$ list vertex coloring problem, each v has a palette of size $d(v) + 1$. Currently $\mathsf{Det}_d(n') = \log^5 \log(n')$ [216, 114], but two years ago, $\mathsf{Det}_d(n') = 2^{O(\sqrt{\log\log n'})}$ [208].

The CLP algorithm follows the shattering technique, where the randomized shattering phase takes $O(\log^* \Delta)$ rounds. After that, the remaining uncolored vertices form connected components of size $\operatorname{poly}\log n$. The post-shattering phase then applies a $(d + 1)$ list vertex coloring deterministic algorithm on each of the remaining connected components to color all these vertices.

The (main part of the) CLP algorithm, the shattering phase, works in $O(\log^* \Delta)$ iterations. We do not delve into the details of these iterations but comment the following: to run each iteration of this algorithm, it suffices if each vertex knows its neighbors and their color palettes as well as the neighbors of its neighbors. The rest of the process of the iteration is some local computation which needs no communication. After these $O(\log^* \Delta)$ iterations, the vertices that remain uncolored induce a much simpler graph, with $O(n)$ edges, and where each component has size poly $\log n$. This remaining graph can be handled much easier. The main interesting point is these $O(\log^* \Delta)$ iterations of the shattering algorithm.

As mentioned before, for each iteration, it suffices if each vertex knows its neighbors and their color palettes as well as the neighbors of its neighbors. If $\Delta = O(\sqrt{n})$, this information can be learned easily in $O(1)$ many CC rounds, using Lenzen's routing. The more interesting case is when $\Delta = \Omega(\sqrt{n})$.

The Algorithm of Parter [209]

For the $\Delta = \Omega(\sqrt{n})$ case, Parter's [209] recursive sparsification procedure randomly breaks the graph of degree Δ into many subgraphs with degree roughly $\Delta' = O(\Delta^2 \log n/n)$ and one special leftover subgraph with degree $O(\sqrt{n})$. This last part will be handled using CLP. The earlier subgraphs receive fixed and disjoint colors, by a deterministic partitioning of the original color. They will be solved first in parallel via recursion. Then the leftover part will be colored using CLP, where all colors already taken by neighbors are removed from the palette of a vertex. For these subgraphs of degree Δ', the sparsification procedure is called recursively. Given that each iteration reduces the degree from Δ to $O(\Delta^2 \log n/n)$, it can take $O(\log\log \Delta) = O(\log\log n)$ iterations for this process to bring down the degree of each part to $\sqrt{n}$, which can be handled using CLP. This explains the $O(\log\log \Delta \log^* \Delta)$ complexity of [209].

Now, let us discuss why this partitioning method cannot set the size of each of the parts to be less than $O(\Delta^2 \log n/n)$. The random partitioning has some deviation and each part should receive a number of colors slightly larger than the expected degree in order to prepare for the deviation in the maximum degree of the part. Since the expected degree in each part is Δ', the expected additive deviation per part will be roughly $O(\sqrt{\Delta' \log n})$. We will have roughly $k = \Delta/\Delta'$ such parts. Thus, we should put roughly $\Delta\sqrt{\log n/\Delta'}$ colors aside. These are taken basically from the share of the leftover part, so we should have $k\sqrt{\Delta' \log n} \leq \sqrt{n}$, which implies that $\Delta' = O(\Delta^2 \log n/n)$. Notice that this limitation on Δ', which itself is rooted in the upper bound on what degrees can be handled by CLP, is the reason of why $O(\log\log\Delta)$ recursions are needed here.

The Algorithm of Parter and Su [210, 209]

In the improvement provided in [210], Parter and Su increase the threshold for the degree that can be handled directly by CLP from $O(\sqrt{n})$ to $O(n^{5/8})$. This is done by a somewhat complex refinement of the internal parts of the CLP algorithm so that it can work without knowing all of the second neighborhood. Particularly, it just needs a large enough sampling of the palette of the neighbors. Given this increased threshold of $O(n^{5/8})$, the recursive partitioning of [209] can be adjusted to break graphs into many subgraphs with degree $\Delta' = O(\Delta^2 \log n/n^{5/4})$, and one leftover subgraph of degree $O(n^{5/8})$ which can be handled using this modified CLP. Hence, $O(1)$ repetitions suffice to bring down the degree to the range that CLP can handle; the round complexity becomes $O(\log^* \Delta)$. Besides the need for delving into the complex details of CLP, one more critical drawback of this algorithm is that still each vertex will need to learn about $O(n)$ bits of information, which can be $O(n^2)$ bits in total.

Our Algorithm

We provide a simple random partitioning that significantly simplifies and extends the above [210, 209]. The main change will be that, besides partitioning the vertices randomly, we also partition the colors randomly (using shared randomness, which can be produced offline and shared easily). In particular, this new procedure partitions the vertices and colors in a way that allows us to easily apply CLP in a black-box manner.

Naïve Divide-and-Conquer: A naïve divide-and-conquer attempt to solve the coloring problem works as follows. Decompose the vertex set V and the color set C randomly into $k = \sqrt{\Delta}$ parts: $V = B_1 \cup \cdots \cup B_k$ and $C = C_1 \cup \cdots \cup C_k$. We hope to be able to color each part B_i by its associated color set C_i, and so we need to make sure that for each $v \in B_i$, its palette size restricted to C_i is higher than its degree in B_i. Consider a vertex $v \in V$. For ease of calculation, let us assume $d(v) = \Delta$ and $|\Psi(v)| = \Delta + 1$. In expectation, its degree in B_i will be $d(v)/\sqrt{\Delta} = \sqrt{\Delta}$, and its palette size restricted to C_i will be $|\Psi(v)|/\sqrt{\Delta} > \sqrt{\Delta}$, but due to the anti-concentration behavior of summation of independent random variables, the gap between the degree and the palette size can be as high as roughly $\Delta^{1/4}$.

Leftover Part: This issue can be solved by introducing a leftover part L in the partition $V = B_1 \cup \cdots \cup B_k \cup L$. Specifically, we include each $v \in V$ to the set L with probability $q = \widetilde{O}(\Delta^{-1/4})$, and then each remaining vertex joins one of $B_1, \ldots, B_k$ uniformly at random. The introduction of L decreases the expected degree of each vertex v in each B_i by roughly $\Delta^{1/4}$, and so with high probability this is smaller than its palette size restricted to C_i. Hence now we are able to color B_i by C_i with high probability.

Indeed, each B_i is of maximum degree $O(\sqrt{\Delta})$ and has size $O(n/\sqrt{\Delta})$, in expectation, and so it is expected that the number of edges within

B_i is $O(n)$. Moreover, each vertex in these subgraphs receives a random part of its palette with size close to the maximum degree of the subgraph. The palettes for each part are disjoint, which allows us to color all parts in parallel. Since each subgraph has $O(n)$ edges, all of $B_1, \ldots, B_k$ can be colored in $O(1)$ rounds, using Lenzen's routing, by sending the subgraph B_i to node i and letting it compute a solution offline.

After finishing the coloring all of $B_1, \ldots, B_k$, we update the set of available colors of each remaining uncolored vertices. In expectation, the subgraph induced by L has maximum degree $\widetilde{O}(\Delta^{3/4})$ and contains $\widetilde{O}(n/\Delta^{1/4})$ vertices. Hence the number of edges in L is $\widetilde{O}(n\sqrt{\Delta})$. We deal with this leftover part L by recursion.

The second iteration—with $n_2 = \widetilde{O}(n/\Delta^{1/4})$ and $\Delta_2 = \widetilde{O}(\Delta^{3/4})$—provides a new leftover set with $\widetilde{O}(n/\Delta^{7/16})$ vertices, $\widetilde{O}(n\Delta^{1/8})$ edges, and maximum degree $\widetilde{O}(\Delta^{9/16})$. Repeating the same calculation again for the third iteration, if $\Delta = \log^{4+\Omega(1)} n$ initially, we find that the subgraph induced by L after the third iteration has a sublinear number of edges, and so we can also deliver this subgraph to one node and compute a proper coloring offline. The graph partitioning technique hence suffices in that case.

We note that the previous partitioning approach [209, 210] is unable to reduce the maximum degree to below $\sqrt{n}$; this is a significant limitation that our partitioning overcomes. We also note that the CC and MPC coloring algorithms of [141, 140] also use the approach of randomly partitioning the palette. However, their algorithms need a palette of size that is much larger than $\Delta + 1$ to ensure that the set of colors associated with each vertex set B_i is higher than its maximum degree. We avoid the use of extra colors by having a sufficiently large leftover part L and recursively applying the partitioning algorithm on L.

9.2.2 Formal Description of Graph Partitioning

Our graph partitioning algorithm will be applied recursively. It is required that the failure probability is at most $1 - n^{-\Omega(1)}$ in all recursive calls, where n is the number of vertices in the original graph. For the sake of presentation, throughout this section, we use $G = (V, E)$ to refer to the current subgraph under consideration, not the original input graph. Accordingly, graph G does not always have exactly n vertices, but only at most n vertices.

We first give a formal description of our graph partitioning, and then show that it satisfies some nice properties.

Graph Partitioning Algorithm

We consider a graph partitioning algorithm parameterized by two constants γ and ν satisfying $\gamma \geq 2$ and $\nu = \frac{1}{2} + \frac{2}{3\gamma+2}$. Consider a graph $G = (V, E)$ with maximum degree Δ. Recall that in this section G is assumed to be a subgraph of the n-vertex original graph, and so $n \geq |V|$. Each vertex $v \in V$ has a palette $\Psi(v)$ of size $|\Psi(v)| \geq \max\{d_G(v), \Delta'\} + 1$, where $\Delta' = \Delta - \Delta^{\nu}$. Denote $G[S]$ as the subgraph induced by the vertices $S \subseteq V$. For each vertex $v \in V$, denote $d_S(v)$ as $|N(v) \cap S|$. The algorithm is as follows, where we set $k := \sqrt{\Delta}$.

Vertices: The partition $V = B_1 \cup \cdots \cup B_k \cup L$ is defined by the following procedure. Include each $v \in V$ to the set L with probability $q = \Theta\left(\sqrt{\log n}/\Delta^{1/4}\right)$. Each remaining vertex joins one of $B_1, \ldots, B_k$ uniformly at random. Note that $\mathsf{Pr}[v \in B_i] = p(1-q)$ for $p = 1/k = 1/\sqrt{\Delta}$.

Palettes: Let $C = \bigcup_{v \in V} \Psi(v)$ denote the set of all colors. The partition $C = C_1 \cup \cdots \cup C_k$ is defined by having each color $c \in C$ join one of $C_1, \ldots, C_k$ uniformly at random. Note that $\mathsf{Pr}[c \in C_i] = p = 1/k$.

Properties of Graph Partitioning

Lemma 9.3. *Suppose* $|\Psi(v)| \geq \max\{d_G(v), \Delta'\} + 1$ *with* $\Delta' := \Delta - \Delta^\nu$, *and* $|V| > \Delta = \omega(\log^\gamma n)$, *where* γ *and* ν *are two constants satisfying* $\gamma \geq 2$ *and* $\nu := \frac{1}{2} + \frac{2}{3\gamma+2}$. *The two partitions* $V = B_1 \cup \cdots \cup B_k \cup L$ *and* $C = \bigcup_{v \in V} \Psi(v) = C_1 \cup \cdots \cup C_k$ *satisfy the following four properties with probability* $1 - n^{-\Omega(1)}$.

(i) Size of Each Part: *We have*

$$|E(G[B_i])| = O(|V|)$$

for each $i \in [k]$ *and*

$$|L| = O(q|V|) = O\Big(\sqrt{\log n}/\Delta^{1/4}\Big)|V|.$$

(ii) Available Colors in B_i***:*** *For each* $i \in \{1, \ldots, k\}$ *and* $v \in B_i$, *for the number* $g_i(v)$ *of available colors for* v *in the subgraph* B_i, *it holds*

$$g_i(v) := |\Psi(v) \cap C_i| \geq \max\{d_{B_i}(v), \Delta_i - \Delta_i^\nu\} + 1,$$

where $\Delta_i := \max_{v \in B_i} d_{B_i}(v)$.

(iii) Available Colors in L***:*** *For each* $v \in L$, *we have*

$$g_L(v) := |\Psi(v)| - (d_G(v) - d_L(v)) \geq \max\{d_L(v), \Delta_L - \Delta_L^\nu\} + 1,$$

for each $v \in L$, *where* $\Delta_L := \max_{v \in L} d_L(v)$. *Note that* $g_L(v)$ *represents a lower bound on the number of available colors in* v after *all of* $B_1, \ldots, B_k$ *have been colored.*

(iv) Remaining Degrees: *The maximum degree of* B_i *is*

$$\max_{v \in B_i} d_{B_i}(v) \leq \Delta_i = O\Big(\sqrt{\Delta}\Big)$$

and the maximum degree of L is

$$\max_{v\in L} d_L(v) \le \Delta_L = O(q\Delta) = O\left(\frac{\sqrt{\log n}}{\Delta^{1/4}}\right) \cdot \Delta.$$

Moreover, for each vertex v, we have

$$d_{B_i}(v) \le \max\left\{O(\log n), O\left(\frac{1}{\sqrt{\Delta}}\right)\cdot d(v)\right\} \text{ if } v \in B_i,$$

and

$$d_L(v) \le \max\left\{O(\log n), O(q)\cdot d(v)\right\} \text{ if } v \in L.$$

Proof. We split the proof in four parts, according to the four properties in the lemma statement.

Property (i): We first show the bound on the B_i, that is, that $|E(G[B_i])| = O(|V|)$, for each $i \in [k]$, with probability $1 - n^{-\Omega(1)}$. To have $|E(G[B_i])| = O(|V|)$, it suffices to have $d_{B_i}(v) = O(p\Delta)$ for each v, and $|B_i| = O(p|V|)$, since $p = 1/\sqrt{\Delta}$. Note that we already have $\mathbb{E}[d_{B_i}(v)] \le (1-q)p\Delta < p\Delta$ and $\mathbb{E}[|B_i|] = (1-q)p|V| < p|V|$, so we only need to show that these parameters concentrate at their expected values with high probability. A Chernoff bound yields

$$\begin{aligned}\Pr[d_{B_i}(v) \le (1+\varepsilon_1)(1-q)p\Delta] &= 1 - e^{-\Omega\left(\varepsilon_1^2(1-q)p\Delta\right)} \\ &= 1 - n^{-\Omega(1)},\end{aligned} \tag{9.1}$$

for

$$\varepsilon_1 = \Theta\left(\sqrt{\frac{\log n}{(1-q)p\Delta}}\right) = \Theta\left(\sqrt{\frac{\log n}{p\Delta}}\right),$$

and

$$\begin{aligned}\Pr\left[|B_i| \le (1+\varepsilon_2)(1-q)p|V|\right] &= 1 - e^{-\Omega\left(\varepsilon_2^2(1-q)p|V|\right)} \\ &= 1 - n^{-\Omega(1)},\end{aligned}$$

for

$$\varepsilon_2 = \Theta\left(\sqrt{\frac{\log n}{(1-q)p|V|}}\right) = \Theta\left(\sqrt{\frac{\log n}{p|V|}}\right).$$

Note that we have $\varepsilon_1 < 1$ and $\varepsilon_2 < 1$ in these calculations. In particular, the inequality $\varepsilon_1 < 1$ holds because of the assumption $\Delta = \omega(\log^\gamma n) = \omega(\log^2 n)$.

Next, we show the analogous results for L, i.e., that both $|L|/|V|$ and Δ_L/Δ have size $O(q) = O\left(\sqrt{\log n}/\Delta^{1/4}\right)$ with probability $1 - n^{-\Omega(1)}$. Similarly to before, we already have $\mathbb{E}[d_L(v)] \leq q\Delta$ and $\mathbb{E}[|L|] = q|V|$. Since $q = O\left(\frac{\sqrt{\log n}}{\Delta^{1/4}}\right)$, we only need to show that these parameters concentrate around their expected values with high probability, using a Chernoff bound. Indeed, we get

$$\begin{aligned}\mathsf{Pr}[d_L(v) \leq (1+\varepsilon_3)q\Delta] &= 1 - e^{-\Omega(\varepsilon_3^2 q\Delta)} \\ &= 1 - n^{-\Omega(1)},\end{aligned} \tag{9.2}$$

for $\varepsilon_3 = \Theta\left(\sqrt{\log n/(q\Delta)}\right)$, and

$$\mathsf{Pr}[|L| \leq (1+\varepsilon_4)q|V|] = 1 - e^{-\Omega\left(\varepsilon_4^2 q|V|\right)} = 1 - n^{-\Omega(1)},$$

for $\varepsilon_4 = \Theta\left(\sqrt{\log n/(q|V|)}\right)$. Again, observe that we have $\varepsilon_3 < 1$ and $\varepsilon_4 < 1$. In particular, $\varepsilon_3 < 1$ because $\Delta = \omega(\log^\gamma n) = \omega(\log^2 n)$.

Property (ii): Now we analyze the number of available color for each set B_i. We will show that with probability $1 - n^{-\Omega(1)}$, we have $|\Psi(v) \cap C_i| \geq \Delta_i + 1$ for each B_i and each $v \in B_i$, which implies $|\Psi(v) \cap C_i| \geq d_{B_i}(v) + 1$ and $|\Psi(v) \cap C_i| \geq \Delta_i - \Delta_i^\nu + 1$.

Observing that $q = \Theta\left(\frac{\sqrt{\log n}}{\Delta^{1/4}}\right) \geq \Delta^{-(1-\nu)} \geq \Delta^{-1/4} \gg \Delta^{-(1-\nu)}$, since $\gamma \geq 2$ and $\nu = \frac{1}{2} + \frac{2}{3\gamma+2}$ imply that $\nu \in (1/2, 3/4]$, and selecting $q \geq 3\varepsilon_1 = \Theta\left(\sqrt{\log n}/\Delta^{1/4}\right)$, we get

$$(1-\varepsilon_1)p\Delta' = (1-\varepsilon_1)\left(1-\frac{1}{\Delta^{(1-\nu)}}\right)p\Delta \geq (1+\varepsilon_1)(1-q)p\Delta + 1.$$

We already know that $\Delta_i \leq (1+\varepsilon_1)(1-q)p\Delta$ with probability $1-n^{-\Omega(1)}$. In order to have $|\Psi(v)\cap C_i| \geq \Delta_i + 1$, we only need to show that $|\Psi(v)\cap C_i| \geq (1-\varepsilon_1)p\Delta'$ with probability $1-n^{-\Omega(1)}$. For the expected value, we know that $\mathbb{E}[|\Psi(v)\cap C_i|] = p|\Psi(v)| \geq p\Delta'$. A Chernoff bound gives

$$\mathsf{Pr}[|\Psi(v)\cap C_i| \geq (1-\varepsilon_1)p\Delta'] = 1 - e^{-\Omega(\varepsilon_1^2 p\Delta')} = 1 - n^{-\Omega(1)},$$

which concludes the proof of (ii).

Property (iii): Next, we consider the number of available colors in L. It is straightforward to see that $g_L(v) \geq d_L(v)+1$, since

$$g_L(v) = (|\Psi(v)| - d_G(v)) + d_L(v) \geq 1 + d_L(v).$$

Thus, we only need to show that $g_L(v) \geq \Delta_L - \Delta_L^{\nu} + 1$.

We have

$$\begin{aligned}\mathbb{E}[g_L] &= (|\Psi(v)| - d_G(v)) + \mathbb{E}[d_L(v)] = (|\Psi(v)| - d_G(v)) + qd_G(v)\\ &= |\Psi(v)| - (1-q)d_G(v) \geq q\Delta',\end{aligned}$$

where we use the assumption that $d_G(v) \leq |\Psi(v)| - 1$ and $\Delta' \leq |\Psi(v)| - 1$ in the last inequality.

Next, we prove that $g_L(v) \geq (1-\varepsilon_3)q\Delta'$ with probability $1-n^{-\Omega(1)}$. A Chernoff bound yields

$$\begin{aligned}\mathsf{Pr}[g_L(v) \geq (1-\varepsilon_3)q\Delta'] &= \mathsf{Pr}[d_L(v) \geq qd_G(v) - \varepsilon_3 q\Delta']\\ &= 1 - e^{-\Omega(\varepsilon_3^2 q\Delta')} = 1 - n^{-\Omega(1)},\end{aligned}$$

recalling that $\varepsilon_3 = \Theta\left(\sqrt{\frac{\log n}{q\Delta}}\right) = \Theta\left(\sqrt{\frac{\log n}{q\Delta'}}\right)$, and that $\varepsilon_3 < 1$. Using the above concentration bound, we infer that

$$\begin{aligned} g_L(v) &\geq (1-\varepsilon_3)q\Delta' \\ &= q\Delta' - O\left(\sqrt{q\Delta' \log n}\right) \\ &= q\Delta - q\Delta^\nu - O\left(\sqrt{q\Delta \log n}\right) \end{aligned}$$

with probability $1 - n^{-\Omega(1)}$. Combining this with

$$\Delta_L \leq (1+\varepsilon_3)q\Delta = q\Delta + O(\sqrt{q\Delta \log n}),$$

we obtain

$$g_L(v) = \Delta_L - q\Delta^\nu - O(\sqrt{q\Delta \log n}).$$

Note that $q\Delta^\nu + O(\sqrt{q\Delta \log n}) = o\left((q\Delta)^\nu\right) = o\left(\Delta_L^\nu\right)$, where we use

$$\sqrt{q\Delta \log n} \ll (q\Delta)^{\frac{1}{2}}(q\Delta)^{\frac{1}{2}\left(\frac{3}{4}\gamma+\frac{1}{2}\right)^{-1}} = (q\Delta)^\nu,$$

since $q\Delta = \Theta(\Delta^{\frac{3}{4}} \log^{\frac{1}{2}} n) = \omega(\log^{\frac{3}{4}\gamma+\frac{1}{2}} n)$. We thus finally obtain $g_L(v) \geq \Delta_L - \Delta_L^\nu + 1$.

Property (iv): The upper bounds on Δ_i and Δ_L follow immediately from (9.1) and (9.2). The claimed bounds for $d_{B_i}(v)$ and $d_{B_i}(v)$ can be derived using a straightforward application of Chernoff and union bound. □

9.3 Sparsification of Local Coloring

We next present our second novel technical ingredient: a sparsification for the LOCAL $(\Delta+1)$ list vertex coloring algorithm CLP of Chang, Li, and Pettie [58]. As a consequence, combined with the opportunistic speedup lemma from Lemma 3.5, this sparsification gives us an $O(1)$-round CC algorithm solving $(\Delta+1)$ list vertex coloring for the case $\Delta = \text{poly}\log n$.

9.3.1 Overview and Outline

The algorithm of [58] is based on the graph shattering framework. The $O(\log^* \Delta)$-round shattering phase leaves connected components of size $\text{poly}\log n$ with $O(n)$ edges among uncolored vertices. On these remaining components, in the post-shattering phase of their LOCAL algorithm, they run a deterministic $(d{+}1)$ list vertex coloring algorithm. Since the remaining components have only $O(n)$ edges, in CC, the post-shattering can be directly solved using Lenzen's routing. Thus, in order to simulate the CLP algorithm in $O(1)$ rounds in CC, we only need to focus on the shattering phase.

The shattering phase of the CLP algorithm consists of three parts, an $O(1)$-round *initial coloring*, an $O(1)$-round *dense coloring*, and an $O(\log^* \Delta)$-round *color bidding* part. In Section 9.3.2 in Lemma 9.4, we present a summary (and minor adaptations) of the first two steps as a black-box algorithm. This black-box algorithm runs in 2 rounds in the LOCAL model. Since $\Delta = \text{poly}\log n$, the 2-hop neighborhoods have size $\Delta^2 = \text{poly}\log n$, and hence can be easily gathered in $O(1)$ CC rounds using Lenzen's routing.

The black-box algorithm partitions the vertices into the sets V_{good}, V_{bad}, and R. The latter two sets will have size $\text{poly}\log n$ and $O(n)$, respectively, so that they will admit an easy solution with Lenzen's routing algorithm as well. The only remaining, and most interesting part, is thus V_{good}. In the CLP algorithm, this set is taken care of in the last part, the color bidding. It will thus be enough to sparsify the color bidding part of the CLP algorithm.

In Section 9.3.3, we present a sparsified version of the color bidding where every vertex needs to receive information from only $\text{poly}\log \Delta = \text{poly}\log\log n$ of its neighbors to decide its output. In Section 9.3.4, we prove some useful properties of our sparsified color bidding algorithm. In Section 9.3.5, we show how to implement this color bidding algorithm in the LOCAL model.

9.3.2 Black-Box Partial Coloring

We will use the first two parts of the CLP algorithm of [58] as a black box. We will later see in (the proof of) Lemma 9.13, that this black-box algorithm can be easily simulated in CC in $O(1)$ rounds. Indeed, for the black-box algorithm it is enough for each node to know the 2-hop neighborhood of its vertex. If $\Delta = O(\sqrt{n})$, and hence 2-hop neighborhoods have size $\Delta^2 = O(n)$, this can be done in $O(1)$ rounds using Lenzen's routing. Moreover, observe that vertices in V_{bad} and R form components of size $\operatorname{poly}\log n$ and $O(n)$ respectively, hence they can be solved easily using Lenzen's routing algorithm as well. The only remaining, and most interesting part, is thus V_{good}.

Lemma 9.4 (Chang, Li, and Pettie [58, 59])**.** *There is an $O(1)$-round LOCAL algorithm with messages of $O(\Delta^2 \log n)$ bits that colors a subset of vertices so that the remaining uncolored vertices are partitioned into subsets V_{good}, V_{bad}, and R as follows.*

Good Vertices: *The edges within V_{good} are oriented as a directed acyclic graph, and each vertex $v \in V_{\mathsf{good}}$ is associated with a parameter $p_v \leq |\Psi(v)| - d(v)$ satisfying the conditions $p^\star = \min_{v\in V} p_v \geq \Delta/\log\Delta$ and $\sum_{u\in N^{\mathsf{out}}(v)} 1/p_u \leq 1/C$, where $C > 0$ can be any specified constant. Here $\Psi(v)$ is the set of available colors at v, i.e., the colors in the palette of v that have not been taken by v's neighbors and $d(v)$ refers to the number of uncolored neighbors of v in V_{good}. Intuitively, $p_v \leq |\Psi(v)| - d(v)$ is a lower bound on the number of excess colors at v.*

Bad Vertices: *The probability that a vertex $v \in V$ joins V_{bad} is $1 - \Delta^{-\Omega(1)}$. In particular, using Lemma 3.3, they w.h.p. form connected components of size $\Delta^{O(1)} \cdot O(\log n)$ and the number of edges within the bad vertices is $O(n)$.*

Remaining Vertices: *The subgraph induced by R has maximum degree $O(1)$.*

Proof. We briefly review the algorithm of [58, 59] and show how to obtain Lemma 9.4 from [58, 59]. The algorithm uses a sparsity sequence defined by $\varepsilon_1 = \Delta^{-1/10}$, $\varepsilon_i = \sqrt{\varepsilon_{i-1}}$ for $i > 1$, and $\ell = \Theta(\log\log\Delta)$ is the largest index such that $\varepsilon_\ell K \leq 1$ for some sufficiently large constant K.

Initial Coloring: In an initial coloring phase, the algorithm performs an $O(1)$-round procedure to color a fraction of the vertex set V, leaving $V^\star$ as the set of remaining uncolored vertices. The set $V^\star$ is decomposed into $\ell+1$ subsets $(V_1, \ldots, V_\ell, V_{\mathsf{sparse}})$ according to their local sparsity.

Dense Coloring: In the dense coloring phase, The algorithm then applies another $O(1)$-round procedure to color a fraction of vertices in $V_1 \cup \cdots \cup V_\ell$. The remaining uncolored vertices in $V^\star$ after the first two phases are partitioned into three subsets: U, R, and V_{bad}.[2] The set R induces a constant-degree graph. The set V_{bad} satisfies the property that each vertex is added to V_{bad} with probability $\Delta^{-\Omega(1)}$. The vertices in U satisfy the following properties.

Excess Colors: We have $V_1 \cap U = \varnothing$. Each $v \in V_i \cap U$, with $i > 1$, has $\Omega(\varepsilon_{i-1}^2\Delta)$ excess colors. Each $v \in V_{\mathsf{sparse}} \cap U$ has $\Omega(\varepsilon_\ell^2\Delta) = \Omega(\Delta)$ excess colors. The number of excess colors at a vertex v is defined by the number of available colors of v minus the number of uncolored neighbors of v.

Number of Neighbors: For each $v \in U$, and for each $i \in [2, \ell]$, the number of uncolored neighbors of v in $V_i \cap U$ is $O(\varepsilon_i^5\Delta) = O(\varepsilon_{i-1}^{2.5}\Delta)$. The number of uncolored neighbors of v in $V_{\mathsf{sparse}} \cap U$ is of course at most $\Delta = O(\varepsilon_\ell^{2.5}\Delta)$, since ε_ℓ is a constant.

At this moment, the two sets V_{bad} and R satisfy the required condition specified in Lemma 9.4. In what follows, we focus on U.

[2] The algorithm in [58] for coloring layer-1 large blocks has two alternatives. Here we always use the one that puts the remaining uncolored vertices in one of R or V_{bad}, where each vertex is added to V_{bad} with probability $\Delta^{-\Omega(1)}$.

Orientation: We orient the graph induced by the uncolored vertices in U as follows. For any edge $\{u, v\}$, we orient it as (u, v) if one of the following is true: (a) $u \in V_{\mathsf{sparse}}$ but $v \notin V_{\mathsf{sparse}}$, (b) $u \in V_i$ and $v \in V_j$ with $i > j$, (c) u and v are within the same part in the partition $V^\star = V_1 \cup \ldots V_\ell \cup V_{\mathsf{sparse}}$ and $\mathsf{ID}(v) < \mathsf{ID}(u)$. This results in a directed acyclic graph. We write $N^{\mathsf{out}}(v)$ to denote the set of out-neighbors of v in this graph.

Lower Bound on Excess Colors: In view of the above, there exist universal constants $\eta > 0$ and $C > 0$ such that the following is true. For each $i \in [2, \ell]$ and each uncolored vertex $v \in V_i \setminus V_{\mathsf{bad}}$, we set $p_v = \eta\varepsilon_{i-1}^2\Delta$. For each $v \in V_{\mathsf{sparse}} \setminus V_{\mathsf{bad}}$, we set $p_v = \eta\varepsilon_\ell^2\Delta$. By selecting a sufficiently small η, the number p_v is always a lower bound on the number of excess colors at v.

Recall that to color the graph quickly we need the number of excess colors to be sufficiently large with respect to the outdegree. If $v \in V_i \cap U$ with $i \geq 2$, it satisfies $|N^{\mathsf{out}}(v)| = \sum_{j=2}^{i} O(\varepsilon_{j-1}^{2.5}\Delta) = O(\varepsilon_{i-1}^{2.5}\Delta)$. In this case, $p_v/|N^{\mathsf{out}}(v)| = \Omega(\varepsilon_{i-1}^{-0.5})$. If $v \in V_{\mathsf{sparse}} \cap U$, then of course $|N^{\mathsf{out}}(v)| \leq \Delta = O(\varepsilon_\ell^2\Delta)$, since ε_ℓ is a constant. In this case, $p_v/|N^{\mathsf{out}}(v)| = \Omega(\varepsilon_\ell^{-0.5})$.

However, due to the high variation on the palette size in our setting, $p_v/|N^{\mathsf{out}}(v)|$ is not a good measurement for the gap between the number of excess colors and outdegree at v. The inverse of the expression $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u$ turns out to be a better measurement, as it takes into account the number of excess colors in each out-neighbor.

There is a constant $C > 0$ such that for each uncolored vertex $v \in V^\star \setminus (V_{\mathsf{bad}} \cup R)$, we have $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u \leq 1/C$. The calculation is as follows. If $v \in V_i \cap U$ for $i > 1$, then

$$\sum_{u \in N^{\mathsf{out}}(v)} \frac{1}{p_u} = \sum_{j=2}^{i} O\left(\frac{\varepsilon_{j-1}^{2.5}\Delta}{\varepsilon_{j-1}^{2}\Delta}\right) = \sum_{j=2}^{i} O\big(\varepsilon_{j-1}^{0.5}\big) = O\big(\varepsilon_{i-1}^{0.5}\big) < \frac{1}{C}.$$

If $v \in V_{\mathsf{sparse}} \cap U$, then

$$\sum_{u \in N^{\mathsf{out}}(v)} \frac{1}{p_u} = \sum_{j=2}^{\ell+1} O\left(\frac{\varepsilon_{j-1}^{2.5}\Delta}{\varepsilon_{j-1}^{2}\Delta}\right) = \sum_{j=2}^{\ell+1} O\big(\varepsilon_{j-1}^{0.5}\big) = O\big(\varepsilon_{\ell}^{0.5}\big) < \frac{1}{C}.$$

For a specific example, if v is an uncolored vertex in $V_2 \setminus V_{\mathsf{bad}}$, then $p_v = \eta\varepsilon_1^2\Delta = \eta\Delta^{0.8}$ is the lower bound on the number of excess colors at v, and v has outdegree $|N^{\mathsf{out}}(v)| = O(\varepsilon_1^{2.5}\Delta) = O(\Delta^{0.75})$, and we have $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u = O(\varepsilon_1^{0.5}) = O(\Delta^{-0.05}) < 1/C$. Intuitively, this means that the gap between the number of excess colors and the outdegree at v is $\Omega(\Delta^{0.05})$.

Summary: The graph induced by U satisfies the following conditions. Each vertex v is associated with a parameter $p_v = \eta\varepsilon_j^2\Delta$ (for some $j \in [1, \ell]$) so that $p_v = \Omega(\varepsilon_j^2\Delta)$. Moreover, v has $O(\varepsilon_j^{2.5}\Delta)$ out-neighbors. In particular, we have $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u = O(\varepsilon_j^{0.5}) < 1/C$, where $C > 0$ is a universal constant. The current p_v-values for vertices in U almost satisfy the required condition for V_{good}, but only almost.

(1) Let $p^\star$ be the minimum p_v-value among all uncolored vertices $v \in V^\star$. Currently we only have $p^\star \geq \eta\varepsilon_1^2\Delta = \eta\Delta^{0.8}$, but it is required that $p^\star \geq \Delta/\log\Delta$.

(2) Currently we have $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u \leq 1/C$ for some universal constant C, but it is required that $C > 0$ can be any given constant.

For the rest of the proof, we show that there are $O(1)$-round algorithms that are able to improve these conditions, by coloring a fraction of vertices in U and putting some vertices from U to V_{bad}.

We first consider improving the lower bound on $p^\star$. This is done by letting all vertices whose p_v-value is too small (i.e., less than $\Delta/\log\Delta$) jointly run the following algorithm.

Lemma 9.5 (Chang, Li, and Pettie [58, 59]). *Consider a directed acyclic graph, where vertex v is associated with a parameter $p_v \le |\Psi(v)| - d(v)$. We write $p^\star = \min_{v \in V} p_v$ and let $d^\star$ be the maximum outdegree of the graph. Suppose that there is a number $C = \Omega(1)$ such that all vertices v satisfy $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u \le 1/C$. Then there is an $O(\log^*(p^\star) - \log^*(C))$-round* LOCAL *algorithm achieving the following. Each vertex v remains uncolored with probability at most $e^{-\Omega(\sqrt{p^\star})} + d^\star e^{-\Omega(p^\star)}$. This is true even if the random bits generated outside a constant radius around v are determined adversarially.*

As $p_v = \eta\varepsilon_j^2\Delta$, we have $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u = O(\varepsilon_j^{0.5}) = O(p_v^{1/4}) = O(\log^{-1/4}\Delta)$, and thus can use $C = \Omega\left(\log^{1/4}\Delta\right)$ in Lemma 9.5. All remaining uncolored vertices join V_{bad}.

We show how to increase C to any given constant in $O(1)$ rounds. We apply the following algorithm using the current $p^\star$ and C.

Lemma 9.6 (Chang, Li, and Pettie [58, 59]). *There is a 1-round* LOCAL *coloring algorithm that satisfies*

$$\mathit{Pr}\left[d \ge \frac{1+\lambda}{e^{\frac{C}{6}}C}\right] \le e^{-\frac{2\tau^2 p^\star e^{-\frac{C}{3}}}{C}} + d^\star e^{-\Omega(p^\star)}.$$

for any v and τ, and d being the summation of $1/p_u$ over all vertices u in $N^{\mathsf{out}}(v)$ that remain uncolored after the algorithm.

After that, we can set the new C-value to be $C' = Ce^{C/6}/(1+\tau)$. Each vertex v that does not meet the condition that the sum of $1/p_u$ over all remaining uncolored vertices u in $N^{\mathsf{out}}(v)$ is at most $1/C' = (1+\tau)/(e^{C/6}C)$ is put into V_{bad}. If τ is chosen as a small enough constant, we have $C' > C$. After a constant number of iterations, we can increase the C-value to any constant we like. Now, all conditions in Lemma 9.4 are met for the three sets R, V_{bad}, and $V_{\mathsf{good}} \leftarrow U$. $\square$

9.3.3 Sparsified Color Bidding

In view of Lemma 9.4, we focus on the subgraph induced by V_{good}, and denote it as $G_0 = (V_0, E_0)$. The graph G_0 is a directed acyclic graph. The set of available colors for v is denoted as $\Psi_0(v)$. Our goal is to find a proper coloring of G_0. An important property of G_0 is that each vertex $v \in V$ is associated with a parameter $p_v \leq |\Psi_0(v)| - d_{G_0}(v)$ such that $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u \leq 1/C_0$, where C_0 can be any specified large constant. Intuitively, p_v gives the lower bound of the number of *excess colors* at vertex v. It is guaranteed that $p^\star = \min_{v \in V_0} p_v \geq \Delta / \log \Delta$. All vertices in V_0 initially know the parameters C_0 and $p^\star$.

Review of the Color Bidding Algorithm

The above conditions might look a bit strange, but it allows us to find a proper coloring in $O(\log^* \Delta)$ rounds in the LOCAL model in $O(\log^* \Delta)$ iterations of the $\mathsf{ColorBidding}$ procedure by [58], as follows.

1. Each $c \in \Psi(v)$ is added to S_v with probability $\frac{C}{2|\Psi(v)|}$.
2. If there exists a color $c^\star \in S_v$ that is not selected by any vertex in $N^{\mathsf{out}}(v)$, then v colors itself with $c^\star$.

We give a very high-level explanation about how this works. For the first iteration we use $C = C_0$. Intuitively, for each color $c \in S_v$, the probability that c is selected by an out-neighbor of v is

$$\sum_{u \in N^{\mathsf{out}}(v)} \frac{C}{2|\Psi(u)|} \leq \sum_{u \in N^{\mathsf{out}}(v)} \frac{C}{2p_u} \leq 1/2,$$

where we use the inequality $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u \leq 1/C_0$ that is guaranteed by Lemma 9.4. The probability that v fails to color itself is roughly $2^{-|S_v|}$, which is exponentially small in C_0, as $\mathbb{E}[|S_v|] = C_0/2$. Thus, for the next iteration we may use a parameter C that is exponentially small in C_0, and so after $O(\log^* \Delta)$ iterations, we are done.

Parameters

Let $\zeta > 0$ be a constant to be determined. Let $p^\star \in [\Delta/\log\Delta, \Delta]$ be the parameter specified in the conditions for Lemma 9.4. The C-parameters $C_0, \ldots, C_{k-1}$ used in the algorithms are defined as follows. For the base case, C_0 is the parameter C specified in the conditions for Lemma 9.4. Given that C_i has been defined, we set

$$C_{i+1} = 2\left\lceil \frac{\left(\min\left\{\frac{1}{2}e^{\frac{C_i}{6}}C_i, \log^\zeta p^\star\right\}\right)}{2}\right\rceil - 2.$$

In other words, C_{i+1} is the result of rounding the minimum of $\frac{1}{2}e^{\frac{C_i}{6}}C_i$ and $\log^\zeta p^\star$ down to the nearest even number. We choose the number k of iterations as the smallest index such that $C_{k-1} = 2\left\lceil \log^\zeta \frac{p^\star}{2}\right\rceil - 2$. It is clear that $k = O(\log^* \Delta)$, as $p^\star \leq \Delta + 1$.

We will use this sequence $C_0, \ldots, C_{k-1}$ in our sparsified color bidding algorithm. This sequence is slightly different from the one used in [58]. The last number in the sequence used in [58] is set to be $\sqrt{p^\star}$, but here we set it to be $\operatorname{poly}\log p^\star$. Having a larger C-parameter leads to a smaller failure probability, but it comes at a cost that we have to sample more colors. This means that each vertex needs to communicate with more neighbors to check for conflict.

Overview of the Proof

We first review the analysis of ColorBidding in [58], and then we discuss how we sparsify this algorithm. The proof in [58] maintains an invariant for each vertex v that is uncolored at the beginning of each iteration i which says

$$\mathcal{I}_i(v)\colon \qquad \sum_{u \in N_G^{\mathsf{out}}(v)} 1/p_u \leq 1/C_i.$$

We will use the *same* p_u in every iteration because the number of excess colors of a vertex never decreases. Here, G refers the current graph under consideration, i.e., it excludes all vertices that have been colored or removed in previous iterations. We use G_0 to refer to the original graph (i.e., the one induced by V_{good}).

By Lemma 9.4, this invariant is met for $i = 0$. The vertices u not satisfying the invariant $\mathcal{I}_i(v)$ are considered *bad*, and are removed from consideration. The analysis of [58] shows that

(1) Suppose all vertices u in G at the beginning of iteration i satisfy $\mathcal{I}_i(u)$. Then at the end of this iteration, for each vertex u, with probability $1 - \Delta^{-\Omega(1)}$, either u has been successfully colored or $\mathcal{I}_{i+1}(u)$ is satisfied.

(2) For the last iteration, given that all vertices u in G satisfy $\mathcal{I}_{k-1}(u)$, then v is successfully colored at iteration k with probability $1 - \Delta^{-\Omega(1)}$.

By the shattering lemma in Lemma 3.3, all vertices that remain uncolored at the end of the algorithm induce a subgraph with $O(n)$ edges. In particular, in CC we are able to color them in $O(1)$ additional rounds using Lenzen's routing.

To sparsify the algorithm, our strategy is to let each node sample the colors needed in all iterations at the beginning of the algorithm. It is straightforward to see that each node only needs to use $\operatorname{poly}\log\Delta$ colors throughout the algorithm, with probability $1 - \Delta^{-\Omega(1)}$. After sampling the colors, if u finds that $v \in N^{\mathsf{out}}(u)$ does not share any sampled color, then there is no need for u to communicate with v. This effectively reduces the maximum degree to $\Delta' = \operatorname{poly}\log\Delta$. If $\Delta = \operatorname{poly}\log n$, then $\Delta' = \operatorname{poly}\log\log n$, which is enough to apply the opportunistic speedup lemma in Lemma 3.5.

Verifying Invariant: There is one issue needed to overcome: verifying whether $\mathcal{I}_i(u)$ is met has to be done on the original graph G, as we have to go over all vertices $v \in N_G^{\mathsf{out}}(u)$, regardless of whether u and v have shared sampled colors. One way to deal with this issue is to simply not remove the vertices u violating $\mathcal{I}_i(u)$, but if we do it this way, then when we calculate the failure probability of a vertex v, we have to apply a union bound over all vertices u within radius $r = O(\log^* \Delta)$ to v that u does not violate the invariant for each iteration. Due to this union bound, we can only upper bound the size of the connected components of bad vertices by $\Delta^{O(\log^* \Delta)} \cdot O(\log n)$. To resolve this issue, we observe that the invariant $\mathcal{I}_i(u)$ might be too strong for our purpose, since intuitively if $v \in N^{\mathsf{out}}(u)$ does not share any sampled colors with u, then v should not be able to affect u throughout the algorithm.

Alternative Invariant and Rich Vertices: We here consider an alternative weaker invariant $\mathcal{I}'_i(u)$ that can be checked in the sparsified graph. More precisely, in each iteration, each vertex v will do a two-stage sampling to obtain two color sets $S_v \subseteq T_v \subseteq \Psi(v)$. The set S_v has size $C/2$, and the set T_v has size $\log^\zeta \Delta$, where $\zeta > 0$ is a constant to be determined. The alternative invariant is defined as

$$\mathcal{I}'_i(v)\colon \qquad \left| T_v \setminus \bigcup_{u \in N_G^{\mathsf{out}}(v)} S_u \right| \geq \frac{|T_v|}{3}.$$

This invariant can be checked by having v communicate only with neighbors that share a sampled color with v. Intuitively, if $\mathcal{I}_i(v)$ holds, then $\mathcal{I}'_i(v)$ holds with probability $1 - \Delta^{-\Omega(1)}$. It is also straightforward to see that $\mathcal{I}'_i(v)$ implies that v has a high probability of successfully coloring itself in this iteration, as S_v is a uniformly random subset of T_v of size $C_i/2$. In the subsequent discussion, we say that v is *rich* if $\mathcal{I}'_i(v)$ is met.

Overloaded and Lazy Vertices: Other than not satisfying $\mathcal{I}'_i(v)$, there are two other bad events that we need to consider. If v has too many neighbors that share a sampled color with v, we say that

v is *overloaded.* This is a bad event since the goal of the palette sparsification is to reduce the number of neighbors that v needs to receive information from. More concretely, we say a vertex is overloaded, if

If most of the sampled colors of v reserved for iteration i have already been taken by the neighbors of v during the previous iterations $1, \ldots, i-1$, then v does not have enough colors to correctly run the algorithm for iteration i. In this case, we say that v is *lazy.*

The Sparsified Color Bidding Algorithm

We are now in a position to describe the sparsified version of the ColorBidding method. For the sake of clarity we use the following notation to describe the palette of a vertex u. Recall that $\Psi_0(u)$ refers to the palette of u initially in the original graph G_0. At the beginning of an iteration, we write $\Psi^+(u)$ to denote the set of available colors at u, and write $\Psi^-(u)$ to denote the set of colors already taken by vertices in $N_{G_0}(u)$. Note that $\Psi^+(u) = \Psi_0(u) \setminus \Psi^-(u)$.

SparsifiedColoring: The SparsifiedColoring algorithm—which represents the entire coloring algorithm—works as follows. Let $K = \log^{3+\zeta} p^\star$, consider the graph of uncolored and not Bad vertices, where each vertex v knows the set $\Psi^-(v)$ of colors already used by neighbors.

It consists of k iterations of SparsifiedColorBidding. More concretely, for $k = O(\log^* p^* - \log^* C_0) = O(\log^* \Delta)$ iterations, in iteration i, each node v does the following.

(1) It generates a sequence $\mathcal{R}_v^{(i)}(1), \ldots, \mathcal{R}_v^{(i)}(K)$ of colors, by picking each uniformly at random and independently from $\Psi_0(v)$. Then, it collects the information about $\left\{\mathcal{R}_u^{(i')} : 0 \leq i' \leq i\right\}$ from each neighbor $u \in N(v)$. If there exist three indices $i' \in [0, i], j \in$

$[1, K]$, and $j' \in [1, K]$ so that $\mathcal{R}_v^{(i)}(j) = \mathcal{R}_v^{(i')}(j')$, we say u is a significant neighbor of v. If v has more than $K^2 \log \Delta$ significant neighbors, we call v *overloaded.*

(2) It calls the function SparsifiedColorBidding is called with parameters $C \leftarrow C_i$, and $\mathcal{R}_v \leftarrow \mathcal{R}_v^{(i)}$.

Intuitively, an overloaded vertex v means that the colors in $\mathcal{R}_v^{(i)}$ have appeared in $\bigcup_{0 \le i' \le i} \mathcal{R}_u^{(i')}$ for too many neighbors $u \in N(v)$. This is undesirable as we want the degree of the sparsified graph to be small.

SparsifiedColorBidding: We present a sparsified version of ColorBidding: the procedure SparsifiedColorBidding$(C, \{\mathcal{R}_v\}_v)$ is the sparsified version of ColorBidding. It consists of four steps, executed by all nodes v in parallel.

(1) Node v calls SampleColors with $k_1 \leftarrow C/2$, $k_2 \leftarrow \log^{\zeta} p^\star$, $S^- \leftarrow \Psi^-(v)$, and $\mathcal{R} \leftarrow \mathcal{R}_v$) to sample two sets S_v and T_v of k_1 and k_2 colors, respectively, from $\mathcal{R} \leftarrow \mathcal{R}_v \setminus \Psi^-(v)$. If v is overloaded, it sets $S_v = T_v = \varnothing$. If $S_v = \varnothing$, we say that v is *lazy.*

(2) Node v then collects the information S_u from all neighbors $u \in N^{\mathsf{out}}(v)$. If at most $2/3$ of the colors in T_v are selected in S_u of neighbors $u \in N^{\mathsf{out}}(v)$, thus $\mathcal{I}'_i(v)$ is met, we call v *rich.*

(3) If v is not rich or if v is lazy, then v marks itself Bad and stops the algorithm, thus skips the next step.

(4) If there is a color $c \in S_v$ that is not in $\bigcup_{u \in N^{\mathsf{out}}(v)} S_u$, we say that v is *lucky* with color c. Node v then colors itself with color c. If v is lucky with more than one color, ties are broken arbitrarily.

Note that a Bad vertex does not attempt to color itself; but it still might need to provide information to other vertices in subsequent iterations.

SampleColors: The function SampleColors$(k_1, k_2, S^-, \mathcal{R})$ describes how a vertex u samples the colors S_u and T_u in a single iteration. We use k_1 and k_2 as the target set sizes of S_u and T_u, respectively. The set $\mathcal{R}$ represents a length-K sequence of colors that u *presampled* for the iteration i, where $K = \log^{3+\zeta} p^\star$, and $\mathcal{R}(j)$ stands for color j of $\mathcal{R}$. We will later see that $\mathcal{R}$ is generated in such a way that each $\mathcal{R}(j)$ is a uniformly random color chosen from $\Psi_0(u)$, where $\Psi_0(u)$ is the set of initially available colors of v in G_0. The set S^- represents the set $\Psi^-(u)$ which consists of the colors already taken by the vertices in $N_{G_0}(u)$ before iteration i. The method SampleColors$(k_1, k_2, S^-, \mathcal{R})$ goes through the colors in $\mathcal{R} \setminus S^-$ and stores the first k_1 such colors in S_u, and the first k_2 such colors in T_u, if there are enough colors, and $\varnothing$ otherwise.

It is straightforward to verify that the output (S_v, T_v) of the method SampleColors$(C/2, \log^\zeta p^*, \Psi^-(v), \mathcal{R}_v)$ satisfies one of the following.

(a) $S_v = T_v = \varnothing$. This happens when most of the pre-sampled colors for this iteration have been taken by the neighboring vertices. We will alter show that this occurs with probability $\Delta^{-\Omega(1)}$.

(b) When each $\mathcal{R}_v(j)$ is a uniformly random color of $\Psi_0(v)$, then T_v is a uniformly random subset of $\Psi_0(v) \setminus \Psi^-(v)$ of size $\log^\zeta p^*$ and S_v is a uniformly random subset of T_v of size $C/2$.

9.3.4 Analysis of Sparsified Color Bidding

For each iteration i, recall that $\Psi^+(v) = \Psi_0(v) \setminus \Psi^-(v)$ is the set of available colors at v at the beginning of this iteration. Consider the beginning of iteration i of SparsifiedColoring. In the graph under consideration, we say that v is (C, D)-honest if the following two conditions are met.

(i) $\sum_{u \in N^{\mathsf{out}}(v)} 1/p_u \leq 1/C$.

(ii) For each color $c \in \Psi_0(v)$, vertex v has at most D many c-

significant neighbors u in the previous iteration.

All vertices are $(C_0, 0)$-honest at the beginning of iteration $i = 0$.

Honest Vertices are Well-Behaved

We first show that all (C, D)-honest vertices are well-behaved.

Lemma 9.7. *Let U be the set of yet uncolored but not Bad vertices after iteration i. If a vertex v is (C, D)-honest, with $C \le \log^{\beta} p^{\star}$ and $D \le 2Kk = O(K \log^{*} \Delta)$, at the beginning of iteration i of SparsifiedColorBidding in SparsifiedColoring, the following holds.*

(i) The probability that v does not successfully color itself is at most $e^{-\frac{C}{6}} + e^{-\Omega\left(\log^{\zeta} \Delta\right)}$.

(ii) The probability that v marks itself Bad is at most $e^{-\Omega\left(\log^{\zeta} \Delta\right)}$.

(iii) The probability that at the beginning of the next iteration, $v \in U$ or v is not (C', D')-honest is at most $e^{-\Omega\left(\log^{\zeta} \Delta\right)}$, where $C' = \min\left\{\frac{1}{2} e^{\frac{C}{6}} C,\ \log^{\zeta} p^{\star}\right\}$ and $D' = D + 2K$.

The probability calculation only relies on the distribution of random bits generated in $N^2_{G_0}(v)$ in this iteration, i.e., $\{\mathcal{R}^{(i)}_u\}_{u \in N^2_{G_0}(v)}$. In particular, the result holds even if random bits generated outside $N^2_{G_0}(v)$ are determined adversarially.

Proof. We focus on the iteration i of the algorithm where the vertex v is (C, D)-honest. There is no guarantee about the (C, D)-honesty of all other vertices. For this vertex v, we write $\mathcal{E}^{\mathsf{overload}}_v$, $\mathcal{E}^{\mathsf{lazy}}_v$, $\mathcal{E}^{\mathsf{rich}}_v$, and $\mathcal{E}^{\mathsf{lucky}}_v$ to denote the event that v is overloaded, lazy, rich, and lucky. Note that a lucky vertex must be rich and not lazy, and an overloaded vertex must be lazy.

In this proof we frequently use the inequality

$$\Delta + 1 \geq |\Psi_0(v)| \geq |\Psi^+(v)| \geq p_v \geq p^\star = \Omega\left(\frac{\Delta}{\log \Delta}\right).$$

Our analysis only relies on the random bits generated by vertices within $N^2_{G_0}(v)$ in this iteration.

We next bound the probabilities of the events $\mathcal{E}_v^{\mathsf{overload}}$, $\mathcal{E}_v^{\mathsf{lazy}}$, and $\mathcal{E}_v^{\mathsf{rich}}$ separately, and then combine them to prove Lemma 9.7.

Claim 9.8. *We have* $\mathsf{Pr}[\mathcal{E}_v^{\mathsf{overload}}] = e^{-\Omega(\log^{3+\zeta} \Delta)}$.

Proof. Since v is (C, D)-honest, our plan is to show that for each color $c \in \Psi_0(v)$, the number $\mathcal{R}_u^{(i)}$ of *new* c-significant neighbor $u \in N_{G_0}(v)$ brought by the color sequences in iteration i is at most $2K$ with probability $1 - e^{-\Omega(\log^{3+\zeta} \Delta)}$.

Set $N_{G_0}(v) = \{u_1, \ldots, u_s\}$, use X_r for the indicator variable that color c appears in $\mathcal{R}_{u_r}^{(i)}$, and define $Y = \sum_{1 \leq r \leq s} X_j$. Then Y is an upper bound on the number of new c-significant neighbors. Since v is (C, D)-honest, the total number of c-significant neighbors is at most $D + Y$. Note that $X_1, \ldots, X_s$ are independent, and

$$\begin{aligned}\mathbb{E}[Y] = \sum_{1 \leq r \leq s} \mathbb{E}[X_r] &\leq \sum_{1 \leq r \leq s} \left(1 - \left(1 - \frac{1}{|\Psi_0(u_r)|}\right)^K\right) \\ &\leq s\frac{K}{\Delta + 1} \leq \frac{K\Delta}{\Delta + 1} < K.\end{aligned}$$

By a Chernoff bound, we have

$$\mathsf{Pr}[Y \geq 2K] \leq e^{-\Omega(K)} = e^{-\Omega(\log^{3+\zeta} \Delta)}.$$

A union bound over all $c \in \Psi_0(v)$ thus shows that v has more than $D' = D + 2K$ c-significant neighbors $u \in N_{G_0}(v)$ for some color $c \in \Psi_0(v)$ in iteration i with probability at most $e^{-\Omega(\log^{3+\zeta}\Delta)}$. Given that v has no more than D' many c-significant neighbors for every color $c \in \Psi_0(v)$, we infer that v has at most

$$|\mathcal{R}_u^{(i)}|D' \le KD' = K(D+2K) \ll K^2 \log \Delta$$

significant neighbors, which implies that v is not overloaded, which concludes the proof. □

Claim 9.9. *We have* $\mathsf{Pr}[\mathcal{E}_v^{\mathsf{lazy}}] = e^{-\Omega(\log^{2+\beta}\Delta)}$.

Proof. Remember that v is lazy if either (a) v is overloaded, or if (b) $\mathsf{SampleColors}$ outputs $(\varnothing, \varnothing)$. In view of Claim 9.8, we only need to show that $\mathsf{SampleColors}$ gives $(\varnothing, \varnothing)$ with probability at most $e^{-\Omega(\log^{2+\beta}\Delta)}$.

Consider the sampling process in $\mathsf{SampleColors}$, and suppose that we are in the middle of the process, and T is the current set of colors that we have obtained. Suppose $|T| = r$ currently, i.e., we have selected r colors from $\Psi^+(v)$. The probability that the next color $\mathcal{R}_j$ we consider is different from these r colors in T is at least

$$\frac{|\Psi^+(v)| - r}{|\Psi_0(v)|} \ge \frac{|\Psi^+(v)| - \log^\zeta p^\star}{|\Psi_0(v)|} \ge \frac{\Omega\frac{\Delta}{\log \Delta}}{\Delta+1} = \Omega\left(\frac{1}{\log \Delta}\right).$$

Remember that $|\Psi^+(v)| \ge p_v = \Omega(\Delta/\log\Delta)$, as well as $\log^\zeta p^\star = O(\log^\zeta \Delta)$, and that $\mathsf{SampleColors}$ outputs $(\varnothing, \varnothing)$ if after we go over all $k_2 \log^3 p^\star = \log^{3+\beta} p^\star$ elements in the sequence $\mathcal{R}$, the size of T is still less than $k_2 = \log^\beta p^\star$. The probability that this event occurs can be bounded by the probability that a binomial random variable with expected value $n'p'$ is smaller than t' and hence less than $n'p'/2$, where where $n' = \log^{3+\beta} p^\star = \Theta(\log^{3+\beta}\Delta)$, and $p' = \Omega(1/\log\Delta)$, as well as $t' = \log^\beta p^\star = \Theta(\log^\beta \Delta) \ll n'p'$. By a Chernoff bound, this is $e^{-\Omega(n'p')} = e^{-\Omega(\log^{2+\beta}\Delta)}$. □

Claim 9.10. *We have* $Pr\left[\overline{\mathcal{E}_v^{\mathsf{rich}}}\right] = e^{-\Omega(\log^\zeta \Delta)}$.

Proof. Recall that v is rich if $\left|T_v \setminus \bigcup_{u \in N_G^{\mathsf{out}}(v)} S_u\right| \geq |T_v|/3$. If v is lazy, then $T_v = \varnothing$, so v is automatically rich. Thus, in the subsequent discussion, we assume that v is not lazy. Let $N_G^{\mathsf{out}}(v) = \{u_1, \ldots, u_s\}$ and $X_r = |T_v \cap S_{u_r}|$. To prove the lemma, it suffices to show that we have $\mathsf{Pr}[Y \geq 2/3|T_v|] = e^{-\Omega(\log^\zeta p^\star))}$ for $Y = \sum_{r=1}^{s} X_r,$.

We consider the random variable $X_r = |T_v \cap S_{u_r}|$. For notational simplicity, we write $u = u_r$. If u is lazy, then $X_r = 0$. Suppose u is not lazy. Then let $S_u \subseteq \Psi^+(u)$ be the result of randomly choosing $C/2$ distinct colors $c_1, \ldots, c_{C/2}$ from $\Psi^+(u)$, one by one. For each $j \in [1, C/2]$, let $Z_{u,j}$ be the indicator random variable that $c_j \in T_v$. Then $X_r = \sum_{j=1}^{C/2} Z_{u,j}$. Observe that in the process, when we pick the color j, the probability that the color picked is in T_j is at most

$$\frac{|T_v|}{|\Psi^+(u)| - (j-1)} \leq \frac{|T_v|}{|\Psi^+(u)| - \frac{C}{2}},$$

regardless of the already chosen colors $c_1, \ldots, c_{j-1}$. Thus, we have

$$\mathbb{E}[Z_{u,j}] \leq \frac{|T_v|}{|\Psi^+(u)| - \frac{C}{2}} \leq \frac{|T_v|}{p_u - \frac{C}{2}} \leq \frac{1.1|T_v|}{p_u},$$

since $C \leq \log^\beta p^\star = \operatorname{poly}\log \Delta$ and $p_u = \Omega(\Delta/\log \Delta)$.

Therefore, in order to bound $Y = X_1 + \cdots + X_s$ from above, we can assume without loss of generality that each X_r is the sum of $C/2$ independent and identically distributed random variables, and each of them is a bernoulli random variable with $p = \frac{1.1|T_v|}{p_u}$, and so Y is the summation of $s \cdot (C/2)$ independent 0-1 random variables. Since v is (C, D)-honest, we have $\sum_{u \in N_G^{\mathsf{out}}(v)} 1/p_u \leq 1/C$. The expected value of Y can be upper bounded as follows.

$$\mathbb{E}[Y] \leq \frac{C}{2} \sum_{v \in N_G^{\mathsf{out}}(u)} \frac{1.1|T_v|}{p_v} \leq \frac{1.1}{2}|T_v|.$$

A Chernoff bound yields

$$\begin{aligned}\mathsf{Pr}\left[\overline{\mathcal{E}_v^{\mathsf{rich}}}\right] &\le \mathsf{Pr}\left[Y \ge \left(\frac{1}{1.1}\cdot\frac{4}{3}\right)\left(\frac{1.1}{2}|T_v|\right)\right] \\ &= e^{-\Omega(|T_v|)} = e^{-\Omega\left(\log^{\zeta} p^\star\right)},\end{aligned}$$

and concludes the proof. □

Using Claims 9.8 to 9.10, we now prove the three conditions specified in Lemma 9.7.

Condition (i): Conditioning on $\mathcal{E}_v^{\mathsf{rich}} \cap \overline{\mathcal{E}_v^{\mathsf{lazy}}}$, we have that v is lucky with some color unless it fails to select *any* of $|T_v|/3$ specific colors from $T_v \subseteq \Psi^+(v)$. Remember that $\mathcal{E}_v^{\mathsf{rich}}$ implies that $\left|T_v \setminus \bigcup_{u\in N_G^{\mathsf{out}}(v)} S_u\right| \ge |T_v|/3$, and if any one of them is in S_v, then v successfully colors itself. Also remember that S_v is a size-$(C/2)$ subset of T_v chosen uniformly at random. Thus,

$$\mathsf{Pr}\left[\ \overline{\mathcal{E}_v^{\mathsf{lucky}}} \mid \mathcal{E}_v^{\mathsf{rich}} \cap \overline{\mathcal{E}_v^{\mathsf{lazy}}}\ \right] \le \frac{\binom{2/3|T_v|}{C/2}}{\binom{|T_v|}{C/2}} \le \left(\frac{2}{3}\right)^{C/2} \le e^{-\frac{C}{6}}.$$

Combining this with Claims 9.9 and 9.10, we get

$$\begin{aligned}\mathsf{Pr}\left[\ \overline{\mathcal{E}_v^{\mathsf{lucky}}}\ \right] &\le \mathsf{Pr}\left[\ \overline{\mathcal{E}_v^{\mathsf{lucky}}} \mid \mathcal{E}_v^{\mathsf{rich}} \cap \overline{\mathcal{E}_v^{\mathsf{lazy}}}\ \right] + \mathsf{Pr}\left[\mathcal{E}_v^{\mathsf{lazy}}\right] + \mathsf{Pr}\left[\overline{\mathcal{E}_v^{\mathsf{rich}}}\right] \\ &\le e^{-\frac{C}{6}} + e^{-\Omega(\log^{\zeta} p^\star)},\end{aligned}$$

which shows (i).

Condition (ii): Vertex v marks itself Bad if $\overline{\mathcal{E}_v^{\mathsf{rich}}} \cup \mathcal{E}_v^{\mathsf{lazy}}$ occurs, which happens with probability at most $\mathsf{Pr}\left[\overline{\mathcal{E}_v^{\mathsf{rich}}}\right] + \mathsf{Pr}\left[\mathcal{E}_v^{\mathsf{lazy}}\right] = e^{-\Omega(\log^{\zeta} p^\star)}$, using Claims 9.9 and 9.10.

Condition (iii): Define Y as the summation of $1/p_u$ over all vertices $u \in N_G^{\mathsf{out}}(v)$ such that $u \notin U$ in the next iteration. We prove that the probabilities of (a) $Y \leq 1/C'$ and (b) v has more than D' c-significant neighbors $u \in N_{G_0}(v)$ in this iteration are both at most $e^{-\Omega(\log^{\zeta} \Delta)}$.

For (b), this directly follows from Claim 9.8. For the rest of the proof, we deal with (a). Let $N_G^{\mathsf{out}}(v) = \{u_1, \ldots, u_s\}$. Consider the event $\mathcal{E}_r^* = \mathcal{E}_{u_r}^{\text{lucky}} \cup \overline{\mathcal{E}_{u_r}^{\text{rich}}} \cup \mathcal{E}_{u_r}^{\text{lazy}}$ that u_r does not join U in the next iteration, i.e., u_r successfully colors itself or marks itself Bad.

For each $r \in [1, s]$, define the random variable Z_r as follows. Let $Z_r = 0$ if the event $\mathcal{E}_{u_r}^{\text{lucky}} \cup \overline{\mathcal{E}_{u_r}^{\text{rich}}} \cup \mathcal{E}_{u_r}^{\text{lazy}}$ occurs, and $Z_r = 1/p_{u_r}$ otherwise. Clearly we have $Y = \sum_{r=1}^{s} Z_r$. Note that as calculated above in the proof of Condition (i), we have

$$\mathsf{Pr}\left[\, \overline{\mathcal{E}_r^*} \,\right] = \mathsf{Pr}\left[\, \overline{\mathcal{E}_{u_r}^{\text{lucky}}} \cap \mathcal{E}_{u_r}^{\text{rich}} \cap \overline{\mathcal{E}_{u_r}^{\text{lazy}}} \,\right] \leq \mathsf{Pr}\left[\, \overline{\mathcal{E}_{u_r}^{\text{lucky}}} \mid \mathcal{E}_{u_r}^{\text{rich}} \cap \overline{\mathcal{E}_{u_r}^{\text{lazy}}} \,\right],$$

which is at most $e^{-\frac{C}{6}}$, hence $\mathbb{E}[Y] \leq e^{-\frac{C}{6}}/C$. Since v is (C, D)-honest, we have $\sum_{u \in N_G^{\mathsf{out}}(v)} 1/p_u \leq 1/C$. Combining these two inequalities, we obtain that $\mathbb{E}[Y] \leq e^{-\frac{C}{6}}/C \leq 1/(2C')$.

Next, we prove the desired concentration bound for Y using Hoeffdings's inequality. Each variable Z_r is within the range $[0, 1/p_{u_r}]$. We have

$$\sum_{u \in N_G^{\mathsf{out}}(v)} \frac{1}{p_u^2} \leq \sum_{u \in N_G^{\mathsf{out}}(v)} \frac{1}{p_u p^\star} \leq \frac{1}{Cp^\star},$$

and thus obtain

$$\mathsf{Pr}\left[Y \geq \frac{1}{C'}\right] \leq e^{\frac{-\frac{2}{(2C')^2}}{\frac{1}{Cp^\star}}} = e^{-\Omega(p^\star \log^{-3\beta} p^\star)}$$
$$\leq e^{-\Omega(\sqrt{p^\star})} \ll e^{-\Omega(\log^{\beta} p^\star)},$$

as $(1/C')^2 = \Omega(1/\log^{2\beta} p^\star)$ and $Cp^\star = \Omega(p^\star/\log^\beta p^\star)$, by the assumptions specified in the lemma statement.

There is a subtle issue regarding the applicability of Hoeffding's inequality. The variables $\{X_1, \ldots, X_k\}$ are not independent, but we argue that we are still able to apply Hoeffding's inequality. Assume that $N^{\mathsf{out}}(v) = (u_1, \ldots, u_s)$ is sorted in reverse topological order, and so for each $1 \le a \le s$, we have $N^{\mathsf{out}}(u_a) \cap \{u_a, \ldots, u_s\} = \varnothing$. We reveal the random bits in the following manner. First of all, we reveal the set T_u for all vertices u. Now the event regarding whether a vertex is rich or is lazy has been determined. Then, for $r = 1$ to s, we reveal the set $\{S_u \mid u = u_r \text{ or } u \in N^{\mathsf{out}}(u_r)\}$. This information is enough for us to decide the outcome of Z_r. Note that in this process, conditioning on *arbitrary* outcome of $Z_1, \ldots, Z_{r-1}$ and all random bits revealed prior to revealing the set S_{u_r}, The probability that $\overline{\mathcal{E}_r^*}$ occurs is still at most $e^{-C/6}$. □

Remark 9.11. *Note that Lemma 9.7 only relies on the assumption that the vertex v under consideration is (C, D)-honest, and works even if many neighbors of v are not (C, D)-honest. This is in contrast to most of the analysis of graph shattering algorithms where the analysis relies on the assumption that* all *vertices at the beginning of each iteration have to satisfy certain invariants.*

Probability of an Uncolored Vertex

Based on Lemma 9.7, we show that SparsifiedColoring colors a vertex with a sufficiently high probability that enables us to apply the shattering lemma.

Lemma 9.12. *The algorithm SparsifiedColoring gives a partial coloring of G_0 such that the probability that a vertex v does not successfully color itself with a color in $\Psi_0(v)$ is*

$$O(k) \cdot e^{-\Omega\left(\log^\zeta \Delta\right)} = \Delta^{-\Omega(1)},$$

and this holds even if the random bits generated outside $N^2_{G_0}(v)$ are determined adversarially.

Proof. We consider the sequence $D_0 = 0$ and $D_{i+1} = D_i + 2K$. Suppose the algorithm does not color a vertex v, then v must fall into one of the following categories.

(a) There is an index $i \in [0, k-2]$ such that v is (C_i, D_i)-honest at the beginning of iteration i, but v is not (C_{i+1}, D_{i+1})-honest at the beginning of iteration $i+1$. By Lemma 9.7 (iii), this occurs with probability at most $(k-1)e^{-\Omega(\log^\zeta \Delta)}$.

(b) There is an index $i \in [0, k-1]$ such that v is (C_i, D_i)-honest at the beginning of iteration i, but v marks itself Bad in iteration i. By Lemma 9.7 (ii), this happens with probability at most $ke^{-\Omega(\log^\zeta \Delta)}$.

(c) For the last iteration $i = k-1$, the vertex v is (C_{k-1}, D_{k-1})-honest at the beginning of iteration $k-1$, but v does not successfully color itself with a color in its palette in iteration $k-1$. By Lemma 9.7 (iii), this occurs with probability at most $e^{-\Omega(\log^\zeta \Delta)}$.

Note that our analysis only relies on the distribution of random bits generated in $N^2_{G_0}(v)$, as guaranteed by Lemma 9.7. That is, even if an adversary is able to decide the random bits of vertices outside of $N^2_{G_0}(v)$ throughout the algorithm SparsifiedColoring, the probability that v does not successfully color itself is still at most $O(k) \cdot e^{-\Omega(\log^\zeta \Delta)}$. □

9.3.5 Implementation of Sparsified Color Bidding

In this section, we present an implementation of SparsifiedColoring in the LOCAL model so that after an $O(1)$-round preprocessing step, each node v is able to identify a $\operatorname{poly}\log \Delta$-size subset $N_*(v) \subseteq N_{G_0}(v)$ of neighbors such that v only needs to receive information from these nodes during SparsifiedColoring. Put simply, in the preprocessing step, we let each vertex v sample the color sequences $\mathcal{R}_v^{(i)}$ for $0 \le i \le k-1$ and let each vertex learn the set of colors sampled by its neighbors. Based on this information, before the first iteration begins, v is able to identify at most $K^2 \log \Delta = \operatorname{poly}\log \Delta$ neighbors of v for each iteration i such that v is sure that v does not need to receive information from all other neighbors during this iteration.

Fixing All Random Bits

Instead of having each node v generate the color sequence $\mathcal{R}_v^{(i)}$ at iteration i, we determine all of $\{\mathcal{R}_v^{(i)}\}_{0 \le i \le k-1}$ in the preprocessing step. After fixing these sequences, we can regard $\mathsf{SparsifiedColoring}$ as a *deterministic* LOCAL algorithm, where $\{\mathcal{R}_v^{(i)}\}_{v \in V_0,\ 0 \le i \le k-1}$ can be seen as the input for the algorithm. To gather this information, we need to use messages of $k \cdot K \cdot O(\log n) = \operatorname{poly}\log \Delta \cdot \log n$ bits, where $k = O(\log^* \Delta)$ is the number of iterations, and K is the length of the color sequence $\mathcal{R}_v^{(i)}$ for a single iteration.

Determining the Set $N_*(v)$

We show how to let each vertex v determine a set $N_*(v) \subseteq N_{G_0}(v)$ of size $\operatorname{poly}\log \Delta$ so that v only needs to receive messages from $N_*(v)$ during the execution of $\mathsf{SparsifiedColoring}$, based on information in $\{\mathcal{R}_u^{(i)}\}_{u \in N_{G_0}(v),\ 0 \le i \le k-1}$. We make the following observations, which follow straightforwardly from the description of $\mathsf{SparsifiedColoring}$. In order for v to execute the method $\mathsf{SparsifiedColorBidding}$ in iteration i correctly, v does not need to receive information from $u \in N_{G_0}(v)$ if all colors in $\{\mathcal{R}_u^{(i')}\}_{u \in N_{G_0}(v),\ 0 \le i' \le i}$ do not overlap with the colors in $\mathcal{R}_u^{(i)}$. In other words, v only needs information from its significant neighbors. Moreover, if v is overloaded in iteration i, then v knows that it is lazy in this iteration, and so the outcome of $\mathsf{SparsifiedColorBidding}$ in iteration i is that v sets $S_v = T_v = \varnothing$ and marks itself Bad. This allows us to define the set $N_*(v)$ as follows.

Add $u \in N_{G_0}(v)$ to $N_*(v)$ if there exists an index $i \in [0, k-1]$ such that u is a significant neighbor of v in iteration i and if v is not overloaded in iteration i.

By the definition of overloaded vertices, we know that if v is not overloaded, then v has at most $K^2 \log \Delta = \operatorname{poly}\log \Delta$ significant neighbors for this iteration, meaning $|N_*(v)| = \operatorname{poly}\log \Delta$. Note that the set $N_*(v)$ can be calculated offline at the node v during the preprocessing step.

Summary

In the LOCAL model, the SparsifiedColoring algorithm can be summarized as follows. In a randomized 1-round preprocessing phase with messages of $\operatorname{poly}\log \Delta \cdot \log n$ bits, it is achieved that each node has calculated a set $N_*(v)$ of size $|N_*(v)| = \operatorname{poly}\log \Delta$ so that in the second phase, each node v only receives messages from $N_*(v)$. In the main phase—a deterministic $O(\log^* \Delta)$-round procedure—each node receives its color sequences for all iterations as input and outputs a color (or a special symbol $\perp$ indicating that it is uncolored). The input can be represented with $\ell_{\mathsf{in}} = \operatorname{poly}\log \Delta \cdot \log n$ bits; the output requires $\ell_{\mathsf{out}} = O(\log n)$ bits.

9.4 Vertex Coloring in CC

We split the proof into two parts, according to $\Delta = \operatorname{poly}\log n$ or $\Delta = \log^{4+\Omega(1)} n$.

9.4.1 Coloring Low-Degree Graphs

We show that by applying SparsifiedColoring with the opportunistic speedup lemma in Lemma 3.5, we can solve $(\Delta + 1)$ list vertex coloring in CC in $O(1)$ rounds when $\Delta = \operatorname{poly}\log n$.

Lemma 9.13. *There is an $O(1)$-round* CC *algorithm that w.h.p. solves the $(\Delta + 1)$ list vertex coloring problem in any graph with maximum degree $\Delta = \operatorname{poly}\log n$.*

Proof. First, recall that it directly follows from Lenzen's routing algorithm that an r-round LOCAL algorithm with messages of size s can be implemented in $O(1)$ rounds of CC if $\Delta^r s = O(n)$. For our application, this in particular includes $O(1)$-round algorithms with message size $\operatorname{poly}\Delta \cdot \log n = \operatorname{poly}\log n$. We simulate the LOCAL coloring algorithm as follows.

Black-Box Partial Coloring: The first step of the LOCAL algorithm is to run the black-box algorithm from Lemma 9.4, which takes $O(1)$ rounds in LOCAL with messages of size $O(\Delta^2 \log n) = \operatorname{poly}\log n$. This phase thus can be implemented in CC in $O(1)$ rounds using Lenzen's routing.

Graphs R and V_{bad}: The set R trivially induces a subgraph with $O(n)$ edges. By Lemma 3.3, the set V_{bad} w.h.p. induces a subgraph with $O(n)$ edges. We use Lenzen's routing to color these two subgraphs in $O(1)$ rounds each.

Graph V_{good}: The SparsifiedColoring algorithm can be implemented in CC as follows.

The preprocessing phase takes $O(1)$ rounds in LOCAL with messages of size $\operatorname{poly}\log\Delta \cdot \log n$, which can be implemented in $O(1)$ rounds of CC using Lenzen's routing.

For the main phase, we apply the speedup lemma in Lemma 3.5 with $r = O(\log^* \Delta)$, $\ell_{\mathsf{out}} = O(\log n)$, $\ell_{\mathsf{in}} = \operatorname{poly}\log\Delta \cdot \log n$, and $\Delta_* = \operatorname{poly}\log\Delta$. Since $\Delta = \operatorname{poly}\log n$, the criterion for Lemma 3.5 is satisfied. We thus can simulate this phase in $O(1)$ many CC rounds.

The algorithm SparsifiedColoring does not color all vertices in V_{good}. However, by Lemma 9.12 and Lemma 3.3, we know that these uncolored vertices induce a subgraph with $O(n)$ edges. We use Lenzen's routing to color them in $O(1)$ rounds. □

9.4.2 Coloring High-Degree Graphs

We show that the $(\Delta+1)$ list vertex coloring problem can be solved in $O(1)$ rounds in CC using graph partitioning when the degrees are assumed to be sufficiently high, as captured by the following lemma.

Lemma 9.14. *There is an $O(1)$-round CC algorithm that solves the $(\Delta + 1)$ list vertex coloring problem w.h.p. in any graph with maximum degree $\Delta = \log^{4+\Omega(1)} n$.*

We prove this result in two parts. First, we show that our graph partitioning can be implemented in $O(1)$ rounds in CC. Then, we use recursive application of this graph partitioning.

Implementation of Graph Partitioning in CC

We show how this graph partitioning can be implemented in $O(1)$ rounds in the Congested Clique.

Lemma 9.15. *The graph partitioning, as described in Section 9.2.2, can be computed w.h.p. in $O(1)$ rounds in CC.*

Proof. Partitioning the vertex set V is straightforward, as every vertex can make the decision independently and offline, whereas it is not obvious how to partition C to make all vertices agree on the same partition. Note that we can assume $|C| \leq (\Delta+1)|V|$; if $|C|$ is greater than $(\Delta+1)|V|$ initially, then we can let each vertex decrease its palette size to $\Delta+1$ by removing some colors in its palette, and we will have $|C| \leq (\Delta+1)|V|$ after removing these colors.

A straightforward way of partitioning C is to generate $\Theta(|C| \log n)$ random bits at a vertex v offline, and then v broadcasts this information to all other vertices. Note that it takes $O(\log k) = O(\log |V|) = O(\log n)$ bits to encode which part of $C_1 \cup \cdots \cup C_k$ each $c \in C$ is in. A direct implementation of the approach cannot be done in $O(1)$ rounds, due to the message size constraint of CC, as each vertex can send at most $\Theta(n \log n)$ bits in each round.

To solve this issue, observe that it is not necessary to use total independent random bits for each color $c \in C$ in Lemma 9.3. Indeed, $\Theta(\log n)$-wise independence suffices to guarantee a failure probability of $n^{-\Omega(1)}$ in all applications of the Chernoff bound in Lemma 9.3 when using a variant of Chernoff for bounded independence [219]. To compute the decomposition $C = C_1 \cup \cdots \cup C_k$ with $\Theta(\log n)$-wise independent random bits, we thus only need $O(\log n \cdot \log(|C| \log k)) = O(\log^2 n)$ total independent random bits (also see [7, Section 16.2]). Broadcasting $O(\log^2 n)$ bits of information to all nodes can be done in $O(1)$ rounds via Lenzen's routing. □

Recursive CC Coloring Algorithm

We prove that a constant-depth recursive application of Lemma 9.3 suffices to give an $O(1)$-round CC $(\Delta + 1)$ list vertex coloring algorithm for graphs with $\Delta = \log^{4+\Omega(1)} n)$.

We will use the graph partitioning to color each B_i using the colors in C_i in parallel as follows. Since $|E(G[B_i])| = O(|V|) = O(n)$, using Lenzen's routing, we are able to send the entire graph $G[B_i]$ to a single distinguished node $v_i^\star$, which then can compute a proper coloring of $G[B_i]$ offline.

If $|E(G[L])| = O(n)$, then similarly we can send $G[L]$ to a single distinguished node to compute the coloring offline. Otherwise, we apply the graph partitioning recursively on $G[L]$, with the *same* parameter n (to guarantee a failure probability of $n^{-\Omega(1)}$ in every step). What remains to show is that $O(1)$ recursive applications of the partitioning are enough.

Proof of Lemma 9.14. Given a graph $G = (V, E)$, we apply the graph partitioning algorithm of Lemma 9.3 to partition V into subsets $B_1, \ldots, B_k, L$ with parameter $n = |V|$, and $k = \sqrt{\Delta}$. After that, we select arbitrary $k = \sqrt{\Delta}$ different nodes $v_1^*, \ldots, v_k^*$ to be responsible for coloring $G[B_i]$, as follows. Each v_i^* in parallel gathers all information of $G[B_i]$, and then computes a proper coloring of $G[B_i]$ using the palettes $\Psi(v) \cap C_i$ for each vertex $v \in B_i$ offline. The existence of such a proper coloring is guaranteed by Property (ii) of Lemma 9.3. The gathering can be done in $O(1)$ rounds using Lenzen's routing, since Property (i) of Lemma 9.3 guarantees that $|E(G[B_i])| = O(n)$. We thus can color $V \setminus L$ in $O(1)$ rounds.

Finally, each vertex $v \in L$ removes the colors that have been taken by its neighbors in $V \backslash L$ from its palette $\Psi(v)$. By Property (iii), after this operation, the number of available colors for each $v \in L$ is at least $g_L(v) \geq \max\{d_L(v), \Delta_L - \Delta_L^{\nu}\} + 1$. Now the subgraph $G[L]$ satisfies all conditions required to apply Lemma 9.3, as long as $\Delta_L = \omega(\log^{\gamma} n)$. We will see that this condition is always met.

We then recursively apply the algorithm of the lemma on the subgraph induced by vertices L *with the same parameter n*. The recursion stops once we reach a point that $|E(G[L])| = O(n)$, and so we can apply Lenzen's routing to let one distinguished node gather all information of $G[L]$ and compute its proper coloring offline.

Now we analyze the number of iterations needed to reach a point that $|E(G[L])| = O(n)$. Here we use $\gamma = 2$ and $\nu = 3/4$. Let $V_1 = V$ be the vertex set, $\Delta_1 = \Delta$ the maximum degree, and $V_1 = B_1 \cup \cdots \cup B_k \cup L$ the graph partitioning in the first iteration. Define $V_2 = L$ and $\Delta_2 = \Delta_L$. Similarly, for $i > 2$, we define V_i and Δ_i based on the set L in the outcome of the graph partitioning algorithm in iteration $i - 1$. We have

$$\Delta_i = \Delta_{i-1} \cdot O\left(\frac{\sqrt{\log n}}{\Delta_{i-1}^{1/4}}\right)$$

by Property (iv) of Lemma 9.3 and

$$|V_i| = |V_{i-1}| \cdot O\left(\frac{\sqrt{\log n}}{\Delta_{i-1}^{1/4}}\right)$$

by Property (i) of Lemma 9.3. We choose $\alpha > 0$ such that $\Delta_1 = (\log n)^{2+\alpha}$, and assume $\alpha = \Omega(1)$ and $i = O(1)$. This yields

$$\Delta_i = O\left((\log n)^{2+\alpha(3/4)^{i-1}}\right)$$

as well as

$$|V_i| = O(n/\Delta) \cdot \Delta_i = n \cdot O\left((\log n)^{\alpha\left((3/4)^{i-1}-1\right)}\right).$$

Thus, the condition of $\Delta_i = \omega(\log^\gamma n) = \omega(\log^2 n)$ for applying Lemma 9.3 must be met.

Next, we analyze the number of iterations it takes to make $\Delta_i|V_i|$ sufficiently small. In the CC model, if $\Delta_i|V_i| = O(n)$, then we are able to compute a proper coloring of V_i in $O(1)$ rounds by Lenzen's routing. Let us write $\Delta = \log^{2+\alpha} n$, where $\alpha = 2+\beta$. The lemma statement implies that $\beta = \Omega(1)$. Note that the condition for $\Delta_i|V_i| = O(n)$ can be rewritten as $(2-\alpha) + 2\alpha(3/4)^{i-1} \leq 0$. Combining this with $\alpha = 2+\beta$, we obtain the formula $-\beta + 2(2+\beta)(3/4)^{i-1} \leq 0$, which can also be read as $(4/3)^{i-1} \geq 2(2+\beta)/\beta$.

Now we can calculate the minimum i needed so that the condition for $\Delta_i|V_i| = O(n)$ is met. We have $i \geq 1 + \log_{4/3} 2(2+\beta)/\beta = O(1)$, since $\beta = \Omega(1)$. Hence, our algorithm takes only $O(1)$ iterations. In particular, when $\beta \geq 10.8$, i.e., $\Delta = \Omega(\log^{12.8} n)$, we have $i \geq 1 + \log_{4/3} 2(2+\beta)/\beta$ for $i \geq 4$. Therefore, $\Delta_4|V_4| = O(n)$, which means that three iterations suffice. Since each iteration can be implemented in CC in $O(1)$ rounds, by Lemma 9.15, overall we get an algorithm with round complexity $O(1)$. $\square$

9.5 Vertex Coloring in MPC

First, note that the graph partitioning directly also leads to an $O(1)$-round MPC coloring algorithm with $\mathcal{S} = \widetilde{O}(n)$ memory per node and $\mathcal{MS} = \widetilde{O}(m)$ total memory, similar to the CC algorithm described in Section 9.4.2. This gives a simple alternative proof of a result by [13] that $(\Delta+1)$ (list) vertex coloring can be solved in the linear-memory regime. However, the coloring algorithm of [13] takes only one round of communication, and it uses only $\widetilde{O}(n)$ bits in total.

The case of sublinear-memory MPC is a little more complicated than the graph partitioning for CC and linear-memory MPC in that an $O(n)$-edge subgraph no longer fits into the memory of a node. In this case, we still apply the above divide-and-conquer approach, but we also recurse on each B_i. When the maximum degree of the current subgraph is small enough, we solve the coloring problem directly by simulating the CLP algorithm. In order to so, we need to make sure that the palette size of each vertex is not only higher than its degree, but also sufficiently close to the maximum degree of the subgraph under consideration.

We first show in Section 9.5.1 how to simulate the CLP algorithm in MPC under certain conditions. In Section 9.5.2, we then use this simulation together with a recursive application of the graph partitioning to prove our MPC result from Theorem 9.2.

9.5.1 Simulation of Local Coloring

We show how to simulate the CLP in CC.

Lemma 9.16 (Chang, Li, and Pettie [58, 209]). *The $(\Delta + 1)$ list vertex coloring problem can be solved w.h.p. in $O(\sqrt{\log\log n})$ rounds of MPC with total memory $\mathcal{MS} = \widetilde{O}\left(\sum_v d(v)^2\right)$ if $\Delta^2 = O\left(n^\delta\right)$ and if $|\Psi(v)| \geq \max\left\{d(v)+1, \Delta - \Delta^{3/5}\right\}$ for every vertex v.*

Note that combining this with [216, 114] gives a round complexity of $O(\log\log\log n)$.

The proof of Lemma 9.16 almost immediately follows from [58, 209]; there are only few changes that have to be made in order to turn their CC algorithm into a sublinear-memory MPC algorithm.

Proof of Lemma 9.16. There are two main issues in the sublinear-memory MPC model that we need to take care of. First, the total memory of the system is limited to $\mathcal{MS} = \widetilde{\Theta}(m+n)$. Second, the local memory per node is restricted to $O(n^\delta)$, for a constant $\delta > 0$. These two restrictions force us to be careful about the amount of information sent between the nodes. In particular, no vertex can receive messages from more than $O(n^\delta)$ other vertices in one round (as opposed to CC where a vertex can receive up to $O(n)$ messages per round).

The key feature of our partitioning algorithm is that we can reduce the coloring problem to several instances of graph coloring with maximum degree $\Delta = O(n^{\delta/2})$. Given this assumption, we can implement the CLP algorithm in the sublinear-memory MPC model almost line by line as done by Parter [209, Appendix A.2] for CC. Therefore, here we simply point out the differences in the algorithm and refer the reader to the paper of Parter for further technical details.

Dense Vertices: Put briefly, a vertex is γ-dense, if a $(1-\gamma)$-fraction of the edges incident on it belong to at least $(1-\gamma)\Delta$ triangles. An γ-almost clique is a connected component of γ-dense vertices that have at most $\gamma\Delta$ vertices outside the component. Each such component has a weak diameter of at most 2. These components can be computed in 2 rounds by each vertex learning its 2-hop neighborhood. This process is performed $O(\log\log\Delta)$ times in parallel which incurs a factor of $O(\log\log\Delta)$ in the memory requirements, which is negligible. Furthermore, the algorithm requires running a coloring algorithm within the dense components. Since the component size is at most $\Delta \ll \Delta^2$, we can choose one vertex in the component as a leader. The node containing the leader vertex can simulate the coloring algorithm offline without breaking the local memory restriction.

Memory Bounds: Once the 2-hop neighborhoods of nodes have been learned, no more memory overhead is required. Since we have $\Delta \ll n^{\delta/2}$, learning the 2-hop neighborhoods does not violate the local memory restriction of $O(n^\delta)$. For the total memory bound, storing 2-hop neighborhoods requires $\widetilde{O}(\sum_v (d(v))^2)$ memory.

Post-Shattering and Clean-up: Another step that we cannot use as a black box is an algorithm that colors a graph consisting of connected components of poly log n size. Regardless of the component sizes being small, all vertices over all components might not fit the memory of a single node. Hence, similarly to the CLP algorithm in the LOCAL model, we use the best deterministic list coloring algorithm to color the components. For general graphs, currently the best round complexity in the LOCAL model is obtained by applying the network decomposition algorithm of Panconesi and Srinivasan [208] with round complexity $2^{O(\sqrt{\log n'})}$, where $n' = O(\text{poly} \log n)$ is the maximum size of the small components.[3] We can speed up this bound exponentially in the MPC model by using the graph exponentiation technique, thus obtain a running time of $O(\sqrt{\log \log n})$.

We observe that for the graph exponentiation technique, if we have components of size poly log n, then the 2^i-hop neighborhood of any vertex for any i fits into the memory of a single node, since the number of vertices in the neighborhood is clearly bounded by poly log n. The same observation yields that total memory of $\widetilde{O}(m)$ suffices. □

[3]With [216, 114], we could further improve this to $O(\log^5 n')$.

9.5.2 Recursive Coloring in MPC

We show how to use recursive application of Lemma 9.3 as well as an efficient simulation of the CLP algorithm of [58], as summarized in Lemma 9.16, to prove Theorem 9.2.

Proof of Theorem 9.2. In the case that $\Delta = \operatorname{poly}\log n$, the conditions of Lemma 9.16 are satisfied trivially; we can solve the problem in $O(\log^* \Delta + \sqrt{\log\log n}) = O(\sqrt{\log\log n})$ rounds of sublinear-memory MPC with total memory $\widetilde{O}(n\Delta^2) = \widetilde{O}(m)$. Otherwise, we execute the following recursive algorithm.

Randomized Partitioning: We apply the randomized partitioning algorithm of Lemma 9.3 to G, which gives us sets $B_1, \ldots, B_k$ and L, as well as color sets $C_1, \ldots, C_k$. The goal is now to first color $B_1, \ldots, B_k$ with colors from $C_1, \ldots, C_k$, respectively. Since the colors in the sets C_i are disjoint, this gives a proper coloring of $B := \bigcup_{i=1}^{k} B_i$. Then, for every vertex in L, we remove all colors already used by neighbors in B from the palettes, leaving us with a list vertex coloring problem of the graph induced by L with maximum degree Δ_L.

Note that the implementation of the graph partitioning is straightforward in MPC.

We first describe how to color each set B_i with colors in C_i, and then how to solve the remaining list vertex coloring problem in L. For the parameters in Lemma 9.3, we use $\gamma = 6$ and $\nu = 3/5$.[4]

List-Coloring Problem in B_i: If $\Delta_i^2 = O(n^{\delta})$, then, because of Lemma 9.3 (ii), B_i satisfies the conditions of Lemma 9.16. We thus can apply the algorithm of Lemma 9.16 to B_i. Otherwise, we recurse on B_i. Note that this is possible since, by Lemma 9.3 (ii) applied

[4]The choice $\nu = 3/5$ is to ensure that the number of available colors for each vertex in each subgraph meets the palette size constraint specified in Lemma 9.16.

to G, B_i satisfies the conditions of Lemma 9.3.

List-Coloring Problem in L: If $\Delta_L^2 = O(n^\delta)$, then, due to Lemma 9.3 (iii) applied to G, the graph L satisfies the conditions of Lemma 9.16. We thus can apply the algorithm of Lemma 9.16 to L. Otherwise, we recurse on L. Note that this is possible since L satisfies the conditions of Lemma 9.3 due to Lemma 9.3 (iii).

Number of Iterations: Since the maximum degree in L reduces by a polynomial factor in every step, after at most $O(1/\delta)$ steps, the resulting graph has maximum degree at most $O(n^{\delta/2})$, where we satisfy the conditions of Lemma 9.16, and hence do not recurse further. Note that when recursing on sets B_i, the degree drop is even larger, and hence the same reasoning applies to bound the number of iterations.

Memory Requirement: It is obvious that the recursive partitioning of the input graph G does not incur any asymptotic overhead in the memory, neither local nor global. Now, let $\mathcal{H}$ be the set of all graphs H on which we apply the algorithm of Lemma 9.16. As we only apply this algorithm when the maximum degree Δ_H of H is $O(n^{\delta/2})$ or $\text{poly}(\log n)$, we clearly have $\Delta_H^2 = O(n^\delta)$, so the algorithm Lemma 9.16 is guaranteed to run with local memory $O(n^\delta)$.

It remains to show how to guarantee the total memory requirement of $\widetilde{O}(m)$, where m is the number of edges in the input graph G. First, observe that due to the specifications of Lemma 9.16, we can write the total memory requirement as $\sum_{H\in\mathcal{H}}\sum_{v\in H}(d_H(v))^2$. First, assume that the graph G has been partitioned at least three times to get to H. By Lemma 9.3 (iv), the degree of any vertex v in H is either $\widetilde{O}(1)$ or at most

$$\begin{aligned} d_G(v)\cdot\widetilde{O}\left(\Delta^{-\frac{1}{4}}\right)\widetilde{O}\left(\Delta^{-\frac{1}{4}\cdot\frac{3}{4}}\right)\widetilde{O}\left(\Delta^{-\frac{1}{4}\cdot\left(\frac{3}{4}\right)^2}\right) &= d_G(v)\cdot\widetilde{O}\left(\Delta^{-37/64}\right) \\ &= \widetilde{O}\left(\sqrt{d_G(v)}\right). \end{aligned}$$

Note that in the above calculation we assume that v goes to the leftover part L in all three iterations. If v goes to B_i, then the degree shrinks even faster. Remember that we set $q = \widetilde{O}(\Delta^{-1/4})$. Hence, we need a total memory of

$$\widetilde{O}\left(\sum_{H\in\mathcal{H}}\sum_{v\in H}(d_H(v))^2\right) = \widetilde{O}\left(\sum_{H\in\mathcal{H}}\sum_{v\in H}d_G(v)\right) = \widetilde{O}\left(\sum_{v\in G}d_G(v)\right),$$

which is in $\widetilde{O}(m)$. Note that the algorithm can be easily adapted to always perform at least three partitioning steps if Δ_H is bounded from below by a sufficiently large $\operatorname{poly}\log n$, because then the conditions of Lemma 9.3 are satisfied. On the other hand, if $\Delta_H = \operatorname{poly}\log n$, it follows immediately that $\widetilde{O}\left(\sum_v (d_H(v))^2\right) = \operatorname{poly}\log n$, which is $\widetilde{O}(1)$. Put together, we have $\sum_{H\in\mathcal{H}}\sum_{v\in H}(d_H(v))^2 = \widetilde{O}(m)$, which concludes the proof. $\square$

CHAPTER 10

MIS in Trees

10.1 Introduction

In this chapter—based on the publications 'Breaking the Linear-Memory Barrier in MPC: Fast MIS on Trees with Strongly Sublinear Memory' [44, 45]—we see how to adopt the shattering technique to the MPC setting, i.e., how to efficiently shatter a graph in this model, for maximal independent sets in trees.

10.1.1 Our Results and Related Work

We provide two different algorithms for the problem of MIS in trees. Our first algorithm in Theorem 10.1 is surprisingly simple and intuitive, but comes with a small overhead in the global memory, meaning that $\mathcal{MS}$ is superlinear in the input size $\Theta(n)$.

Theorem 10.1. *There is an $O(\log^2 \log n)$-round MPC algorithm that w.h.p. computes an MIS in n-vertex trees with $\mathcal{S} = \widetilde{O}\left(n^{\delta}\right)$ memory on each of $\mathcal{M} = \widetilde{O}\left(n^{1-\delta/3}\right)$ nodes, for any $0 < \delta < 1$.*

Our second algorithm in Theorem 10.2 gets rid of this overhead at the cost of some elegance and a factor $\log \log n$ in the running time.

Theorem 10.2. *There is an $O(\log^3 \log n)$-round MPC algorithm that w.h.p. computes an MIS in n-vertex trees with $\mathcal{S} = \widetilde{O}\left(n^{\delta}\right)$ memory on each of $\mathcal{M} = \widetilde{O}\left(n^{1-\delta}\right)$ nodes, for any $0 < \delta < 1$.*

The algorithms in Theorems 10.1 and 10.2 almost match the conditional lower bound of $\Omega(\log \log n)$ for MIS (on general graphs) due to Ghaffari, Kuhn, and Uitto [121], which holds unless there is an $o(\log n)$-round sublinear-memory MPC algorithm for connected components. This, in turn, is believed to be impossible under a popular conjecture [231].

Our algorithms improve almost exponentially on the $\widetilde{O}(\sqrt{\log n})$-round sublinear-memory MPC algorithms in concurrent works—for bounded-arboricity by Onak [201] and for general graphs by Ghaffari and Uitto [123] and by Czumaj, Davies, and Parter [71]—as well as on the algorithms directly adopted from the PRAM and LOCAL model: an $O(\log n)$-round algorithm for general graphs due to Luby [181] and independently Alon, Babai, Itai [5], and the $O(\sqrt{\log n})$-round algorithm for trees by Barenboim et al. and Ghaffari [28, 109], improving on $O(\sqrt{\log n} \cdot \log \log n)$ by Lenzen and Wattenhofer [170]. Note that for rooted trees, the PRAM and LOCAL algorithm of Cole and Vishkin [68] directly gives rise to an $O(\log^* n)$-round sublinear-memory MPC algorithm.

Moreover, our result shows that the memory per node can be reduced substantially from $\widetilde{\Omega}(n)$ to n^{δ} while not incurring a significant loss in the round complexity, compared to the recent $O(\log \log n)$-round MIS algorithm of Ghaffari et al. [112].

In independent concurrent works, Ghaffari and Uitto [123] and Onak have provided algorithms for the problems of maximal independent set and matching in general graphs in $\widetilde{O}(\sqrt{\log n})$ rounds. Czumaj, Davies, and Parter [71] provided a $O(\log \Delta + \log\log n)$-round deterministic algorithm for general graphs. In a subsequent work, Ghaffari, Grunau, and Jin [113] improved our result to $O(\log\log n)$ rounds.

10.1.2 Overview and Outline

We first explain some initial clean-up preprocessing which allows a simplified description of the algorithm. Then, we present a frequently used tool: a *gathering* algorithm that allows us to collect connected components of diameter D onto single nodes in $O(\log D)$ rounds. After this, we are ready to outline our algorithm.

Clean-Up in Preprocessing Phase

We argue how to get rid of the issue that potentially $\Delta = \omega(\mathcal{S})$ by a simple clean-up phase in the very beginning. To make the statements and arguments more readable, we think of this clean-up as having taken place already.

If the degree of a vertex is larger than the memory of a node, one needs to store several lower-degree copies of this vertex on different nodes. Here, we give a short argument for why one can assume without loss of generality that all incident edges of a vertex are stored on the same node. Notice that in a tree with n vertices, there can be at most $n^{1-\delta/2}$ vertices with degree at least $n^{\delta/2}$. If we now just ignore all these high-degree vertices and find an MIS among the remaining vertices, the resulting graph, after removal of all MIS vertices and their neighbors, has at most $n^{1-\delta/2}$ vertices. Repeating this argument roughly $2/\delta$ times gives an MIS in the whole input graph.

Efficient Gathering of Connected Components

One recurring theme of our algorithm is the following frequently used gathering tool which allows us to quickly gather all vertices of a connected component onto one node (provided that they fit there), and hence to quickly run local algorithms. It will come in two variants, which naturally give rise to Theorems 10.1 and 10.2, respectively. The proof is deferred to Section 10.3.

Lemma 10.3 (Gathering)**.** *Let G be any n'-vertex subgraph of a tree G consisting of connected components of size at most $k = O\left(n^{\delta/3}\right)$ and diameter at most D. Then there are*

a) an $O(\log D)$-round sublinear-memory MPC algorithm with $\mathcal{M} = \widetilde{O}\left(n^{1-\delta/3}\right)$ nodes and

b) an $O(\log D \cdot \log\log n)$-round sublinear-memory MPC algorithm with $\mathcal{M} = \widetilde{O}\left(n^{1-\delta}\right)$ nodes, if $n'D^3 = O(n)$,

that compute an assignment of vertices to nodes so that all the vertices of a connected component of G' are on the same node.

In concurrent works, independent of ours, [9, 15, 34] studied, among other problems, finding connected components in the sublinear-memory setting of MPC. In particular, Andoni et al. [9] give algorithms to find connected components and to root a forest with constant success probability, with $O(m)$ total memory in time $O(\log D \cdot \log\log n)$. While their results are more general, ours have the advantages of being (arguably) much simpler and deterministic. Furthermore, to turn their algorithm to work with high probability, the straightforward approach requires a logarithmic overhead in the total memory.

Algorithm Outline

Our algorithm is based on the shattering technique and correspondingly will consist of two phases: a shattering and a post-shattering.

Shattering: In Section 10.2, we prove the following lemma.

Lemma 10.4 (Tree MIS Shattering). *There are*

a) an $O(\log\log n \cdot \log\log \Delta)$*-round sublinear-memory* MPC *algorithm that uses* $\mathcal{M} = \widetilde{O}(n^{1-\delta/3})$ *nodes and*

b) an $O(\log^2 \log n \cdot \log\log \Delta)$*-round sublinear-memory* MPC *algorithm with* $\mathcal{M} = \widetilde{O}(n^{1-\delta})$ *nodes*

that compute an independent set on an n*-vertex tree with maximum degree* Δ *so that the remainder graph, after removal of the independent set vertices and their neighbors, w.h.p. has only components of size at most* $\operatorname{poly}\log n$*.*

Post-Shattering: The following Tree MIS Post-Shattering Lemma is a direct consequence of our Gathering Lemma in Lemma 10.3.

Lemma 10.5 (Tree MIS Post-Shattering). *There are*

a) an $O(\log k)$*-round* MPC *algorithm with* $\mathcal{M} = \widetilde{O}\left(n^{1-\delta/3}\right)$*, and*

b) an $O(\log k \cdot \log\log n)$*-round* MPC *algorithm with* $\mathcal{M} = \widetilde{O}\left(n^{1-\delta}\right)$

that find an MIS *in an* n*-vertex tree consisting of connected components of size* $k = O\left(n^{\delta/3}\right)$*.*

Proof. By Lemma 10.3, we can gather the connected components in $O(\log k)$ rounds and compute an MIS on each connected component offline. Theorem 1.1 by Ghaffari [109] certifies that the number of vertices remaining after our shattering process can be made small enough to satisfy the conditions required by Lemma 10.3. □

Note that the naïve simulation of the corresponding LOCAL post-shattering algorithm [109, 207] would lead to a round complexity of $2^{O(\sqrt{\log\log n})}$. Combining it with the recent work by Rozhoň and Ghaffari [216] and Ghaffari, Grunau, and Rozhoň [114] would yield an $O(\log^5 \log n)$-round algorithm.

10.2 Tree MIS Shattering Algorithm

10.2.1 Overview and Outline

Our shattering algorithm consists of a degree reduction phase and a low-degree LOCAL shattering phase.

Degree Reduction

The degree reduction phase reduces the maxmium degree[1] of the tree from Δ to $\operatorname{poly}\log n$ in $\log\log\Delta$ rounds.

Lemma 10.6 (Iterated Subsample-and-Conquer). *There are*

a) an $O(\log_{1+\delta}\log\Delta \cdot \log\log n)$-round sublinear-memory MPC algorithm with $\mathcal{M} = \widetilde{O}(n^{1-\delta/3})$ nodes and

b) an $O(\log_{1+\delta}\log\Delta \cdot \log^2\log n)$-round sublinear-memory MPC algorithm with $\mathcal{M} = \widetilde{O}(n^{1-\delta})$ nodes

that compute an independent set on an n-vertex tree with maximum degree Δ such that the remainder graph, after removal of the independent set vertices and their neighbors, w.h.p. has maximum degree $\operatorname{poly}\log n$.

The proof of this lemma can be found in Section 10.2.2.

[1]Note that in the MPC model it is easy to keep track of the maximum degree.

Low-Degree Shattering

Once the degree has dropped to $\Delta' = \operatorname{poly}\log n$, we apply the shattering part of the LOCAL MIS algorithm of Ghaffari [109], which runs in $O(\log \Delta') = O(\log\log n)$ rounds and w.h.p. leads to connected components of size $\operatorname{poly}\Delta' \cdot \log n = \operatorname{poly}\log n$ in the remainder graph. Observe that the simulation of this algorithm in the MPC model is straightforward.

Lemma 10.7 (Low-Degree LOCAL Shattering of [109])**.** *There is an $O(\log \Delta)$-round LOCAL algorithm that computes an independent set on an n-vertex graph with maximum degree Δ so that the remainder graph, after removal of all vertices in the independent set and their neighbors, w.h.p. has connected components of size* $\operatorname{poly}\Delta \cdot \log n$.

Wrap-Up

We now combine these two results to prove our Tree MIS Shattering Lemma in Lemma 10.4.

Proof of Lemma 10.4. We apply the algorithm of Lemma 10.6 which w.h.p. yields an independent set with a remainder graph that has maximum degree $\Delta' = \operatorname{poly}\log n$. On this low-degree graph, we simulate the LOCAL algorithm of Lemma 10.7 in a straightforward manner, by collecting the r-hop neighborhood of each vertex to a single node. Then a node with the r-hop neighborhood of a vertex u can derive what the r-round LOCAL algorithm would output for vertex u. This simulation takes $O(\log \Delta') = O(\log\log n)$ rounds and w.h.p. leaves us with connected components of size $\operatorname{poly}\Delta' \cdot \log n = \operatorname{poly}\log n$. □

10.2.2 Degree Reduction

Our degree reduction algorithm from Lemma 10.6 basically consists of $\operatorname{poly}\log\log\Delta$ repetitions of the *subsample-and-conquer* method.

Each iteration roughly reduces the maximum degree from Δ to Δ^{β} for some $0 < \beta < 1$ and can be summarized as follows.

Lemma 10.8 (Subsample-and-Conquer Algorithm)**.** *There are*

a) an $O(\log\log n)$-round sublinear-memory MPC algorithm with $\mathcal{M} = \widetilde{O}(n^{1-\delta/3})$ nodes and

b) an $O(\log^2 \log n)$-round sublinear-memory MPC algorithm with $\mathcal{M} = \widetilde{O}(n^{1-\delta})$ nodes

that compute an independent set in an n-vertex tree G with maximum degree $\Delta = \log^{\Omega(1)} n$ such that the remainder graph, after removal of the independent set vertices and their neighbors, w.h.p. has maximum degree at most Δ^{β} for some $\beta = \Theta\left(1/(1+\delta)\right)$.

The proof of this result is presented in the next section. Here we quickly show how it implies Lemma 10.6.

Proof of Lemma 10.6. This directly follows from Lemma 10.8, with $\log_{\frac{1}{\beta}} \log \Delta = O(\log_{1+\delta} \log \Delta)$ many applications. □

Degree Reduction via Iterated Subsampling

Before we dive into the proof of Lemma 10.8, we provide an intuitive discussion. The Subsample-and-Conquer algorithm of Lemma 10.8 can be summarized as follows.

Subsample: We sample the vertices independently with probability roughly $\Delta^{-\beta}$, This subsampling step guarantees, roughly speaking, the following three very desirable properties of the graph G' induced by the sampled vertices.

(i) The diameter of each connected component of G' is bounded by $O(\log_{\Delta} n)$.

(ii) The number of vertices in each connected component of G' is small enough to fit into the memory of a single node.

(iii) Every vertex with degree Δ^{β} or higher in G has $\Omega(\log n)$ neighbors in G'.

Conquer: We find a random MIS in all the connected components of G' in parallel. This can be done as follows. We first gather each connected component on a node, using the graph exponentiation technique. Here, Properties (i) and (ii) are crucial to ensure that the gathering can be done efficiently. In particular, storing the components on a single node is possible due to the small size of the components, and the gathering is fast due to the small diameter. Now every remaining component sits on a single node and hence can be processed offline. For every connected component, which again is just a tree, we pick one of the two possible 2-colorings uniformly at random, and add one of the two color classes to the MIS.

This allows us to conclude that every high-degree vertex (with degree at least Δ^{β}) will be removed from the graph. Indeed, a high-degree vertex v either is subsampled, in which case either v itself or one of its neighbors is added to the MIS; or v is not subsampled, but because of (iii) has many subsampled neighbors. Each of these neighbors will be part of a different connected component (since they are connected only via v, due to the acyclicity of the tree), and hence added to the MIS independently with probability $1/2$. A simple Chernoff and union bound argument hence lets us conclude that every non-subsampled vertex with high degree will have at least one neighbor that joins the MIS. Summarizing, all high-degree vertices (sampled or not), w.h.p. either join the MIS or have an adjacent independent set vertex, and thus are removed from the graph for the next iteration. A schematic depiction can be found in Figure 10.1.

For the purpose of the proof of Lemma 10.8 we assume that $\Delta = \log^{\Omega(1)} n$. Notice that from the perspective of the final round com-

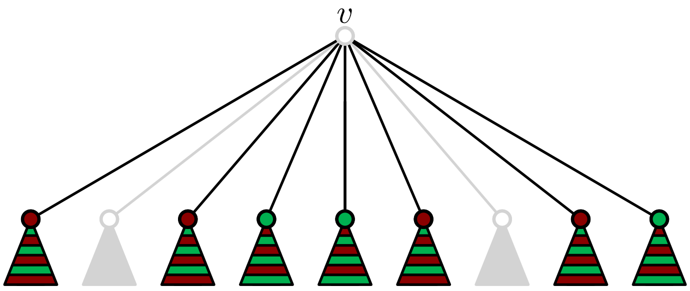


Figure 10.1: A high-degree vertex v has many subsampled neighbors. Each of those is added to the MIS (green) with probability 1/2 independently, depending on the 2-coloring of the corresponding subtree. Thus, w.h.p., v will have at least one neighboring vertex added to the MIS, and hence will be removed for the next iteration.

plexity, the exponent of the logarithm turns into a constant factor hidden in the O-notation.

Proof of Lemma 10.8. We first outline the algorithm and then slowly go through the steps of the algorithm again while proving its key properties. Throughout, let $\alpha' > 0$ be a small constant and let α be such that $(1 + \alpha')\alpha = \beta$.

Algorithm: Every vertex is sampled independently with probability $\Delta^{-\alpha}$ into a set V'. The connected components of $G' = G[V']$ are gathered by Lemma 10.3, and one of the two 2-colorings is picked uniformly at random, independently for every connected component. This can be done offline. All the black vertices, say, are added to the MIS, and are removed from the graph along with their neighbors.

Subsampling: We first prove that the random subsampling leads to nice properties of the graph induced by subsampled vertices that allows us to efficiently gather its components using our Gathering Lemma in Lemma 10.3.

Claim 10.9. *After the subsampling, w.h.p. the following holds.*

(i) Every connected component of G' has diameter $O\left(\frac{1}{\alpha}\log_\Delta n\right)$.

(ii) Every connected component of G' consists of $n^{O((1-\alpha)/\alpha)}$ vertices.

(iii) Every vertex with degree $\Omega\left(\Delta^\beta\right)$ in G has degree $\log^{\Omega(1)} n$ in G'.

Proof. Consider an arbitrary path of length $\ell = \Omega\left(\frac{1}{\alpha}\log_\Delta n\right)$ in G. This path is in G' only if all its vertices are subsampled into V', which happens with probability at most $\Delta^{-\alpha\ell} = n^{-\Omega(1)}$. A union bound over all—at most n^2 many—paths in the tree T shows that w.h.p. the length of every path, and hence in particular also the diameter of every connected component, in G' is bounded by $O\left(\frac{1}{\alpha}\log_\Delta n\right)$. Since the degree among the subsampled vertices w.h.p. is bounded by $O\left(\Delta^{1-\alpha}\right)$, which is a simple application of Chernoff and union bound, it follows that every connected component consists of at most $O\left(\Delta^{(1-\alpha)\ell}\right) = n^{O((1-\alpha)/\alpha)}$ vertices. Finally, another simple Chernoff and union bound argument shows that every vertex with degree $\Omega\left(\Delta^\beta\right)$ in the graph G has at least $\Omega\left(\Delta^{\alpha'\alpha}\right) = \log^{\Omega(1)} n$ neighbors in G', which concludes the proof of Claim 10.9. □

Gathering: Since G' consists of components that have a low diameter by Claim 10.9 (i) and that are small enough to fit on a single node by Claim 10.9 (ii)—by choosing $\alpha = \Theta\left(1/(1+\delta)\right)$ small enough such that $n^{O(\alpha/(1-\alpha))} = O\left(n^{\delta/3}\right)$—we can gather them efficiently by Lemma 10.3, in either $O(\log\log n)$ or $O(\log^2\log n)$ rounds. The random MIS can then be easily computed offline.

Random MIS: It remains to show that w.h.p. every high-degree vertex in G has at least one adjacent vertex that joins the random MIS, which leads to the removal of this high-degree vertex from the graph. This is trivially true for all subsampled vertices.

Now consider an arbitrary non-subsampled vertex v with degree $\Omega\left(\Delta^{\beta}\right)$ and its $\log^{\Omega(1)} n$ subsampled neighbors, by Claim 10.9 (iii). Observe that, since we are in a tree and thus in particular in a triangle-free graph, there cannot be edges between these neighbors. Therefore no two neighbors of a non-subsampled vertex belong to the same connected component in G', which means that all the neighbors in V' of v are colored independently, and hence are added to an MIS independently with probability 1/2. By the Chernoff inequality, w.h.p. at least one of v's neighbors must have been added to an MIS, and a union bound over all vertices concludes the proof of the degree reduction, and hence of Lemma 10.8. □

10.3 Gathering Connected Components

We provide a proof of the *Gathering Lemma* in Lemma 10.3. Our approach is essentially a tuned version of the Hash-to-Min algorithm of Chitnis et al. [66] and the graph exponentiation idea by Lenzen and Wattenhofer [169]. Notice that, however, Chitnis et al. only show an $O(\log n)$ bound for the round complexity; it is not possible to just use their method as a black box. The section is divided into two subsections, where we first give a simple and fast but memory-inefficient algorithm and then present a slightly slower algorithm that only needs a constant space overhead.

We present the naïve gathering algorithm in Section 10.3.1 and the in-space gathering in Section 10.3.2.

10.3.1 Naïve Gathering

We first present the algorithm. The underlying idea of the algorithm is to find a minimum-ID[2] vertex within every component and to cre-

[2]We assume without loss of generality that every vertex has a unique identifier. If not, every vertex can draw an $O(\log n)$-bit identifier at random, which w.h.p. will be unique.

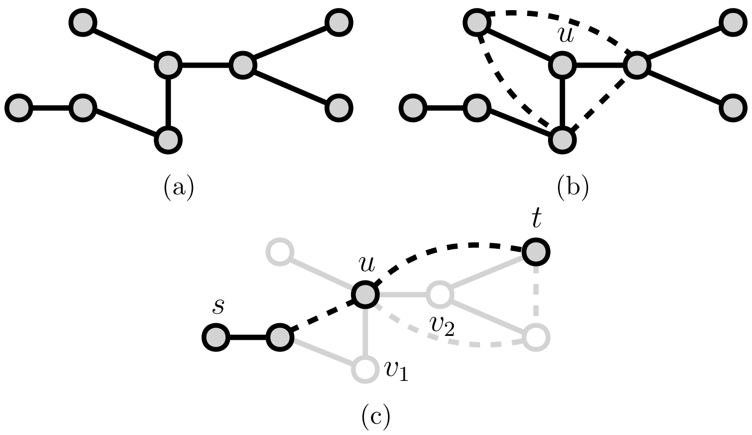


Figure 10.2: We illustrate the gathering algorithm with help of the tree depicted in (a). The edges added by vertex u are illustrated in (b) by dashed arcs. In (c) we display how the edges added by vertices v_1 and v_2, drawn as dashed arcs, shortcut the shortest path between vertices s and t.

ate a virtual graph that connects all the vertices of that component to this minimum-ID vertex, the *leader*.

Gathering Algorithm

In every round, every vertex u completes its 1-hop neighborhood to a clique. Once a round is reached in which there are no more edges to be added, u stops and selects its minimum-ID neighbor as its leader. We refer to Figure 10.2 for an illustration.

Observe that once there is a round in which u does not add any edges, the component of u forms a clique, and thus all vertices in this component have the same leader, namely the minimum-ID vertex in this clique.

Round Complexity

Next, we prove that this algorithm terminates quickly.

Claim 10.10. *The gathering algorithm takes $O(\log D)$ rounds on a graph with diameter D.*

Proof. Consider any shortest path $u_1, \ldots, u_\ell$ of length $2 \leq \ell \leq D$. After the first round, every u_i gets connected to u_{i-2} and u_{i+2} for $2 < i < \ell - 1$. Thus, the diameter of the new graph is at most $\lceil 2D/3 \rceil$. After $O(\log D)$ iterations, the diameter within each component has reduced to 1, and the algorithm halts. □

Memory Analysis

It remains to show that not too many edges are added, so that the virtual graph of any component still fits into the memory of a node.

Claim 10.11. *The number*[3] *of edges in the virtual graph created by the gathering algorithm in a component of size k is $O(k^3)$.*

Proof. During the execution of the algorithm, each vertex in a component may create an edge between any other two vertices in the corresponding component, thus at most k^3. □

Since the components need to be of size at most $O(n^{\delta/3})$, the previous claim guarantees that the virtual graph of any connected component indeed fits into the memory. So as to not overload any node with too many components, we assume that the shuffling distributes the components to the nodes in an arbitrary feasible way, e.g., greedily[4].

[3]Note that here we count edges multiple times if they are (possibly) stored multiple times (on different nodes), which allows for a number of edges that is larger than the number of possible distinct edges in a component of size k.

[4]An alternative and simple way to prevent overloading is to add a factor

10.3.2 In-Space Gathering for Trees

The simple and naïve gathering algorithm can be very wasteful in terms of space usage over the whole system. A main weakness is that we need $O(k^3)$ memory to store a connected component of size k, even if this component originally just consisted of as few as $k-1$ edges. This is because a single edge can exist on up to k nodes. In the worst case, the required memory is blown up by a power 3. This leads to a super-linear overall memory requirement, that is, we need roughly $N^{1+2\delta/3}$ total memory in the system. Notice that this can be implemented either by adding more nodes or by adding more memory to the nodes, since we do not care on which nodes the resulting components lie, as long as they fit in the memory.

We now provide a fine-tuned version of the gathering method that works asymptotically in space. In other words, the total space requirement drops to $O(n)$. This proves part (b) of Lemma 10.3.

Informally, our algorithm first turns every connected component into a rooted tree and then determines which vertices are contained in the same tree component by making sure that each vertex learns the ID of the root of its tree.

Determining the Root

Lemma 10.12. *There is an $O(\log D)$-round MPC algorithm that works in an n-vertex forest of rooted trees with maximum diameter D and, for every vertex, determines the root of the corresponding tree. The algorithm requires $\mathcal{M} = O\left(n^{1-\delta}\right)$ nodes.*

Proof. Let $\mathsf{parent}(v)$ denote the parent of vertex v, where we define $\mathsf{parent}(r) = r$ for a root vertex r. Consider the following pointer-forwarding algorithm (inspired by the graph exponentiation tech-

$O(\log n)$ of memory per node and consider a random assignment of components to nodes as a balls-into-bins process.

nique) that is run in parallel for every vertex v. In every round, for every child u of v, we set $\mathsf{parent}(u) := \mathsf{parent}(v)$. The process terminates once v points to a root, i.e., to a vertex r for which $\mathsf{parent}(r) = r$. Notice that after every step, following the parent pointers still leads to the root vertex.

Let $(v_1, v_2, \ldots, v_k)$ be the directed path from vertex v_1 to the root $r = v_k$ of its subtree in round t. After one round of the algorithm, every v_i is connected to $v_{\min\{i+2,k\}}$. Thus, the length of the path is at most $\lceil k/2 \rceil$. After $O(\log k) = O(\log D)$ rounds the algorithm terminates yielding the claim. □

Rooting a Tree

Given Lemma 10.12, what remains to show for our algorithm is how to root a tree. The idea is to once more use the graph exponentiation method to learn an ℓ-hop neighborhood of a vertex in $\log \ell$ steps. However, in order to prevent the space requirement from getting out of hand, each vertex performs only a bounded number of exponentiation steps, after which all vertices that already know their parent in the output orientation are removed from the graph. Then this process is iterated until at most one vertex (per connected component) remains.

Tree Rooting Algorithm $\mathcal{A}$: In the following, we give a formal description of an algorithm $\mathcal{A}$ for rooting a tree of diameter D. The algorithm takes an integer B as input parameter that describes the initial memory *budget* for each vertex v, i.e., an upper bound on the number of edges that v may add before the first vertex removal. The execution of $\mathcal{A}$ is subdivided in phases $i = 0, 1, \ldots$ which consist of $O(\log D)$ rounds each. Set $B_0 = B$.

Phase i of $\mathcal{A}$: In phase i, each vertex v does the following:

In round 0, vertex v sets its local budget B_v to B_i. In each following round $j = 1, 2, \ldots$, vertex v first connects its 1-hop neighborhood to a clique by adding edges between all its neighbors that are not connected yet, but it does so only if the number of added edges is at most B_v. Then v updates its local budget by decreasing B_v by the number of edges that v added. If B_v was not large enough to connect v's 1-hop neighborhood to a clique, then v does not add any edges in round j. This concludes the description of round j, of which there are $O(\log D)$ many.

Denote the tree at the beginning of phase i by T_i, and for each neighbor u of v, denote the set of vertices that are closer to u than v in T_i by $S^i_u(v)$. Phase i concludes with a number of special rounds: First, v checks whether it has a neighbor u' in T_i with the following properties:

(i) $S^i_u(v)$ is contained in the current 1-hop neighborhood of v, for each neighbor u of v in T_i satisfying $u \neq u'$.

(ii) $S^i_{u'}(v)$ is not (entirely) contained in the current 1-hop neighborhood of v.

If such a neighbor u' exists (which, by definition, is unique), then v sets $\mathsf{parent}(v) = u'$. Second, v removes all edges that it added during phase i (regardless of whether a parent is set). Third, v is removed from T_i if it already chose its parent, i.e., if it set $\mathsf{parent}(v)$. Fourth, the budget per vertex is updated, by setting $B_{i+1} = B_i \cdot n_i/n_{i+1}$, where n_i and n_{i+1} are the numbers of vertices of T_i and T_{i+1}, respectively. This concludes the description of phase i.

We execute this process until at most one vertex remains.

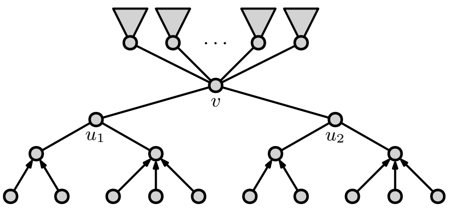
Termination of $\mathcal{A}$: Since in each phase (at the very least) all leaves are removed, this process eventually terminates.

It is straightforward to check that if a vertex v chooses its parent $u' = \mathsf{parent}(v)$ in phase i, then any neighbor $u \neq u'$ of v in T_i also chooses its parent in phase i, and, what is more, u chooses v as its parent (which, combined with the following observations, shows that the orientation of the input tree induced by the parent choices of the vertices yields indeed a rooted tree). Hence, given the above process, one of two things happens in the end: either exactly one vertex remains, or all vertices are removed but there is exactly one pair of vertices that chose each other as their parent. In the former case, no action has to be taken, as the remaining vertex is simple the root of our rooted tree. In order to handle the latter case, we add a simple fifth special round at the end of each phase i: Each vertex v removed in phase i checks whether the vertex it chose as its parent chose v as its parent. If this is the case, then the vertex with the higher ID removes its choice of parent and becomes the root vertex of the input tree. See Figure 10.3 for an illustration of algorithm $\mathcal{A}$.

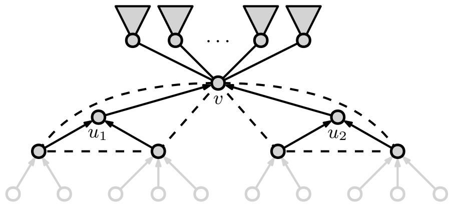
Running Time of $\mathcal{A}$: We present a number of lemmas in order to determine the round complexity of algorithm $\mathcal{A}$. Here, a subtree $T(v)$ rooted at some vertex v corresponds to the descendants of v in the rooted tree T returned by $\mathcal{A}$ (or in the rooted subtree of T induced by the vertices of some T_i).

Lemma 10.13. *Consider some arbitrary phase i, and let $T(v)$ be the subtree of T_i rooted at v. If $|T(v)| \leq \sqrt{B_i}$, then v chooses its parent in phase i and is removed from the tree.*

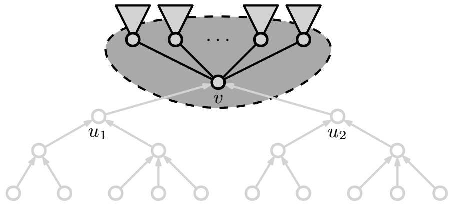
Proof. Let k be some arbitrary non-negative integer, and consider any vertex u in $T(v)$ with distance at most 2^k to v. Observe that, according to $\mathcal{A}$, the distance between any two vertices in T_i decreases by a factor of at most 2 per round. Hence, after round k of phase i, all vertices contained in the 1-hop neighborhood of u are actually also contained in $T(v)$. Thus, each edge that u would have added if it had connected its 1-hop neighborhood to a clique in each of the rounds $1, \dots, k+1$, disregarding any budget constraints, is an edge between vertices from $T(v)$. Since $|T(v)| \leq \sqrt{B_i}$, the number of edges between vertices from $T(v)$ is bounded from above by B_i; it follows that u had enough budget to indeed connect its 1-hop



(a) The leaves have only one neighbor, which becomes their parent. Vertices u_1, u_2, and v do not have enough budget to add edges.



(b) Once the leaves are removed, enough budget is freed for vertices u_1 and u_2 to add edges that connect their neighbors.



(c) Once u_1 and u_2 know that v is their parent, vertex v can focus its budget to the remainder of the tree (grey area).

Figure 10.3: The steps performed by algorithm $\mathcal{A}$. Assume that the initial budget is 2. As illustrated in (a), leaves are always able to determine their parent. Assuming that the tree has non-leaf minimum degree 3, removing all the leaves at least roughly doubles the budget of all vertices. Thus, in the second step, shown in (b), vertices with degree at most 4 are able to complete their 1-hop neighborhoods into a clique. Small subtrees rooted at (or connected to) v are removed quickly in our process, as depicted in (c). Therefore, v requires large subtrees to survive for many phases.

neighborhood to a clique in each round up to and including round $k+1$.

Now consider any vertex w whose distance to v in T_i is at least 2^k, but at most $2^{k+1}-1$. Let $w_0, \ldots, w_k$ be vertices on the unique path between v and w with distance $2^0, \ldots, 2^k$ to v. Due to the observations above, it is straightforward to check that, in each round $1 \leq h \leq k$, vertex w_{h-1} connects vertex w_h to v, while in round $k+1$, vertex w_k connects vertex u to v. Since the depth of $T(v)$ is upper bounded by $\log D$, it follows that after $\log D$ rounds, all vertices from $T(v)$ are contained in v's 1-hop neighborhood. Hence, v will choose the only neighbor that is not contained in $T(v)$ as its parent, and v is removed in phase i. Since k was chosen arbitrarily, the lemma statement follows. □

Lemma 10.14. *Let T be a rooted tree with n vertices and diameter at most D. Let $1 \leq \beta \leq n$, and let C be the set of vertices v with the property that $|T(v)| \leq \beta$. Then, $|C| \geq n(\beta/(D+\beta))$.*

Proof. Assign one dollar to each vertex that is not contained in a subtree of size at most β. Every such vertex then distributes its dollar evenly among all of its descendants in C. Note that, for each leaf vertex w of the tree obtained from T by deleting all vertices in C, the number of descendants of w in C is at least β since otherwise w would be in C, by the definition of C. Hence, all vertices that are not contained in C have at least β descendants in C.

Consider then any vertex $v \in C$. Since the diameter of the tree is D, vertex v can have at most D ancestors in T. Every ancestor of v distributes at most $1/\beta$ dollars to v and therefore, v receives at most D/β dollars. As the amount of dollars did not change during its redistribution from vertices not contained in C to vertices in C, we can conclude that $|C|(D/\beta) \geq n - |C|$ which implies that $|C| \geq n(\beta/(D+\beta))$. □

Lemma 10.15. *Assume that the input parameter B for our algorithm $\mathcal{A}$ satisfies $B \geq D^3$. Then the round complexity of $\mathcal{A}$ in trees with n vertices and diameter D is $O(\log D \cdot \log\log n)$.*

Proof. Observe that the sequence $B_0, B_1, \ldots$ of budgets at the beginning of phases $0, 1, \ldots$ is monotonically non-decreasing, by definition. Hence, $B_i \geq D^3$ for all phases i. Now consider some arbitrary phase i, and let n_i denote the number of vertices of T_i. By Lemma 10.13 and Lemma 10.14, the number of vertices that are removed in phase i is at least $n_i \cdot (\sqrt{B_i}/(D + \sqrt{B_i}))$. Thus, for the new budget B_{i+1}, it holds by definition that

$$B_{i+1} \geq B_i \frac{1}{1 - \frac{\sqrt{B_i}}{D+\sqrt{B_i}}} = B_i \left(\frac{D + \sqrt{B_i}}{D} \right) \geq \frac{B_i^{\frac{3}{2}}}{D} \ .$$

Since, as observed above, $D \leq B_i^{1/3}$, we obtain $B_{i+1} \geq B_i^{7/6}$, which implies $B_{i+5} \geq B_i^2$. Recall that in each phase i, at least a $(\sqrt{B_i}/D)$-fraction of vertices is removed. Thus, after $O(\log\log n)$ phases, all vertices (except possibly for one vertex) have been removed and the termination condition of $\mathcal{A}$ is satisfied. Since every phase takes $O(\log D)$ time, the claim follows. □

Now we have all the ingredients to prove the second part of our gathering lemma in Lemma 10.3. It is a simple corollary of the following theorem.

Theorem 10.16. *Consider a forest F of n vertices where every tree is of diameter at most D. There is an* MPC *algorithm that finds the connected components of F in time $O(\log D \cdot \log\log n)$ where $\mathcal{MS} = O(nD^3)$.*

Proof. Imagine that we run algorithm $\mathcal{A}$ in parallel on all trees of the input forest F, with input parameter $B = D^3$. There are only two parts of $\mathcal{A}$ that are of a global nature, i.e., where the actions of vertices do not depend on their immediate neighborhood: the termination condition that all vertices, possibly except for one, have been removed, and the part where the vertex's budgets are updated from B_i to B_{i+1}. The former is easily adapted to the case of forests; each vertex simply terminates when itself or all its neighbors are removed. Regarding the updating of the budget, we adapt the tree rooting algorithm as follows: we still set the new budget B_{i+1} to $B_i n_i / n_{i+1}$, but now n_i and n_{i+1} denote the total number of vertices (i.e., in all trees of the forest) that have not been removed yet at the beginning of phase i, respectively phase $i + 1$.

In the following, we verify that Lemmas 10.13 to 10.15 also hold for forests instead of trees. In the case of Lemma 10.13, this is obvious as the argumentation is local and thus also applies to forests. Lemma 10.14 trivially also holds for forests since the lemma statement holds for all trees in the forest. Finally, since the argumentation of the proof of Lemma 10.15 does not make use of the fact that the input graph is a tree except when applying Lemmas 10.13 and 10.14, it follows that Lemma 10.15 also holds for forests.

Hence, our adapted tree rooting algorithm actually transforms the forest into a rooted forest in time $O(\log D \cdot \log\log n)$. Now we can apply Lemma 10.12, and, e.g., color each component with the color of the root vertex, thereby marking the connected components. Due to the round complexity given in Lemma 10.12, our total round complexity is still $O(\log D \cdot \log\log n)$.

It remains to show that the claimed memory constraints are satisfied. Due to the space guarantee given in Lemma 10.12, it is sufficient to show that the memory overhead induced by adding edges during the execution of out forest rooting algorithm does not exceed the allowed amount. Thus, consider the number of edges added in an arbitrary phase i. Since each vertex adds at most as many edges as its budget allows, i.e., at most B_i edges, the total number of edges added in phase i is upper bounded by $n_i B_i$. By the definition of B_{j+1}, we have $n_{j+1}B_{j+1} = n_jB_j$, for any phase j. Hence, the value of n_iB_i is the same for every phase i, and we obtain $n_iB_i = n_0B_0 = nD^3$. Therefore, the number of edges added in any phase i does not exceed nD^3, and since all added edges are removed again at the end of each phase, the lemma statement follows. □

Remark 10.17. *In the analysis, we implicitly assumed that edges incident on vertices are always added only once. It could, however, be the case that some vertex is unlucky and many of its neighbors add a copy of the same edge many times. This misfortune could potentially result in adding n^δ copies of the same (virtual) edge, which could, in turn, overload the memory per node constraint on the nodes containing these unlucky vertices. For the sake of simplicity, we decided to leave this problem to the shuffling algorithm of the underlying* MPC *framework that can, for example, load the vertices onto the nodes greedily after each communication step. Since the total memory constraint is satisfied, this is always feasible. Alternatively, the shuffling algorithm could simply drop duplicate messages.*

CHAPTER 11

MM and MIS in Uniformly Sparse Graphs

11.1 Introduction

In the following—based on the manuscript 'Matching and MIS for Uniformly Sparse Graphs in the Low-Memory MPC Model' [43] and the publication 'Massively Parallel Computation of Matching and MIS in Sparse Graphs' [32]—we drastically generalize the degree reduction method from Chapter 10 from trees to uniformly sparse graphs, i.e., graphs with arboricity $\lambda = \operatorname{poly}\log n$.

11.1.1 Our Results and Related Work

Maximal Matching and Maximal Independent Set

Our two main results can be summarized as follows.

Theorem 11.1. *There is a sublinear-memory MPC algorithm that w.h.p. computes a maximal independent set in a graph with arboricity λ in $O\left(\sqrt{\log \lambda} \cdot \log\log \lambda + \log\log n \cdot \log\log \Delta\right)$ rounds.*

Theorem 11.2. *There is a sublinear-memory MPC algorithm that w.h.p. computes a maximal matching in a graph with arboricity λ in $O\left(\sqrt{\log \lambda} \cdot \log\log \lambda + \log\log n \cdot \log\log \Delta\right)$ rounds.*

This improves on the $O\left(\log \lambda + \sqrt{\log n}\right)$-round LOCAL algorithm of [28, 109] and on the $\widetilde{O}\left(\sqrt{\log \Delta}\right)$-round algorithm of [123]. The only previous subpolylogarithmic round maximal matching algorithm due to Lattanzi et al. [167], requires space per node of $n^{1+\Omega(1)}$. Recently, Czumaj, Davies, and Parter [71] provided a $O(\log \Delta + \log\log n)$-round deterministic algorithm for general graphs.

For all graphs with arboricity up to poly $\log n$, our algorithms take only $O(\log^2 \log n)$ rounds. This improves almost exponentially on the known sublinear-memory MPC algorithms: the $\widetilde{O}(\sqrt{\log \Delta})$-round algorithm due to Ghaffari and Uitto [123] and the $O\left(\log \lambda + \sqrt{\log n}\right)$-round algorithm due to a straightforward simulation of the LOCAL algorithm of Barenboim et al. [27, 28]. Previously, the only known subpolylogarithmic algorithm, due to Lattanzi et al. [167], required strongly superlinear memory per node;

We also note that the round complexity of all previous poly $\log\log n$-round MPC matching approximation algorithms for general graphs (even if we allow linear memory) [73, 12, 112] blows up by an $\Omega(\log n)$ overhead for the case of maximal matching. In fact, the problem of finding a maximal matching seems to be much more difficult than finding a $(1+\varepsilon)$-approximate maximum matching. Indeed, currently, an $O(1)$-approximation can be found almost exponentially faster than a maximal matching, and the approximation ratio can be easily improved from any constant to $1+\varepsilon$ using a reduction of McGregor [183] (also see Corollary 11.3).

In a subsequent work, Ghaffari, Grunau, and Jin [113] provided an $O(\sqrt{\log \lambda} \cdot \log\log \lambda + \log\log n)$-round algorithm for MIS, MM, and coloring.

Approximate Maximum Matching and Vertex Cover

As a maximal matching automatically provides 2-approximations for maximum matching and minimum vertex cover, Theorem 11.2 directly implies the following result.

Corollary 11.3. *There is a sublinear-memory* MPC *algorithm that w.h.p. computes a* 2*-approximate maximum matching as well as a* 2*-approximate minimum vertex cover in a graph with arboricity* λ *in* $O\left(\sqrt{\log \lambda} \log\log \lambda + \log\log n \cdot \log\log \Delta\right)$ *rounds.*

McGregor's reduction [183] allows us to further improve the approximation to $1 + \varepsilon$.

Corollary 11.4. *There is a sublinear-memory* MPC *algorithm that for any* $\varepsilon > 0$ *w.h.p. computes a* $(1+\varepsilon)$*-approximate maximum matching in* $O\left(\left(\frac{1}{\varepsilon}\right)^{O(1/\varepsilon)} \left(\sqrt{\log \lambda} \cdot \log\log \lambda + \log\log n \cdot \log\log \Delta\right)\right)$ *rounds.*

Due to a reduction by to Lotker, Patt-Shamir, and Pettie [177], our constant-approximate matching algorithm can also be employed to find a $(2+\varepsilon)$-approximate maximum weighted matching.

Corollary 11.5. *There is a sublinear-memory* MPC *algorithm that in* $O\left(\log \frac{1}{\varepsilon} \left(\sqrt{\log \lambda} \cdot \log\log \lambda + \log\log n \cdot \log\log \Delta\right)\right)$ *rounds gives a* $(2+\varepsilon)$*-approximate weighted matching w.h.p., for any* $\varepsilon > 0$.

11.1.2 Overview and Outline

Degree Reduction

The main ingredient of our approach is a degree reduction technique that reduces the problems of MM and MIS in a graph with arboricity λ to the corresponding problems in graphs with maximum degree poly λ in $O\left(\log^2 \log n\right)$ rounds.

Theorem 11.6 (Degree Reduction). *There is a sublinear-memory MPC algorithm that w.h.p. reduces maximal matching and maximal independent set in graphs with arboricity $\lambda = n^{o(1)}$ to the respective problems in graphs with maximum degree $O\left(\max\{\lambda^{20}, \log^{20} n\}\right)$[1] in $O\left(\log\log_{\Delta} n \cdot \log\log_{\lambda} \Delta\right)$ rounds.*

This improves on the degree reduction algorithm of [28, Theorem 7.2], which runs in $O(\log_{\lambda} n)$ rounds in the LOCAL model and can be straightforwardly implemented in sublinear-memory MPC.

Polynomial Degree Reduction

Our degree reduction algorithm in Theorem 11.6 consists of several phases, each reducing the maximum degree by a polynomial factor, as long as the degree is still large enough.

Lemma 11.7. *There are $O\left(\log\log n\right)$-round sublinear-memory MPC algorithms that compute a matching and an independent set, respectively, in a graph with arboricity $\lambda = n^{o(1)}$ and maximum degree $\Delta = \Omega\left((\max\{\lambda, \log n\})^{20}\right)$ so that the remainder graph w.h.p. has maximum degree $O(\Delta^{0.4})$.*

We first show that indeed iterated applications of this polynomial degree reduction lead to the desired degree reduction in Theorem 11.6.

[1]The purpose of the choice of all the constants in this work is merely to simplify presentation.

Proof of Theorem 11.6. We iteratively apply the polynomial degree reduction from Lemma 11.7, observing that as long as the maximum degree is still in $\Omega(\lambda^{20})$ and $\Omega(\log^{20} n)$, we reduce the maximum degree by a polynomial factor from Δ to $O(\Delta^{0.4})$ in each phase, resulting in at most $O(\log\log\Delta)$ phases. □

The polynomial degree reduction, as claimed in Lemma 11.7, is proved in two parts. First, in Section 11.2, we provide a centralized algorithm, and then, in Section 11.2.2, we show how to implement this centralized algorithm efficiently in MPC.

Wrap-Up

Theorems 11.1 and 11.2 follow from Theorem 11.6 and from an efficient simulation of LOCAL algorithms due to [123].

Proof of Theorems 11.1 and 11.2. If λ, and hence Δ, is at least polynomial in n, we directly apply the algorithm of [123], which runs in $O(\sqrt{\log\Delta}\cdot\log\log\Delta+\sqrt{\log\log n}) = O(\sqrt{\log n}\cdot\log\log n)$ rounds. Otherwise, we first apply the algorithm of Theorem 11.6 to obtain a partial solution that reduces the degree in the remainder graph to $\Delta' = O(\lambda^{20})$ if $\lambda \geq \log n$, or to $\Delta' = O(\log^{20} n)$ if $\lambda \leq \log n$. It runs in $O(\log\log n\cdot\log\log\Delta)$ rounds. In $O(\sqrt{\log\Delta'}\cdot\log\log\Delta'+\sqrt{\log\log n}) = O(\sqrt{\log\lambda}\cdot\log\log\lambda+\sqrt{\log\log n})$ rounds, we then apply the algorithm of [123] on the remainder graph. □

Remark 11.8. *While our algorithms, at first sight, seem to need to know λ, we can easily get rid of this assumption by employing the standard technique [155] of running the algorithm with doubly-exponentially increasing estimates for λ. Roughly speaking, the idea is as follows. We run $O(\log\log\Delta)$ copies, one for every estimate of λ, of the algorithm in parallel and pick (one of) the correct solution(s), which is easy to find in MPC.*

11.2 Degree Reduction

11.2.1 Centralized Degree Reduction Algorithm

In this section, we present a centralized algorithm for the polynomial degree reduction as stated in Lemma 11.7. For details on how this algorithm can be implemented in the sublinear-memory MPC model, we refer to Section 11.2.2. In Section 11.2.1, we give a formal description of the (centralized) algorithm. Then, in Section 11.2.1, we prove that this algorithm indeed leads to a polynomial degree reduction.

Algorithm Description

In the following, we set $d = \Delta^{1/10}$, and observe that $d = \Omega(\lambda^2)$ as well as $d = \Omega(\log^2 n)$, due to the assumptions on Δ in the lemma statement. We present an algorithm that reduces the maximum degree to $O(d^4)$. This algorithm consists of three phases: a *partition* phase, in which the vertices are partitioned into layers so that every vertex has at most d neighbors in higher-index layers, a *mark-and-propose* phase in which a random set of candidates is proposed independently in every layer, and a *selection* phase in which a valid subset of the candidate set is selected as partial solution by resolving potential conflicts across layers.

Partition Phase: We compute an H-partition, that is, a partition of the vertices into layers so that every vertex has at most d neighbors in layers with higher (or equal) index. Such a partition can be computed in $O(\log_{d/\lambda} n)$ rounds using the sequential peeling algorithm described in Section 3.1.3.

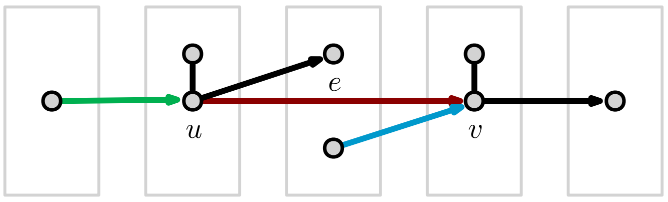
Mark-and-Propose Phase: We first mark a random set of candidates (either edges or vertices) and then propose a subset of these marked candidates for the partial solution as follows. In the case of maximal matching, every vertex first marks an outgoing edge chosen uniformly at random and then proposes one of its incoming marked edges, if any, uniformly at random. In the case of maximal independent set, every vertex marks itself independently with probability $p = d^{-2}$. Then, if a vertex is marked and none of its neighbors in the same layer is marked, this vertex is proposed. Note that whether a marked vertex gets proposed only depends on vertices in the same layer, thus on neighbors with respect to unoriented edges.

Selection Phase: The set of proposed candidates might not be a valid solution, meaning that it might have some conflicts (i.e., two incident edges or two neighboring vertices). In the selection phase, possible conflicts are resolved (deterministically) by picking an appropriate subset of the proposed candidates, as follows. Iteratively, for $i = \ell, \ldots, 1$, all (remaining) proposed candidates in layer i are added to the partial solution and then removed from the graph. In the case of maximal matching, we add all (remaining) proposed edges directed to a vertex in layer i to the matching and remove both their endpoints from the graph. In the case of maximal independent set, we add all (remaining) proposed vertices in layer i to the independent set and remove them along with their neighbors from the graph.

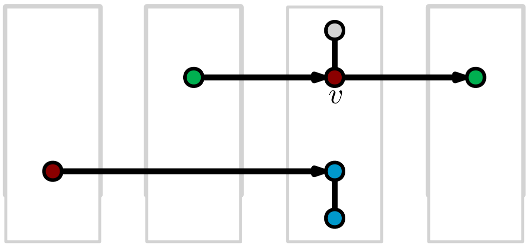
Proof of Correctness

Conflict-Free: It is easy to see that the selected solution is a valid partial solution, that is, that there are no conflicts. See Figure 11.1 for an illustration.

Indeed, for the case of the matching, observe the following: An (oriented) edge $e = (u, v)$ that is selected to be added to the matching cannot have an incident edge that is also selected: an unoriented incident edge cannot be marked as only oriented edges are marked; an oriented edge with the same starting point u cannot be marked as u marks only one outgoing edge; an oriented edge with the same endpoint v cannot be proposed as v proposes only one incoming



(a) the case of maximal matching



(b) the case of maximal independent set

Figure 11.1: Illustration of the mark-and-propose and the selection phase for matching in (a) and independent set in (b). Blue indicates marked but not proposed, green stands for (marked and) proposed but not selected, and red means (marked and proposed and) selected. Note that we omitted all (but a few) irrelevant edges from the figure; the partition into layers thus might not correspond to a valid H-partition.

edge; all other oriented edges f are either processed before (in the case of an outgoing edge from v) or after (in the case of an incoming edge to u) edge e in the selection phase. In the former case, the selection of f would lead to the removal of e before e is processed; e thus would not be selected. In the latter case, the edge f is removed immediately after e is selected (and thus before f is processed), and thus cannot be selected.

Similarly, for independent set, the following holds: If two neighboring vertices are marked, none of them will be proposed, and consequently, none of them will be selected. A vertex v that is selected to be added to the independent set cannot have a neighbor that is also selected: a neighbor in the same layer cannot be marked as otherwise v would not be proposed; a neighbor in a lower-index layer is removed from the graph when v joins the independent set, and hence before it potentially could be selected; a selected neighbor in a higher-index layer would lead to v's immediate removal from the graph; when v's layer is processed, v is not part of the graph anymore, and thus could not be selected.

Degree Reduction: It remains to show that the degree drops to $O(d^4)$. As the outdegree is bounded by d, the following is enough.

Lemma 11.9. *Every vertex with indegree at least d^4 gets removed or all but d^4 of its incoming edges get removed, with high probability.*

Proof of Lemma 11.9 for matching. Let v be a vertex with degree at least d^4. First, observe that if at least one incoming edge of v is proposed, then an edge incident to v (not necessarily incoming) will be selected to be added to the matching. This is because the only reason why v would not select a proposed incoming edge is that v has already been removed from the graph, and this happens only if its proposed edge has been selected to be added to the matching in a previous step. It thus remains to show that every vertex v with indegree at least d^4 with high probability will have at least one incoming edge that has been proposed by the respective child. As every incoming edge of v is marked independently with probability at least $1/d$, the probability of v not having a marked, and hence not having a proposed incoming edge is at most $(1-1/d)^{d^4} \leq e^{-d^3} = e^{-\Omega(\log^6 n)} = n^{-\omega(1)}$. A union bound over all vertices with degree at least d^4 concludes the proof. □

Proof of Lemma 11.9 for independent set. Let v be a vertex in layer i that is still in the graph and has at least d^4 children after all layers with index $\geq i$ have been processed. We show that then at least one of these children will be selected to join the independent set with high probability. Note that this then concludes the proof, as in all the cases either v will be removed from the graph or will not have a high indegree anymore. Moreover, observe that such a child u of v (that is still there after having processed layers $\geq i$) will be selected to join the independent set iff it is proposed. This is because if it did not join even though it is proposed, then a parent of u would had been selected to join the independent set, in which case u would not have been part of the graph anymore, at latest after i has been processed, thus would not count towards v's high degree at that point.

Every such child u of v is marked independently with probability $p = d^{-2}$. The probability of u being proposed and hence joining the independent set is at least $p(1-p)^d$, as it has at most d neighbors in its layer, and it is proposed iff it is marked and none of its neighbors in the same layer is marked. vertex v thus in expectation has at least $\mu := d^4 p(1-p)^d \geq d^2 e^{-2/d} = \Omega(d^2)$ children that join the independent set.

Since whether a vertex u proposes and hence joins the independent set depends on at most d other vertices (namely u's neighbors in the same layer), it follows from a variant of the Chernoff bounds for bounded dependence, e.g., from Theorem 2.1 in [212], that the probability of v having, say, 0.5μ neighbors that join the independent set is at most $e^{-\Omega(\mu/d)} = e^{-\Omega(d)} = e^{-\Omega\left(\log^2 n\right)} = n^{-\omega(1)}$. A union bound over all vertices v with degree at least d^4 concludes the proof. □

11.2.2 Degree Reduction in MPC

We show how to simulate the centralized degree reduction algorithm from Section 11.2 in the sublinear-memory MPC model, for the partition phase and for the mark-and-propose and selection phase.

The degree reduction algorithm described in Section 11.2 can be implemented in the LOCAL model in a relatively straightforward way in $\operatorname{poly}\log n$ rounds. We show how to exponentially speed this up using a variant of the graph exponentiation technique. To deal with the issues of the local and global memory barriers of the graph exponentiation technique, the key observation is that a large fraction of the vertices in the graph are contained in the first layers of the H-partition. In particular, we show that if we focus on the graph remaining after $\ell/2$ iterations of peeling, we can perform roughly $\log \ell$ exponentiation steps without violating the memory constraints. Hence, we can perform 2^i steps of the degree reduction process in roughly i communication rounds.

Partition Phase

In the following, we describe how to compute the H-partition with parameter $d = \Delta^{1/10}$ in the sublinear-memory MPC model. This is done by simulating the greedy sequential peeling algorithm described in Section 3.1.3, also recalling the properties from Lemma 3.1 Throughout this section, we assume that $\Delta \geq (2\lambda)^{20}$, i.e., that $d \geq (2\lambda)^2$. Observe that if $\Delta^2 > n^\delta$, then the H-partition with parameter $d = \Delta^{1/10}$ consists of $O(\log_{d/\lambda} n) = O(\log_\Delta n) = O(1/\delta)$ layers, in which case the arguments in this section imply that going through the layers one by one will easily yield at least as good round complexities as for the more difficult case of $\Delta^2 \leq n^\delta$. Hence, throughout this section, it is safe to assume that $\Delta^2 \leq n^\delta$.

The goal of the algorithm for computing the H-partition is that each vertex (or, more formally, the node storing the vertex) knows in which layer of the H-partition it is contained. The algorithm proceeds in iterations, where each iteration consists of two parts: first, the output, i.e., the layer index, is determined for a large fraction of the vertices, and second, these vertices are removed for the remainder of the computation. The latter ensures that the remaining small fraction of vertices can use essentially all of the total available memory in the next iteration, resulting in a larger memory budget per vertex. However, there is a caveat: When the memory budget per vertex exceeds $\Theta(n^\delta)$, i.e., the memory capacity of a single node, then it is not sufficient anymore to merely argue that the used memory of all vertices together does not exceed the total memory of all nodes together. We circumvent this issue by starting the above process repeatedly from anew (in the remaining graph) each time the memory requirement per vertex reaches the memory capacity of a single node.

As we will see, the number of repetitions, called *phases*, is bounded by $O(1/\delta)$. We now examine the phases and iterations in more detail.

Algorithm Details: Let k be the largest integer s.t. $\Delta^{2^k+1} \leq n^\delta$ (which implies that $k \geq 0$). The algorithm consists of phases and each phase consists of $k+1$ iterations.

In each iteration $i = 0, 1, \ldots, k$, we do the following. Let $G_i = G_i^{(0)}$ be the graph at the beginning of iteration i. Each vertex connects its current 1-hop neighborhood to a clique by adding virtual edges to G_i; if $i = 0$, omit this step. We perform 20 repetitions of the following process if $i \geq 1$, and 60 repetitions if $i = 0$. In repetition $0 \leq j \leq 19$ (or $0 \leq j \leq 59$), each vertex computes its layer index in the H-partition of $G_i^{(j)}$ (with parameter d) or determines that its layer index is strictly larger than 2^i, upon which all vertices in layer at most 2^i (and all its incident edges) are removed from the graph, resulting in a graph $G_i^{(j+1)}$.

For the next iteration, we set $G_{i+1} = G_i^{(20)}$ (and $G_{i+1} = G_i^{(60)}$ if $i = 0$). At the end of a phase, we remove all added edges. The algorithm terminates when each vertex knows its layer.

Note that each time a vertex is removed from the graph, the whole layer that contains this vertex is removed, and each time such a layer is removed, all layers with smaller index are removed at the same time or before. By the definition of the H-partition, if we remove the ℓ layers with smallest index from a graph, then there is a 1-to-1 correspondence between the layers of the resulting graph and the layers with layer index at least $\ell + 1$ of the original graph. More specifically, layer ℓ' of the resulting graph contains exactly the same vertices as layer $\ell + \ell'$ of the original graph. Hence, if a vertex knows its layer index in some $G_i^{(j)}$, it can easily compute its layer index in our original input graph G, by keeping track of the number of deleted layers, which is uniquely defined by i, j and the number of the phase. We implicitly assume that each vertex performs this computation upon determining its layer index in some $G_i^{(j)}$ and only consider how to determine the layer index in the current graph.

Implementation in the MPC Model: Let us take a look at one iteration. Connecting the 1-hop neighborhoods to cliques is done by adding the edges that are missing. Edges that are added by multiple vertices are only added once (since the edge in question is stored by the nodes that contain an endpoint of the edge, this is straightforward to realize). Note that during a phase, the 1-hop neighborhoods of the vertices grow in each iteration (if not too many close-by vertices are removed from the graph); more specifically, after i iterations of connecting 1-hop neighborhoods to cliques, the new 1-hop neighborhood of a vertex contains exactly the vertices that were contained in its 2^i-hop neighborhood at the beginning of the phase (and were not removed so far).

In iteration i, the layer of a vertex is computed as follows: First each vertex locally gathers the topology of its 2^i-hop neighborhood (without any added edges)[2]. Since this step is performed after con-

[2]Note that it is easy to keep track of which edges are original and which are added, incurring only a small constant memory overhead; later we will also argue why storing the added edges does not violate our memory constraints.

necting the 2^{i-1}-hop neighborhood of each vertex to a clique (by repeatedly connecting 1-hop neighborhoods to cliques), i.e., after connecting each vertex to any other vertex in its 2^i-hop neighborhood, only 1 round of communication is required for gathering the topology. Moreover, since a vertex that knows the topology of its 2^i-hop neighborhood can simulate any (2^i-1)-round distributed process locally, it follows from the definition of the H-partition, that knowledge of the topology of the 2^i-hop neighborhood is sufficient for a vertex to determine whether its layer index is at most 2^i and, if this is the case, in exactly which layer it is contained. Thus, the only tasks remaining are to bound the round complexity of our algorithm and to show that the memory restrictions of our model are not violated by the algorithm.

Round Complexity: It is easy to see that every iteration takes $O(1)$ rounds. Thus, in order to bound the round complexity of our algorithm, it is sufficient to bound the number of iterations by $O((1/\delta) \cdot \log\log n)$. By Lemma 3.1 (ii) the number of layers in the H-partition of our original input graph G is $O(\log_{d/\lambda} n)$, which is $O(\log_\Delta n)$ since $d/(2\lambda) \geq \sqrt{d} = \Delta^{1/20}$. Consider an arbitrary phase. According to the algorithm description, in iteration $i \geq 1$, all vertices in the $20 \cdot 2^i$ lowest layers are removed from the current graph. Hence, ignoring iteration 0, the number of removed layers doubles in each iteration, and we obtain that the number of layers removed in the $k+1$ iterations of our phase is $\Omega(2^k)$. By the definition of k, we have $\Delta^{2^{k+1}+1} > n^\delta$, which implies $2^k > 1/3\delta \log_\Delta n$. Combining this inequality with the observations about the total number of layers and the number of layers removed per phase, we see that the algorithm terminates after $O(1/\delta)$ phases. Since there are $k+1 = O(\log\log n)$ iterations per phase, the bound on the number of iterations follows.

Memory Footprint: As during the course of the algorithm edges are added and vertices collect the topology of certain neighborhoods, we have to show that adding these edges and collecting these neighborhoods does not violate our memory constraints of $O(n^\delta)$ per node. As a first step towards this end, the following lemma bounds the number of vertices contained in graph G_i.

Lemma 11.10. *Graph G_i from phase i contains at most n'/Δ^{2^i} vertices, for all $i \geq 1$, where $n' = n/\Delta$.*

Proof. By Lemma 3.1 (i), removing the vertices in the layer with smallest index from the current graph decreases the number of vertices by a factor of at least $d/(2\lambda) \geq d^{1/2} = \Delta^{1/20}$. We show the lemma statement by induction. Since in iteration 0 the vertices in the 60 layers with smallest index are removed, we know that G_1 contains at most $n/\Delta^3 = n'/\Delta^{2^1}$ vertices. Now assume that G_i contains at most n'/Δ^{2^i} vertices, for an arbitrary $i \geq 1$. According to the design of our algorithm, G_{i+1} is obtained from G_i by removing the vertices in the $20 \cdot 2^i$ layers with smallest index. Combining this fact with our observation about the decrease in the number of vertices per removed layer, we obtain that G_{i+1} contains at most

$$\frac{n'}{\Delta^{2^i}} \cdot \frac{1}{\Delta^{2^i}} = \frac{n'}{\Delta^{2^{i+1}}}$$

vertices, concluding the proof. □

Using Lemma 11.10, we now show that the memory constraints of the sublinear-memory MPC model are not violated by our algorithm. Consider an arbitrary phase and an arbitrary iteration i during that phase. If $i = 0$, then no edges are added and each vertex already knows the topology of its 2^i-hop neighborhood, so no additional memory is required. Hence, assume that $i \geq 1$.

Due to Lemma 11.10, the number of vertices considered in iteration i is at most $n'/(\Delta^{2^i})$, where, again, $n' = n/\Delta$. After the initial step of connecting 1-hop neighborhoods to cliques in iteration i, each remaining vertex is connected to all vertices that were contained in its 2^i-hop neighborhood in the original graph G (and were not removed so far). Hence, each remaining vertex is connected to at most $O(\Delta^{2^i})$ other vertices, resulting in a memory requirement of $O(\Delta^{2^i})$ per vertex, or $O(n/\Delta)$ in total. Similarly, when collecting the topology of its 2^i-hop neighborhood, each vertex has to store $O(\Delta^{2^i} \cdot \Delta)$ edges, which requires at most $O(\Delta^{2^i} \cdot \Delta)$ memory, resulting in a total memory requirement of $O(n)$. Hence, the described algorithm does not exceed the total memory available in the sublinear-memory MPC model. Moreover, due to the choice of k, the memory requirement of each single vertex does not exceed the memory capacity of a single node.

Simulation of Mark-and-Propose and Selection

For the simulation of the mark-and-propose and selection phase, we rely heavily on the approach in Section 11.2.2. Recall that vertices were removed in *chunks* consisting of several consecutive layers and that before a vertex v was removed, v was directly connected to all vertices contained in a large neighborhood around v by adding the respective edges. For the simulation, we go through these chunks in the reverse order in which they were removed. Note that in which chunk a vertex is contained is uniquely determined by the layer index of the vertex. As a vertex computes its layer index during the construction of the H-partition, it can easily determine in which part of the simulation it will actively participate.

However, there is a problem we need to address. For communication, we want the edges added during the construction of the H-partition to be available also for the simulation. Unfortunately, during the course of the construction, we removed added edges to free memory for adding other edges. Fortunately, there is a conceptually simple way to circumvent this problem: in the construction of the H-partition, add a preprocessing step in the beginning, in which we remove the lowest $c \log(\log \log n/\delta)$ layers, for a sufficiently large constant c, one by one in $\log(\log\log n/\delta)$ rounds. This increases the available memory (compared to the number of (remaining) vertices) by a factor of $\Omega(\log\log n/\delta)$, by Lemma 3.1. Since the algorithm for constructing the H-partition consists of $O(\log\log n/\delta)$ iterations, this implies that we can store all edges that we add during the further course of the construction simultaneously without violating the memory restriction, by an argument similar to the analogous statement for the old construction of the H-partition. Similarly, the number of added edges incident to one particular vertex does not exceed the memory capacity of a single node. We assume that this preprocessing step took place and all edges added during the construction of the H-partition are also available for the simulation.

Matching Algorithm: As mentioned above, we process the chunks one by one, in decreasing order with respect to the indices of the contained layers. After processing a chunk, we want each vertex contained in the chunk to know the output of all incident edges according to the centralized matching algorithm. In the following, we describe how to process a chunk, after some preliminary steps.

The mark-and-propose phase of the algorithm is straightforward to implement in the sublinear-memory MPC model: each vertex (in each chunk at the same time) performs the marking of an outgoing edge as specified in the algorithm description. Note that, formally, the algorithm for the construction of the H-partition only returns the layer index for each vertex; however, from this information each vertex can easily determine which edges are outgoing, unoriented, or incoming according to the partial orientation induced by the H-partition. The proposing is performed for all vertices before going through the chunks sequentially: each vertex proposes one of its marked incoming edges (if there is at least one) uniformly at random. Note that proposes an edge does not necessarily indicate that this edge will be added to the matching; more specifically, an edge proposed by some vertex v will be added to the matching iff the edge that v marked is not selected to be added to the matching. In other words, only proposed edges can go into the matching and whether such an edge indeed goes into the matching can be determined by going through the layers in decreasing order and only adding a proposed edge if there is no conflict.

After this mark-and-propose phase, the processing of the chunks begins. Consider an arbitrary chunk. Let i be the iteration (in some phase) in which this chunk was removed in the construction of the H-partition, i.e., the chunk consists of 2^i layers. Each vertex in the chunk collects the topology of its 2^i-hop neighborhood in the chunk including the information contained therein about proposed edges. Due to the edges added during the construction of the H-partition, this can be achieved in a constant number of rounds, and by an analogous argument to the one at the end of Section 11.2.2, collecting the indicated information does not violate the memory restrictions of our model. Lemma 11.11 shows that the information contained in the 2^i-hop neighborhood of a vertex is sufficient for the vertex to determine the output for each incident edge in the centralized matching algorithm.

Lemma 11.11. *The information about which edges are proposed in the 2^i-hop neighborhood of a vertex v uniquely determines the output of all edges incident to v according to the centralized matching algorithm.*

Proof. From the design of the centralized matching algorithm, it follows that an edge is part of the matching iff 1) the edge is proposed and 2) either the higher-layer endpoint of the edge has no outgoing edges or the outgoing edge marked by the higher-layer endpoint is not part of the matching. Hence, in order to check whether an incident edge is in the matching, vertex v only has to consider the unique directed chain of proposed edges (in the chunk) starting in v. Clearly, the information which of the edges in this chain are proposed uniquely defines the output of the first edge in the chain, from which v can infer the output of all other incident edges. Since the number of edges in the chain is bounded by $2^i - 1$ as the chain is directed, the lemma statement follows. □

It thus follows from the bound on the number of iterations that the simulation of the selection phase for the matching algorithm can be performed in $O(\log \log n/\delta)$ rounds of communication, finishing the proof of Theorem 10.1.

Independent Set Algorithm: The simulation of the independent set algorithm proceeds analogously to the case of the matching algorithm. First, each vertex performs the marking and proposing in a distributed fashion in a constant number of rounds. Then, the chunks are processed one by one, as above, where during the processing of a chunk removed in iteration i, each vertex contained in the chunk collects its 2^i-hop neighborhood, including the information about which vertices are proposed, and then computes its own output locally. By analogous arguments to the ones presented in the case of the matching algorithm, the algorithm adheres to the memory constraints of our model and the total number of communication rounds is $O(\log \log n/\delta)$. The only part of the argumentation where a bit of care is required is the analogue of Lemma 11.11. In the case of the independent set algorithm the output of a vertex v may depend on *each* of its parents since each of those could be part of the independent set, which would prevent v from joining the independent set. However, *all* vertices in the chunk that can be reached from v via a directed chain of edges are contained in v's 2^i-hop neighborhood; therefore, collecting the own 2^i-hop neighborhood is sufficient for determining one's output. Note that at the end of processing a chunk, if we follow the above implementation, we have to spend an extra round for removing the neighbors of all selected independent set vertices since these may be contained in another chunk.

Bibliography

[1] Y. Afek, S. Kutten, and M. Yung. The Local Detection Paradigm and its Applications to Self-Stabilization. *Theoretical Computer Science*, 186(1):199–229, 1997.

[2] M. Ahmadi, F. Kuhn, and R. Oshman. Distributed Approximate Maximum Matching in the CONGEST Model. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, 2018.

[3] N. Ailon, M. Charikar, and A. Newman. Aggregating Inconsistent Information: Ranking and Clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.

[4] N. Alon. A Parallel Algorithmic Version of the Local Lemma. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 586–593. IEEE, 1991.

[5] 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.

[6] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-Efficient Local Computation Algorithms. In *Proceedings of ACM-SIAM*

Symposium on Discrete Algorithms (SODA), pages 1132–1139, 2012.

[7] N. Alon and J. H. Spencer. *The Probabilistic Method.* Wiley Publishing, 4th edition, 2016.

[8] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel Algorithms for Geometric Graph Problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 574–583, 2014.

[9] A. Andoni, C. Stein, Z. Song, Z. Wang, and P. Zhong. Parallel Graph Connectivity in Log Diameter Rounds. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 674–685, 2018.

[10] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan. An Introduction to MCMC for Machine Learning. *Machine Learning*, 50(1-2):5–43, 2003.

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

[12] S. Assadi, M. Bateni, A. Bernstein, V. Mirrokni, and C. Stein. Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1616–1635, 2019.

[13] S. Assadi, Y. Chen, and S. Khanna. Sublinear Algorithms for $(\Delta + 1)$ Vertex Coloring. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019.

[14] S. Assadi and S. Khanna. Randomized Composable Coresets for Matching and Vertex Cover. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 3–12, 2017.

[15] S. Assadi, X. Sun, and O. Weinstein. Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs. *ArXiv e-prints*, 2018. `arXiv:1805.02974`.

[16] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast Distributed Network Decompositions and Covers. *Journal of Parallel and Distributed Computing*, 39(2):105–114, 1996.

[17] B. Awerbuch, M. Luby, A. V. Goldberg, and S. A. Plotkin. Network Decomposition and Locality in Distributed Computation. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 364–369, 1989.

[18] B. Awerbuch and D. Peleg. Sparse Partitions. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 503–513, 1990.

[19] A. Balliu, S. Brandt, J. Hirvonen, D. Olivetti, M. Rabie, and J. Suomela. Lower Bounds for Maximal Matchings and Maximal Independent Dets. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 481–497, 2019.

[20] A. Balliu, F. Kuhn, and D. Olivetti. Distributed Edge Coloring in Time Quasi-Polylogarithmic in Delta. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 289–298, 2020.

[21] P. Bamberger, M. Ghaffari, F. Kuhn, Y. Maus, and J. Uitto. On the Complexity of Distributed Splitting Problems. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 280–289, 2019.

[22] N. Bansal, A. Blum, and S. Chawla. Correlation Clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[23] L. Barenboim and M. Elkin. Sublogarithmic Distributed MIS Algorithm for Sparse Graphs using Nash-Williams Decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.

[24] L. Barenboim and M. Elkin. Distributed Graph Coloring: Fundamentals and Recent Developments. *Synthesis Lectures on Distributed Computing Theory*, 4(1):1–171, 2013.

[25] L. Barenboim, M. Elkin, and U. Goldenberg. Locally-Iterative Distributed $(\Delta + 1)$-Coloring Below Szegedy-Vishwanathan Barrier, and Applications to Self-Stabilization and to Restricted-Bandwidth Models. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 437–446, 2018.

[26] L. Barenboim, M. Elkin, and T. Maimon. Deterministic Distributed $(\Delta + o(\Delta))$-Edge-Coloring, and Vertex-Coloring of Graphs with Bounded Diversity. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 175–184, 2017.

[27] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The Locality of Distributed Symmetry Breaking. In *Foundations of Computer Science (FOCS) 2012*, pages 321–330. IEEE, 2012.

[28] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The Locality of Distributed Symmetry Breaking. *J. ACM*, 63(3):20:1–20:45, 2016.

[29] P. Beame, P. Koutris, and D. Suciu. Skew in Parallel Query Processing. In *the Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 212–223, 2014.

[30] P. Beame, P. Koutris, and D. Suciu. Communication Steps for Parallel Query Processing. *Journal of the ACM (JACM)*, 64(6):40, 2017.

[31] J. Beck. An Algorithmic Approach to the Lovász Local Lemma. I. *Random Structures & Algorithms*, 2(4):343–365, 1991.

[32] S. Behnezhad, S. Brandt, M. Derakhshan, M. Fischer, M. Hajiaghayi, R. M. Karp, and J. Uitto. Massively Parallel Computation of Matching and MIS in Sparse Graphs. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 481–490, 2019.

[33] S. Behnezhad, M. Derakhshan, M. Hajiaghayi, C. Stein, and M. Sudan. Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 382–405, 2019.

[34] S. Behnezhad, L. Dhulipala, H. Esfandiari, J. Lacki, and V. Mirrokni. Near-Optimal Massively Parallel Graph Connectivity. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1615–1636, 2019.

[35] S. Behnezhad, L. Dhulipala, H. Esfandiari, J. Łacki, V. Mirrokni, and W. Schudy. Parallel Graph Algorithms in Constant Adaptive Rounds: Theory Meets Practice. *Proceedings on the VLDB Endowment (PVLDB)*, 13(13):3588–3602, 2020.

[36] A. Bernshteyn. Distributed Algorithms, the Lovász Local Lemma, and Descriptive Combinatorics. *arXiv preprint arXiv:2004.04905*, 2020.

[37] A. S. Biswas, R. Rubinfeld, and A. Yodpinyanee. Local Access to Huge Random Objects Through Partial Sampling. In *Innovations in Theoretical Computer Science Conference, ITCS*, volume 151, pages 27:1–27:65, 2020.

[38] G. E. Blelloch, J. T. Fineman, P. B. Gibbons, and J. Shun. Internally Deterministic Parallel Algorithms Can Be Fast. In *ACM SIGPLAN Notices*, pages 181–192, 2012.

[39] G. E. Blelloch, J. T. Fineman, and J. Shun. Greedy Sequential Maximal Independent Set and Matching are Parallel on Average. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 308–317, 2012.

[40] R. Bocchino, V. Adve, S. Adve, and M. Snir. Parallel Programming Must Be Deterministic By Default. In *USENIX Conference on Hot Topics in Parallelism*, pages 4–4, 2009.

[41] B. Bollobás. *Modern Graph Theory*, volume 184. Springer Science & Business Media, 2013.

[42] O. Borůvka. O Jistém Problému Minimálním. 1926.

[43] S. Brandt, M. Fischer, and J. Uitto. Matching and MIS for Uniformly Sparse Graphs in the Low-Memory MPC Model. *arXiv preprint arXiv:1807.05374*, 2018.

[44] S. Brandt, M. Fischer, and J. Uitto. Breaking the Linear-Memory Barrier in MPC: Fast MIS on Trees with n^{ε} Memory per Machine. In *the Proceedings of the International Colloquium on Structural Information and Communication Complexity*, volume 11639, pages 124–138, 2019.

[45] S. Brandt, M. Fischer, and J. Uitto. Breaking the Linear-Memory Barrier in MPC: Fast MIS on Trees with Strongly Sublinear Memory. *Theoretical Computer Science*, 849:22–34, 2021.

[46] S. Brandt, O. Fischer, J. Hirvonen, B. Keller, T. Lempiäinen, J. Rybicki, J. Suomela, and J. Uitto. A Lower Bound for the Distributed Lovász Local Lemma. In *Proceedings of the*

Symposium on Theory of Computing (STOC), pages 479–488, 2016.

[47] S. Brandt, Y. Maus, and J. Uitto. A Sharp Threshold Phenomenon for the Distributed Complexity of the Lovász Local Lemma. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 389–398, 2019.

[48] R. L. Brooks. On Colouring the Nodes of a Network. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 37, pages 194–197. Cambridge University Press, 1941.

[49] R. Bubley and M. Dyer. Path Coupling: A Technique for Proving Rapid Mixing in Markov Chains. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 223–231, 1997.

[50] N. Calkin and A. Frieze. Probabilistic Analysis of a Parallel Algorithm for Finding Maximal Independent Sets. *Random Structures & Algorithms*, 1(1):39–50, 1990.

[51] N. Calkin, A. Frieze, and L. Kučera. On the Expected Performance of a Parallel Algorithm for Finding Maximal Independent Subsets of a Random Graph. *Random Structures & Algorithms*, 3(2):215–221, 1992.

[52] M. Ceccarello, A. Pietracaprina, G. Pucci, and E. Upfal. Space and Time Efficient Parallel Graph Decomposition, Clustering, and Diameter Approximation. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 182–191, 2015.

[53] Y.-J. Chang. *Locality of Distributed Graph Problems.* PhD thesis, University of Michigan, 2019.

[54] Y.-J. Chang, M. Fischer, M. Ghaffari, J. Uitto, and Y. Zheng. The Complexity of (Δ+1)-Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 471–480, 2019.

[55] Y.-J. Chang, Q. He, W. Li, S. Pettie, and J. Uitto. The Complexity of Distributed Edge Coloring with Small Palettes. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2633–2652, 2018.

[56] Y.-J. Chang, Q. He, W. Li, S. Pettie, and J. Uitto. Distributed Edge Coloring and a Special Case of the Constructive Lovász Local Lemma. *ACM Transactions on Algorithms (TALG)*, 16(1):1–51, 2019.

[57] Y.-J. Chang, T. Kopelowitz, and S. Pettie. An Exponential Separation Between Randomized and Deterministic Complexity in the LOCAL Model. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 615–624, 2016.

[58] Y.-J. Chang, W. Li, and S. Pettie. An Optimal Distributed (Δ+1)-Coloring Algorithm? In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 445–456, 2018.

[59] Y.-J. Chang, W. Li, and S. Pettie. Distributed ($\Delta + 1$)-Coloring via Ultrafast Graph Shattering. *SIAM Journal on Computing*, 49(3):497–539, 2020.

[60] Y.-J. Chang and S. Pettie. A Time Hierarchy Theorem for the LOCAL Model. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 156–167, 2017.

[61] Y.-J. Chang and S. Pettie. A Time Hierarchy Theorem for the LOCAL Model. *SIAM Journal on Computing*, 48(1):33–69, 2019.

[62] S. Chatterjee, R. Gmyr, and G. Pandurangan. Sleeping is Efficient: MIS in $O(1)$-Rounds Node-Averaged Awake Complexity. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 99–108, 2020.

[63] S. Chen, M. Delcourt, A. Moitra, G. Perarnau, and L. Postle. Improved Bounds for Randomly Sampling Colorings via Linear Programming. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2216–2234. SIAM, 2019.

[64] S. Chen and A. Moitra. Linear Programming Bounds for Randomly Sampling Colorings. *arXiv preprint arXiv:1804.03156*, 2018.

[65] F. Chierichetti, N. Dalvi, and R. Kumar. Correlation Clustering in MapReduce. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 641–650. ACM, 2014.

[66] L. Chitnis, A. Das Sarma, A. Machanavajjhala, and V. Rastogi. Finding Connected Components in MapReduce in Logarithmic Rounds. In *IEEE International Conference on Data Engineering ICDE*, pages 50–61, 2013.

[67] K.-M. Chung, S. Pettie, and H.-H. Su. Distributed Algorithms for the Lovász Local Lemma and Graph Coloring. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 134–143, 2014.

[68] R. Cole and U. Vishkin. Deterministic Coin Tossing and Accelerating Cascades: Micro and Macro Techniques for Design-

ing Parallel Algorithms. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 206–219, 1986.

[69] D. Coppersmith, P. Raghavan, and M. Tompa. Parallel Graph Algorithms that Are Efficient on Average. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 260–269, 1987.

[70] G. Cormode, H. Jowhari, M. Monemizadeh, and S. Muthukrishnan. The Sparse Awakens: Streaming Algorithms for Matching Size Estimation in Sparse Graphs. In *the Proceedings of the Annual European Symposium on Algorithms*, pages 29:1–29:15, 2017.

[71] A. Czumaj, P. Davies, and M. Parter. Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 175–185, 2020.

[72] A. Czumaj, P. Davies, and M. Parter. Simple, Deterministic, Constant-Round Coloring in the Congested Clique. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 309–318, 2020.

[73] A. Czumaj, J. Łacki, A. Madry, S. Mitrović, K. Onak, and P. Sankowski. Round Compression for Parallel Matching Algorithms. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 471–484, 2018.

[74] A. Czumaj and C. Scheideler. A New Algorithm Approach to the General Lovász Local Lemma with Applications to Scheduling and Satisfiability Problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 38–47, 2000.

[75] A. Czygrinow and M. Hańćkowiak. Distributed Algorithm for Better Approximation of the Maximum Matching. In *In-*

ternational Computing and Combinatorics Conference, pages 242–251, 2003.

[76] A. Czygrinow, M. Hańćkowiak, and E. Szymańska. A Fast Distributed Algorithm for Approximating the Maximum Matching. In *the Proceedings of the Annual European Symposium on Algorithms*, pages 252–263, 2004.

[77] A. Czygrinow, M. Hańćkowiak, and E. Szymańska. Distributed Algorithm for Approximating the Maximum Matching. *Discrete Applied Mathematics*, 143(1):62–71, 2004.

[78] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the Symposium on Operating Systems Design & Implementation (OSDI)*, pages 10–10, 2004.

[79] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, pages 107–113, 2008.

[80] M. Delcourt, G. Perarnau, and L. Postle. Rapid Mixing of Glauber Dynamics for Colorings Below Vigoda's 11/6 Threshold. *arXiv preprint arXiv:1804.04025*, 2018.

[81] L. Dhulipala. Provably Efficient and Scalable Shared-Memory Graph Processing. 2020.

[82] L. Dhulipala, G. E. Blelloch, and J. Shun. Theoretically Efficient Parallel Graph Algorithms Can Be Fast and Scalable. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 393–404, 2018.

[83] L. Dhulipala, C. McGuffey, H. Kang, Y. Gu, G. Blelloch, P. Gibbons, and J. Shun. Sage: Parallel Semi-Asymmetric Graph Algorithms for NVRAMs. *Proceedings of the VLDB Endowment (PVLDB)*, 13(9), 2020.

[84] A. Drucker, F. Kuhn, and R. Oshman. On the Power of the Congested Clique Model. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 367–376, 2014.

[85] J. Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[86] M. Elkin and O. Neiman. Distributed Strong Diameter Network Decomposition. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 211–216, 2016.

[87] M. Elkin, S. Pettie, and H.-H. Su. $(2\Delta - 1)$-Edge-Coloring Is Much Easier Than Maximal Matching in the Distributed Setting. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 355–370, 2015.

[88] P. Erdős and L. Lovász. Problems and Results on 3-Chromatic Hypergraphs and Some Related Questions. *Infinite and finite sets*, 10(2):609–627, 1975.

[89] H. Esfandiari, M. T. Hajiaghayi, V. Liaghat, M. Monemizadeh, and K. Onak. Streaming Algorithms for Estimating the Matching Size in Planar Graphs and Beyond. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1217–1233, 2015.

[90] L. Euler. Solutio Problematis ad Geometriam Situs Pertinentis. *Commentarii Academiae Ccientiarum Petropolitanae*, pages 128–140, 1741.

[91] S. Even. *Graph Algorithms*. Cambridge University Press, 2011.

[92] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On Distributing Symmetric Streaming Computa-

tions. *ACM Transactions on Algorithms (TALG)*, 6(4):1–19, 2010.

[93] W. Feng, T. P. Hayes, and Y. Yin. Distributed Symmetry Breaking in Sampling (Optimal Distributed Randomly Coloring with Fewer Colors). *arXiv preprint arXiv:1802.06953*, 2018.

[94] W. Feng, T. P. Hayes, and Y. Yin. Fully-Asynchronous Distributed Metropolis Sampler with Optimal Speedup. *arXiv preprint arXiv:1904.00943*, 2019.

[95] W. Feng, T. P. Hayes, and Y. Yin. Distributed Metropolis Sampler with Optimal Parallelism. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2121–2140, 2021.

[96] W. Feng, Y. Sun, and Y. Yin. What Can Be Sampled Locally? In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 121–130, 2017.

[97] W. Feng, N. K. Vishnoi, and Y. Yin. Dynamic Sampling from Graphical Models. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 1070–1081, 2019.

[98] W. Feng and Y. Yin. On Local Distributed Sampling and Counting. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 189–198, 2018.

[99] M. Fischer. Improved Deterministic Distributed Matching via Rounding. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, volume 91, pages 17:1–17:15, 2017.

[100] M. Fischer. Improved Deterministic Distributed Matching via Rounding. *Distributed Computing*, 33(3-4):279–291, 2020.

[101] M. Fischer and M. Ghaffari. Sublogarithmic Distributed Algorithms for Lovász Local Lemma with Implications on Complexity Hierarchies. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 18:1–18:16, 2017.

[102] M. Fischer and M. Ghaffari. A Simple Parallel and Distributed Sampling Technique: Local Glauber Dynamics. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, volume 121, pages 26:1–26:11, 2018.

[103] M. Fischer, M. Ghaffari, and F. Kuhn. Deterministic Distributed Edge-Coloring via Hypergraph Maximal Matching. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 180–191, 2017.

[104] M. Fischer, M. Ghaffari, and J. Uitto. Simple Graph Coloring Algorithms for Congested Clique and Massively Parallel Computation. *CoRR*, abs/1808.08419, 2018.

[105] M. Fischer and A. Noever. Tight Analysis of Randomized Greedy MIS. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2152–2160, 2018.

[106] M. Fischer and A. Noever. Tight Analysis of Parallel Randomized Greedy MIS. *ACM Transaction on Algorithms (TALG)*, 16(1):6:1–6:13, 2020.

[107] P. Fraigniaud, M. Heinrich, and A. Kosowski. Local Conflict Coloring. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 625–634, 2016.

[108] B. Gfeller and E. Vicari. A Randomized Distributed Algorithm for the Maximal Independent Set Problem in Growth-

Bounded Graphs. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 53–60, 2007.

[109] M. Ghaffari. An Improved Distributed Algorithm for Maximal Independent Set. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 270–277, 2016.

[110] M. Ghaffari. Distributed MIS via All-to-All Communication. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 141–149, 2017.

[111] M. Ghaffari. Distributed Maximal Independent Set using Small Messages. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 805–820, 2019.

[112] M. Ghaffari, T. Gouleakis, C. Konrad, S. Mitrović, and R. Rubinfeld. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 129–138, 2018.

[113] M. Ghaffari, C. Grunau, and C. Jin. Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, volume 179, pages 34:1–34:18, 2020.

[114] M. Ghaffari, C. Grunau, and V. Rozhoň. Improved Deterministic Network Decomposition. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2904–2923, 2021.

[115] M. Ghaffari, D. G. Harris, and F. Kuhn. On Derandomizing Local Distributed Algorithms. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 662–673, 2018.

[116] M. Ghaffari, J. Hirvonen, F. Kuhn, Y. Maus, J. Suomela, and J. Uitto. Improved Distributed Degree Splitting and Edge Coloring. *Distributed Computing*, 33(3):293–310, 2020.

[117] M. Ghaffari, C. Jin, and D. Nilis. A Massively Parallel Algorithm for Minimum Weight Vertex Cover. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 259–268, 2020.

[118] M. Ghaffari and F. Kuhn. Deterministic Distributed Vertex Coloring: Simpler, Faster, and without Network Decomposition. *arXiv preprint arXiv:2011.04511*, 2020.

[119] M. Ghaffari, F. Kuhn, and Y. Maus. On the Complexity of Local Distributed Graph Problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 784–797, 2017.

[120] M. Ghaffari, F. Kuhn, Y. Maus, and J. Uitto. Deterministic Distributed Edge-Coloring with Fewer Colors. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 418–430, 2018.

[121] M. Ghaffari, F. Kuhn, and J. Uitto. Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1650–1663, 2019.

[122] M. Ghaffari and H.-H. Su. Distributed Degree Splitting, Edge Coloring, and Orientations. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2505–2523, 2017.

[123] M. Ghaffari and J. Uitto. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *Proceedings of ACM-*

SIAM Symposium on Discrete Algorithms (SODA), pages 1636–1653, 2019.

[124] G. Goel and J. Gustedt. Bounded Arboricity to Determine the Local Structure of Sparse Graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 159–167. Springer, 2006.

[125] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin. Parallel Gibbs Sampling: From Colored Fields to Thin Junction Trees. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 324–332, 2011.

[126] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, Searching, and Simulation in the MapReduce Framework. In *Proc. ISAAC*, pages 374–383. Springer, 2011.

[127] M. Göös, J. Hirvonen, and J. Suomela. Linear-in-Delta Lower Bounds in the LOCAL Model. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 86–95, 2014.

[128] E. Grigorescu, M. Monemizadeh, and S. Zhou. Estimating Weighted Matchings in o(n) Space. *CoRR*, abs/1604.07467, 2016. URL: `http://arxiv.org/abs/1604.07467`, `arXiv:1604.07467`.

[129] H. Guo, M. Jerrum, and J. Liu. Uniform Sampling Through the Lovász Local Lemma. In H. Hatami, P. McKenzie, and V. King, editors, *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 342–355, 2017.

[130] H. Guo, M. Jerrum, and J. Liu. Uniform Sampling Through the Lovász Local Lemma. *Journal of the ACM (JACM)*, 66(3):1–31, 2019.

[131] M. Hańćkowiak, M. Karoński, and A. Panconesi. On the Distributed Complexity of Computing Maximal Matchings. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 219–225, 1998.

[132] M. Hańćkowiak, M. Karoński, and A. Panconesi. A Faster Distributed Algorithm for Computing Maximal Matchings Deterministically. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 219–228, 1999.

[133] M. Hańćkowiak, M. Karoński, and A. Panconesi. On the Distributed Complexity of Computing Maximal Matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001.

[134] D. Harris. Derandomizing the Lovász Local Lemma via Log-Space Statistical Yests. *arXiv preprint arXiv:1807.06672*, 2018.

[135] D. G. Harris. Deterministic Parallel Algorithms for Fooling Polylogarithmic Juntas and the Lovász Local Lemma. *ACM Transactions on Algorithms (TALG)*, 14(4):1–24, 2018.

[136] D. G. Harris. Deterministic Algorithms for the Lovász Local Lemma: Simpler, More General, and More Parallel. *arXiv preprint arXiv:1909.08065*, 2019.

[137] D. G. Harris. Distributed Local Approximation Algorithms for Maximum Matching in Graphs and Hypergraphs. In *the Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 700–724. IEEE, 2019.

[138] D. G. Harris. Oblivious Resampling Oracles and Parallel Algorithms for the Lopsided Lovász Local Lemma. *ACM Transactions on Algorithms (TALG)*, 17(1):1–32, 2020.

[139] D. G. Harris, J. Schneider, and H.-H. Su. Distributed $(\Delta + 1)$-Coloring in Sublogarithmic Rounds. *Journal of the ACM*, 65(4):19:1–19:21, 2018.

[140] N. J. Harvey, C. Liaw, and P. Liu. Greedy and Local Ratio Algorithms in the MapReduce Model. *arXiv preprint arXiv:1806.06421*, 2018.

[141] J. W. Hegeman and S. V. Pemmaraju. Lessons from the Congested Clique Applied to MapReduce. *Theoretical Computer Science*, 608:268–281, 2015.

[142] J. Hirvonen and J. Suomela. Distributed Maximal Matching: Greedy is Optimal. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 165–174, 2012.

[143] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[144] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. *SIGOPS Operating Systems Review*, 41(3):59–72, 2007.

[145] A. Israeli and A. Itai. A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Information Processing Letters*, 22(2):77–80, 1986.

[146] A. Israeli and Y. Shiloach. An Improved Parallel Algorithm for Maximal Matching. *Information Processing Letters*, 22(2):57–60, 1986.

[147] M. Jerrum. A Very Simple Algorithm for Estimating the Number of k-Colorings of a Low-Degree Graph. *Random Structures & Algorithms*, 7(2):157–165, 1995.

[148] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random Generation of Combinatorial Structures From a Uniform Distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[149] Ö. Johansson. Simple Distributed $(\Delta+1)$-Coloring of Graphs. *Information Processing Letters*, 70(5):229–232, 1999.

[150] T. Jurdzinski and K. Nowicki. MST in $O(1)$ Rounds of Congested Clique. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2620–2632, 2018.

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

[152] R. M. Karp. *Reducibility among Combinatorial Problems*. Springer, 1972.

[153] H. Klauck, D. Nanongkai, G. Pandurangan, and P. Robinson. Distributed Computation of Large-Scale Graph Problems. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–410. SIAM, 2014.

[154] D. König. *Theorie der endlichen und unendlichen Graphen: Kombinatorische Topologie der Streckenkomplexe*, volume 16. Akademische Verlagsgesellschaft mbH, 1936.

[155] A. Korman, J. Sereni, and L. Viennot. Toward More Localized Local Algorithms: Removing Assumptions Concerning Global Knowledge. *Distributed Computing*, 26(5-6):289–308, 2013.

[156] C. Koufogiannakis and N. E. Young. Distributed Fractional Packing and Maximum Weighted b-Matching via Tail-Recursive Duality. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 221–238. Springer, 2009.

[157] C. Koufogiannakis and N. E. Young. Distributed Algorithms for Covering, Packing and Maximum Weighted Matching. *Distributed Computing*, 24(1):45–63, 2011.

[158] F. Kuhn. *The Price of Locality: Exploring the Complexity of Distributed Coordination Primitives.* PhD thesis, ETH Zurich, 2005.

[159] F. Kuhn. Weak Graph Colorings: Distributed Algorithms and Applications. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 138–144, 2009.

[160] F. Kuhn. Faster Deterministic Distributed Coloring through Recursive List Coloring. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1244–1259. SIAM, 2020.

[161] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Fast Deterministic Distributed Maximal Independent Set Computation on Growth-Bounded Graphs. *Distributed Computing*, pages 273–287, 2005.

[162] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What Cannot Be Computed Locally! In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 300–309, 2004.

[163] F. Kuhn, T. Moscibroda, and R. Wattenhofer. The Price of Being Near-Sighted. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 980–989, 2006.

[164] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local Computation: Lower and Upper Bounds. *Journal of the ACM (JACM)*, 63:17:1–17:44, 2016.

[165] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast Greedy Algorithms in MapReduce and Streaming. *ACM Transactions on Parallel Computing (TOPC)*, 2(3):14, 2015.

[166] S. Kutten, D. Nanongkai, G. Pandurangan, and P. Robinson. Distributed Symmetry Breaking in Hypergraphs. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 469–483, 2014.

[167] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a Method for Solving Graph Problems in MapReduce. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 85–94, 2011.

[168] C. Lenzen. Optimal Deterministic Routing and Sorting on the Congested Clique. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 42–50, 2013.

[169] C. Lenzen and R. Wattenhofer. Brief Announcement: Exponential Speed-up of Local Algorithms Using Non-Local Communication. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 295–296, 2010.

[170] C. Lenzen and R. Wattenhofer. MIS on Trees. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 41–48. ACM, 2011.

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

[172] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[173] N. Linial. Local-Global Phenomena in Graphs. *Combinatorics, Probability and Computing*, 2(4):491–503, 1993.

[174] N. Linial and M. Saks. Decomposing graphs into regions of small diameter. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, pages 320–330, 1991.

[175] Z. Lotker, B. Patt-Shamir, E. Pavlov, and D. Peleg. Minimum-Weight Spanning Tree Construction in $O(\log\log n)$ Communication Rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.

[176] Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved Distributed Approximate Matching. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 129–136, 2008.

[177] Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved Distributed Approximate Matching. *Journal of the ACM (JACM)*, 62(5), 2015.

[178] Z. Lotker, B. Patt-Shamir, and A. Rosen. Distributed Approximate Matching. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 167–174, 2007.

[179] Z. Lotker, E. Pavlov, B. Patt-Shamir, and D. Peleg. MST Construction in $O(\log\log n)$ Communication Rounds. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 94–100, 2003.

[180] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 1–10, 1985.

[181] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[182] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a System for Large-Scale Graph Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 135–146, 2010.

[183] A. McGregor. Finding Graph Matchings in Data Streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181. Springer, 2005.

[184] Y. Métivier, J. M. Robson, N. Saheb-Djahromi, and A. Zemmari. An Optimal Bit Complexity Randomized Distributed MIS Algorithm. In *Structural Information and Communication Complexity*, pages 323–337. Springer, 2010.

[185] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[186] M. Molloy and B. Reed. Further Algorithmic Aspects of the Local Lemma. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 524–529, 1998.

[187] M. Molloy and B. Reed. Graph Coloring and the Probabilistic Method, 2002.

[188] R. A. Moser. A Constructive Proof of the Lovász Local Lemma. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 343–350, 2009.

[189] R. A. Moser and G. Tardos. A Constructive Proof of the General Lovász Local Lemma. *Journal of the ACM (JACM)*, 57(2):11, 2010.

[190] H. M. Mulder. Julius Petersen's Theory of Regular Graphs. *Discrete mathematics*, 100(1-3):157–175, 1992.

[191] S. Nahar, S. Sahni, and E. Shragowitz. Simulated Annealing and Combinatorial Optimization. In *ACM/IEEE Design Automation Conference*, pages 293–299. IEEE, 1986.

[192] D. Nanongkai, T. Saranurak, and S. Yingchareonthawornchai. Breaking Quadratic Time for Small Vertex Connectivity and an Approximation Scheme. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 241–252, 2019.

[193] D. Nanongkai and M. Scquizzato. Equivalence Classes and Conditional Hardness in Massively Parallel Computations. In *the Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, volume 153, pages 33:1–33:16, 2019.

[194] M. Naor. A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991.

[195] M. Naor and L. Stockmeyer. What Can Be Computed Locally? In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 184–193. SIAM, 1993.

[196] M. Naor and L. Stockmeyer. What Can Be Computed Locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

[197] C. S. J. Nash-Williams. Edge-Disjoint Spanning Trees of Finite Graphs. *Journal of the London Mathematical Society*, 1(1):445–450, 1961.

[198] C. S. J. Nash-Williams. Decomposition of Finite Graphs into Forests. *Journal of the London Mathematical Society*, 1(1):12–12, 1964.

[199] C. S. J. A. Nash-Williams. Edge-Disjoint Spanning Trees of Finite Graphs. *Journal of the London Mathematical Society*, 36:445–450, 1961.

[200] D. Newman, P. Smyth, M. Welling, and A. U. Asuncion. Distributed Inference for Latent Dirichlet Allocation. In *Advances in neural information processing systems*, pages 1081–1088, 2008.

[201] K. Onak. Round Compression for Parallel Graph Algorithms in Strongly Sublinear Space. *arXiv preprint arXiv:1807.08745*, 2018.

[202] K. Onak, B. Schieber, S. Solomon, and N. Wein. Fully Dynamic MIS in Uniformly Sparse Graphs. In *the Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 92:1–92:14, 2018.

[203] K. Onak, B. Schieber, S. Solomon, and N. Wein. Fully Dynamic MIS in Uniformly Sparse Graphs. *ACM Transactions on Algorithms (TALG)*, 16(2):1–19, 2020.

[204] J. Pach and G. Tardos. Conflict-Free Colorings of Graphs and Hypergraphs Probability and Computing. *Combinatorics*, 18:819–834, 2009.

[205] A. Panconesi and R. Rizzi. Some Simple Distributed Algorithms for Sparse Networks. *Distributed computing*, 14(2):97–100, 2001.

[206] A. Panconesi and M. Sozio. Fast Primal-Dual Distributed Algorithms for Scheduling and Matching Problems. *Distributed Computing*, 22(4):269–283, 2010.

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

[208] A. Panconesi and A. Srinivasan. On the Complexity of Distributed Network Decomposition. *J. Algor.*, 20(2):356–374, 1996.

[209] M. Parter. $(\Delta+1)$ Coloring in the Congested Clique Model. In *the Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 160:1–160:14, 2018.

[210] M. Parter and H.-H. Su. Randomized $(\Delta + 1)$ Coloring in $O(\log^* \Delta)$ Congested Clique Rounds. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 39:1–39:18, 2018.

[211] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, 2000.

[212] S. V. Pemmaraju. Equitable Coloring Extends Chernoff-Hoeffding Bounds. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, pages 285–296. Springer, 2001.

[213] S. Pettie and H.-H. Su. Distributed Coloring Algorithms for Triangle-Free Graphs. *Information and Computation*, 243:263–280, 2015.

[214] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-Round Tradeoffs for MapReduce Computations. In *Proceedings of the International Conference on Supercomputing*, pages 235–244, 2012.

[215] T. Roughgarden, S. Vassilvitskii, and J. R. Wang. Shuffles and Circuits: (On Lower Bounds for Modern Parallel Computation). In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–12, 2016.

[216] V. Rozhoň and M. Ghaffari. Polylogarithmic-Time Deterministic Network Decomposition and Distributed Derandomization. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 350–363, 2020.

[217] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast Local Computation Algorithms. In *Symposium on Innovations in Theoretical of Computer Science*, pages 223–238, 2011.

[218] J. Salas and A. D. Sokal. Absence of Phase Transition for Antiferromagnetic Potts Models via the Dobrushin Uniqueness Theorem. *Journal of Statistical Physics*, 86(3-4):551–579, 1997.

[219] J. P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding Bounds for Applications with Limited Independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

[220] J. Schneider, M. Elkin, and R. Wattenhofer. Symmetry Breaking Depending on the Chromatic Number or the Neighborhood Growth. *Theoretical Computer Science*, 509:40–50, 2013.

[221] J. Schneider and R. Wattenhofer. An Optimal Maximal Independent Set Algorithm For Bounded-Independence Graphs. *Distributed Computing*, 22(5):349–361, 2010.

[222] A. Srinivasan. Improved Algorithmic Versions of the Lovász Local Lemma. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 611–620. Society for Industrial and Applied Mathematics, 2008.

[223] H.-H. Su and H. T. Vu. Distributed Dense Subgraph Detection and Low Outdegree Orientation. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, volume 179, pages 15:1–15:18, 2020.

[224] J. Suomela. Distributed Algorithms for Edge Dominating Sets. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 365–374, 2010.

[225] E. Vigoda. Improved Bounds for Sampling Colorings. *Journal of Mathematical Physics*, 41(3):1555–1569, 2000.

[226] V. G. Vizing. On an Estimate of the Chromatic Class of a p-Graph. *Diskret Analiz*, 3(7):25–30, 1964.

[227] M. Wattenhofer and R. Wattenhofer. Distributed Weighted Matching. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 335–348. Springer, 2004.

[228] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 2012.

[229] F. Yan, N. Xu, and Y. Qi. Parallel Inference for Latent Dirichlet Allocation on Graphics Processing Units. In *Advances in Neural Information Processing Systems*, pages 2134–2142, 2009.

[230] M. Yannakakis and F. Gavril. Edge Dominating Sets in Graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980.

[231] G. Yaroslavtsev and A. Vadapalli. Massively Parallel Algorithms and Hardness for Single-Linkage Clustering under ℓ_p Distances. In *International Conference on Machine Learning*, pages 5600–5609. PMLR, 2018.

[232] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2010.