Improved Distributed Algorithms for Fundamental Graph Problems

by

Mohsen Ghaffari

B.S., Electrical Engineering, Sharif University of Technology (2010)
B.S., Computer Science, Sharif University of Technology (2010)
S.M., Computer Science, Massachusetts Institute of Technology (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2017

Author .....
Department of Electrical Engineering and Computer Science
October 14, 2016

Certified by .....
Nancy Lynch
NEC Professor of Software Science and Engineering
Thesis Supervisor

Accepted by .....
Leslie Kolodziejski
Chair, Department Committee on Graduate Theses

Improved Distributed Algorithms for Fundamental Graph Problems

by

Mohsen Ghaffari

Submitted to the Department of Electrical Engineering and Computer Science
on October 14, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Distributed graph algorithms provide efficient and theoretically sound methods for solving graph problems in distributed settings and more generally for performing distributed computation in networks. These algorithms are applicable in a wide variety of settings, ranging from computer networks to massively parallel computing and beyond.

This thesis addresses a number of the central problems of distributed graph algorithms. These problems generally revolve around two of the principal challenges of the area, *locality* and *congestion*. The problems include computing *maximal independent set*, *minimum spanning tree*, *minimum edge cut* and *minimum vertex cut*, *graph connectivity decompositions*, *network information dissemination*, *minimum-weight connected dominating set*, and *scheduling distributed protocols*.

We develop novel techniques, concepts, and tools for these problems, and present algorithms and impossibility results which improve considerably on the state of the art, in several cases resolving or advancing long-standing open problems.

Thesis Supervisor: Nancy Lynch
Title: NEC Professor of Software Science and Engineering

A Note on the Content

This thesis presents a sampling of the author's results during the PhD period [GSar, GKPar, GKS16, BYCHGS16, GP16a, GP16c, GP16b, GH16a, GN16b, GN16a, Gha16b, GH16b, Gha15b, GKK+15, GMRL15, CGG+15, Gha15a, CHGK14a, CHGK14b, GKLN14, GHS14, GH14, Gha14, GL14, AGHK14, GGLS14, GK13, GH13a, GHK13, GHK13, DGG+13, GLN13, GH13b, GGNT12, GHLN12, GLS11].

In particular, the thesis covers [Gha16b, Gha15b, CHGK14a, CHGK14b, Gha14, GK13, CGG+15] essentially completely, and it also discusses some results from [GH16b] partially. The presented results either fit squarely within the area of *distributed graph algorithms*, or provide purely graph-theoretic basis for some results in distributed graph algorithms. In the following, we specify which results appear in each chapter.

Awards Among the covered papers, the following were honored with awards: [Gha16b] received both a Best Paper Award and the Best Student Paper Award at SODA'16, [Gha15b] received the Best Student Paper Award at PODC'15, [CHGK14a] received the Best Student Paper Award at PODC'14, [Gha14] received a Best Student Paper Award at ICALP'14, and [GK13] received the Best Paper Award at DISC'13.

Chapter 3 is based on the following previous publication:

- [Gha16b] Mohsen Ghaffari. "An Improved Distributed Algorithm for Maximal Independent Set". In the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 270–277, 2016. **SODA'16 Best Paper and Best Student Paper awards.**

Chapter 4 is based on the following previous publications:

- [CHGK14a] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. "Distributed connectivity decomposition". In the Proceedings of the International Symposium on Principles of Distributed Computing (PODC), pages 156–165, 2014. **PODC'14 Best Student Paper award**
- [CHGK14b] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. "A new perspective on vertex connectivity". In the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), 2014.

Chapter 5 is based on the following previous publications:

- [CHGK14b] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. "A new perspective on vertex connectivity". In Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), 2014.

- [CGG+15] Keren Censor-Hillel, Mohsen Ghaffari, George Giakkoupis, Bernhard Haeupler, and Fabian Kuhn. "Tight bounds on vertex connectivity under vertex sampling". In Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 2006–2018, 2015.

Chapter 6 is based in part on the following previous publication:

- [GH16b] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for Planar Networks II: Low-congestion shortcuts, MST, and Min-Cut. In the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), 2016.

Chapter 7 is based on the following previous publications:

- [GK13] Mohsen Ghaffari and Fabian Kuhn. "Distributed minimum cut approximation". In Proceedings of of the International Symp. on Dist. Comp. (DISC), pages 1–15, 2013. **DISC'14 Best Paper award**.

- [GH16b] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for Planar Networks II: Low-congestion shortcuts, MST, and Min-Cut. In the Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), 2016.

Chapter 8 is based on the following previous publication:

- [Gha14] Mohsen Ghaffari. Near-optimal distributed approximation of minimum-weight connected dominating set. In the Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP), 2014. **ICALP'14 Best Student Paper award, track C.**

Chapter 9 is based on the following previous publication:

- [CHGK14a] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. "Distributed connectivity decomposition". In the Proceedings of the International Symposium on Principles of Distributed Computing (PODC), pages 156–165, 2014. **PODC'14 Best Student Papaer award**

Chapter 10 is based on the following previous publication:

- [Gha15b] Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In the Proceedings of the International Symposium on Principles of Distributed Computing. (PODC), pages 3–12, 2015. **PODC'15 Best Student Paper award**

Acknowledgments

This thesis is centered on *distributed network algorithms*; it is appropriate to start it by expressing my gratitude and appreciation to the nodes of my (academic) distributed network.

First and foremost, I would like to thank my parent in the academic tree[1]—that is, my advisor—Nancy Lynch. Over the past few years, I literally grew up under Nancy's guidance and supervision. Sometimes, we had close collaborations and worked together on research problems. Other times, she gave me the freedom to pursue my own interests and instincts in research; though, always being there to support and advise. Nancy, I could not be more grateful for your trust, support, and encouragement!

Next, I want to thank my thesis defense committee, Ronitt Rubinfeld, Nir Shavit, and Fabian Kuhn. Ronitt and Nir, I surely enjoyed our numerous small chats throughout the years, and I also learned a lot in your courses. I hope that now that I leave MIT, we will get to actually work with each other on some research problems. Nir, I am grateful for your advice on various matters. Fabian, I am proud to have you as my colleague. I have had a lot of fun throughout our collaborations, and I am eagerly looking forward to much more of that fun, as I move to Zurich. Let us see which problem stands in our way!

I also want to thank a few of the other faculties at our theory group: Silvio Micali for being on my Research Qualification Exam committee, Piotr Indyk for advice on a number of matters, and Madhu Sudan and David Karger, for our collaborations and for their extremely enjoyable and instructive courses on Coding Theory and Randomized Algorithms, respectively. The latter course, and especially David's intuitive way of explaining things, certainly influenced my way of thinking about algorithm design and analysis, and for that, I owe a big thanks to David!

I want to thank two of my colleagues, Bernhard Haeupler and Merav Parter, who certainly have been far beyond just colleagues for me. Guys, I feel very fortunate to have you as friends! I have had tremendous joy in all the time we spent together, all the whiteboards that we filled, and all our conversations, research and non-research related. I am hopeful to have you as friends and colleagues for many many more years to come.

I would also like to thank my coauthors, who have made the research experience far more enjoyable, and from whom I have learned so much: Noga Alon, Keren Censor-Hillel, Sebastian Daum, Rati Gelashvili, George Giakkoupis, Seth Gilbert, Bernhard Haeupler, David Karger, Erez Kantor, Andreas Karrenbauer, Majid Khabbazian, Fabian Kuhn, Christoph Lenzen, Jerry Li, Nancy Lynch, Cameron Musco, Calvin Newport, Debmalya Panigrahi, Merav Parter, Boaz Patt-Shamir, Tsvetomira Radeva, Srikanth Sastry, Nir Shavit, Hsin-Hao Su, Madhu Sudan, and Henry Tan.

[1] If I have learned something during my PhD from Nancy's exemplary devotion to precision, I should note that, formally, the advisorship graph is not necessarily a tree, but rather a Directed Acyclic Graph (DAG).

During the past few years, I had an absolute bliss and MIT truly became a home for me. For that, I owe thanks to a great many people: the administrative assistants, Joanne Hanely, Be Blackburn, Rebecca Yadegar, Linda Lynch, and Debbie Goodwin, who do an amazing job and literally carry the theory of computation group of MIT on their shoulders; my officemates throughout the years, Andrew Drucker, Jelani Nelson, Mira Radeva, Rotem Oshman, Alejandro Cornejo, Prasant Gopal, Cameron Musco, Badih Ghazi, and Aikaterini Sotiraki, with whom we had numerous conversations about all sorts of things, and from whom I learned amazing things about all beautiful subareas of the theory of computation; our lunch group, Hsin-Hao Su, Merav Parter, Stephan Hozer, Erez Kantor, Christoph Lenzen, Bernhard Haeupler, Alex Cornjeo, and Keren Censor-Hillel, who made the lunch break and its random conversations an indispensable part of my day; my squash friends from MIT, Arturs Backurs, Nishanth Dikkala, Badih Ghazi, Alex Cornjeo, and Bernhard Haeupler; and many other fellow students, Rati Gelashvilli, Elad Haramaty, Sepideh Mahabadi, Ali Vakilian, Mohammad Bavarian, etc. Sorry if I am forgetting a name. Thank you all!

Special thanks to Faith Ellen for hosting me at the University of Toronto, while I got stuck in Canada this past summer, and to Jimmy (Leqi) Zhu and Mika Göös for numerous enjoyable conversations.

Last but not least, this thesis is dedicated to my parents and my sister. Thanks for everything, and so sorry that I had to stay in the US throughout my PhD and I was not able to see you for the past few years.

Contents

1 Introduction **13**

1.1 Part I: Locality . . . 14

1.1.1 The MIS Problem, its Centrality, and State of the Art . . . 15

1.1.2 Our Result On MIS and Its Implications . . . 17

1.2 Part II: Congestion . . . 18

1.2.1 Graph-Theoretic Foundations . . . 19

1.2.2 Distributed Algorithmic Aspects . . . 24

1.2.3 Information Dissemination via Connectivity Decomposition . . . 27

1.2.4 Scheduling Many Distributed Protocols . . . 28

2 Models **31**

I Locality **34**

3 Maximal Independent Set **35**

3.1 Introduction & Related Work . . . 35

3.1.1 Our Result on Distributed MIS . . . 36

3.1.2 Technical Overview of Our Approach . . . 36

3.2 Preliminaries . . . 39

3.3 Our MIS Algorithm, & Its Local Complexity . . . 40

3.4 Our MIS Algorithm, and Improved Global Complexity . . . 44

3.5 Implications and Applications . . . 49

3.6 Open Questions . . . 54

II Congestion—Graph-Theoretic Foundations & Results **56**

4 Connectivity Decompositions **57**

4.1 Introduction & Related Work . . . 57

4.1.1 Introducing Connectivity Decompositions . . . 58

4.1.2 Known Results On Edge Connectivity Decomposition . . . 59

4.1.3 Our Results on Vertex Connectivity Decomposition . . . 60

4.1.4 Implications & Applications of Our Results . . . 61
4.2 Preliminaries . . . 63
4.3 Fractional CDS Packing . . . 64
4.3.1 High-Level Outline of the Construction . . . 64
4.3.2 Domination . . . 66
4.3.3 Connectivity . . . 67
4.3.4 Wrap Up . . . 71
4.4 Integral CDS Packing . . . 72
4.5 Bad Graphs with No Large Fractional CDS Packing . . . 74
4.6 Vertex Connectivity Decomposition and Throughput in Network Information Dissemination . . . 76

5 Vertex Connectivity Under Random Sampling **79**
5.1 Introduction & Related Work . . . 79
5.2 Probability Concentration for Connectivity . . . 81
5.2.1 Warm Up . . . 81
5.2.2 Intuition and Challenges in Proving Theorem 5.2.1 . . . 82
5.2.3 Proof of Theorem 5.2.1 via Semi-Connectivity . . . 84
5.2.4 Proof of Lemma 5.2.7: Sampling for $\boldsymbol{\lambda}$-Semi-Connectivity . . . 87
5.3 From Concentration for Simple Connectivity to High Connectivity . . . 98
5.3.1 Vertex Connectivity under Vertex Sampling . . . 98
5.3.2 Vertex Connectivity under Edge Sampling . . . 99
5.4 Optimality of Our Sampling Results . . . 100
5.5 Discussion and Open Problems . . . 102

III Congestion—Distributed Algorithms **104**

6 Low-Congestion Shortcuts, and Minimum Spanning Tree **105**
6.1 Introduction . . . 105
6.1.1 The Motivation and Definition of Low-Congestion Shortcuts . . . 106
6.1.2 Our Results . . . 107
6.2 Shortcuts in Action: Minimum Spanning Tree . . . 108
6.2.1 The Generic Method For Using Shortcuts . . . 108
6.2.2 Computing an MST via Shortcuts . . . 109

7 Minimum Edge Cut Approximation **113**
7.1 Introduction & Related Work . . . 113
7.1.1 Our Contribution . . . 114
7.1.2 Related Work in the Centralized Setting . . . 115
7.2 Preliminaries . . . 116
7.3 Edge Sampling and The Random Layering Technique . . . 118

7.4 A Constant Approximation of Minimum Edge Cut . . . 120
7.4.1 Algorithm Outline . . . 121
7.4.2 Testing Cuts . . . 123
7.4.3 Wrap up . . . 125
7.5 A $(2+\varepsilon)$ Approximation of Minimum Edge Cut . . . 126
7.6 A $(1+\varepsilon)$ Approximation of Minimum Edge Cut . . . 128
7.6.1 High-Level Description of the Algorithm . . . 129
7.6.2 Subtree Sums . . . 130
7.6.3 Approximating Tree-Edge Induced Cuts . . . 133
7.7 Lower Bounds . . . 135
7.7.1 Generalized Simulation Theorem . . . 135
7.7.2 Lower Bound for Approximating Minimum Cut: Weighted Graphs . . . 138
7.7.3 Lower Bound for Approximating Minimum Cut: Simple Unweighted Graphs . . . 141

8 Minimum-Weight Connected Dominating Set 145
8.1 Introduction & Related Work . . . 145
8.1.1 Result . . . 146
8.1.2 Other Related Work . . . 147
8.2 Preliminaries . . . 147
8.3 The Algorithm for MCDS . . . 148
8.3.1 The Outline . . . 148
8.3.2 A High-level View of the Algorithm for One Phase . . . 149
8.3.3 The Algorithm For One Phase . . . 152
8.4 Analysis . . . 155
8.4.1 Cost Related Analysis . . . 156
8.4.2 Speed Related Analysis . . . 157
8.5 Round Complexity Lower Bound . . . 160
8.6 Open Problems and Future Work . . . 161

9 Distributed Connectivity Decomposition 163
9.1 Introduction & Related Work . . . 163
9.1.1 Applications . . . 164
9.1.2 Other Related Work . . . 166
9.2 Preliminaries . . . 168
9.3 Dominating Tree Packing Algorithm . . . 169
9.4 Dominating Tree Packing Analysis . . . 172
9.4.1 Connector Paths . . . 173
9.4.2 The Fast Merger Lemma . . . 175
9.5 Distributed Dominating Tree Packing . . . 179
9.5.1 Identifying the Connected Components of Old Nodes . . . 180

9.5.2 Creating the Bridging Graph . . . 180
9.5.3 Maximal Matching in the Bridging Graph . . . 181
9.5.4 Wrap Up . . . 182
9.6 Centralized Dominating Tree Packing . . . 182
9.7 Testing A Dominating Tree Packing . . . 184
9.8 A Simple Application Example: Gossiping . . . 187
9.9 Distributed Fractional Spanning-Tree Packing . . . 188
9.9.1 Fractional Spanning Tree Packing for Small Edge Connectivity . . . 188
9.9.2 Generalized Fractional Spanning Tree Packing . . . 190
9.10 Lower Bounds . . . 192
9.10.1 Lower Bound Construction . . . 193
9.10.2 Reduction . . . 196

IV Congestion—Scheduling Distributed Protocols 200

10 Scheduling Distributed Protocols 201
10.1 Introduction & Related Work . . . 201
10.1.1 General Motivation and Background . . . 201
10.1.2 A Zoomed-In View of Scheduling and Our Terminology . . . 202
10.1.3 Our Technical Contributions . . . 204
10.2 Preliminaries . . . 206
10.2.1 Communication Patterns and their Simulations . . . 206
10.2.2 The Distributed Protocol Scheduling Problem . . . 208
10.3 Lower Bound . . . 210
10.4 Upper Bound . . . 218
10.4.1 Warm-Up: Existence Proof for a Short Schedule . . . 218
10.4.2 The Main Challenge and the Outline of Our Approach . . . 219
10.4.3 Scheduling Distributed Protocols via Locally Sharing Randomness . . . 221
10.5 A Recipe for Removing Shared Randomness in Bellagio Distributed Algorithms 230
10.6 Concluding Remarks and Future Work . . . 234

Chapter 1

Introduction

Distributed graph algorithms—also known as *network algorithms*—is an area whose objective is to provide efficient methods for networked computations and more generally a rigorous understanding of the fundamental principles of distributed computing in networks. The prototypical setup is as follows: a number of individual entities, each with some computational power, are connected to each other via a potentially large and complex network; the entities interact and communicate, by exchanging messages through the network, in order to perform some computational task. The classical motivation for studying distributed graph algorithms was to address the problems faced by computer networks, for example, routing in packet-switched networks. However, this area has grown well beyond that original motive and is now relevant for a far wider range of settings: it also applies to processors in a supercomputer, cores on a chip, and gates in a logic circuit, or even more broadly to human beings in a social network, insects in a colony, and neurons in a brain. With the ever-increasing role that networks play in all aspects of life, and the clear trend of the computational world moving towards decentralized processing, the relevance of network algorithms and distributed computing is constantly growing. Perhaps it is more certain now than ever before that

the future is distributed!

This thesis tackles some of the central problems of distributed graph algorithms, in two subareas revolving around two of its principal challenges, *locality* and *congestion*. We next briefly overview our contribution in each part. A more detailed description follows afterward.

In the subarea centered on locality, the objective is to characterize network computations that can be performed very efficiently, without the need to learn global information about the network. Note that the latter is undesirable as it can typically be quite time-consuming. Arguably, the most central problem of this subarea has been the Maximal Independent Set (MIS) problem. This is especially true because most of the other classical problems reduce to MIS. We present a novel algorithmic method which leads to significant improvements for MIS. We also explain how, thanks to the centrality of MIS, this new algorithm leads to improvements in about a dozen other problems of the area.

In the subarea centered on congestion, the objective is to understand how network computations can cope with the limited communication bandwidth of the network and to develop methods for utilizing this bandwidth efficiently. We provide a novel and systematic approach for exploiting the network's bandwidth, almost as much as possible. In an informal sense, the approach is to decompose the connectivity of the network into small and more manageable units, and then work with each of these units separately. For that, we first develop the purely graph-theoretic foundations of these connectivity decompositions, which itself leads to answers for a couple of problems in graph theory. We then develop efficient distributed algorithms which allow us to take these connectivity decompositions to the distributed setting. Along the way, we provide distributed algorithms for a number of graph problems, which improve on the respective state of the art, and solve a few related open problems.

Finally, we conclude our treatment of congestion by investigating a different aspect of this challenge: we study how to schedule and run many network computation protocols simultaneously, given the network's bandwidth limitations, and we provide a nearly tight characterization of this problem in a particular mathematical abstraction of it.

1.1 Part I: Locality

A fundamental challenge at the heart of the study of network algorithms is *locality*. In a network, each processor can communicate directly with its adjacent processors. However, accessing the information known only to the far-away processors requires more time. As such, in any time-efficient networked computation, each processor's output is dictated by the information known to the processor itself or to the others nearby. But this is clearly only a partial and local part of the global problem. That leads to the following general question, which forms the underpinning of the subarea of *locality*:

To what extent can a global solution be obtained from local data?

The classical mathematical formulation in investigating this question uses a standard message passing model, due to Linial [Lin92], called the LOCAL model: the network is abstracted as an undirected graph $G = (V, E)$ where one processor resides in each node of the graph. Initially, each processor/node knows only its neighbors[1]. Communication happens in synchronous rounds where per round each processor can send one message to each of its neighbors.

The standard objective is to characterize the round complexity of each given computational problem, i.e., to determine the smallest number of rounds necessary for solving the problem, modulo constant factors. Besides determining the time needed for solving the problem in distributed settings, this round complexity characterizes the locality of the problem in a precise mathematical sense. The reason is as follows: It is easy to see that, in any t-round algorithm, each node can learn at most the information in nodes within its distance t. Thus,

[1] At the risk of some level of informality, we use the terms processor and node interchangeably in the remained of this section.

each node's output depends only on this information, and particularly on the structure of the subgraph induced by the nodes within distance t. Moreover, in its basic form, the model does not restrict the message sizes. Therefore, in t rounds, each node can learn all the information held initially by the nodes within its distance t. Hence, providing a t-round algorithm for a given problem, or ruling out the possibility of the existence of such an algorithm, show bounds on what local radius around each node dictates its solution in the problem.

1.1.1 The MIS Problem, its Centrality, and State of the Art

Locality is the oldest branch of distributed graph algorithms. Since its beginning in the 1980's, the Maximal Independent Set (MIS) problem has been the most central problem of this subarea, and it has received extensive attention. This is symbolized in the two Dijkstra prizes awarded for upper and lower bounds on MIS, respectively to Linial [Lin92] in 2013, and to Luby [Lub86] and Alon, Itai, and Babai [ABI86] in 2016. We next briefly review the reasons for this primal role.

The Centrality of MIS As explicitly stated by Panconesi and Rizzi [PR01] in the early 1990's, there are four classical problems which have been central to the subarea of locality:

1. Maximal Independent Set, i.e., finding a set of maximal mutually non-adjacent vertices, where maximality implies that every vertex is either in the set or has a neighbor in it,
2. $(\Delta+1)$-vertex coloring, i.e., assigning a color from $1, 2, \ldots, \Delta+1$ to each vertex such that no two adjacent vertices have the same color,
3. $(2\Delta-1)$-edge-coloring, i.e., assigning a color from $1, 2, \ldots, 2\Delta-1$ to each edge such that no two edges that share an endpoint vertex have the same color, and
4. Maximal Matching, i.e., finding a maximal set of edges no two of which share an endpoint.

As observed early on [Lin92], all these problems can be *reduced* locally to MIS. This means that the MIS problem is the hardest of them and given a LOCAL-model algorithm for MIS, one can solve all these other problems in essentially the same round complexity, using only a few simple extra steps. This fact gave the MIS problem a unique and paramount role in the subarea of locality, e.g., the citation of the 2016 Dijkstra prize calls it "*a crown jewel of distributed symmetry breaking problems*".

The MIS problem also has various applications in a wide range of practical settings: from job scheduling in multi-processor computing, to medium access control in wireless networks, and even to biological settings such as the selection of Drosophila sensory organ precursor (SOP) in the nervous system of the fly.

The common denominator to all these settings is captured by the following toy example: Imagine that a number of tasks need to be performed but there is interdependency

between these tasks, abstracted by a graph, and no two adjacent tasks can be performed simultaneously. For instance, the tasks can be transmissions in a wireless network and no two adjacent antennas should transmit together due to the signal interferences that they would cause. Then we wish to find an *independent* set of tasks to be performed, and we also wish this set to be as large as possible, especially in a local sense. Particularly, we usually desire the set to be a *maximal* independent set[2] which means that for each task, either the task is in the set chosen to be performed, or there is clear excuse for not having it included because an adjacent task is included in the set.

State of the Art for MIS MIS has received extensive attention since the 1980's. The story can be traced back to the surveys of Valiant [Val82] and Cook [Coo83] which mentioned MIS as an interesting problem in non-centralized computation. Shortly after, Karp and Wigderson [KW84], presented the first $\text{poly}(\log n)$ round randomized distributed/parallel algorithm. That was followed by the ingenious $O(\log n)$ round randomized distributed algorithm of Luby [Lub86] and Alon, Babai, and Itai [ABI86], presented independently and simultaneously in 1986. This algorithm has played a prominent role throughout the subarea of distributed graph algorithms. Moreover, in the extreme of high-degree graphs when $\log \Delta = \Omega(\log n)$, where Δ denotes the maximum degree, this bound still remains the best known.

As progress on the algorithmic side slowed down and seemed less feasible, researchers turned to proving lower bounds, showing that distributedly solving MIS requires at least so many rounds. In a seminal work, Linial [Lin92] showed in 1992 that even on a simple ring graph—i.e., a cycle of n vertices—solving MIS requires $\Omega(\log^* n)$ rounds. A major breakthrough came in 2004 when Kuhn, Moscibroda, and Wattenhofer [KMW04, KMW16] showed that any MIS algorithm requires at least $\Omega(\frac{\log \Delta}{\log \log \Delta})$ rounds on some graphs with maximum degree Δ. Though, it should be noted that there is a restriction on the range of Δ as a function of the network size n. In other words, the lower bound graph requires a large network size n as a function of Δ. When expressed as a function of n and Δ, the lower bound becomes $\Omega(\min\{\frac{\log \Delta}{\log \log \Delta}, \sqrt{\frac{\log n}{\log \log n}}\})$.

This gap between the $O(\log n)$ upper bound and the $\Omega(\min\{\frac{\log \Delta}{\log \log \Delta}, \sqrt{\frac{\log n}{\log \log n}}\})$ lower bound lasted for a few years, until in 2011, Barenboim, Elkin, Pettie, and Schneider [BEPSv3] presented a new algorithm with a round complexity of $O(\log^2 \Delta) + 2^{O(\sqrt{\log \log n})}$, and more importantly, a general new technique for coping with local problems, now referred to as *graph shattering*. This new bound improves on the $O(\log n)$ upper bound of [Lub86, ABI86] when $\log \Delta = o(\sqrt{\log n})$. Still, the gap remained significant. Especially when focusing on the dependency on Δ, the remaining gap was approximately quadratic, between $O(\log^2 \Delta)$ and $\Omega(\frac{\log \Delta}{\log \log \Delta})$.

[2] We remark that one may wish to alternatively find a *maximum* independent set. However, it is well-known that computing a maximum independent set distributedly can be prohibitively time-consuming, it would require $\Omega(n)$ rounds in a cycle graph.

1.1.2 Our Result On MIS and Its Implications

Our Result We present a randomized distributed MIS algorithm that achieves a round complexity of $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$. This result appeared in [Gha16b]. This is the first algorithm, after about 30 years of research on MIS, to attain a nearly optimal bound for a wide range of the parameter values. Particularly, its $O(\log \Delta)$ dependency on Δ is nearly optimal, almost matching the $\Omega(\frac{\log \Delta}{\log \log \Delta})$ lower bound of [KMW04, KMW16]. Furthermore, as we will explain in the technical sections, we now understand why this bound might be the best achievable given the current techniques: e.g., by an argument of Chang et al. [CKP16], it is known that improving the latter term $2^{O(\sqrt{\log \log n})}$ would improve the $2^{O(\sqrt{\log n})}$ round complexity of deterministic distributed algorithms for MIS [PS92], which would be considered as a major breakthrough.

Besides the end result, the algorithm exhibits many desired and some surprising properties, as exemplified by its very local guarantees: it gives that the (expected) time till the removal of each node v depends only on the degree of v itself, and not even its neighbors! The dependency on the degree is logarithmic, which is nearly optimal as implied by [KMW04, KMW16]. This time complexity also has a nice probabilistic concentration, with an exponential decay. We will formalize this later. Moreover, this local guarantee relies only on the randomness used by nodes within distance 2 of v, that is, the guarantee would hold even if the random coins used by the processors outside this 2-hop neighborhood are determined adversarially.

Perhaps most surprisingly, the core of the algorithm is extremely simple, natural, and practical. The chief idea is a careful but clean *negative feedback* that over time dynamically adjusts the probability of each node trying to join the MIS based on the probabilities of its neighbors. In an intuitive sense, this method can be viewed as a blending of ideas from the *Multiplicative Weight Update* with those of the *Belief Propagation* methods.

Implications and Applications Our result on MIS leads to improvements on several other problems: Using standard reductions, our algorithm leads to a unified and simple approach for achieving a round complexity of $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ for the other three classical local problems mentioned above, namely $(\Delta + 1)$-vertex-coloring, $(2\Delta - 1)$-edge-coloring, and Maximal Matching. However, we note that there are better bounds known for some of these problems. That will be reviewed carefully in the related work section.

Combined with other techniques, we also get a faster LOCAL algorithm for the *Lovász Local Lemma*, improving on results of [CPS14]; even faster algorithms for MIS in graphs with low arboricity, improving on results of [BEPSv3, BE10, LW11]; faster ruling set algorithms, improving on results of [BKP14, BEPSv3]; an $O(\sqrt{\log n})$-round MIS algorithm for Erdös-Rényi random graphs $G(n, p)$, improving on the previous best bound of $O(\log n)$ [Lub86], and an $O(\log \Delta)$-round MIS algorithm for the CONGESTED-CLIQUE model of distributed computing.

Perhaps more instructive than all the above, the new result highlights in a concrete and

precise sense the barrier of the current lower bound techniques. It shows that in graphs with *tree-like* local topologies—which are the only family for which we have distributed lower bound techniques—the lower bound of $\Omega(\min\{\frac{\log \Delta}{\log\log \Delta}, \sqrt{\frac{\log n}{\log\log n}}\})$ is tight, up to doubly logarithmic factors. Hence, to prove noticeably better lower bounds, we need to come up with lower bound techniques that can handle graphs with *non-tree-like* local topologies. This again will be made precise in the related section.

Further Developments and Extensions Subsequent to the appearance of the conference version of our MIS result in [Gha16b], there have been a number of works that use this algorithmic idea, sometimes with some modifications, to solve other problems:

- Bar-Yehuda, Censor-Hillel, Ghaffari, and Schwartzman [BYCHGS16] used modifications of this idea to achieve a round complexity of $O(\frac{\log \Delta}{\log\log \Delta})$ for distributed $(1+\varepsilon)$ approximation of maximum cardinality matching and $(2+\varepsilon)$ approximation of maximum weight matching, improving on $O(\log n)$ round algorithms of [LPSR09, LPSP15],
- Censor-Hillel, Parter, and Schwartzman [CHPS16] used a direct derandomization of this algorithm to present an $O(\log \Delta \log n)$ round deterministic distributed MIS algorithm in the CONGESTED-CLIQUE model,
- Holzer and Lynch [HL16] extended and modified this algorithm to present an efficient MIS algorithm in the beeping model of wireless networks, and
- Ghaffari [Gha16a] used this algorithm as a starting point for an MIS algorithm with a round complexity of $\tilde{O}(\frac{\log \Delta}{\sqrt{\log n}} + 1) = \tilde{O}(\sqrt{\log \Delta})$ in the CONGESTED-CLIQUE model, which is the first such algorithm with a sublogarithmic complexity.

1.2 Part II: Congestion

Another fundamental challenge in network algorithms is to cope with the limited communication capacities of the networks. Communication networks usually have certain bandwidth limitations, e.g., each communication link can transfer messages only up to a certain rate, or each node can upload information only up to a certain rate. Since in any networked computation or communication, many messages might need to pass from one node to the other, this limited bandwidth naturally gives rise to the possibility of *congestion*. This makes congestion a central challenge for virtually all network algorithms. As such, a significant portion of the research in the area of distributed graph algorithms deals with issues rooted in congestion. In this subarea, the prototypical questions are as follows:

What is the maximum achievable throughput in network information dissemination?

and

Given limited bandwidth, how fast can the network solve a computational problem?

To mathematically formulate these and other related problems, the standard model in the area is a variant of the LOCAL message passing model explained above, which is called CONGEST [Pel00]. We consider two variants of this model: In the first version called E-CONGEST, which stands for edge congestion, per round at most B bits can be sent along each edge. In the second version called V-CONGEST, which stands for vertex congestion, per round each node can send at most B bits in total, which are delivered to all of its neighbors. Notice that V-CONGEST is a more stringent model and any algorithm in this model can be run in the E-CONGEST model. This is because the B bits that are sent by each node to all of its neighbors in the V-CONGEST model can also be transferred to all of them in the E-CONGEST model which admits B-bit messages per edge per round. We also note that the most common parameter setting is $B = O(\log n)$. This essentially means that each message can carry the equivalent of constant many "words", each describing the identifier of one node or one edge in the graph.

The standard objective is to characterize the round complexity of each given computational or communication problem, that is, to determine the smallest number of rounds necessary for performing a particular computational task or for communicating some information from some nodes to some others.

We take a systematic approach in addressing this challenge, as we describe next. We first explain the graph theoretic foundations of this approach and the graph theoretic results that we develop for it, in Section 1.2.1. These cover results that appeared in [CHGK14b, CGG+15]. Then, in Section 1.2.2, we explain the related distributed computation aspects, our distributed algorithms, and their applications. These distributed algorithms also provide answers for some classical network optimization graph problems. This part covers results that appeared in [Gha14, GK13, GH16b]. In Section 1.2.3, we explain the applications of our connectivity decomposition results for distributed information dissemination in networks. This covers results that appeared in [CHGK14a] and some remaining parts of [CHGK14b]. Finally, in Section 1.2.4, which covers the results of [Gha15b], we explain a treatment of a general scheduling questions centered on congestion, which also relates to the algorithms of Section 1.2.2.

1.2.1 Graph-Theoretic Foundations

Transferring information throughout the network is one of the central objectives of any communication network and also a core issue in distributed computing [Pel00, Section 1.3.1]. The flow of information in a network has a clear connection to the notion of *graph connectivity* in graph theory. In simple and colloquial terms, networks that have bandwidth bottlenecks cannot admit large flows of information. Mathematically, this concept is captured by graph connectivity, namely *edge connectivity* and *vertex connectivity*, which are among the most basic and well-studied graph-theoretic concepts.

Formally, *edge connectivity* and *vertex connectivity* are defined as the smallest number of edges or vertices, respectively, whose removal disconnects the graph. Edge and vertex connectivity characterize limitations on how well information can be transferred throughout the network. This is because each edge or vertex cut—a set of edges or vertices whose removal disconnects the graph—defines an upper bound on the flow across the cut. Naturally, we expect networks with larger connectivity—that is, those in which the minimum cut size is larger—to provide a better communication medium and support larger flow. However, designing distributed algorithms that leverage large connectivity remains challenging.

We propose a systematic method for exploiting large connectivity towards the goal of obtaining a large flow of information. We use a rather natural approach, which we call *connectivity decomposition*, of decomposing connectivity into smaller and more manageable units. In simple terms, we decompose a graph with large connectivity into many (essentially) "disjoint" trees, while almost preserving the total connectivity through the trees. This will be made formal shortly. These decompositions allow us to parallelize the flow of information along the trees and thus achieve a total flow value close to the connectivity of the network.

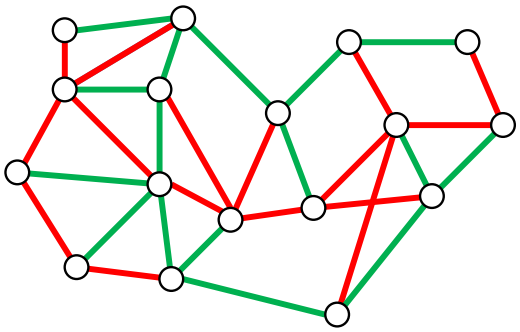
An interesting comparison of our approach of handling *congestion* is to that of addressing *locality*. Classically, locality has received significantly more attention and many general techniques are known for dealing with it. Congestion on the other hand is less understood and the methods used for handling congestion in different problems appear to be rather ad hoc, each being well-suited for a particular problem. A fundamental and generic technique centered around locality is *locality-based decomposition* [Pel00], which groups nodes in small-diameter clusters with certain properties; classical examples include [ALGP89, ABCP92, ABCP96, PS92, KP95]. We believe that connectivity decompositions can be viewed as a direction orthogonal to that of *locality-based* decompositions, and we hope they are a first step towards systematically addressing congestion.

We next explain these connectivity decomposition concepts, while putting them in the context of the classical graph theoretic results on graph connectivity.

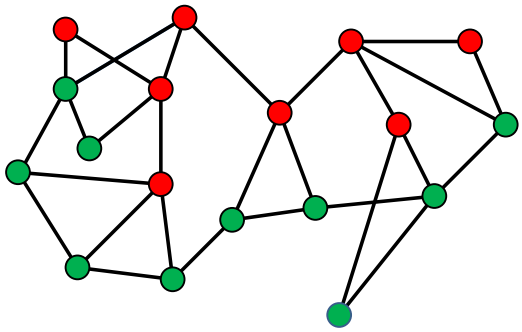
Connectivity Decomposition and Tree Packings

Menger's theorem (see [BM08, Chapter 9])—which is one of the most basic results concerning graph connectivity—states that in each graph with edge connectivity λ or vertex connectivity k, each pair of vertices are connected via λ edge-disjoint paths or k internally vertex-disjoint paths, respectively. However, when we have to deal with more than two nodes, this theorem does not provide a strong enough characterization. This is especially because it does not provide any information about the structure of the overlaps between paths of different vertex pairs, or on structures connecting three or more nodes (i.e., *Steiner trees*).

To organize the connectivity in a way that makes it accessible to algorithms, and especially for distributed algorithms, we consider edge and vertex connectivity decompositions. When we are interested in edge connectivity and edge cuts, this will be by decomposing the graph into *edge-disjoint spanning* trees. When we are interested in vertex connectivity and



(a) Two edge-disjoint spanning trees, i.e., a spanning tree packing of size 2.



(b) Two vertex-disjoint dominating trees, i.e., a dominating tree packing of size 2.

vertex cuts, this will decompose the graph into *vertex-disjoint dominating* trees.

Recall that a tree $H = (V_T, E_T)$ is a *spanning tree* of $G = (V_G, E_G)$ if $E_T \subseteq E_G$ and $V_T = V_G$. On the other hand, a tree $H = (V_T, E_T)$ is called a *dominating tree* of $G = (V_G, E_G)$ if $E_T \subseteq E_G$ and V_T dominates G, i.e., each node in $V_G \setminus V_T$ has a G-neighbor in V_T[3].

We next describe and discuss the concepts of edge and vertex connectivity decomposition, via spanning and dominating tree packings, respectively:

- **Edge Connectivity Decomposition via Spanning Tree Packing:** We define a *spanning tree packing of size* λ' to be a collection of λ' edge-disjoint spanning trees. Given such a packing, for each pair of vertices, we get λ' edge-disjoint paths, one through each tree. More importantly, for any number of vertex pairs, the paths going through different trees are edge-disjoint. Hence, a spanning tree packing can be viewed as a decomposition of edge connectivity. Figure 1-1a shows an example graph with a spanning tree packing of size 2.

- **Vertex Connectivity Decomposition via Dominating Tree Packing:** We define a *dominating tree packing of size* k' to be a collection of k' vertex-disjoint dominating trees. Given such a packing, for each vertex pair we get k' internally vertex-disjoint paths, one through each tree. More importantly, for any number of pairs, the paths going through different trees are internally vertex-disjoint. Hence, a dominating tree packing can be viewed as a decomposition of vertex connectivity. Figure 1-1b shows an example graph with a dominating tree packing of size 2.

Let us explain informally how these connectivity decompositions allow us to readily leverage a network's connectivity in information dissemination. Consider the E-CONGEST model and imagine that there are a large number $N \to \infty$ of messages, spread throughout the network, which all need to be broadcast to all nodes. Without network decompositions, the best known solution—which is just by standard message pipelining techniques [Pel00]—could

[3]Note that if we want to have many vertex-disjoint subgraphs, we cannot ask them to be *"spanning"* subgraphs. As we soon see, in this case, the *"dominating"* condition turns out to be the natural and the practically relevant requirement.

only broadcast messages with a throughput of one message per round. Given an edge connectivity decomposition into λ' edge-disjoint spanning trees, we can improve this throughput to $\Theta(\lambda')$. For that, we split the message among the trees, say by random assignments, and then we simultaneously perform broadcasts in each of the trees, where each tree is used only to broadcast the messages assigned to it.

Fractional Relaxation In both spanning and dominating tree packings, we can relax the disjointness requirement to *fractional disjointness*. That is, we allow the trees to overlap but now each tree τ has a weight $\omega_\tau \in [0, 1]$ and for each edge or vertex, respectively, the total weight of the trees including it has to be at most 1. In applications, this naturally corresponds to sharing the edge (or the vertex) between the trees proportional to their weights. For instance, in information dissemination, we would use time-sharing and each edge (or vertex) gets used in each of the trees that include it in a fraction of time proportional to the weight of that tree. In fact, for all the applications that we have in mind, these fractional tree packings are as useful as their stronger integral packing counterparts.

Previously Known Results and Our Results on Connectivity Decomposition Edge connectivity decompositions have been well-known[4] for a long time, thanks to beautiful (existential) results of Tutte [Tut61] and Nash-Williams [NW61] from 1960. These results show that each graph with edge-connectivity λ contains a spanning tree packing of size $\lceil \frac{\lambda-1}{2} \rceil$, see [Kun74]. This bound is existentially tight even for the fractional version and has numerous important applications.

We proposed vertex connectivity decompositions in [CHGK14a]. We show that each graph with vertex-connectivity k contains a fractional dominating tree packing of size $\Omega(k/\log n)$, and that the $\Omega(k/\log n)$ fractional packing bound is existentially best possible. For disjoint dominating tree packings, we shows that each graph with vertex-connectivity k contains a dominating tree packing of size $\Omega(\kappa/\log^2 n)$, where κ is the vertex-connectivity of the sampled graph when each vertex is randomly sampled with probability $1/2$. Separately, we prove that with high probability $\kappa = \Omega(k)$. This is a special case of a more general result that we discuss shortly.

These results also have implications for other problems: we get a $\Theta(\log^2 n)$ approximation of the Itai-Zehavi conjecture on vertex-independent trees [ZI89], and we also can show that the network coding gap in networks with bandwidth limitations on nodes has a tight bound of $\Theta(\log n)$. Both of these points will be made more clear in the technical sections.

Graph Connectivity Under Random Sampling

As alluded to above, to obtain our vertex connectivity decomposition result, we need to analyze how a graph's vertex connectivity changes as vertices are removed randomly. This

[4] Although we remark that they were not called "edge-connectivity decomposition". Often, the phrase "edge-disjoint spanning trees" was used to refer to the related concept.

itself touches on central questions in areas of broad interest: *random graph processes under node removals*, *network reliability under node failures*, and *site percolation theory*.

The problem of characterizing the vertex connectivity under random vertex sampling had remained open, while the analogues on edge connectivity under edge sampling were resolved more than two decades ago due to results of Karger [Kar94b]. As we will discuss, despite the apparent resemblance, the two problems are quite different intrinsically and the known methods for analyzing edge connectivity under edge sampling do not extend to the vertex connectivity case. We next explain our result and put it in the broader context of graphs under random sampling.

Consider a random process where given a base graph G, each edge or node of G is sampled with some probability p. Given such a random graph process, it is interesting to see how various global connectivity properties of the graph induced by the sampled edges or nodes change as a function of the sampling probability p. If G is the complete n-node graph, sampling each edge independently with probability p results in the classic Erdős-Rényi random graph $G_{n,p}$, for which exact thresholds for the formation of a giant component, global connectivity, and many other properties have been studied (e.g., in [Bol98]). Thresholds for the formation of a giant component are further studied in percolation theory [BR06]—mostly for regular or random lattice graphs G. In the context of percolation theory, edge sampling is called bond percolation whereas vertex sampling is referred to as site percolation.

We are particularly interested in how the vertex connectivity of a general graph G changes under uniform random vertex (or edge sampling). For edge connectivity and edge sampling, the analogous question has been resolved two decades ago. Karger's seminal result [Kar94b] showed that for any λ-edge-connected graph with n vertices, sampling edges independently at random with probability $p = \Omega(\log(n)/\lambda)$ results in an $\Omega(\lambda p)$-edge-connected subgraph, with high probability[5]. This was a strong extension of the earlier result by Lomonosov and Polesskii [LP72], which stated that sampling each edge with probability $\Theta(\log(n)/\lambda)$ leads to a connected subgraph, w.h.p. These sampling results and their extensions were cornerstone tools for addressing various important problems such as various min-cut problems [Kar94a, Kar94b], constructing cut-preserving graph sparsifiers [BK96, ST04], max-flow problems [Kar94a, KL98], and network reliability estimations [Kar95].

As in the case of edge connectivity, studying the vertex connectivity of the subgraph obtained by independently sampling vertices or edges of a k-vertex-connected graph is of fundamental interest. However, the vertex connectivity case has been recognized as being much harder and progress has been scarce. It was not even known whether a $\Theta(n)$-vertex-connected graph stays (simply) connected when nodes are sampled with probability $p = 1/2$.

We prove that if $p = \Omega(\log n/\sqrt{k})$, then with high probability the sampled graph is connected and indeed has vertex connectivity $\Omega(kp^2)$. This covers results of [CHGK14b,

[5]As standard throughout theory of computation, we use the phrase '*with high probability*' (*w.h.p.*) to indicate that some event has a probability of at least $1 - n^{-\Theta(1)}$. Intuitively, this means that within the life time of system/algorithm, which are usually presumed to be at most polynomial in the network size, we are unlikely to see that this high probability event did not happen.

CGG+15].

1.2.2 Distributed Algorithmic Aspects

Here we overview the distributed algorithms that we present on various problems related to graph connectivity. We divide the exposition into two parts, those relating to edge connectivity and edge cuts, and those relating to vertex connectivity and vertex cut. Generally, the results in the former are more basic and can serve as a warm up for those of the latter. Finally, we also explain the information dissemination applications of our distributed connectivity decomposition algorithms.

Edge Connectivity Decomposition, Edge Cuts, and Spanning Trees

We next discuss our results on distributedly computing minimum spanning trees, minimum edge cuts, and edge connectivity decompositions. With the exception of the result on MST, all others improve on the state of the art. The MST algorithm is mainly explained as a subroutine for the other results. However, this algorithm itself provides a significantly simpler and more modular method for achieving almost the state of art bound and because of that, and especially due to the significance of MST in distributed graph algorithms, we present it as a separate result.

Minimum Spanning Tree (MST) Recall that in the MST problem, given an undirected graph $G = (V, E)$, we assume that each edge e of the graph has a real weight $w(e)$[6] and the objective is to compute a spanning tree of the graph with the minimum possible weight. MST is a well-studied problem throughout network optimizations, but it plays a much more significant role in distributed algorithms. Its algorithms and lower bounds have been used to derive algorithms and lower bounds for a wide range of other distributed graph problems.

The distributed round complexity of MST is essentially resolved, due to an influential line of work on algorithms [GHS83, GKP93, KP95] and lower bounds [Elk04c, PR99, DSHK+11]. These culminated in the best known upper bound being $O(D + \sqrt{n}\log^* n)$, due to Kutten and Peleg [KP95], and the best known lower bound being $\Omega(D + \sqrt{n/\log n})$, due to Das Sarma et al. [DSHK+11]. Here, n and D denote respectively the number of nodes and the diameter of the network graph.

We present a clean and general framework, centered around the concept of *low-congestion shortcuts* which we introduce, that leads to an $O(D+\sqrt{n}\log n)$ round MST algorithm. This in part discusses the framework set forth in [GH16b]. Note that this $O(D + \sqrt{n}\log n)$ round algorithm is slightly slower than the $O(D + \sqrt{n}\log^* n)$ round algorithm of [KP95]. However, the algorithm is significantly simpler and cleaner and thus, we believe that it is a more suitable choice when teaching distributed MST algorithms. On the other hand, the concept of low-congestion shortcuts is quite general and, in a certain sense, it can be used to

[6] These real weights are usually assumed to be truncated to a certain level of precision

(combinatorially) characterize the distributed round complexity of various graph problems in different graph families. We also use this framework in some of the latter results.

Edge Connectivity Decomposition via Spanning Tree Packing Recall that in edge connectivity decomposition, the objective is to decomposition the graph into as many as possible edge disjoint spanning trees. The classical 1961 results of Tutte [Tut61] and Nash-Williams [NW61] show that in each graph with edge connectivity λ, there exists an edge connectivity decomposition into $\lceil \frac{\lambda-1}{2} \rceil$ edge-disjoint spanning trees. We are not aware of an efficient distributed method for computing truly edge-disjoint spanning trees. However, we provide a nearly-optimal distributed algorithm for computing a fractional relaxation of such a spanning tree packing, which for all our applications, is functionally as good as having the actual spanning tree packing.

In particular, we show a randomized distributed algorithm with a round complexity of $\tilde{O}(D + \sqrt{n\lambda})$ that finds a fractional spanning tree packing of size $\lceil \frac{\lambda-1}{2} \rceil (1-\varepsilon)$, for any small constant $\varepsilon > 0$. This covers a results from [CHGK14a]. The algorithm follows a classical and generic approach (see e.g. [PST91, SM90, KPST94, Kar96]) which can be viewed as an adaptation of the Lagrangian relaxation method of optimization theory. Tailored to our problem, this general approach goes as follows: we gradually build a collection of weighted trees, by adding trees to the collection one by one. Each time, we define new weights on the graph edges and then compute a minimum spanning tree based on these new weights. The weights are defined by an exponential function of the hitherto load, which effectively penalizes the edges with large load and incentivizes the new tree to pass through those with lower loads. As such, the overall collection tends not to overload one or few edges, when possible. We will present a particular instantiation of this general approach which admits to a very simple and clean analysis.

Minimum Edge Cut Approximation Recall that an edge cut is defined by partitioning the vertex set into two non-empty sets; the edges with one end-point in each part are the *cut edges*, and removing them disconnected the graph. The minimum edge cut asks for finding a cut (i.e., a bipartition) for which the set of cuts edges has the smallest possible cardinality, or smallest possible weight, in the case of weighted graphs.

We present distributed algorithms that compute approximations of the (weighted) minimum edge cut with a nearly optimal round complexity. This covers results of [GK13]. In particular, we provide a $(1+\varepsilon)$ approximation for any constant approximation $\varepsilon > 0$ in $\tilde{O}(D+\sqrt{n})$ rounds. This matches the $\tilde{\Omega}(D+\sqrt{n})$ lower bound of Das Sarma et al. [DSHK+11], modulo logarithmic factors, and thus provides an answer to open questions of Elkin [Elk04c] and Das Sarma et al. [DSHK+11].

Vertex Connectivity Decomposition, Vertex Cuts, and Dominating Trees

We now overview our analogous distributed algorithmic results on graph problems relating to vertex connectivity. While in the world of decomposing edge connectivity, spanning trees are the building blocks, when decomposing vertex connectivity, the building blocks are dominating trees. As we will see, the right counterpart of minimum cost spanning trees is minimum cost dominating trees, where now the costs are on the vertices. In an informal level, the reason for having the costs on vertices is that now we are concerned with vertex connectivity, disjointness on vertices, and congestion on vertices.

Minimum Cost Connected Dominating Set (MCDS) In the MCDS problem, each vertex of the graph $G = (V, E)$ has a real weight $w(v)$ and the objective is to compute a connected dominating set of vertices—that is a set S which induces a connected subgraph $G[S]$ and also such that each node $v \in V \setminus S$ has at least one neighbor in S—such that the summation of the weights of S is minimized. The MCDS problem is often viewed as the node-weighted analogue of MST. Here, we recap this connection. The natural interpretation of the definition of CDS is that a CDS is a selection of *nodes* that provides *global connectivity*—that is, any two nodes of the graph are connected via a path that its internal nodes are in the CDS. On the counterpart, a *spanning tree* is a (minimal) selection of *edges* that provides global connectivity. In both cases, the problem of interest is to minimize the total weight needed for global connectivity. In one case, each edge has a weight and the problem becomes MST; in the other, each node has a weight and the problem becomes MCDS.

Despite the seemingly analogous nature of the two problems, MCDS turns out to be a significantly harder problem: The MST problem can be computed sequentially in $\mathcal{O}(m)$ time, where m is the number of edges [KKT95]. On the other hand, MCDS is NP-hard [GJ90], and in fact, unless P = NP, no polynomial time algorithm can find any approximation better than $\Theta(\log n)$-factor for it (see [Fei96, RS97, AMS06]). Furthermore, the known sequential algorithms for $O(\log n)$ approximation of MCDS (see [GK98, GK99]) have unspecified polynomial time complexity, which are at least $\Theta(n^3)$ and thus much larger than the linear time complexity of computing an MST.

We show that in the distributed setting, MCDS can be solved—that is, approximated optimally—in a time close to that of MST. More concretely, we present a randomized distributed algorithm in the CONGEST model that, with high probability, finds an $O(\log n)$ approximation of the minimum-weight connected dominating set, in $\tilde{O}(D + \sqrt{n})$ rounds. This covers results that appeared in [Gha14]. This algorithm is (near) optimal in both round complexity and approximation factor: Using techniques of [DSHK+11], we can reduce the *two-party set-disjointness communication complexity* problem on $\Theta(\sqrt{n})$-bit inputs to MCDS, proving that the round complexity is optimal, up to logarithmic factors, for any (non-trivial) approximation. As mentioned above, the $O(\log n)$ approximation factor is known to be optimal up to a constant factor, unless P = NP, and with the standard assumption that processors can perform polynomial-time computations.

Vertex Connectivity Decomposition via Dominating Tree Packing Recall from the previous subsection that, in the graph-theoretic part, we prove the existence of large vertex connectivity decompositions. We particularly prove that each graph with vertex-connectivity k contains a fractional dominating tree packing of size $\Omega(k/\log n+1)$ and a dominating tree packing of size $\Omega(k/\log^2 n+1)$.

We present efficient and indeed near-optimal distributed algorithms for obtaining such vertex connectivity decompositions. This covers results that appeared in [CHGK14a]. Concretely, we present a randomized distributed algorithm with a round complexity of $\tilde{O}(\min\{D+\sqrt{n}, \frac{n}{k}\})$-rounds randomized V-CONGEST model that with high probability finds a fractional dominating tree packing of size $\Omega(\frac{k}{\log n})$ and another algorithm with the same round complexity that produces a dominating tree packing of size $\Omega(\frac{k}{\log^2 n})$. We also prove this round complexity to be near-optimal. By extending results of [DHK+12], we show that any such decomposition algorithm requires $\tilde{\Omega}(D+\sqrt{\frac{n}{k}})$ rounds. Interestingly, this near-optimal distributed algorithm can also be turned into an $\tilde{O}(m)$-time centralized algorithm for the same problem.

Minimum Vetex Cut Approximation Our connectivity decomposition approach also leads to efficient algorithms for approximating vertex connectivity, both in the centralized and distributed models. This result appeared in [CHGK14a]. Let us put this result in the context of what is known.

Extensive attention has been paid to developing algorithms that compute or approximate vertex connectivity in the centralized model, and it is widely known that this problems is significantly harder than the counterparts in the edge connectivity case. In 1974, Aho, Hopcraft and Ulman [AHU74, Problem 5.30] conjectured that there should be an $O(m)$ time algorithm for computing the vertex connectivity. Despite much interesting work in this direction—e.g., [Tar74, Eve75, Gal80, Hen97, HRG96, Gab00]—finding $O(m)$ time algorithms for vertex connectivity has yet to succeed. The current state of the art is an $O(\min\{n^2k+nk^{3.5}, n^2k+n^{1.75}k^2\})$ time exact algorithm by Gabow [Gab00] and an $O(\min\{n^{2.5}, n^2k\})$ time 2-approximation by Henzinger [Hen97]. The situation is considerably worse in distributed settings and the problem of upper bounds has remained widely open.

Our dominating tree packing algorithm allows us to compute an $O(\log n)$ approximation of vertex connectivity, in $\tilde{O}(m)$ time in the centralized model, in $\tilde{O}(D+\sqrt{n})$ rounds of the V-CONGEST model of distributed computing. The former is the first sub-quadratic algorithm for approximating vertex connectivity. In terms of the time-complexity, both of the algorithms are nearly optimal.

1.2.3 Information Dissemination via Connectivity Decomposition

As their primary application, our connectivity decomposition results lead to time-efficient distributed constructions for broadcast algorithms with existentially optimal throughput. This result appeared first in [CHGK14a]. Note that in the V-CONGEST model, vertex cuts

characterize the main limits on the information flow, and k messages per round is the clear limit on the broadcast throughput in each graph with vertex connectivity k. Similarly, in the E-CONGEST model, edge cuts characterize the main limits on the information flow, and λ messages per round is the limit on the broadcast throughput in each graph with edge connectivity λ.

We get the following two results using our connectivity decomposition algorithms.

- In the V-CONGEST model, using our $\tilde{O}(D + \sqrt{n})$-round construction of fractional dominating tree packings, and then broadcasting each message along a random tree, we get a broadcast algorithm with throughput of $\Omega(\frac{k}{\log n})$ messages per round. We also prove that this throughput is existentially optimal.

- In the E-CONGEST model, using our $\tilde{O}(D + \sqrt{\lambda n})$-round construction of spanning dominating tree packings, and then broadcasting each message along a random tree, we get a broadcast algorithm with throughput of $\lceil \frac{\lambda-1}{2} \rceil (1 - \varepsilon)$ messages per round, which is asymptotically optimal.

1.2.4 Scheduling Many Distributed Protocols

In most of the distributed graph algorithms mentioned above, and in fact more generally in most of the distributed algorithms, we need to simultaneously perform many subroutines. Each of these subroutines might compute a different function, or they might compute different instances of the same function but for different parts of the network. For instance, this is clearly the case when we discuss low-congestion shortcuts and distributed computation of minimum spanning tree.

Simultaneously having many computational tasks is a core issue when dealing with congestion in networked computations and it can be cast much more generally, as follows: Computer networks are constantly running many applications at the same time and because of the bandwidth limitations, each application gets slowed down due to the activities of the others. Despite that, for the vast majority of the distributed algorithms, the design and analysis are carried out with the assumption that each algorithm uses the network alone. We investigate what happens when many distributed protocols are to be run together. The results related to this investigation appeared in [Gha15b].

Specifically, we study the questions of *how to run these distributed protocols simultaneously as fast as possible* and *what are the limitations on how fast this can be done*. While being arguably a fundamental issue, to the best of our knowledge, these questions have not been investigated in their full generality. The general scenario we consider is as follows: We want to run independent distributed protocols $\mathcal{A}_1$, $\mathcal{A}_2$, ..., $\mathcal{A}_k$ together in the CONGEST model, but we do not know what problem is being solved by each protocol. Hence, we must run each protocol essentially in a black-box manner without altering the content of its messages, except for potentially adding a small amount of information to its header.

This problem has two trivial lower bounds: if one of the protocols $\mathcal{A}_1$, $\mathcal{A}_2$, ..., $\mathcal{A}_k$ that are to be run together takes d rounds, running all of them together will clearly require at least d rounds. We refer to the maximum running time of the protocols as $\mathsf{dilation}$. Moreover, consider a particular edge e of the graph. Let $c_i(e)$ be the number of rounds in which protocol $\mathcal{A}_i$ sends a message over e. Then, running all the protocols together will also require at least $\mathsf{congestion} = \max_{e \in E} \mathsf{congestion}(e)$ rounds where $\mathsf{congestion}(e) = \sum_{i=1}^{k} c_i(e)$. Hence, we can conclude that running all protocols together will need at least $\max\{\mathsf{congestion}, \mathsf{dilation}\} \geq (\mathsf{congestion} + \mathsf{dilation})/2$ rounds.

The key question of interest is "*can we always find a schedule close to the* $\Omega(\mathsf{congestion} + \mathsf{dilation})$ *lower bound and if yes, can we do that distributedly?*" We provide two complementary answers:

- The result that is technically more interesting is showing an impossibility. We particularly use the probabilistic method to prove that the trivial lower bound is not always achievable. This might seem somewhat surprising, especially if contrasted with the well-known packet-routing results of [LMR94], which show that if each protocol was simply routing (i.e., forwarding) a packet along an arbitrary fixed path, then the trivial lower bound could be matched. We prove that there are instances of the distributed protocol scheduling problem for which any schedule, even if constructed in a centralized manner and with no computational restriction, requires $\Omega(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log\log n)$ rounds.
- We complement the above by presenting an efficient distributed algorithm for finding a near-optimal schedule. Particularly, we show that there is a randomized distributed algorithm that for any instance of distributed protocol scheduling, produces a schedule of length $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$, in $O(\mathsf{dilation} \log^2 n)$ rounds of pre-computation.

We remark that a technique that we developed for the latter result is in fact more general and perhaps of interest well beyond this particular result. It can be used to remove the assumption of having shared randomness in a broad family of randomized distributed algorithms, at the cost of an $O(\log^2 n)$ factor increase in their running time. Roughly speaking, the family that this method applies to is those algorithms in which each node outputs one (canonical) output in the majority of the executions of the algorithm (for each given input), that is, algorithms where the randomness does not affect the output (significantly) and is used only to speed up the computation. This class was termed *Bellagio algorithms* [Gol12, GG11], and it is studied as a subclass of randomized algorithms with some desirable pseudo-deterministic behavior.

Roadmap

The rest of this thesis is structured as follows. In the next chapter we present the models of distributed computation that we use throughout the thesis. Then, we turn to the technical parts of the thesis, which are organized in 4 parts. Among these, the first part deals with the issue of *locality* and the other three parts are centered around *congestion*.

- Part I consists of one chapter, Chapter 3, which explains our results on *local distributed graph algorithms* and particularly on Maximal Independent Set, together with their implications and applications.
- Part II consists of two chapters, and it presents the graph-theoretic foundations of our work on graph connectivity. In particular, Chapter 4 explains connectivity decompositions, proves their existence, and shows some of their applications, and Chapter 5 presents our results on how a graph's connectivity changes when various elements of it (vertices or edge) are sampled randomly.
- Part III consists of four chapters, which present our distributed algorithms for various graph problems related to graph connectivity. In particular, in Chapter 6 we present the concept of *low-congestion shortcuts* and we use it to derive a simple near-optimal distributed algorithm for *minimum spanning trees*. Chapter 7 presents our near-optimal distributed algorithms for approximating *edge connectivity and minimum edge cut*. Chapter 8 presents our near-optimal distributed algorithms for approximating *minimum weight connected dominating sets*. Finally, in Chapter 9, we present our distributed algorithms for constructing *connectivity decompositions*, and their applications.
- Part IV consists of one chapter, Chapter 10, which presents our results on scheduling distributed algorithms, that is, it investigates the issue of congestion when multiple distributed algorithms need to be run together.

Chapter 2

Models

In this chapter, we review the mathematical models of distributed graph algorithms that we use throughout this thesis. We consider two standard models, LOCAL and CONGEST [Pel00], where the latter has two variants: E-CONGEST and V-CONGEST. These models are all the same, with only one exception: different models have different limitations on the communication. We next describe these models in a unified manner, and then explain their communication limitations.

Communication Network and Processes All of the aforementioned models are synchronous message passing models. The communication network is abstracted as an undirected graph $G = (V, E)$ with $n = |V|$ nodes. There is one processor residing in each node of the graph and each processor can communicate directly with only the processors residing in adjacent nodes. From now on, we use the terms *processor* and *node* interchangeably.

Initial Knowledge We assume that at the beginning of the execution, the processors know only their adjacent processors, and especially that they do not know the full topology of the network $G = (V, E)$. However, we sometimes assume that nodes know certain estimates on some of the global parameters of the network. For instance, we usually assume that they all know a polynomial upper bound N on the network size n such that $N \leq n^c$ for some fixed constant $c \geq 1$.

Synchronous Executions We here provide a brief description of the model for the round-by-round executions in synchronous network algorithms. This minimal and simplified description suffices for the discussions in this thesis. We refer the reader to [Lyn96, Chapter 2.1] for a detailed and more formal description.

In all the models in consideration in this thesis, the execution proceeds in synchronous rounds and particularly, communication happens in lock-step rounds 1, 2, 3, ... Each round works as follows: In the beginning of the round, each processor generates messages to send to its neighbors, and puts these messages in the appropriate communication channels. These

messages get sent at the beginning of the round and they get delivered to the neighbors before the end of the round. At the end of the round, each processor performs some local computations and changes its states, based on the messages it received from the communication channels. We discuss these local computations shortly. The messages are then removed from the channel. Then the execution proceeds to the next round.

Communication Limitations and Message Sizes In the LOCAL model, there is no limitation on the size of the messages, and as far as the model is concerned, each node can send all the information that it holds to its neighbors. The CONGEST model on the other hand takes bandwidth-related communication limitations into account. In particular, the CONGEST model restricts the message size, and this restriction comes in two variants: In the V-CONGEST variant of the model—abbreviating congestion on vertices—in each round, each node can send one $O(\log n)$-bit message to *all* of its neighbors. That is, the same message is sent to all the neighbors. In the E-CONGEST variant of the model—abbreviating congestion on edges—in each round, each node can send one $O(\log n)$-bit message to each of its neighbors and the model allows the node to send different messages to different neighbors.

Notice that V-CONGEST is more restrictive than E-CONGEST. In the V-CONGEST model, each node sends at most one message per round to all neighbors. Clearly, this one message can be sent to all neighbors also in the E-CONGEST model, which allows the node to send different messages to different neighbors. There is no relation in the other direction, meaning that the V-CONGEST model cannot simulate the E-CONGEST model (without a significant round complexity overhead).

We remark that E-CONGEST, often just called CONGEST [Pel00], is the more classical distributed model that considers *congestion* and assumes bounded size messages. Considering congestion on vertices is however also not new. It is motivated by application domains such as wireless networks where each node can broadcast one message to its neighbors per time-unit, or peer-to-peer overlay networks and social networks where each node can upload one message per time-unit.

Remark on Local Computations The distributed models usually do not assume any limitation on the local computation power of the processors. However, in all the algorithms that we present, the local computations are quite simple, always being at most polynomial time computations in the network size n and most often being at most linear, up to logarithmic factors.

Performance Measure In essentially all of our results, the main performance measure is the number of rounds required to perform a given computation or communication task. If a given algorithm solves the task in a particular number of rounds, we refer to this number as the *round complexity* of that algorithm. Usually, this round complexity is expressed as a function of the parameters of the network graph. Most importantly, many of our bounds

depend on the number of nodes n. Some of the bounds depend on the maximum degree Δ, the diameter D, or potentially some other parameters which will be specified when needed.

Randomized Algorithms The vast majority of the algorithms presented in this thesis are randomized distributed algorithms. In these algorithms, each processor follows a probabilistic state transition process, and in particular, the state transitions at the end of each round are probabilistic. Then, the messages to be sent to the neighbors in the next round are a deterministic function of the new state. We do not specify the details of this point further, as it is similar to the probabilistic process in centralized randomized algorithms. We refer the interested reader to [Lyn96, Chapter 2.7] for a formal description.

We remark that in our models, the randomness used by the processors is the only source of randomness in the execution. Moreover, we assume that each processor has access to a sufficiently long sequence of independent uniformly-distributed random bits, which are used for determining the probabilistic state transitions at the end of the rounds. We also assume that these randomness sequences for different processors are distributed independently.

When working with randomized algorithms, we usually desire them to solve each given problem "with high probability". Concretely, this means that the algorithm should provide the following two guarantees with high probability: (1) within the claimed number of rounds, all the processors should have computed their individual outputs, (2) the computed outputs should form a valid solution for the given problem. Following the convention in the theory of computation, and especially in the area of distributed algorithms, we use the phrase '*with high probability*' (*w.h.p.*) to indicate that some event has a probability of at least $1 - 1/n^c$ for a desirably large constant $c \geq 2$. Recall that n denotes the number of the nodes in the network/graph. Intuitively, this high probability guarantee means that within the life-time of the system/algorithm, which is usually presumed to be at most polynomial in the network size n, we are unlikely to see that a high probability event does not happen.

Part I

Locality

Chapter 3

Maximal Independent Set

3.1 Introduction & Related Work

In this chapter, we present a randomized distributed algorithm for computing a Maximal Independent Set (MIS) which improves significantly over the state of the art and achieves a near-optimal round complexity. Informally, an MIS is an independent set of vertices, meaning that no two of them are adjacent, such that the set is maximal with regard to independence, meaning that adding any other vertex to it would violate independence. See Section 3.2 for the formal definition. Our MIS result is especially important considering the centrality of the MIS problem in the subarea of locality in distributed graph algorithms. See Section 1.1 for a discussion about the reasons of this centrality, and also see Section 3.5, which shows how this progress on MIS leads to improvements in about a dozen other classical problems.

State of the Art for Distributed MIS: Here, we briefly review the most relevant work on MIS to ours, which constitute the state of the art. See [BEPSv3, Section 1.1] and the monograph of Barenboim and Elkin [BE13] for more thorough reviews of the related work.

The first related work is the $O(\log^4 n)$ round randomized distributed algorithms of Karp and Wigderson [KW84], which was presented in 1984. This was shortly afterwards followed in 1986 by the influential work of Luby [Lub85], and Alon, Babai, and Itai [ABI86], which provided an $O(\log n)$ round randomized distributed MIS algorithm. After that, the only significant progress that is relevant to our work happened in 2011 when Barenboim, Elkin, Pettie, and Schneider [BEPSv3] presented a randomized distributed algorithm with round complexity of $O(\log^2 \Delta) + 2^{O(\sqrt{\log \log n})}$. Here, Δ denotes the maximum degree in the graph.

There has been also some beautiful work on the lower bound side. In 1992, Linial [Lin92] showed a lower bound of $\Omega(\log^* n)$, which holds even on a simple cycle graph. In 2004, Kuhn, Wattenhofer, and Moscibroda [KMW04,KMW16] proved a lower bound of $\Omega(\frac{\log \Delta}{\log \log \Delta})$ on some (large) graphs with maximum degree Δ. When expressed as a function of both Δ and n, the lower bound becomes $\Omega(\min\{\frac{\log \Delta}{\log \log n}, \sqrt{\frac{\log n}{\log \log n}}\})$.

Roadmap: This chapter is organized as follows. In the remainder of this section, we present the statement of our result and some high-level discussions about the algorithm, respectively, in Subsections 3.1.1 and 3.1.2. Some preliminaries are presented in Section 3.2. We present the algorithm in two parts in Section 3.3 and Section 3.4. Then, Section 3.5 presents some implications and applications of our MIS result. We conclude the chapter with some open questions in Section 3.6.

3.1.1 Our Result on Distributed MIS

We present an improved distributed randomized MIS algorithm which achieves a round complexity of $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$. More formally, we prove the following theorem:

Theorem 3.1.1. *There is a randomized distributed that computes an MIS in* $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ *rounds, with probability at least* $1 - 1/n^c$ *for any arbitrary constant* $c \geq 2$.

Remarks on Near-Optimality: There are three concrete points to state here regarding the optimality of this bound, or about the difficulty in improving on it.

- This $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ round complexity improves the best-known bound for MIS and gets close to the $\Omega(\min\{\frac{\log \Delta}{\log \log n}, \sqrt{\frac{\log n}{\log \log n}}\})$ lower bound of Kuhn et al. [KMW04, KMW16]. In particular, the upper bound is optimal, modulo doubly-logarithmic factors, when $\log \Delta \in [2^{\sqrt{\log \log n}}, \sqrt{\log n}]$. This is the first time since the beginning of research on distributed MIS in 1984 that such a provable near-optimality is obtained.

- The new result matches the lower bound in a stronger and much more instructive sense: as we elaborate in Section 3.5, it perfectly pinpoints why the current lower bound techniques cannot prove a lower bound better than $\Omega(\min\{\log \Delta, \sqrt{\log n}\})$.

- Thanks to a recent observation of Chang, Kopelowitz, and Pettie [CKP16], we also know why the $2^{O(\sqrt{\log \log n})}$ term in the bound is the best-achievable dependency on n, given the current techniques. In other words, improving this $2^{O(\sqrt{\log \log n})}$ term would lead to an improvement on the currently best known $2^{O(\sqrt{\log n})}$ round complexity of deterministic distributed MIS algorithms, which has remained the best known for about 25 years. Such an improvement would be considered a major breakthrough in the area of distributed graph algorithms, especially if the improvement is significant.

3.1.2 Technical Overview of Our Approach

Here, we present a high-level description of the algorithm. To focus on the core novelty in Theorem 3.1.1, we present a new perspective for measuring the complexity of distributed algorithms. We consider two separate measures, a *local complexity* measure, and a *global complexity* measure. We do not formalize these measures completely in their full generality;

we just explain them in the context of our MIS problem. In the following, we motivate this division and explain the contribution of our new algorithm in local and global complexity parts separately. As a side remark, we believe that this separation between local and global complexities might be helpful and/or instructive also in studying other problems in the subarea of *local distributed graph algorithms*, and it might be interesting also for applications in practical settings.

Local Complexity of Our MIS Result

Despite the local nature of the MIS problem, classically the main focus has been on the so-called *global complexity*, which measures the time until all nodes terminate. Moreover, somewhat strikingly, essentially all the standard analyses also take a non-local approach by considering the whole graph and showing guarantees on how the algorithm makes *global progress* towards its *local objectives*. A prominent example is the analysis of [Lub85, ABI86] which show that per round, in expectation, half of the edges of the whole network get removed, hence leading to the *global complexity* guarantee that after $O(\log n)$ rounds, with high probability, these algorithm terminates everywhere. See [MR10, Section 12.3], [Pel00, Section 8.4], [Lyn96, Section 4.5] for textbook treatments or [Win04] for a simpler analysis.

The fact that essentially all the tight analysis for the local problem of MIS are based on global progress arguments seemingly suggests a gap in our understanding of *locality*. Our starting point is to seek a *local analysis* instead. Particularly, what we desire is to have an analysis that looks only at a node and some small neighborhood of it, and provides a guarantee for that one node independent of n. To be concrete, our starting question is:

> **Local Complexity Question**: How long does it take till each node v terminates, and knows whether it is in the (eventual) MIS or not, with probability at least $1 - \varepsilon$?

We remark that this question can be directly relevant for practical network algorithms, when one wishes to compute an MIS. It is reasonable to imagine that in various practical settings, it is not crucial to complete the computation of MIS in all nodes. Instead, if for instance only 1% of nodes remain undecided, the found solution may be regarded as satisfactory. If as the above question suggests, each node has some small ε probability, say $\varepsilon = 0.01$, of not knowing whether it is in the MIS or not, we could infer that overall only a small fraction of nodes remain undecided. The latter would hold in expectation, simply by linearity of expectation. It can also be strengthened to a high probability guarantee if we have certain independence properties between the events of different nodes remaining undecided. See Theorem 3.1.2 for such an independence property.

Using Δ to denote the maximum degree, one can obtain answers such as $O(\log^2 \Delta + \log 1/\varepsilon)$ rounds for Luby's algorithm, or $O(\log \Delta \log\log \Delta + \log \Delta \log 1/\varepsilon)$ rounds for (a parameter optimized version) of the variant of Luby's algorithm used by Barenboim, Elkin, Pettie, and Schneider [BEPSv3] and Chung, Pettie, and Su [CPS14]. However, neither of these two bounds is optimal. For instance, consider the case where we set $\varepsilon = 1/n$ and

$\Delta = n^\delta$ for a constant $\delta > 0$. With these parameters, both of the above bounds become $\Theta(\log^2 n)$. However, standard analysis of Luby's algorithm provides this per-node termination guarantee with probability $1 - \varepsilon$ in just $O(\log n)$ rounds.

We present an extremely simple algorithm that overcomes this problem and provides a local complexity of $O(\log \Delta + \log 1/\varepsilon)$. More formally, we prove that:

Theorem 3.1.2. *There is a randomized distributed MIS algorithm such that for each node v, the probability that v has not made its decision after the first $O(\log \deg(v) + \log 1/\varepsilon)$ rounds is at most ε. Furthermore, this holds even if the bits of randomness outside the 2-hop neighborhood of v are determined adversarially.*

The perhaps surprising fact that the bound depends only on the degree of node v, even allowing its neighbors to have infinite degree (as $n \to \infty$), demonstrates the *truly local* nature of this algorithm. The logarithmic degree-dependency in the bound is optimal, modulo a doubly-logarithmic factor, following a lower bound of Kuhn, Moscibroda and Wattenhofer [KMW04, KMW16], in the following sense: As indicated by [Kuh15], with minor changes in the arguments of [KMW04,KMW16], one can prove that there are graphs in which the time until each node v knows whether it is in MIS or not with constant probability mus be at least $\Omega(\frac{\log \Delta}{\log \log \Delta})$.

We note that the fact that the analysis looks only at the 2-hop neighborhood of node v, and particularly, that the guarantee relies only on the coin tosses within the 2-hop neighborhood of node v, will prove vital as we move to global complexity.

Finally, we remark that the locality of this guarantee might be interesting for practical purposes as well. This locality shows that even if things go wrong outside the 2-hop neighborhood of v—e.g., even if far-away nodes do not follow the algorithm properly—still v will have made its decision about whether it is in the chosen set or not. Notice that in such a scenario, we cannot guarantee that the chosen set is a maximal independent set in the whole graph, because those corrupted far-away nodes may behave arbitrarily. However, we can say that we have the following two properties: (1) it cannot be the case that both v and a neighbor of it are in the chosen set, (2) either v in the chosen set or it has a neighbor in the chosen set. These are in a sense local parts of the definition of MIS—as formalized in Section 3.2—for node v. For further details, see the work of Holzer and Lynch [HL16, Definition 1], which formalizes this notion of *local correctness* of MIS algorithms.

Global Complexity of Our MIS Result

Notice that one can easily infer from Theorem 3.1.2 that after $O(\log n)$ rounds, w.h.p., all nodes have terminated. Thus, this reproves the standard global complexity of $O(\log n)$ rounds, but now with the advantage that the related analysis is local. We now explain how we improve the global complexity bound to $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds.

The overall plan is based on the following nice and natural intuition, which was used in the MIS results of Alon et al. [ARVX12] and Barenboim et al. [BEPSv3]. We note that

this general strategy is often attributed to Beck, as he used it first in his breakthrough algorithmic version of the Lovász Local Lemma [Bec91]. Applied to MIS, the intuition is as follows: when we run any of the usual randomized MIS algorithms, we gradually add nodes to the eventual MIS and as a result, we remove them and also their neighbors from the graph. Thus, over time, the remaining graph becomes sparser. After running this *base algorithm* for a certain number of rounds, a certain *graph shattering* phenomenon occurs, where what remains of the graph is a number of "small" components. Here, "small" might be in regard to size, diameter, the maximum size of some specially defined independent sets, or some other measure. Once the graph is shattered, one switches to a deterministic algorithm to solve the problem in these remaining small components.

It can be shown that the graph shattering phenomenon starts to show up around the time that the probability ε of each node remaining falls below $1/\Delta$. This can be inferred for instance from analysis of *Galton-Watson branching processes* [WG75] (ignoring some technicalities, which we deal with in the actual proofs in Section 3.4)[1]. In a very informal sense, when the probability ε of each node remaining is below $1/\Delta$, we expect each particular node to have at most $\varepsilon\Delta < 1$ neighbors remaining, and thus at this point the graph starts to "break down". Alon et al. [ARVX12] used an argument of [PR07], showing that Luby's algorithm reaches this shattering threshold after $O(\Delta \log \Delta)$ rounds. Barenboim et al. [BEPSv3] used a variant of Luby's algorithm, with a small but clever modification, and showed that it reaches the threshold after $O(\log^2 \Delta)$ rounds. As Barenboim et al. [BEPSv3] show, after the shattering, the remaining pieces can be solved deterministically, via the help of known deterministic MIS algorithms (and some other ideas), in $\log \Delta \cdot 2^{O(\sqrt{\log \log n})}$ rounds. Thus, the overall complexity of [BEPSv3] is $O(\log^2 \Delta) + \log \Delta \cdot 2^{O(\sqrt{\log \log n})} = O(\log^2 \Delta) + 2^{O(\sqrt{\log \log n})}$.

To improve this, we use our new MIS algorithm as the base, instead of Luby's algorithm. As Theorem 3.1.2 suggests, this algorithm reaches the shattering threshold after $O(\log \Delta)$ rounds. This is formalized in Section 3.4. We also use some minor modifications for the *post-shattering* phase to reduce its complexity from $\log \Delta \cdot 2^{O(\sqrt{\log \log n})}$ to $2^{O(\sqrt{\log \log n})}$. The overall round complexity of the algorithm thus becomes $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$, providing the round complexity claimed in Theorem 3.1.1.

3.2 Preliminaries

Graph Notations As stated before, we work with an undirected graph $G = (V, E)$ where $n = |V|$ and Δ denotes an upper bound on the maximum degree of the graph. For a node v, we use the notation $N(v)$ to denote the set of its neighbors in the graph, and we also define $N^+(v) = N(v) \cup \{v\}$. Given a subset of vertices S, we use $G[S]$ to denote the subgraph induced by S, that is, $G[S] = (S, E_S)$ where E_s is the set of all edges in E for which both

[1] This ignores a few technicalities. In truth, the probability threshold is $1/\operatorname{poly}(\Delta)$, instead of $1/\Delta$, because of some unavoidable dependencies. However, due to the exponential concentration, the time to reach the $1/\operatorname{poly}(\Delta)$ threshold is within a constant factor of that of the $1/\Delta$ threshold. We will also need to establish some independence. See Section 3.4 for details.

of the endpoints are in E. Given an integer $x \geq 1$, we use the notation $G^{\leq x}$ to denote the graph with the same vertex set as G but where we put edges between each two G-nodes that have distance at most x. A set S of vertices is called *x-independent* if it is an independent set in $G^{\leq x}$, that is if each two of its vertices have distance at least $x+1$.

The Maximal Independent Set (MIS) Problem The MIS problem requires selecting a subset S of the vertices of the graph $G = (V, E)$ that satisfies the following two properties: (1) S is an *independent* set, meaning that no two nodes of S are adjacent in G, (2) S is *maximal* in regard to independence, meaning that for each node $v \in V \setminus S$, there is at least one node $u \in S$ that is adjacent to v in G. In the distributed representation of an MIS S, each processor should output whether it is in S or not.

Ruling Set This is a generalization of the notion of MIS. A subset S of vertices is called an *(α, β)-ruling set* of $G = (V, E)$, for positive integers α and β, if and only if we have the following two properties: (1) each two nodes in S are at distance at least α, (2), each node $v \in V \setminus S$ has a node in S within its distance at most β. In particular, a $(2, 1)$-ruling-set is simply an MIS.

Network Decomposition A *(α, β)-network decomposition* for a graph $G = (V, E)$, for positive integers α and β, is a partitioning of the vertices V into subsets $S_1, S_2, \ldots$, such that the following two properties are satisfied: (1) each two nodes of the same subset have distance at most α in graph G, (2) there exists a *coloring* which assigns a color from $[1, \ldots, \beta]$ to each subset such that each two subsets which have adjacent vertices are colored differently. We refer to each of these subsets as a *cluster*. Typically, we assume that the network decomposition also includes the coloring of the clusters mentioned in property (2). In the standard distributed representation of network decompositions (see e.g., [PS92]), each cluster has a unique leader, each node knows the identifier of its cluster as well as the color of that cluster.

3.3 Our MIS Algorithm, & Its Local Complexity

Here we present a very simple and clean algorithm that guarantees for each node v that after $O(\log \Delta + \log 1/\varepsilon)$ rounds, with probability at least $1 - \varepsilon$, node v has terminated and it knows whether it is in the (eventual) MIS or it has a neighbor in the (eventual) MIS.

The Intuition The intuitive base of the algorithm is as follows: There are two scenarios in which a node v has a good chance of being removed: either (1) v is trying to join the MIS and it does not have too many competing neighbors, in which case v has a chance to join the MIS, or (2) a large number of neighbors of v are trying to join the MIS each of which does not have too much competition, in which case it is likely that one of them joins the MIS and thus v gets removed. These two cases also depend only on v's 2-neighborhood. Our key idea

is to create a *dynamic* which makes each node v spend a significant amount of time in these two scenarios, unless it has been removed already.

The algorithm for node v The algorithm is divided into *iterations*, each of which consists of three consecutive rounds. In the beginning of each iteration $t \geq 1$, node v has a *probability* $p_t(v)$. Initially, we set $p_0(v) = 1/2$. For any $t \geq 0$, we call the total sum of the probabilities of neighbors of v its *effective-degree* $d_t(v)$, that is, $d_t(v) = \sum_{u \in N(v)} p_t(u)$.

Iteration t of the algorithm, for $t \geq 1$: Each iteration works as follows:

(**R1**) At the beginning of the round, node v sends its probability $p_{t-1}(v)$ of the previous iteration to its neighbors. By the end of the round, node v receives the probability $p_{t-1}(u)$ of each neighbor $u \in N(v)$. It then sets $d_{t-1}(v) = \sum_{u \in N(v)} p_{t-1}(u)$ and updates its probability using the following deterministic rule.

$$p_t(v) = \begin{cases} p_{t-1}(v)/2, & \text{if } d_{t-1}(v) \geq 2 \\ \min\{2p_{t-1}(v), 1/2\}, & \text{if } d_{t-1}(v) < 2. \end{cases}$$

At the end, node v *marks* itself with probability $p_t(v)$, and remains unmarked with the rest of the probability.

(**R2**) At the beginning of the round, node v sends a message to its neighbors indicating whether it is marked or not. By the end of the round, it receives those messages of its neighbors, hence learning whether any of the neighbors is marked. If node v is marked and none of its neighbors is marked, then node v joins the MIS.

(**R3**) At the beginning of the round, node v sends a message to its neighbors indicating whether v joined the MIS in the previous round or not. By the end of the round, it receives those messages of its neighbors, hence learning whether any of the neighbors joined the MIS. Then, if node v joined the MIS, or if it received a message from a neighbor u indicating that u joined the MIS, then v terminates. In that case, effectively node v gets removed from the problem.

Remark about the Message Sizes We also note that 1-bit messages would suffice for implementing the communication rounds of this algorithm. In particular, reporting being marked or joining MIS can be done with 1-bit messages. To send the probability $p_t(v)$, it suffices for the node v to inform its neighbors whether $p_t(v)$ decreased or not, which can be done with a 1-bit message.

Remark about the Contrast with Luby's Algorithm The idea of marking processes and taking only isolated marks to MIS is standard and it appears also in (one version of) Luby's algorithm [Lub86]. However, at each round, Luby's algorithm marks each node v

with a fixed probability $\frac{1}{deg(v)+1}$, where $deg(v)$ denotes the number of the neighbors of v remaining at that time. In this regard, the key element in our new algorithm is the process that changes the marking probability dynamically and flexibly over time, trying to push towards the two desirable scenarios mentioned in the intuition paragraph above.

The Analysis The correctness is clear as the set of nodes that join the MIS is an independent set and the algorithm terminates at a node only if the node is either in MIS or adjacent to a node in MIS. We next argue that each node v is likely to terminate quickly.

Throughout the analysis, we use the notation deg to denote the degree of node v degree at the start of the algorithm.

Theorem 3.3.1. *For each node v, the probability that v has not made its decision within the first $\beta(\log \mathsf{deg} + \log 1/\varepsilon)$ iterations, for a large enough constant β is at most ε. This holds even if the outcome of the coin tosses outside $N_2^+(v)$ are determined adversarially.*

Let us say that a node u is *low-degree* at time t if $d_t(u) < 2$, and *high-degree* otherwise. Considering the intuition discussed above, we define two types of *golden iterations* for a node v: (1) iterations in which $d_t(v) < 2$ and $p_t(v) = 1/2$, (2) iterations in which $d_v(t) \geq 1$ and at least $d_t(v)/10$ of it is contributed by low-degree neighbors. These are called golden iterations because, as we will see, in the first type, v has a constant chance of joining the MIS and in the second type there is a constant chance that one of those low-degree neighbors of v joins the MIS and thus v gets removed. For the sake of analysis, we keep track of the number of golden iterations of each type for node v.

To prove Theorem 3.3.1, we first show in Lemma 3.3.2 that node v will have many golden iterations. We then prove in Lemma 3.3.3 that in each of these golden iterations, node v has a constant probability to be removed, either because of joining the MIS or having a neighbor join the MIS. Then, we put these two Lemmas together to prove Theorem 3.3.1.

In the first lemma, we show that node v must have many golden iterations.

Lemma 3.3.2. *By iteration $\beta(\log \mathsf{deg} + \log 1/\varepsilon)$, either v has joined or has a neighbor in the MIS, or at least one of its golden iteration counts reaches $\frac{\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$.*

Proof. We focus only on the first $\beta(\log \mathsf{deg}+\log 1/\varepsilon)$ iterations. Let g_1 and g_2 be, respectively, the number of golden iterations for v of types 1 and 2, during this period. Moreover, let h be the number of iterations during which $d_t(v) \geq 2$. To prove the lemma, we assume that by the end of iteration $\beta(\log \mathsf{deg}+\log 1/\varepsilon)$, node v is not removed and $g_1 \leq \frac{\beta}{13}(\log \mathsf{deg}+\log 1/\varepsilon)$, and we conclude that, then it must have been the case that $g_2 > \frac{\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$. For that, we first argue that assuming $g_1 \leq \frac{\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$ is small implies that $h \geq \frac{4\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$. We then argue that if $h \geq \frac{4\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$, then $g_2 > \frac{\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$.

Small g_1 $\rightarrow$ Large h: Notice that the changes in $p_t(v)$ are governed by the condition $d_t(v) \geq 2$ and the iterations with $d_t(v) \geq 2$ are exactly the ones in which $p_t(v)$ decreases

by a 2 factor. Hence, there are exactly h iterations in which $p_t(v)$ decreases by a 2 factor. Furthermore, the number of 2 factor increases in $p_t(v)$ can be at most equal to the number of 2 factor decreases in it, as $p_t(v)$ is capped to $1/2$. Hence, the total number of iterations in which $p_t(v)$ increases or decreases (by a 2 factor) is at most $2h$. Therefore, there are at least $\beta(\log \mathsf{deg} + \log 1/\varepsilon) - 2h$ iterations in which $p_t(v) = 1/2$. Now out of these iterations, at most h of them can be when $d_t(v) \geq 2$. Hence, $\beta(\log \mathsf{deg} + \log 1/\varepsilon) - 3h \leq g_1$. As we have assumed $g_1 \leq \frac{\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$, we get that $\beta(\log \mathsf{deg} + \log 1/\varepsilon) - 3h \leq \frac{\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$. This implies that $h \geq \frac{4\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$.

Large $h \to$ Large g_2: Recall that h counts the number of iterations in which v in high-degree, that is, iterations in which $d_t(v) \geq 2$. In an iteration in which $d_t(v) \geq 2$, the first requirement for a golden type-2 iteration is satisfied (in fact with some slack as the requirement only asks for $d_t(v) \geq 1$). However, the second requirement might be not satisfied. We show that the number of iterations in which the first requirement is satisfied but the second requirement is not satisfied cannot be too large. We then conclude that a large h implies a large g_2.

Concretely, let us consider the changes in the effective-degree $d_t(v)$ of v over time. If $d_t(v) \geq 1$ and this is not a golden iteration of type-2, then we have

$$d_{t+1}(v) \leq 2\frac{1}{10}d_v(t) + \frac{1}{2}\frac{9}{10}d_t(v) < \frac{2}{3}d_t(v).$$

There are g_2 golden iterations of type-2. Except for these type-2 golden iterations, whenever $d_t(v) \geq 1$, the effective-degree $d_t(v)$ shrinks by at least a $2/3$ factor. In those exceptions, it increases by at most a 2 factor. Each of these exception iterations cancels the effect of at most 2 shrinkage iterations, as $(2/3)^2 \times 2 < 1$. Thus, ignoring the total of at most $3g_2$ iterations lost due to type-2 golden iterations and their cancellation effects, every other iteration with $d_t(v) \geq 2$ pushes the effective-degree down by a $2/3$ factor[2]. This cannot (continue to) happen more than $\log_{3/2} \mathsf{deg}$ times as that would lead the effective degree to exit the $d_t(v) \geq 2$ region. Hence, the number of iterations in which $d_t(v) \geq 2$ is at most $\log_{3/2} \mathsf{deg} + 3g_2$. That is, $h \leq \log_{3/2} \mathsf{deg} + 3g_2$. Since $h \geq \frac{4\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$, we get $g_2 > \frac{\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$. □

In the next lemma, we show that in each golden iteration, node v has at least a constant probability probability to be removed.

[2]Notice the switch to $d_t(v) \geq 2$, instead of $d_t(v) \geq 1$. This slack also appears in the definition of type-2 iterations (which requires $d_t(v) \geq 1$) and those in which v is high-degree (where the requirement is $d_t(v) \geq 2$). We need to allow a small slack here, as done by switching to threshold $d_t(v) \geq 2$, in order to avoid the possible zigzag behaviors on the boundary. This is because, the above argument does not bound the number of 2-factor increases in $d_t(v)$ that start when $d_t(v) \in (1/2, 1)$ but these would lead $d_t(v)$ to go above 1. This can continue to happen even for an unlimited time if $d_t(v)$ keeps zigzagging around 1. However, for $d_t(v)$ to go/stay above 2, it takes increases that start when $d_t(v) > 1$, and the number of these is at most g_2.

Lemma 3.3.3. *In each type-1 golden iteration, with probability at least* $1/200$*,* v *joins the MIS. Moreover, in each type-2 golden iteration, with probability at least* $1/200$*, a neighbor of* v *joins the MIS. These statements hold even if the coin tosses outside* $N_2^+(v)$ *are determined adversarially.*

Proof. In each type-1 golden iteration, node v gets marked with probability $1/2$. The probability that no neighbor of v is marked is $\prod_{u\in N(v)}(1-p_t(u)) \geq 4^{-\sum_{u\in N(v)} p_t(v)} = 4^{-d_t(v)} > 4^{-2} = 1/16$. Hence, v joins the MIS with probability at least $1/32 > 1/200$.

Now consider a type-2 golden iteration. For the sake of analysis, we examine the set L of low-degree neighbors of v one by one, and we expose their randomness—meaning that we check whether they are marked not not based on their randomness—until we reach a node that is marked. We will find a marked node with probability at least

$$\begin{aligned}1-\prod_{u\in L}(1-p_u(t)) &\geq 1-e^{-\sum_{u\in L} p_u(t)}\\ &\geq 1-e^{-d_t(v)/10} \geq 1-e^{-1/10} > 0.08.\end{aligned}$$

When we reach the first low-degree neighbor u that is marked, the probability that no neighbor of u is marked is at least $\prod_{w\in N(u)}(1-p_t(w)) \geq 4^{-\sum_{w\in N(u)} p_t(w)} \geq 4^{-d_t(u)} > 1/16$. Hence, with probability at least $0.08/16 = 1/200$, one of v's neighbors joins the MIS.

We now know that in each golden iteration, v gets removed with probability at least $1/200$, due to joining MIS or having a neighbor join the MIS. Thus, using Lemma 3.3.2, we get that the probability that v does not get removed is at most $(1-1/200)^{\frac{\beta}{13}(\log \mathsf{deg}+\log 1/\varepsilon)} \leq \varepsilon$. □

We now put Lemma 3.3.2 and Lemma 3.3.3 together and prove Theorem 3.3.1.

Proof of Theorem 3.3.1. Lemma 3.3.2 shows that node v has at least $\frac{\beta}{13}(\log \mathsf{deg} + \log 1/\varepsilon)$ golden iterations during the first $\beta(\log \mathsf{deg}+\log 1/\varepsilon)$ iterations. Lemma 3.3.3 shows that node v is removed with probability at least $1/200$ in each of these golden iterations. Hence, the probability that node v has not been removed during the first $\beta(\log \mathsf{deg} + \log 1/\varepsilon)$ iterations is at most $(1-1/200)^{\frac{\beta}{13}(\log \mathsf{deg}+\log 1/\varepsilon)} \leq (1-1/200)^{\frac{\beta}{13}(\log 1/\varepsilon)} \leq \varepsilon$. Since each iteration is simply three communication rounds, this proves the theorem. □

3.4 Our MIS Algorithm, and Improved Global Complexity

Here, we explain how combining the algorithm of the previous section with some known techniques leads to a randomized MIS algorithm with a high probability global complexity of $O(\log \Delta) + 2^{O(\sqrt{\log\log n})}$ rounds.

Recall from Section 3.1.2 that the high-level outline of the algorithm is as follows: we first run the algorithm of the previous section for long enough until the graphs *shatters* and the remaining components are "*small*". We then solve these remaining small components

simultaneously using (a modification of) the deterministic distributed MIS algorithm of [PS92]. The ideal definition of a *small* component would have been that the component has at most $\text{poly}(\log n)$ nodes. However, we cannot achieve that property. The particular definition of "small" that works for us is somewhat technical, and it is made precise in Lemma 3.4.2. Informally, in a *small* component, the size of the maximum size subset such that each two of its nodes have distance at least 5 is at most $O(\log n)$.

Pre-Shattering Phase of the Algorithm and Its Analysis

The pre-shattering phase of the global algorithm simply runs the algorithm of the previous section for $\Theta(\log \Delta)$ rounds. Thanks to the local complexity of this base algorithm, the graph gets *shattered* in $O(\log \Delta)$ rounds, with high probability. We formalize this soon, in Lemma 3.4.2. The fact that Theorem 3.3.1 relies only on the bits of randomness within 2-hop neighborhood of the node under consideration plays a vital role in establishing this shattering phenomenon. Before stating the achieved shattering property in Lemma 3.4.2, we first need to establish a helping lemma:

Lemma 3.4.1. *Let $c > 0$ be an arbitrary constant. For any 5-independent set of nodes S, the probability that all nodes of S remain undecided after $\Theta(c \log \Delta)$ rounds of the algorithm of Section 3.3 is at most $\Delta^{-c|S|}$.*

Proof. We examine the nodes of S one by one: when considering node $v \in S$, we know from Theorem 3.3.1 that the probability that v stays undecided after $\Theta(c \log \Delta)$ rounds is at most Δ^{-c}, and more importantly, this only relies on the coin tosses within distance 2 of v. Because of the 5-independence of set S, the coin tosses we rely on for different nodes of S are non-overlapping and hence, the probability that the whole set S stays undecided is at most $\Delta^{-c|S|}$. □

From Lemma 3.4.1, we can get a *shattering* guarantee, as stated formally in Lemma 3.4.2. Since the proof is similar to that of [BEPSv3, Lemma 3.3], or those of [Bec91, Main Lemma], [ARVX12, Lemma 4.6], and [LRY15, Theorem 3], we provide only a brief sketch.

Lemma 3.4.2. *Let c be a large enough constant and B be the set of nodes remaining undecided after $\Theta(c \log \Delta)$ rounds of the MIS algorithm of the previous section on a graph G. Then, with probability at least $1 - 1/n^c$, we have the following two properties:*

(P1) There is no $(G^{\leq 4})$-independent $(G^{\leq 9})$-connected subset $S \subseteq B$ such that $|S| \geq \log_\Delta n$.

(P2) Each connected component of $G[B]$ has each at most $O(\Delta^4 \cdot \log_\Delta n)$ nodes.

Proof Sketch. Let $H = G^{\leq 9} \setminus G^{\leq 4}$, i.e., H is the result of removing edges of $G^{\leq 4}$ from $G^{\leq 9}$. That is, H has the same vertex set as G and two vertices are connected if and only if their distance is in $[5, 9]$. For (P1), note that the existence of any such set S would mean $H[B]$ contains a $(\log_\Delta n)$-node tree subgraph. There are at most $4^{\log_\Delta n}$ different

$(\log_\Delta n)$-node tree topologies and for each of them, less than $n\Delta^{9\log_\Delta n}$ ways to embed it in H. For each of these trees, by Lemma 3.4.1, the probability that all of its nodes stay is at most $\Delta^{-2c(\log_\Delta n)}$. By a union bound over all trees, we conclude that with probability $1-n(4\Delta^9)^{\log_\Delta n}\Delta^{-2c(\log_\Delta n)} \geq 1-1/n^c$, no such such set S exists. For (P2), note that if $G[B]$ has a component with more than $\Theta(\Delta^4 \cdot \log_\Delta n)$ nodes, then we can find a set S violating (P1): greedily add nodes to the candidate S one-by-one, and each time discard all nodes within 4-hops of the newly added node, which are at most $O(\Delta^4)$ many. □

Post-Shattering Phase of the Algorithm and Its Analysis

Intuitive Discussions Ideally, we would have liked that the pre-shattering phase of the algorithm leaves components of size at most $O(\log n)$. If that was the case, we could solve each of these remaining components in $2^{O(\sqrt{\log\log n})}$ rounds, all in parallel, using the deterministic MIS algorithm of Panconesi and Srinivasan [PS92], which works in $2^{O(\sqrt{\log n'})}$ rounds in graphs of size n'. However, we know only that the size of each remaining component is no larger than $O(\Delta^4 \log_\Delta n)$, as proved by property (P2) of Lemma 3.4.2. Running the deterministic MIS algorithm of Panconesi and Srinivasan [PS92] on these components would take $2^{O(\sqrt{\log\Delta+\log\log n})}$ rounds. However, the appearance of the $\log\Delta$ in the exponent is undesirable. In particular, it would not allow us to prove the global complexity of $O(\log\Delta)+2^{O(\sqrt{\log\log n})}$ claimed in Theorem 3.1.1.

To remedy this, we use an idea similar to [BEPSv3, Section 3.2 and the Algorithm in their Figure 3], which leverages property (P1) of Lemma 3.4.2. Informally, the (P1) property of Lemma 3.4.2 opens the road for the following idea: we can turn the problem into one where the left-over components have size at most $\log_\Delta n$ by contracting nodes around an appropriately chosen 5-hop independent ruling set. By (P1) of Lemma 3.4.2, this ruling set cannot have cardinality larger than $O(\log n)$, and thus after the contraction, we are left with components of size $O(\log n)$. Once the component sizes are in $O(\log n)$, the complexity of deterministically computing an MIS in each component becomes $2^{O(\sqrt{\log\log n})}$ rounds, which would thus avoid the undesirable $\log\Delta$ term in the exponent. We show that these contracted components can be incorporated into the deterministic MIS algorithm, and the overall algorithm computes an MIS for the whole component before contraction. We next explain the *post-shattering* phase of the algorithm that follows this intuition.

The Post-Shattering Phase of the Algorithm We first present the algorithm. We then provide its analysis in the proof of Theorem 3.1.1, which shows that this global algorithm computes an MIS in $O(\log\Delta)+2^{O(\sqrt{\log\log n})}$ rounds, with high probability.

For the sake of simplicity and readability, and following the common practice in this area (e.g., [Pel00,PS92,BEPSv3,SEW13]), we do not describe the distributed algorithm from the viewpoint of each individual node. Instead, we explain the steps of the algorithm using a global perspective. It is easy to see that one can transform this into a description for the steps that each node needs to take.

The algorithm consists of six steps. We note that each step requires a fixed number of rounds, and thus the nodes know when to switch to the next step in a synchronous manner. For the sake of the algorithm description and its analysis, we consider each connected component C of the remaining nodes separately; the algorithm runs in parallel on all the components.

The Post-Shattering Algorithm: In the following, we explain the algorithm for a component C of the remaining nodes after running the algorithm of the previous section for $\Theta(\log \Delta)$ rounds:

1. Run the algorithm of the previous section for $\Theta(\log \Delta)$ further rounds. Let B_C be the left-over nodes of component C afterwards.

2. Run the distributed algorithm of Schneider, Elkin and Wattenhofer [SEW13] to compute a $(5, h)$-ruling set R_C of the set B_c, for $h = \Theta(\log \log n)$, and with regards to the distances in component C. This algorithm takes $\Theta(h)$ rounds. Recall that a $(5, h)$-ruling set R_C means each two nodes of R_C have distance at least 5 while for each node in B_C, there is at least one node in R_C within its distance of h hops.

3. Form *clusters* around R_C-nodes as follows: each node $v \in B_C$ finds its nearest R_C-node (breaking ties arbitrarily) and remembers that R_C node as its leader. This can be done easily in h rounds as each node has its leader within its distance h. These leaders define our clusters where nodes which have the same leader form one cluster. These clusters are known in a distributed way, where each node knows its cluster leader.

4. Define a new virtual graph G'_C as follows: include one node for each cluster and connect two new nodes if their respective clusters contain nodes that are adjacent in C. In other words, G'_C is the graph we get from contracting the clusters to their central nodes in R_C. In the distributed algorithm, nodes of R_C construct this virtual graph and simulate communications on it. Each node $v \in R_C$ acts as one node of G'_C. Each node in B_C sends the identifier of its R_C-leader to all its neighbors. Then, each R_C-node gathers from all B_C-nodes in its cluster the identifiers of all adjacent cluster leaders. This step is performed in $\Theta(h)$ rounds, as each cluster has radius at most h. Then, each R_C node knows its neighbors in the virtual graph G'_C. Since each two neighbors in G'_C have distance at most $h = O(\log \log n)$, a communication round on G'_C can be simulated in $O(h)$ communication rounds on C.

5. Use the deterministic distributed network decomposition algorithm of Panconesi and Srinivasan [PS92] to compute an $(2^{O(\sqrt{\log \log_\Delta n})}, 2^{O(\sqrt{\log \log_\Delta n})})$-network decomposition of G'_C. This step takes $2^{O(\sqrt{\log \log_\Delta n})}$ rounds. See Section 3.2 for the definition of network decompositions. In particular, the algorithm provides a partitioning/clustering of G'_C where each G'_C-cluster has radius at most $2^{O(\sqrt{\log \log_\Delta n})}$, and such that the

clusters are colored with $2^{O(\sqrt{\log\log_\Delta n})}$ colors in a way that adjacent clusters do not have the same color. Use these colors to color the nodes of B_C by each node $v \in B_C$ taking the color of its G'_C cluster.

6. The algorithm processes nodes of different colors one by one, in a sequential order. For each color $j \in [1, 2, \ldots, 2^{O(\sqrt{\log\log_\Delta n})}]$, we solve the problem for nodes of color-j clusters as follows: we first choose an arbitrary cluster-center for each cluster of color j. Then, we make the cluster-center node gather the whole topology of its cluster and also the adjacent MIS nodes of the previous colors. Then, this cluster-center computes the MIS of its cluster locally. In particular, it first removes the nodes of the cluster that have MIS neighbors in the previous colors. Then, it computes an MIS among the remaining nodes in a centralized manner. At the end, this cluster-center returns the solution to the nodes in the cluster, by reporting to each node of the cluster whether it is in the computed MIS or not.

Analysis We now present the analysis that proves Theorem 3.1.1, that is, it shows that the above algorithm computes an MIS in $O(\log\Delta) + 2^{O(\sqrt{\log\log n})}$ rounds, with high probability.

Proof of Theorem 3.1.1. The correctness of the algorithm is immediate, by correctness of the network decomposition that is computed in step 5, and because of the MIS computation procedure in step 6. Recall that in step 6, we process the clusters of different colors sequentially. In each color, nodes which have MIS neighbors in previous colors are removed first, and then we compute an MIS of the remaining nodes is the cluster by gathering the whole topology in a cluster-center.

We now discuss the round complexity of the algorithm. Steps 1 takes $O(\log\Delta)$ rounds. Step 2—i.e., computing the $(5, h)$-ruling set R_C—can be computed in $O(\log\log n)$ rounds using the algorithm[3] of Schneider, Elkin and Wattenhofer [SEW13]. See also [BEPSv3, Table 4]. Step 3 takes at most $h = O(\log\log n)$ rounds as each node in B_C can find its closest R_C node, which is guaranteed to be in h hops by the choice of R_C, in h rounds. Step 4 is simply a local computation step and it does not incur any round complexity.

We next argue that step 5 can be performed in $2^{O(\sqrt{\log\log_\Delta n})}$ rounds. Panconesi and Srinivasan [PS92] present a deterministic algorithm that computes a $(2^{\sqrt{\log n'}}, 2^{\sqrt{\log n'}})$-network decomposition in any network with n' nodes. Thus, we need to argue that with high probability G'_C has at most $O(\log_\Delta n)$ nodes. We use property (P1) of Lemma 3.4.2 for that. We here provide a short sketch of the argument; see [BEPSv3, Page 19, Steps 3 and 4] for a more complete description. Even though R_C might be disconnected in $G^{\leq 9}$, by greedily adding more nodes of C to it, one by one, we can make it connected in $G^{\leq 9}$ while keeping it

[3] This is different than what Barenboim et al. did. They could afford to use the more standard ruling set algorithm, particularly computing a $(5, 32\log\Delta + O(1))$-ruling set for their purposes, because the fact that this $32\log\Delta$ ends up multiplying the complexity of their post-shattering phase did not change (the asymptotics of) their overall complexity.

5-independent. We note that this is done only for the analysis. Since by (P1) of Lemma 3.4.2, the end result should have size at most $\log_\Delta n$, with high probability, we conclude G'_C has at most $\log_\Delta n$ nodes, with high probability.

Finally, in step 6, we process $2^{O(\sqrt{\log\log_\Delta n})}$ colors sequentially. When processing each cluster in each color, the related cluster-center gathers the topology of the cluster, solves the MIS problem locally, and reports it back. Since each cluster has radius at most $\log\log n \cdot 2^{O(\sqrt{\log\log_\Delta n})}$, this takes $\log\log n \cdot 2^{O(\sqrt{\log\log_\Delta n})}$ rounds per color. Thus, over all the colors, the complexity becomes $2^{O(\sqrt{\log\log_\Delta n})} \cdot \log\log n \cdot 2^{O(\sqrt{\log\log_\Delta n})} = 2^{O(\sqrt{\log\log n})}$ rounds.

Thus, considering all the steps, the overall round complexity is $O(\log\Delta)+2^{O(\sqrt{\log\log n})}$. □

3.5 Implications and Applications

The new MIS algorithm turns out to lead to several other interesting implications and applications. Most of these are improvements in various local problems, and some are obtained by combining the new algorithm with some known results and/or techniques. We here provide a summary of these improvements.

In a couple of the cases, we provide only a sketch of the role of the new MIS algorithm in the improvement, rather than explaining the whole result in a self-contained manner. This is simply because doing the latter formally would require restating and repeating prohibitively large portions of known results/techniques from other work.

(C1) We get a faster distributed MIS algorithm for low-arboricity graphs. This follows from a combination of our new MIS algorithm with the low-arboricity to low-degree reduction of Barenboim et al. [BEPSv3]. Recall that the arboricity λ of an undirected graph $G = (V, E)$ is the minimum number of forests into which the edges of the graph can be partitioned. Equivalently, λ is the minimum number of edge-disjoint spanning forests needed to cover all the edges of the graph G. The formal result is as follows:

Corollary 3.5.1. *There is a randomized distributed algorithm that in any graph with arboricity λ, computes an MIS in $O(\log\lambda + \sqrt{\log n})$ rounds, with high probability.*

Proof. Barenboim et al. [BEPSv3] present a general randomized reduction showing that MIS in λ-arboricity graphs is reducible in $O(\log^{1-\gamma} n)$ rounds to MIS in graphs with maximum degree $\lambda \cdot 2^{\log^\gamma n}$, for any $\gamma \in (0, 1)$. Setting $\gamma = 1$, this is an $O(\sqrt{\log n}$-round reduction to MIS in graphs with maximum degree, $\lambda \cdot 2^{\sqrt{\log n}}$. Using our MIS algorithm for the latter, we get the overal round complexity of $O(\log\lambda + \sqrt{\log n})$. Hence, This improves on results of [BEPSv3, BE10, LW11]. □

(C2) The new results highlight the limitations of the current lower bound techniques. In essence, they show that the current techniques cannot prove a lower bound significantly better than the $\Omega(\min\{\frac{\log\Delta}{\log\log n}, \sqrt{\frac{\log n}{\log\log n}}\})$-round lower bound of Kuhn, Wattenhofer,

and Moscibroda [KMW04, KMW16]. In particular, we next explain why the current methods cannot prove a bound better than $\Omega(\min\{\log\Delta, \sqrt{\log n}\})$.

In all the known locality-based lower bound arguments, to establish a T-round lower bound, it is necessary that within T rounds, each node sees only a tree. That is, each T-hops neighborhood must induce a tree. Informally, it is said that these graphs are *locally "tree-like"*. In particular, the graph must be high-girth and concretely, it must have girth at least $2T+1$. Recall that the girth of an undirected graph is the length of the its shortest cycle. The following corollary, which directly follows from our MIS algorithm, shows that MIS can be solved faster in high-girth graphs, in a time essentially matching the lower bound.

Corollary 3.5.2. *There is a distributed randomized algorithm that in any graph with girth $g = \Omega(\min\{\log\Delta, \sqrt{\log n}\})$ computes an MIS in $O(\min\{\log\Delta + 2^{O(\sqrt{\log\log n})}, \sqrt{\log n}\})$ rounds with high probability.*

Proof. If $\log\Delta \leq \sqrt{\log n}$, then Theorem 3.1.1 implies a global round complexity of $O(\log\Delta) + 2^{O(\sqrt{\log\log n})} = O(\min\{\log\Delta + 2^{O(\sqrt{\log\log n})}, \sqrt{\log n}\})$. Suppose that $\log\Delta > \sqrt{\log n}$. Then, to prove the claim, we need to establish a running time of $O(\sqrt{\log n})$ for graphs with girth $g = \Omega(\sqrt{\log n})$. It is well-known that any g-girth graph has arboricity $\lambda = O(n^{\frac{2}{g-2}})$. Thus, from Corollary 3.5.1, we get an $O(\sqrt{\log n})$-round MIS algorithm for graphs with girth $g = \Omega(\sqrt{\log n})$, which completes the argument. □

From the above corollary, we can infer (informally) that one cannot obtain a lower bound better than $\Omega(\min\{\log\Delta, \sqrt{\log n}\})$ when the topology seen by each node within the allowed time must be a tree. Thus, in an informal sense, to prove a better lower bound, one has to part with these topologies which are locally *"tree-like"*. However, that gives rise to intricate challenges and actually, to the best of our knowledge, there is no distributed locality-based lower bound, in fact for any (local) problem, that does not rely on locally *tree-like* topologies.

(C3) We get a faster MIS algorithm for Erdös-Rényi random graphs $G(n,p)$. The formal statement is as follows:

Corollary 3.5.3. *There is a distributed randomized algorithm that in a randomly sampled Erdös-Rényi random graph $G(n,p)$ where the edge between each two nodes is included with probability p, computes an MIS in $O(\sqrt{\log n})$ rounds with high probability.*

Proof. If $p = \Omega(\frac{2^{\sqrt{\log n}}}{n})$, then it is easy to see that w.h.p. the graph has diameter $O(\sqrt{\log n})$ hops. See e.g. [CL01] for a proof. In this case, in $O(\sqrt{\log n})$ rounds, we could make nodes simply learn the topology of the whole and then compute a solution for it locally. The more interesting case is when $p = O(\frac{2^{\sqrt{\log n}}}{n})$. Here, it is easy to see that with high probability $\Delta = O(2^{\sqrt{\log n}})$. Hence, the algorithm of Theorem 3.1.1 runs in at most $O(\sqrt{\log n})$ rounds. □

(C4) Combined with a recursive sparsification method of Bisht et al. [BKP14], we get a faster distributed $(2,\beta)$-ruling-set algorithm, which improves on the complexities of [BEPSv3] and [BKP14]. Recall from Section 3.2 that this is an independent set S such that each node $v \in V \setminus S$ has a node $u \in S$ within its distance β. In particular, a $(2,1)$-ruling-set is simply a Maximal Independent Set.

Corollary 3.5.4. *There is a randomized distributed algorithm that computes a $(2,\beta)$-ruling-set in $O(\beta \log^{1/\beta} \Delta) + 2^{O(\sqrt{\log\log n})}$ rounds, with high probability.*

Proof Sketch. We make use of a simple probabilistic procedure of Kothapalli an Pemmaraju [KP12], called *sparsify*. This procedure receives a parameter f, and when run on a given graph $H = (V_H, E_H)$ with maximum degree Δ_H, in $O(\log_f \Delta_H)$ rounds, it computes a subset $V'_H \subseteq V_H$ with the following two properties with high probability: (1) each node $v \in V_H$ is either in V'_H or has a neighbor in V'_H, and (2) each node $v' \in V'_H$ has at most $O(f \log n)$ neighbors in V'_H.

The method for computing a $(2,\beta)$-ruling set works by $\beta - 1$ recursive calls to the *sparsify* subroutine and then one MIS call. The $(i+1)^{th}$ call to sparsify works on the subgraph induced by the output of the i^{th} call and uses a parameter f_i, to be fixed later. The call to the MIS algorithm runs on the graph induced by the output of the last call to the sparsify subroutine. We use our MIS algorithm for this last step.

It is easy to see that the final output is a $(2,\beta)$-ruling set. The independence of it holds trivially because of the last step being an MIS computation. The ruling property of it—meaning that each node v of G has a node in the finally chosen set within its distance β—follows because in each of the steps, the distance from the v to the output set grows by at most one.

Given the round complexity of each sparsify call and that of the MIS algorithm, the overall round complexity is

$$\frac{\log \Delta}{\log f_1} + \frac{\log(f_1 \log n)}{\log f_2} + \cdots + \frac{\log(f_{\beta-2} \log n)}{\log f_{\beta-1}} + \log(f_{\beta-1} \log n) + 2^{O(\sqrt{\log\log n})}.$$

To optimize this round complexity, we set $f_i = (\log \Delta)^{1-\frac{i}{\beta-1}}$. Thus, the overall round complexity becomes $O(\beta \log^{1/\beta} \Delta) + 2^{O(\sqrt{\log\log n})}$, as claimed. □

(C5) We get a faster distributed algorithm for the Lovász Local Lemma[4], which improves on results of Chung, Pettie, and Su [CPS14]. The exact guarantee provided by an

[4]We note that *having an algorithm for a lemma* might seem strange. This has become the standard terminology in the literature and we just follow the standard. The concrete relation is as follows: The Lovász Local Lemma proves the existence of a certain structure in a probabilistic spaces that satisfy particular properties. The phrase *algorithm for the Lovász Local Lemma* is usually used to refer to an algorithm that makes this existential result algorithmic and finds the related structure efficiently. Similarly, a distributed algorithm for the Lovász Local Lemma is a an efficient distributed algorithm that finds this structure. See [CPS14] for details.

algorithm for the Lovász Local Lemma is somewhat detailed. We refer the interested reader to [CPS14] for the precise description. Here, we just state the informal claim.

Corollary 3.5.5 (Informal)**.** *There is a randomized distributed algorithm for the Lovász Local Lemma with round complexity of* $O(\log_{\frac{1}{ep(\Delta+1)}} n \cdot \log \Delta)$*. Here* e *is the base of the natural logarithm, and* Δ *and* p *are respectively the degree of dependency and the given upper bound on the probability of the each bad event in the LLL instance.*

Proof Sketch. The core piece in the LLL algorithm of [CPS14] is an algorithm that computes a Weak-MIS. Our MIS algorithm allows us to perform this part faster, hence leading to an improvement on the time complexity. Roughly speaking, a Weak-MIS computation should produce an independent set S such that for each node v, with probability at least $1 - 1/\operatorname{poly}(\Delta)$, node v is either in S or has a neighbor in S. This guarantee is achieved by our MIS algorithm in $O(\log \Delta)$ rounds, as Lemma 3.4.2 proves. In contrast, the method of [CPS14] used $O(\log^2 \Delta)$ round to compute a weak-MIS. Thus, our new algorithm improves the round complexity of the distributed algorithmic version of the Lovász Local Lemma presented by Chung, Pettie, and Su [CPS14] from $O(\log_{\frac{1}{ep(\Delta+1)}} n \cdot \log^2 \Delta)$ to $O(\log_{\frac{1}{ep(\Delta+1)}} n \cdot \log \Delta)$. We note that for a weak-MIS to be useful in the constructive Lovász Local Lemma version of [CPS14], there is another technical condition regarding the independence between the events of different nodes. We do not express this condition explicitly, as formally stating it is somewhat cumbersome. But we note that it is satisfied by our MIS algorithm and it follows directly from the independence guarantee stated in Lemma 3.4.2. □

(C6) In the Local Computation Algorithms (LCA) model of Rubinfeld et al. [RTVX11] and Alon et al. [ARVX12], we get improved bounds for computing MIS. This works by replacing a part of the algorithm of Levi et al. [LRY15] with our MIS algorithm. We do not define the model or describe the whole approach here. We simply point out the change that is needed to improve the bound. We refer the interested to [LRY15] for details.

Corollary 3.5.6. *There is a randomized local computation algorithm that computes a maximal independent set with time complexity* $2^{O(\log^2 \Delta)} \log n$ *and space complexity* $2^{O(\log^2 \Delta)} \log^2 n$*.*

Proof Sketch. The result follows from replacing the $O(\log^2 \Delta)$ round Weak-MIS distributed algorithm of [LRY15, Algorithm], which is based on the distributed Weak-MIS algorithm of [CPS14], with our new distributed MIS algorithm. Then, following steps of [LRY15, Section 3.3], this distributed algorithm is turned into an LCA using a transformation approach of Parnas and Ron [PR07]. In this transformed algorithm, when answering the query about each one node, the algorithm needs to process its

$O(\log \Delta)$-neighborhood, instead of its $O(\log^2 \Delta)$-neighborhood done before. Since the graph has maximum degree at most Δ, this means that the algorithm needs to process $\Delta^{O(\log \Delta)} = 2^{O(\log^2 \Delta)}$ nodes, instead of the previous bound of $\Delta^{O(\log^2 \Delta)} = 2^{O(\log^3 \Delta)}$. It is easy to go through the proofs of [LRY15, Lemma 3 and Theorem 4] and see that this change improves the time and space complexities from, respectively, $2^{O(\log^3 \Delta)} \log^3 n$ and $2^{O(\log^3 \Delta)} \log^2 n$ bounds of [LRY15] to $2^{O(\log^2 \Delta)} \log^3 n$ and $2^{O(\log^2 \Delta)} \log^2 n$. □

(C7) We get a faster distributed MIS algorithm for the CONGESTED-CLIQUE model [LPPSP03]. In this model, per round, each node can send one $O(\log n)$-bit message to each of the other nodes (even those not adjacent to it). The formal statement is as follows:

Corollary 3.5.7. *There is a randomized distributed MIS algorithm in the* CONGESTED-CLIQUE *model that computes an MIS in $O(\log \Delta)$ rounds, with high probability.*

Proof. To obtain the claimed MIS algorithm, run the MIS algorithm of Theorem 3.1.2 for $\Theta(\log \Delta)$ rounds. If $\Delta \geq n^{0.1}$, with high probability, all nodes are removed and thus we are already done. Suppose that $\Delta \geq n^{0.1}$. It follows from Lemma 3.4.2 that each edge is left with probability at most $1/\Delta^{10}$. Thus, the expected number of edges that remain is at most $\frac{n\Delta}{\Delta^{10}} \ll n$. The number of remaining edges is in $O(n)$ with high probability. The reason is as follows: Lemma 3.4.2 shows we have independence between the random events of different edges remaining, as each edge can only depend on the events of nodes within 2-hop distance of its endpoints, and thus, each edge depends on no more than Δ^7 edges. Using a standard variant of Chernoff's concentration bound for random variables with bounded dependency degrees [Pem01, Theorem 1], we get that that the total number of edges left is at most $O(n)$, with high probability. Now since only $O(n)$ edges remain, all these edges can be gathered in a leader using Lenzen's routing scheme [Len13] in $O(1)$ rounds. Lenzen's routing scheme in the congested clique provides the following guarantee: in $O(1)$ rounds, it can deliver messages, each from a given source to a given destination, assuming that each node is the source for at most $O(n)$ messages and each node is the destination for at most $O(n)$ messages. Note that in our application, since we desire to deliver $O(n)$ edges to the leader, the latter condition is satisfied. Once the leader has received all these edges, it solves the remaining problem locally and then sends back the outputs to all nodes, reporting to them whether they are in the final MIS or not, in $O(1)$ rounds. □

Further Developments and Extensions Subsequent to the publication of the conference version of our MIS result in [Gha16b], there have been a number of papers that use this algorithmic idea, sometimes with some modifications, to solve other problems. We here simply list these papers and very briefly overview the role of our algorithm in their results.

- Bar-Yehuda, Censor-Hillel, Ghaffari, and Schwartzman [BYCHGS16] used modifications of our MIS algorithm to achieve a round complexity of $O(\frac{\log \Delta}{\log \log \Delta})$ for distributed

$(1+\varepsilon)$ approximation of maximum cardinality matching and $(2+\varepsilon)$ approximation of maximum weight matching. These improve on $O(\log n)$ round algorithms of [LPSR09, LPSP15] for the same problems.

- Censor-Hillel, Parter, and Schwartzman [CHPS16] used a direct derandomization of our MIS algorithm to present an $O(\log \Delta \log n)$ round deterministic distributed MIS algorithm in the CONGESTED-CLIQUE model.

- Holzer and Lynch [HL16] extended and modified this algorithm to present an efficient MIS algorithm in the beeping model of wireless networks.

- Ghaffari [Gha16a] used this algorithm as a starting point for an MIS algorithm with a round complexity of $\tilde{O}(\frac{\log \Delta}{\sqrt{\log n}} + 1) = \tilde{O}(\sqrt{\log \Delta})$ in the CONGESTED-CLIQUE model, which is the first such algorithm with a sublogarithmic complexity.

3.6 Open Questions

For the MIS problem, closing the gap between the new upper bound of $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ and the lower bounds of $\Omega(\min\{\frac{\log \Delta}{\log \log \Delta}, \sqrt{\frac{\log n}{\log \log n}}\})$ is the most obvious and also the most important open question. We can focus on this gap in three regimes, which we discuss below.

- The gap is most significant when $\log \Delta = \omega(\sqrt{\log n})$. It is unclear if here, the upper bound is the one that should be improved—e.g., as suggested by the case of random graphs or high-girth graphs, discussed in (C3) and (C2) above—or the lower bound should be improved. It appears that either would require quite novel techniques. Particularly, we saw in (C2) of Section 3.5 that if the lower-bound is the one that should be improved, we need to develop lower bound arguments that can handle topologies which are not locally "tree-like".

- The additive $2^{O(\sqrt{\log \log n})}$ term becomes the dominant part of the complexity when the maximum degree Δ is very small. The only concrete lower bound known in this regime is the $\Omega(\log^* n)$ round lower bound of Linial [Lin92], which holds for cycle graph where $\Delta = 2$. However, thanks to recent results of Chang et al. [CKP16], we now know why improving this $2^{O(\sqrt{\log \log n})}$ term might be a hard task, or more concretely, that it requires significant new ideas. In particular, Chang et al. show that improving on the $2^{O(\sqrt{\log \log n})}$ n-dependency term in the round complexity of randomized MIS algorithms would lead to an improvement in the n-depenency of deterministic MIS algorithms, where the best known bound is $2^{O(\sqrt{\log n})}$ due to Panconesi and Srinivasan [PS92]. The latter has remained the best known bound for about 25 years. Improving this bound is one of the most well-known open problems in the area of distributed graph algorithms and any (significant) improvement would be considered a major breakthrough.

- Finally, a third curious case in the gap is the $\log\log\Delta$ factor difference between the dependencies on Δ, between the upper and lower bound. Although this is a small gap, it hinges on a very intriguing point. In particular, the lower bound of Kuhn, Wattenhofer, and Moscibroda [KMW04, KMW16] is proven by showing a lower bound on constant approximation of vertex cover and then using reductions from maximal matching to vertex cover, and from maximal independent set to maximal matching, to lift the lower bound to MIS. It is worth noting that the $\Omega(\log\Delta)$ seemed to be the more natural lower bound and indeed, at the time of the publication of the conference version of our result [Gha16b], it was believed that $\Omega(\log\Delta)$ is a lower bound. This is because of a lower bound improvement from $\Omega(\log\Delta/\log\log\Delta)$ to $\Omega(\log\Delta)$, which was claimed in a technical report [KMW10] follow up of [KMW04]. However, this improvement was refuted very recently. Bar-Yehuda et al, Censor-Hillel, and Schwartzman [BYCHS16] presented a $2+\varepsilon$ approximation of vertex cover running in time $O(\log\Delta/\log\log\Delta)$, which proves that the initial lower bound is indeed tight. We note that the journal version of the lower bound [KMW16] has fixed the issue and the provides a correct proof for an $\Omega(\log\Delta/\log\log\Delta)$ lower bound. Hence, clearly the lower bound methods of [KMW04, KMW16], which are based on reductions from MIS to vertex cover, cannot prove a lower bound better than $\Omega(\log\Delta/\log\log\Delta)$ for MIS. This gives the hope that maybe the upper bound for MIS can be improved. Indeed, recently Bar-Yehuda, Censor-Hillel, Ghaffari, and Schwartzman [BYCHGS16] presented an algorithm with a round complexity of $O(\log\Delta/\log\log\Delta)$ for $1+\varepsilon$ approximation of Maximum Matching. A core part of this algorithm of [BYCHGS16] takes a step towards improving the complexity of MIS; it particularly computes an almost-maximal independent set in $O(\log\Delta/\log\log\Delta)$, under a certain strong definition of *almost-maximality*. The algorithm is in fact a simple modification of the MIS algorithm that we presented here. See [BYCHGS16] for details. However, it is not clear if this round complexity bound can be obtained also for maximal independent set.

Part II

Congestion—Graph-Theoretic Foundations & Results

Chapter 4

Connectivity Decompositions

4.1 Introduction & Related Work

Edge and vertex connectivity are two basic graph-theoretic concepts. As an important application, they characterize limits on how well information can be transferred among nodes of a network: one of the central goals of any communication network and also a core issue in distributed computing [Pel00, Section 1.3.1]. This is because each edge or vertex cut defines an upper bound on the flow across the cut. Naturally, we expect networks with larger connectivity—that is, those in which the minimum cut size is larger—to provide a better communication medium and support larger information flow. However, designing distributed algorithms that leverage large connectivity remains challenging.

With the goal of obtaining a large flow of information, we set forth a framework—which we call *connectivity decomposition*—that decomposes the connectivity of the graph into smaller and more manageable units, roughly speaking. More concretely, a connectivity decomposition partitions a graph with large connectivity into many (essentially) "disjoint" trees, while almost preserving the total connectivity through the trees. These decompositions open the road for parallelizing the flow of information along the trees and thus achieving a total flow value close to the connectivity of the network.

The remainder of this section is organized as follows: In Section 4.1.1, we explain the concepts of these connectivity decompositons, while drawing parallels between those of the edge connectivity decomposition and those of the vertex connectivity decomposition. In Section 4.1.2, we then review the edge connectivity decomposition results that follow from previously known work, especially the seminal 1960 work of Tutte [Tut61] and Nash-Williams [NW61]. In Section 4.1.3, we present our results on vertex connectivity decompostions. Finally, in Section 4.1.4, we state some implications and applications of our results.

As a side remark, we note that the graph-theoretic framework and the related existential results presented here provide the basics and set the stage for some of the distributed algorithms that we present in Part III of the thesis. In particular, in Chapter 9 of Part III, we present efficient distributed and centralized algorithms for constructing the connectivity

decompositions that we prove to exist in this chapter, and we also explain the algorithmic applications of these constructions.

4.1.1 Introducing Connectivity Decompositions

Connectivity Decomposition and Tree Packings

Menger's theorem (see [BM08, Chapter 9])—which is one of the most basic results concerning graph connectivity—states that in each graph with edge connectivity λ or vertex connectivity k, each pair of vertices is connected via λ edge-disjoint paths or k internally vertex-disjoint paths, respectively. However, when we have to deal with more than two nodes, this theorem does not provide a strong enough characterization. This is especially because it does not provide any information about the structure of the overlaps between paths of different vertex pairs, or about structures connecting three or more nodes (i.e., *Steiner trees*).

To organize the connectivity in a way that makes it accessible to algorithms, and especially for distributed algorithms, we consider edge and vertex connectivity decompositions. When we are interested in edge connectivity and edge cuts, this will be by decomposing the graph into *edge-disjoint spanning* trees. When we are interested in vertex connectivity and vertex cuts, this will be by decomposing the graph into *vertex-disjoint dominating* trees.

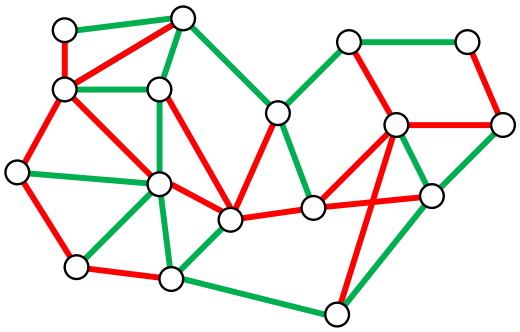
Recall that a tree $H = (V_T, E_T)$ is a *spanning tree* of $G = (V_G, E_G)$ if $E_T \subseteq E_G$ and $V_T = V_G$. On the other hand, a tree $H = (V_T, E_T)$ is called a *dominating tree* of $G = (V_G, E_G)$ if $E_T \subseteq E_G$ and V_T dominates G, that is, each node in $V_G \setminus V_T$ has a G-neighbor in V_T[1].

We next describe and discuss the concepts of edge and vertex connectivity decomposition, via spanning and dominating tree packings, respectively:

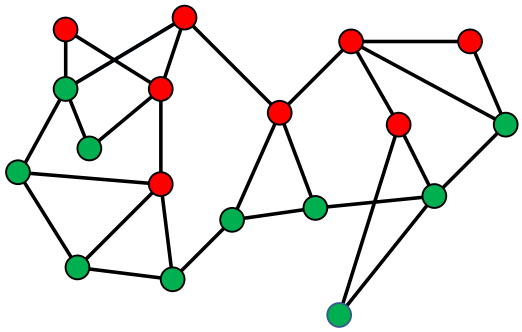
- **Edge Connectivity Decomposition via Spanning Tree Packing:** We define a *spanning tree packing of size* λ', for a non-negative integer λ', to be a collection of λ' edge-disjoint spanning trees. Given such a packing, for each pair of vertices, we get λ' edge-disjoint paths, one through each tree. More importantly, for any number of vertex pairs, the paths going through different trees are edge-disjoint. Hence, a spanning tree packing can be viewed as a decomposition of edge connectivity. Figure 4-1a shows an example graph with a spanning tree packing of size 2.

- **Vertex Connectivity Decomposition via Dominating Tree Packing:** We define a *dominating tree packing of size* k', for a non-negative integer k', to be a collection of k' vertex-disjoint dominating trees. Given such a packing, for each vertex pair we get k' internally vertex-disjoint paths, one through each tree. More importantly, for any number of pairs, the paths going through different trees are internally vertex-disjoint. Hence, a dominating tree packing can be viewed as a decomposition of vertex

[1]Note that if we want to have many vertex-disjoint subgraphs, we cannot ask them to be *"spanning"* subgraphs. As we will see soon, in this case, the *"dominating"* condition turns out to be the natural and the practically relevant requirement.



(a) Two edge-disjoint spanning trees, that is, a spanning tree packing of size 2.



(b) Two vertex-disjoint dominating trees, that is, a dominating tree packing of size 2.

connectivity. Figure 4-1b shows an example graph with a dominating tree packing of size 2.

Fractional Relaxation In both spanning and dominating tree packings, we can relax the disjointness requirement to *fractional disjointness*. That is, we allow the trees to overlap but now each tree τ has a weight $\omega_\tau \in [0, 1]$ and for each edge or vertex, respectively, the total weight of the trees including it has to be at most 1. In applications, this naturally corresponds to sharing the edge (or the vertex) between the trees in a way that is proportional to their weights. For instance, in information dissemination, we could use time-sharing where each edge (or vertex) gets used in each of the trees that include it for a fraction of time proportional to the weight of that tree. In fact, for all the applications that we have in mind, these fractional tree packings are as useful as their stronger integral packing counterparts.

4.1.2 Known Results On Edge Connectivity Decomposition

Edge connectivity decompositions have been well-known[2] for a long time, thanks to beautiful (existential) results of Tutte [Tut61] and Nash-Williams [NW61] from 1960. As shown in [Kun74], the results of these papers imply that each graph with edge-connectivity λ contains a spanning tree packing of size $\lceil \frac{\lambda-1}{2} \rceil$. This bound is existentially tight even for the fractional version. This structural result leads to numerous applications for different problems concerning edge-connectivity. Two well-known examples in the area of graphs algorithms are the first near-linear time (and also the current best) min-cut algorithm [Kar96] and the tightest proof of the upper bound of $O(n^{2\alpha})$ on the number of α-minimum edge-cuts [Kar94a], which itself has many implications including Karger's famous random edge-sampling results [Kar94a, Kar95].

[2]However, we remark that they were not called "*edge-connectivity decompositions*". Often, the phrase "*edge-disjoint spanning trees*" was used to refer to the same concept.

Remark We note that although we do not present a new graph-theoretic result on edge connectivity decomposition, we mentioned them here for two reasons: (1) Edge connectivity decompositions are quite similar, in nature, to those of vertex connectivity decomposition and thus contrasting them seems instructive. (2) In Part III of the thesis, and particularly in Chapter 9, we present distributed algorithms for constructing edge connectivity decompositions, as well as vertex connectivity decompositions, and thus reviewing these basic graph-theoretic concepts here sets the stage for those parts.

4.1.3 Our Results on Vertex Connectivity Decomposition

Our results on vertex connectivity decompostion can be viewed as analogues of the classical results of Tutte [Tut61] and Nash-Williams [NW61] on edge connectivity decomposition. We particularly show matching or nearly matching results on fractional and integral dominating tree packing sizes in k-vertex-connected graphs.

To simplify matters, instead of working directly with dominating trees, we work with closely related objects, Connected Dominating Sets (CDSs), and packings of CDSs. A Connected Dominating Set (CDS) is a dominating set of vertices—meaning that each node of the graph is either in this set or has a neighbor in it—that induces a *connected* subbgraph. Note that packing dominating trees and CDSs are equivalent, in the following sense: On one hand, given a CDS of the graph G, one can easily find a dominating tree in this CDS, simply by picking a spanning tree of the subgraph induced by the CDS. On the other hand, given a dominating tree of the graph G, the set of vertices of this dominating tree form a CDS. We work with CDSs because then we only need to identify the subsets of the nodes in each structure, i.e. each CDS, instead of needing to also identify the related edges. Once the CDSs in the packing are identified, one can use any spanning tree of the subgraph induced by each CDS to get a corresponding dominating tree packing.

Before stating our results, we note that the concept of CDS packing was touched upon in the past in the literature. However, obtaining an optimal CDS packing, or even CDS packings whose size is the vertex connectivity, remained wide open. The CDS packing problem was first studied in [HL83] where the size of a maximum CDS packing is called the *connected domatic number* (CDN). Zelinka [Zel86] showed a number of results on corner cases of the CDN; for instance, the simple observation that it is upper bounded by the vertex connectivity. Moreover, Hartnell and Rall [HF01] showed that the CDN of planar graphs is at most 4.

Our Results: We provide three results on the size of fractional and integral CDS packings in k-vertex-connected graphs, Theorem 4.1.1, Theorem 4.1.2, and Theorem 4.1.3. The first two theorems are constructive, and prove existence of large fractional or integral CDS packings. The third is an impossibility result, which complements the first two and proves their bounds to be tight or nearly tight.

We first show that each graph with vertex-connectivity k contains a fractional CDS packing of size $\Omega(k/\log n)$.

Theorem 4.1.1. *Every k-vertex-connected n-vertex graph has a fractional CDS packing of size $\Omega\big(\frac{k}{\log n}+1\big)$.*

Specifically, we show how to construct a collection of k CDSs, each consisting of $O\big(\frac{n\log n}{k}\big)$ vertices, such that each vertex is in at most $O(\log n)$ of the CDSs.

Using a similar construction style to that of Theorem 4.1.1, we also obtain an efficient way to get a large CDS partition, which leads to Theorem 4.1.2.

Theorem 4.1.2. *Every k-vertex-connected n-vertex graph graph G has an integral CDS packing of size $\Omega\big(\frac{k}{\log^2 n}+1\big)$.*

The following theorem complements the above by showing that the $\Omega\big(\frac{k}{\log n}\big)$ bound achieved for fractional CDS packing bound is optimal.

Theorem 4.1.3. *For any sufficiently large n, and any $k \in [1, n/4]$, there exist n-vertex graphs with vertex connectivity k where the maximum fractional (or integral) CDS packing size is $O\big(\frac{k}{\log n}+1\big)$.*

4.1.4 Implications & Applications of Our Results

Our vertex connectivity decomposition result has implications for problems in graph theory and in network information dissemination. These implications are interesting also as existential results, that is, even without considering the related algorithmic aspects. We overview this existential implications here. In Chapter 9, we see how these implications can be made algorithmic with efficient algorithms.

Vertex-Independent Trees Zehavi and Itai [ZI89] conjectured in 1989 that each k-vertex-connected graph contains k *vertex-independent trees*, that is, k spanning trees all rooted in one node $r \in V$ such that for each vertex $v \in V$, the paths between r and v in different trees are internally vertex-disjoint. This conjecture remains open. Theorem 4.1.2 allows us to obtain an approximation of this conjecture. Concretely, we show that

Corollary 4.1.4. *Any k-vertex-connected n-node graph contains $\Omega(k/\log^2 n+1)$ vertex-independent trees.*

This is because, given a CDS packing of size $\Omega(k/\log^2 n+1)$, we can obtain $\Omega(k/\log^2 n+1)$ vertex-independent trees as follows: define each tree by taking a spanning subtree of the CDS, then adding all nodes that are not in the CDS as the leaves of the tree, and orienting the tree as rooted in vertex r. Notice that since different CDSs are vertex-disjoint, for each vertex $v \in V$, the paths between r and v in different trees will be internally vertex-disjoint.

Throughput of Network Information Dissemination and the Network Coding Gap Vertex connectivity decompositions and CDS packings provide an asymptotically tight combinatorial characterization of the throughput of network information dissemination protocols. This leads to an interesting and important result in network information disseminations. Here, we describe only the existential aspects. The algorithmic aspects are deferred to Part III of this thesis.

Consider the V-CONGEST model where in each round, each node of a network can send one B-bits message to all its neighbors. We show that the achievable total throughput when globally broadcasting messages using *routing-based* algorithms can exactly be characterized by the size of the largest CDS packing of the network graph. A *routing-based* algorithm (also known as *store-and-forward*) corresponds to the classical paradigm of routing where messages are viewed as atomic tokens and each node only stores and forwards messages and cannot combine them or parts of them[3].

Theorem 4.1.5. *Consider any network in the* V-CONGEST *model. Given a (fractional) CDS packing of size K, we can construct a store-and-forward broadcast algorithm with throughput $\Omega(K)$ messages per round. Conversely, given a store-and-forward broadcast algorithm with throughput K messages per round, we can construct a fractional CDS packing of size K.*

The proof of this theorem is presented in Section 4.6. Using Theorem 4.1.5 and our CDS packing result Theorem 4.1.1, we get that there exists a routing-based broadcast algorithm with throughput $\Theta(\frac{k}{\log n})$ messages per round. Moreover, combining Theorem 4.1.5 with Theorem 4.1.3, we get that this throughput is indeed optimal, in the worst case network graph with vertex connectivity k.

The above also tightly characterizes the *network coding gap* (i.e., the throughput gain provided by network coding compared to routing) for the V-CONGEST model. Techniques of [Hae11] can be used to prove, given full knowledge of the topology of the network, one can design a protocol based on network coding that can achieve a throughput of $\Theta(k)$. Thus, our tight $\Theta(\frac{k}{\log n})$ CDS-packing bound—proven in two parts in Theorem 4.1.1 and Theorem 4.1.3—implies that the *network coding gap* for broadcast in V-CONGEST is a tight $\Theta(\log n)$.

We note that network coding has been studied extensively (see [ACLY00] and citations thereof). Since the gains of network coding are usually accompanied by new complications and costs (see e.g. [FS07]), determining the network coding gap for different networking models is one of the important related questions. This has been of interest both in theory (see, e.g., [LLL09, AC04, LM09, GK08, AGHK13]) and in practice (especially for wireless networks—see [KRH+06] and its citations). In particular, in a seminal paper, Li et al. [LLL09] use the Tutte-Nash-Williams edge-disjoint spanning trees result to show that in undirected wired networks—the model where in each round each node can send one distinct message to each of its neighbors—the network coding gap is $\Theta(1)$. Agarwal and Charikar

[3] This is in contrast to the newer (more general and complex) paradigm of *network coding* (see [ACLY00] and citations thereof) where each node can send any B-bit function of the received messages.

[AC04] prove an $\Omega(\log^2 n/\log^2 \log n)$ lower bound on the network coding gap in directed wired networks, while the related upper bound remains wide open. Alon et al. [AGHK13] showed an $\Omega(\log \log n)$ bound on the network coding gap for the wireless network model below the MAC layer (i.e., with collisions).

4.2 Preliminaries

Definition 4.2.1. *(**Dominating Set and Connected Dominating Set**) Given a graph $G = (V, E)$, a set $S \subseteq V$ is called a* dominating set *iff each vertex $u \in V \setminus S$ has a neighbor in S. The set S is called a* connected dominating set *(CDS) iff S is a dominating set and $G[S]$ is connected. If S is a dominating set of G, we also say that S dominates V.*

Definition 4.2.2 (CDS Packing). *A* CDS packing *of a graph $G = (V, E)$ is a partition $V_1 \cup \cdots \cup V_t = V$ of the vertices V such that each set V_i is a CDS. The* size *of a CDS packing is the number of CDSs of the partition. The maximum size of a CDS packing of G is denoted by $K_{CDS}(G)$.*

Definition 4.2.3 (Fractional CDS Packing). *Let $CDS(G)$ be the set of all CDSs of a graph G. A* fractional CDS packing *of G assigns a non-negative real-valued weight x_τ to each $\tau \in CDS(G)$ such that for each vertex $v \in V$, $\sum_{\tau \ni v} x_\tau \leq 1$. The* size *of this fractional CDS packing is $\sum_{\tau \in CDS(G)} x_\tau$. The maximum size of a fractional CDS packing of G is denoted by $K'_{CDS}(G)$.*

Note that a CDS packing is a special case of a fractional CDS packing where each $x_\tau \in \{0, 1\}$. In other words, fractional CDS packing is the LP relaxation of CDS packing when formulating CDS packing as an integer programming problem in the natural way. Consequently, we have $K_{CDS}(G) \leq K'_{CDS}(G)$ for every graph G.

We remark that the maximum CDS packing size of graph G is sometimes also called the *connected domatic number* of G [HL83, Zel86, HF01]. Analogously, the maximum fractional CDS packing size can be referred to as the *fractional connected domatic number* of G.

As each CDS must contain at least one vertex of each vertex cut, we have $K_{CDS}(G) \leq k$. Based on the same basic argument, the same upper bound applies to fractional CDS packings:

Proposition 4.2.4. *For each graph with vertex-connectivity k, we have*

$$K_{CDS}(G) \leq K'_{CDS}(G) \overset{(*)}{\leq} k.$$

Proof. Consider a vertex cut $\mathcal{C} \subseteq V$ of G that has size exactly k. Each CDS τ must include at least one node in $\mathcal{C}$. For each CDS $\tau \in CDS(G)$, pick one node $v \in C$ as a representative of τ in the cut and let us denote it by $Rep(\tau)$. Thus, for any fractional CDS Packing of G, we have

$$\sum_{\tau \in CDS(G)} x_\tau = \sum_{v \in \mathcal{C}} \sum_{\substack{\tau \in CDS(G) \\ s.t.\ v = Rep(\tau)}} x_\tau \leq \sum_{v \in \mathcal{C}} 1 = |\mathcal{C}| = k.$$

Since the above holds for any fractional CDS Packing of G, we get that $K'_{CDS}(G) \leq k$. □

We prove in Theorem 4.1.3, which is presented in Section 4.5, that the gap in the inequality $(*)$ can be as large as $\Omega(\log n)$. On the other hand, Theorem 4.1.1, which we prove in the next section shows that this gap cannot be larger than $O(\log n)$.

4.3 Fractional CDS Packing

In this section, we present a method to construct a fractional CDS packing of size $O(k/\log n + 1)$. This serves mainly as a proof of existence, but it can also be seen as an polynomial-time centralized construction. Recall that Theorem 4.1.3 (proven in Section 4.5) shows that this $O(k/\log n)$ bound is existentially optimal.

We note that in Section 4.4, we use a variant of the construction we explain here to obtain an integral CDS packing. In particular, the construction in Section 4.4 relies on a random *sampling* of vertices. To keep the explanations as close as possible, we generalize the construction of this section, and we present it in a generalized setting where each vertex is kept with probability p. In this generalized setting, we give a fractional CDS packing with size $\Omega(kq^2/\log n)$, where $q = 1 - (1-p)^{1/(3L)}$ and $L = \lambda \log n = \Theta(\log n)$ is the number of *layers* in the construction (to be explained soon). Setting $p = 1$ easily takes us back to the setting where all nodes are kept. In particular, when we set $p = 1$, we get $q = 1$ and thus, the fractional CDS packing size becomes $\Omega(k/\log n)$ as claimed.

4.3.1 High-Level Outline of the Construction

Construction Framework: Transition to a Nicely-Structured Virtual Graph

We construct the fractional CDS packing gradually. We start in a setting where each eventual CDS is an empty set. We then gradually add nodes to the eventual CDSs until they satisfy *domination*. Afterward, we gradually add more nodes until the eventual CDSs satisfy *connectivity*.

To realize the above outline, we ideally would like to partition the vertices of the graph into L *layers* and to construct the CDSs gradually, as we through the layers one-by-one. In each layer, we would add only the vertices of that layer to the (eventual) CDSs. For this layer-by-layer construction to work, we need that the subgraph induced by each layer has a high vertex connectivity. However, it is not easy to partition the graph into many layers and have a high vertex connectivity in the subgraph induced by each layer. That is, having a high per-layer vertex connectivity is hard when partitioning the vertices of the original graph G into layers. Because of this, instead of working directly on G, we transform G into a new graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that we call the *virtual graph*. We next present an informal discussion about this virtual graph and how it helps us in the construction, and then we present its formal definition.

Informal Discussion about the Virtual Graph: In an informal sense, the virtual graph $\mathcal{G}$ is essentially made of $\Theta(\log n)$ copies of the original graph G. In particular, it has $\Theta(\log n)$ layers, where each layer is a copy of the original graph G. Moreover, copies of the same or adjacent vertices are connected even across different layers. Since each layer is a copy of G, each layer has a high vertex connectivity. This opens the road for our layer-by-layer CDS packing construction. In particular, we will construct an integral CDS packing of size $\Theta(kq^2)$ on this virtual graph $\mathcal{G}$ by putting each copy vertex in exactly one CDS. Then, this integral CDS packing of $\mathcal{G}$ can be easily turned into a fractional CDS packing with size $\Theta(kq^2/\log n)$ in the original graph G, simply by giving each CDS weight $1/\Theta(\log n)$. This will be formalized soon.

Formal Definition of the Virtual Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: For each vertex $v \in V$ (and a sufficiently large constant λ), create $3L = 3\lambda \log n$ *virtual vertices* that are copies of v, referred to as $(v, 1), (v, 2), \ldots, (v, 3L)$, three for each layer ℓ in $[1, L]$. Connect two virtual vertices (v, i) and (u, j) if and only if they are copies of the same real vertex, that is $v = u$, or they are copies of two adjacent real vertices, that is, v and u are adjacent in G.

Note that for each layer ℓ, the virtual vertices of layer ℓ induce a copy of G. For each set of virtual vertices $\mathcal{W} \subseteq \mathcal{V}$, define the projection $\Psi(\mathcal{W})$ of $\mathcal{W}$ onto G as the set $W \subseteq V$ of real vertices w, for which at least one virtual copy of w is in $\mathcal{W}$.

Proposition 4.3.1. *Two vertices in $\mathcal{G}$ are connected directly if and only if they project to the same vertex or to neighboring vertices in G. Thus, $\mathcal{G}[\mathcal{W}]$ is connected (resp. dominating) if and only if $G[\Psi(\mathcal{W})]$ is connected (resp. dominating).*

Translating Random Sampling to the Virtual Graph Recall from the second paragraph of the section that we are working in a generalized setting where each node of G is sampled with probability p, and with the remaining probability, the node is discarded. In particular, setting $p = 1$ takes us back to the setting with no sampling where all nodes are kept. To translate this sampling to $\mathcal{G}$, consider the following process: sample each virtual vertex with probability $q = 1 - (1 - p)^{1/(3L)}$ and then sample each real vertex $v \in V$ if and only if at least one of its virtual copies is sampled (that is, the sampled real vertices are obtained by projecting the sampled virtual vertices onto G). The probability of each real vertex being sampled is exactly $1 - (1 - q)^{3L} = p$. Henceforth, we work on $\mathcal{G}$ assuming that each virtual vertex is sampled independently with probability $q = 1 - (1 - p)^{1/(3L)} \geq \frac{p}{6L}$.

Construction Outline on the Virtual Graph

High-Level Discussion In the rest of this section, we work on $\mathcal{G}$ and show how to construct $t = \delta \cdot kq^2$ vertex-disjoint connected dominating sets on the sampled virtual vertices, for a sufficiently small constant $\delta > 0$. That is, we create an *integral* CDS *packing* of size $\Omega(kq^2)$ on the sampled virtual vertices. This can be easily transformed into the claimed fractional CDS packing size of $\Omega(kq^2/\log n)$ on the original sampled graph: since each real

vertex has $\Theta(\log n)$ virtual copies, by giving a weight of $1/\Theta(\log n)$ to each CDS in the integral CDS packing of $\mathcal{G}$, we get the claimed fractional CDS packing size of G.

We have t *classes* and we assign each sampled virtual vertex to one class, such that, eventually each class is a CDS, w.h.p. To organize the construction, we group the virtual vertices in L layers, putting three copies of graph G in each layer. Inside each layer, the three copies are distinguished by a *type number* in $\{1, 2, 3\}$.

Notation We now present the notations that we use throughout the construction and the analysis. Let $\mathcal{S}_\ell^i$ be the set of sampled virtual vertices of layers 1 to ℓ that are assigned to class i. Let N_ℓ^i be the number of connected components of $\mathcal{G}[\mathcal{S}_\ell^i]$. Finally, define $M_\ell := \sum_{i=1}^{t}(N_\ell^i - 1)$ to be the total number of excess components after considering layers $1, \ldots, \ell$, compared to the ideal case where each class is connected.

Construction Plan: Domination and Connectivity Initially $M_1 \leq n$. The idea is to do the assignments of vertices to classes such that M_ℓ decreases essentially exponentially with ℓ, until it becomes zero, meaning each class induces a connected sub-graph.

The assignments are done as follows: We first do a jump start and use all the first $\frac{L}{2}$ layers together in one shot to achieve domination for all the classes. We describe this part in Section 4.3.2. We note this part of the construction is straightforward, and the technically interesting aspect is in achieving connectivity. After obtaining domination in the first half of the layers, we aim for achieving connectivity in each class. This part is described in Section 4.3.3. In particular, we go over the remaining $\frac{L}{2}$ layers one by one, and gradually add vertices of the layers to the classes in order to achieve connectivity. In an informal sense, in each layer $\ell \in [L/2, L-1]$, we assign vertices of layer $\ell + 1$ to classes based on the assignments of vertices of layers 1 to ℓ, using a careful method. This method ensures that the total number M_ℓ of excess connected components over all the classes together decreases essentially as much as possible. We show that in each layer this number decreases by a constant factor (in expectation). After working through all the layers, we reach a setting where $M_\ell = 0$ for $\ell = L$, with high probability, Once $M_\ell = 0$, all the classes have connectivity. Hence, at that point, each class is a CDS.

4.3.2 Domination

Assignments for Domination We begin the assignment with a jump-start, assigning each of the sampled virtual vertices of layers 1 to $\frac{L}{2}$ to a random classes.

We next show in Lemma 4.3.2 that this straightforward random assignment already gives domination.

Lemma 4.3.2 (Domination Lemma). *W.h.p., for each class i, $\mathcal{S}_{L/2}^i$ dominates $\mathcal{V}$.*

Proof. Since G has vertex connectivity k, each real node v has at least k real neighbors and in these k real neighbors, in expectation at least $\frac{kq}{2t} = \Omega(\log n)$ virtual nodes are sampled,

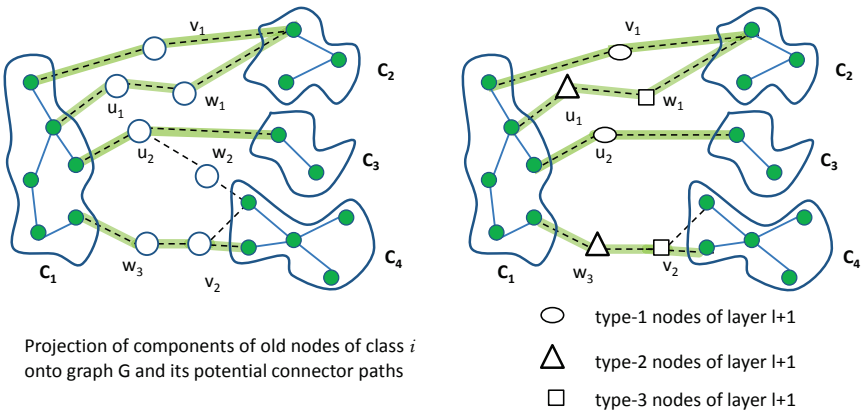


Figure 4-2: Potential Connector Paths for component $\mathcal{C}_1$ in layer $\ell+1$ copies of G

have layer number between 1 and $\frac{L}{2}$, and join class i. These are virtual nodes that are in $\mathcal{S}^i_{\frac{L}{2}}$. Thus, the claim follows from a standard Chernoff bound combined with a union bound over all choices of v and over all classes. □

Note that the domination of each class follows directly from this lemma. For the rest of this section, we assume that for each class i, $\mathcal{S}^i_{L/2}$ dominates $\mathcal{V}$, and we use this property also in achieving connectivity. In particular, as we explain in Lemma 4.3.4, it will be used for obtaining *short connector paths*.

4.3.3 Connectivity

Outline After the first $\frac{L}{2}$ layers, which provide domination of each class, we try to achieve connectivity for all classes, using the last $\frac{L}{2}$ layers. In particular, we work on these remaining layers one by one and for each layer $\ell \in [L/2, L-1]$, we assign vertices of layer $\ell+1$ to classes based on the assignments of vertices of layers 1 to ℓ. We now explain this assignment for layer $\ell+1$. We refer to vertices of layers 1 to ℓ as *old vertices* whereas vertices of layer $\ell+1$ are called *new vertices*. The goal is to perform the class assignment of the newly sampled vertices such that (in expectation) the number of connected components decreases by a constant factor in each layer. This is formalized in the *Fast Merger Lemma* presented as Lemma 4.3.5. During the recursive assignments, our main construction tool will be the concept of *connector paths*, defined next.

Defining Connector Paths and Proving Their Abundance

Consider a class i, suppose $N^i_\ell \geq 2$, and consider a component $\mathcal{C}$ of $\mathcal{G}[\mathcal{S}^i_\ell]$. We use the projection $\Psi(\mathcal{S}^i_\ell)$ onto graph G as defined above. A path P in G is called a *potential connector* for $\Psi(\mathcal{C})$ if it satisfies the following three conditions: (A) P has one endpoint in $\Psi(\mathcal{C})$ and the other endpoint in $\Psi(\mathcal{S}^i_\ell \setminus \mathcal{C})$, (B) P has at most two internal vertices, (C) if P has exactly two internal vertices and has the form s, u, w, t where $s \in \Psi(\mathcal{C})$ and $t \in \Psi(\mathcal{S}^i_\ell \setminus \mathcal{C})$, then w

does not have a neighbor in $\Psi(\mathcal{C})$ and u does not have a neighbor in $\Psi(\mathcal{S}_\ell^i \setminus \mathcal{C})$. Condition (C) is an important condition, requiring *minimality* of each potential connector path. That is, there is no potential connector path connecting $\Psi(\mathcal{C})$ to another component of $\Psi(\mathcal{S}_\ell^i)$ via only u or only w.

From a potential connector path P on graph G, we derive a potential connector path $\mathcal{P}$ on virtual graph $\mathcal{G}$ by determining the types of related internal vertices as follows: (D) If P has one internal real vertex w, then for $\mathcal{P}$ we choose the virtual vertex of w in layer $\ell+1$ in $\mathcal{G}$ with type 1. (E) If P has two internal real vertices w_1 and w_2, where w_1 is adjacent to $\Psi(\mathcal{C})$ and w_2 is adjacent to $\Psi(\mathcal{S}_\ell^i \setminus \mathcal{C})$, then for $\mathcal{P}$ we choose the virtual vertices of w_1 and w_2 in layer $\ell+1$ with types 2 and 3, respectively. Finally, for each endpoint w of P we add the copy of w in $\mathcal{S}_\ell^i$ to $\mathcal{P}$. A given potential connector path $\mathcal{P}$ on the virtual vertices of layer $\ell+1$ is called a *connector path* if and only if the internal vertices of $\mathcal{P}$ are sampled. We call a connector path that has one internal vertex a *short connector path*, whereas a connector path with two internal vertices is called a *long connector path*. Because of condition (C), and rules (D) and (E) above, we get the following fact:

Proposition 4.3.3. *For each class i, each type-2 virtual vertex u of layer $\ell+1$ is on connector paths of at most one connected component of $\mathcal{G}[\mathcal{S}_\ell^i]$.*

Figure 4-2 demonstrates an example of potential connector paths for a component $\mathcal{C}_1 \in \mathcal{G}[\mathcal{S}_\ell^i]$. The figure on the left shows graph G, where the projection $\Psi(\mathcal{S}_\ell^i)$ is indicated via green vertices, and the green paths are potential connector paths of $\Psi(\mathcal{C}_1)$. On the right side, the same potential connector paths are shown, where the type of the related internal vertices are determined according to rules (D) and (E) above, and vertices of different types are distinguished via different shapes (for clarity, virtual vertices of other types are omitted).

The following lemma shows that in each class that has at least two components, each component is likely to have many connector paths.

Lemma 4.3.4 (Connector Abundance Lemma). *Consider a layer $\ell \geq L/2$ and a class i such that $\mathcal{S}_{L/2}^i \subseteq \mathcal{S}_\ell^i$ is a dominating set of $\mathcal{G}$ and $N_\ell^i \geq 2$. Further consider an arbitrary connected component $\mathcal{C}$ of $\mathcal{G}[\mathcal{S}_\ell^i]$. Then, with probability at least $1/4$, $\mathcal{C}$ has at least t internally vertex-disjoint connector paths. Recall that $t = \delta \cdot kq^2$, for a small enough constant δ, is the number of classes.*

Proof. Fix a layer $\ell \in [L/2, L-1]$. Let $\mathcal{D}$ be the set of dominating sets of $\mathcal{G}$ consisting only of nodes from layers $1, \ldots, L/2$. Further, for all $\ell \geq L/2$, let $\mathcal{D}_\ell$ contain all sets $\mathcal{T} \subseteq \mathcal{V}_\ell$ such that there exists a $D \in \mathcal{D}$ for which $D \subseteq \mathcal{T}$. That is, $\mathcal{D}_\ell$ is the collection of all sets of virtual nodes in layers $1, \ldots, \ell$ that contain a dominating set $D \in \mathcal{D}$. Fix an arbitrary set $\mathcal{T} \in \mathcal{D}_\ell$ and fix $\mathcal{S}_\ell^i = \mathcal{T}$.

Consider the projection $\Psi(\mathcal{S}_\ell^i)$ onto G and recall Menger's theorem: Between any pair (u, v) of non-adjacent nodes of a k-vertex connected graph, there are k internally vertex-disjoint paths connecting u and v. Applying Menger's theorem to a node in $\Psi(\mathcal{C})$ and a node in $\Psi(\mathcal{S}_\ell^i \setminus \mathcal{C})$, we obtain at least k internally vertex-disjoint paths between $\Psi(\mathcal{C})$ and $\Psi(\mathcal{S}_\ell^i \setminus \mathcal{C})$

in G. We first show that these paths can be shortened so that each of them has at most 2 internal nodes i.e., to get property (B) of potential connector paths. Pick an arbitrary one of these k paths and denote it $P = v_1, v_2, ..., v_r$, where $v_1 \in \Psi(\mathcal{C})$ and $v_r \in \Psi(\mathcal{S}^i_\ell \setminus \mathcal{C})$. By the assumption that $\mathcal{S}^i_{L/2}$ dominates $\mathcal{G}$, we get that $\Psi(\mathcal{S}^i_\ell)$ dominates G. Hence, since $v_1 \in \Psi(\mathcal{C})$ and $v_r \in \Psi(\mathcal{S}^i_\ell \setminus \mathcal{C})$, either there is a node v_i along P that is connected to both $\Psi(\mathcal{C})$ and $\Psi(\mathcal{S}^i_\ell \setminus \mathcal{C})$, or there must exist two consecutive nodes v_i, v_{i+1} along P, such that one of them is connected to $\Psi(\mathcal{C})$ and the other is connected to $\Psi(\mathcal{S}^i_\ell \setminus \mathcal{C})$. In either case, we can derive a new path P' which satisfies (B) and is internally vertex-disjoint from the other $k-1$ paths since its internal nodes are a subset of the internal nodes of P and are not in $\Psi(\mathcal{S}^i_\ell)$. After shortening all the k internally vertex-disjoint paths, we get k internally vertex-disjoint paths in graph G that satisfy conditions (A) and (B), as stated Section 4.3.3.

Now using rules (D) and (E) in Section 4.3.3, we get k internally vertex-disjoint potential connector paths on the virtual nodes of layer $l+1$. It is clear that during the transition from the real nodes to the virtual nodes, the potential connector paths remain internally vertex-disjoint. Now, for each fixed potential connector path on the virtual nodes, the probability that the internal nodes of this path are sampled is at least q^2. Hence, in expectation, $\mathcal{C}$ has at least kq^2 internally vertex-disjoint connector paths (on virtual nodes). Since the paths are internally vertex-disjoint, we have independence among different paths. Thus, we can apply a Chernoff bound and conclude that with probability at least $1/2$, component $\mathcal{C}$ has at least $t = \Omega(kq^2)$ internally vertex-disjoint connector paths. □

The Algorithm for Class Assignments Using Connector Paths

From Lemma 4.3.4 we know that for each connected component of each class i with $N^i_\ell \geq 1$, with probability at least $1/2$, this component has at least t connector paths. Call a component *poor* if it has less than t connector paths, and *rich* otherwise. For each rich component, pick exactly t of its connector paths. We expect at most $M_\ell/2$ poor components, over all the classes and components. Thus, by Markov's inequality, the probability that there are more than $3M_\ell/4$ poor components is at most $2/3$. Therefore, with probability at least $1/3$, there are at least $M_\ell \cdot t/4$ connector paths in total, over all the class and components. We use these connector paths to assign the class numbers of vertices of layer $\ell+1$. This part is done via a greedy-style algorithm, in three stages as follows:

(I) For each type-1 new vertex v: For each class i, define the *class-i-degree of v* to be the number of connected components of class i that have a *short* connector path through v. Let Δ be the maximum *class-i-degree* of v as i ranges over all classes, and let i^* be a class that attains this maximum. If $\Delta \geq 1$: Assign v to class i^*. Also, remove all connector paths of all classes that go through v and remove all connector paths of all the connected components of class i^* that have v on their short connector paths.

(II) For each type-3 new vertex u: For each class i, define the *class-i-degree of u* to be the number of connected components of class i which have a *long* connector path through

u. Let Δ be the maximum *class-i-degree* of u as i ranges over all classes and let i^* be a class that attains this maximum. If $\Delta \geq 1$: Assign u to class i^*. Moreover, each of the Δ long connector paths of class i^* that goes through u also has a type-2 internal vertex. Let these type-2 vertices be $v_1, \ldots, v_\Delta$. Assign $v_1, \ldots, v_\Delta$ to class i^*. Then, remove all the connector paths that go through u or any of the vertices $v_1, \ldots, v_\Delta$. Also remove all connector paths of each component of class i^* that has a connector path going through u.

(III) Assign each remaining new vertex to a random class.

Analysis of the Class Assignments

We now analyze the above class assignment procedure and prove that it is likely to lead to a constant factor reduction in the total number of excess components. More concretely, we prove that:

Lemma 4.3.5 (Fast Merger Lemma). *For each $\ell \geq \frac{L}{2}$ and every assignment of the sampled vertices of layers $1, \ldots, \ell$ to classes such that for all classes i, $\mathcal{S}^i_{L/2}$ is a dominating set of $\mathcal{G}$, we have (a) $M_{\ell+1} \leq M_\ell$, and (b) with probability at least $1/3$, $M_{\ell+1} \leq \frac{23}{24} \cdot M_\ell$.*

The proof is based on an accounting argument that uses the total number of remaining connector paths over all classes and components as the budget, and shows that number of components that are merged, each with at least one other component, is at least $\frac{M_\ell}{3}$.

Proof. For part (a) of the lemma, note that since for each class i, set $\mathcal{S}^i_{L/2}$ is a dominating set of $\mathcal{G}$ and for each layer $\ell \geq L/2$, $\mathcal{S}^i_{L/2} \subseteq S^i_\ell$, we get that S^i_ℓ dominates set $\mathcal{V}$. Thus, each virtual sampled node in layer $\ell + 1$ has a neighbor in S^i_ℓ which means that each connected component of $\mathcal{G}[\mathcal{S}^i_{\ell+1}]$ contains at least one connected component of $\mathcal{G}[\mathcal{S}^i_\ell]$. Hence, $N^i_{\ell+1} \leq N^i_\ell$, which also means that $M_{\ell+1} \leq M_\ell$.

For part (b), consider a class i such that $N^i_\ell \geq 2$ and let $\mathcal{C}_1$ be a connected component of $\mathcal{G}[\mathcal{S}^i_\ell]$. We say component $\mathcal{C}_1$ is *good* if for at least one connector path p of $\mathcal{C}_1$, all internal nodes of p — one or two nodes depending on whether p is short or long — join class i. Note that if $\mathcal{C}_1$ is good, then it gets connected to another component of $\mathcal{G}[\mathcal{S}^i_\ell]$. In order to prove the lemma, we first show that with probability at least $1/3$, at least $\frac{M_\ell}{12}$ connected components (summed up over all classes) are good. This is achieved using a simple accounting method by considering the number of remaining connector paths as the budget.

We know that with probability at least $1/3$, initially we have a budget of at least $M_\ell \cdot t/4$. We show that the greedy algorithm spends this budget in a manner that at the end, we get $M_{\ell+1} \leq \frac{23}{24} M_\ell$.

In each step of each of stages I or II, if respectively a type-1 node or a type-3 nodes and some associated type-2 nodes join a class, then at least Δ components become good where Δ is defined as explained in the algorithm description. We show that in that case, we remove at most $3\Delta t$ connector paths in the related bookkeeping part. Thus, in the accounting

argument, we get that at most $3\Delta \cdot t$ amount of budget is spent and Δ components become good. Hence, in total over all steps, at least $\frac{M_\ell}{12}$ components become good.

Let us first check the case of short connector paths, which is performed in stage I. Let v be the new type-1 node under consideration in this step and suppose that the related $\Delta \geq 2$, and node v joins class i^*. For class i^*, we remove all paths of all connected components of i^* that have v on their short connector paths. This includes Δ such connected components, and t connector paths for each such component. Thus, in total we remove at most $\Delta \cdot t$ connector paths of components of class i^*. For each class $i \neq i^*$, we remove at most Δ connector paths. This is because v can be on short connector paths of at most Δ components, at most once for each such component. These are respectively because of definition of Δ and due to internally vertex-disjointedness of connector paths of each component. There are less than t classes other than i^*, so in total over all classes other than i^*, we remove at most $\Delta \cdot t$ connector paths. Therefore, we can conclude that the total amount of decrease in budget is at most $2\Delta \cdot t$.

Now we check the case of long connector paths, performed in stage II. Suppose that in this step, we are working on a type-3 new node u, it has $\Delta \geq 1$, and we assign node u and associated type-2 new nodes $v_1, \ldots, v_\Delta$ to class i^*. It follows from Proposition 4.3.3 that nodes $v_1, \ldots, v_\Delta$ are not on long connector paths of components of class i^* other than the Δ components which have long paths through u. Thus, any connector path of class i^* that goes through any of v_1 to v_Δ also goes through u. For each component of class i^* that has a long connector path through u, we remove all the connector paths. By definition of Δ, there are Δ such components and from each such component, we remove at most t paths. Hence, the number of such connector paths that are removed is at most $\Delta \cdot t$. On the other hand, for each class $i \neq i^*$, we remove at most 2Δ connector paths. This is because by definition of Δ, removing just node u removes at most Δ long paths from each class. Moreover, because of Proposition 9.4.2 and internally vertex-disjointedness of connector paths of each component, removing each type-2 node v_j (where $j \in \{1, 2, \ldots, \Delta\}$) removes at most one long connector path of one connected component of class $i \neq i^*$. Over all classes $i \neq i^*$, in total we remove at most $2\Delta \cdot t$ connector paths. Hence, when summed up with removed connector paths related to class i^*, we get that the total amount of decrease in the budget is at most $3\Delta \cdot t$.

Now we know that with probability at least $1/3$, at least $\frac{M_\ell}{12}$ connected components (summed up over all classes) are good. Recall that each good component gets merged with at least one other component of its class. Thus, $\Pr\left[M_{\ell+1} \leq \frac{23}{24} \cdot M_\ell \,|\, \mathcal{S}_\ell = \mathcal{T}\right] \geq 1/3$. □

4.3.4 Wrap Up

We are now ready to put the parts of the construction and its analysis together and prove Theorem 4.1.1 by showing that the above construction provides a fractional CDS packing of size $\Omega(k/\log n + 1)$.

Theorem 4.1.1 *Every k-vertex-connected n-vertex graph G has a fractional CDS packing of size $\Omega\big(\frac{k}{\log n} + 1\big)$.*

Proof of Theorem 4.1.1. We first argue that the above construction provides an integral CDS packing of size $t = \delta \cdot kq^2$ in the virtual graph $\mathcal{G}$. We then explain how this can be easily turned into a fractional CDS packing of size $\Omega(kq^2/\log n)$ in the original graph G.

By the description of the construction, the constructed t classes are vertex-disjoint. By Lemma 4.3.2, each of these classes is a dominating set of $\mathcal{G}$, with high probability. What remains is to prove that each of the classes is also a connected subgraph of $\mathcal{G}$, w.h.p.

This connectivity property follows from Lemma 4.3.5, as follows: In the beginning of the class assignments for connectivity, we have $M_\ell \leq n$ for $\ell = L/2 + 1$. Call a layer ℓ *good* if $M_{\ell+1} \leq \frac{5}{6} M_\ell$. Note that by part (b) of Lemma 4.3.5, we know that each layer is good with probability at least $1/3$. Note that M_ℓ is a non-increasing function of ℓ because we always have $M_{\ell+1} \leq M_\ell$, as shown by part (a) of Lemma 4.3.5. Hence, if we have at least $L' = \log_{24/23} n + 1$ good layers $\ell \in [L/2 + 1, L/2]$, then we can conclude that $M_L \leq M_{L/2+1} \cdot (23/24)^{L'} < n \cdot \frac{1}{n} = 1$. Since M_L is an integer, this would imply that $M_L = 0$, which would mean that all classes are connected. Now since each layer is good with probability at least $1/3$, among layers $\ell \in [L/2+1, L/2]$, we expect at least $L/6$ good layers. As we have independence between different layers, by a simple application of the Chernoff bound, we see that with high probability, we have at least L' good layers. This proves connectivity of all classes. It thus also complete the proof showing that the t constructed classes are vertex-disjoint CDS of the virtual graph $\mathcal{G}$.

Now we transform this integral t-size CDS packing of $\mathcal{G}$, for $t = \delta \cdot kq^2$, to a fractional CDS packing of G. This transformation works as follows. Each CDS $S_{\mathcal{G}}$ of $\mathcal{G}$ is turned into a CDS S_G of $\mathcal{G}$ simply by adding each real node $v \in G$ to S_G if and only if there is a copy $(v, i) \in \mathcal{G}$ of v that is in $S_{\mathcal{G}}$. That is, $S_G = \Psi(S_G)$, which means S_G is the projection of $S_{\mathcal{G}}$ to the original graph G. By Proposition 4.3.1, we get that S_G is a CDS of G.

Since the constructed CDSs were vertex-disjoint in $\mathcal{G}$, and because each real vertex $v \in G$ has exactly $\Theta(\log n)$ virtual copies, each of which can be in one of the CDSs in our integral CDS packing of $\mathcal{G}$, we get that in the projection of the CDSs on G, each vertex is in at most $\Theta(\log n)$ different CDSs. Hence, by giving a weight $1/\Theta(\log n)$ to each of these CDSs, we get a fractional CDS packing of G. Clearly, this reduces the size of CDS packing by exactly the same factor $1/\Theta(\log n)$. That is, we get a fractional CDS packing with size $t/\Theta(\log n) = \Omega(kq^2/\log n)$ of G. By setting $p = 1$ and thus $q = 1 - (1-p)^{1/(3L)} = 1$, this becomes a CDS packing of size $\Omega(k/\log n)$ of G, hence proving Theorem 4.1.1. □

4.4 Integral CDS Packing

We now prove our integral CDS packing result:

Theorem 4.1.2 *Every k-vertex-connected n-vertex graph graph G has an integral CDS packing of size $\Omega\big(\frac{k}{\log^2 n} + 1\big)$.*

To achieve this CDS packing, we use the general construction style of the CDS packing of the previous section. Here, we explain the key changes: since in a CDS packing, each

node can join only one CDS, we cannot use the layering style of the previous section, which uses $\Theta(\log n)$ copies of G and where each node can join $O(\log n)$ CDSs. Instead, we use *random* layering: each node chooses a random *layer number* in $\{1, \dots, L\}$ and a random *type number* in $\{1, 2, 3\}$. The construction is again recursive, with first assigning nodes of layers 1 to $L/2$ randomly to one of t random classes. This suffices to give domination. Here, we set the number of classes to be $t = \delta \frac{k}{\log^2 n}$. After that, for each $\ell \in [L/2, 3L/4 - 1]$, we assign class numbers of nodes of layer $\ell + 1$ based on the configuration of classes in layers 1 to ℓ, using the same greedy algorithm as in Section 4.3.3. The vertices of the remaining layers are assigned to CDSs randomly. Next, we re-define the connector paths, incorporating the random layering.

Connector Paths for Integral CDS packing: Let V_ℓ^i be the set of all nodes of layers 1 to ℓ in class i. Consider a component $\mathcal{C}$ of $G[V_\ell^i]$. Define potential connector paths on G as in Section 4.3.3 (conditions (A) to (C)). Then, for each potential connector path on G, this path is called a *connector path* if its internal nodes are in layer $\ell + 1$ and the types of its internal nodes satisfy rules (D) and (E) in Section 4.3.3.

The key technical change compared to the CDS packing of Section 4.3, appears in obtaining a Connector Abundance Lemma, which we present Lemma 4.4.1.

Lemma 4.4.1 (Connector Abundance Lemma). *For each class i and layer $\ell \in [L/2, 3L/4 - 1]$ such that $N_\ell^i \geq 2$, for each connected component $\mathcal{C}$ of $G[V_\ell^i]$, with probability at least $1/2$, $\mathcal{C}$ has at least $\Omega\big(\frac{k}{\log^2 n}\big)$ internally vertex-disjoint connector paths, with independence between different layers $\ell \leq L/2$.*

To prove Lemma 4.4.1, we use the sampling result of Theorem 5.1.1, which we prove in the next chapter. Roughly speaking, for each layer $\ell \leq L/2$, from Lemma 4.4.1 we get that the graph induced by layers $\ell + 1$ to L has vertex connectivity $\Omega(k)$. Then, we get that each component $\mathcal{C}$ has at least $\Omega(\frac{k}{\log^2 n})$ internally vertex-disjoint connector paths (with internal nodes of right type and layer $\ell + 1$).

Proof. Let W_ℓ^* be the set of all nodes with a layer number in $\{\ell + 1, \dots, L\}$. Since the probability of each node to be in W_ℓ^* is at least $1/4$ (because $\ell \leq 3L/4$), Theorem 5.1.1 shows that, w.h.p, the vertex-connectivity of $G[W_\ell^*]$ is $\Omega(k)$. It is easy to see that therefore, the vertex-connectivity of $G[W_\ell^* \cup V_\ell^i]$ is also $\Omega(k)$. Thus, for each component $\mathcal{C}$ of $G[V_\ell^i]$, we can follow the first part of the proof of Lemma 4.3.4, this time using Menger's theorem on $G[W_\ell^* \cup V_\ell^i]$, and find $\Omega(k)$ internally vertex-disjoint potential connector paths in graph $G[W_\ell^* \cup V_\ell^i]$. It is clear that the internal nodes of these potential connector paths are not in V_ℓ^i, which means they are in W_ℓ^*. For each potential connector path, for each of its internal nodes, given that this node is in W_ℓ^*, the probability that the node is in layer $\ell + 1$ and has the type which satisfies rules (A) and (B) of Section 4.3.3 is at least $\Theta(1/L) = \Theta(1/\log n)$. Hence, the probability of each of these potential connector paths being a connector path is at least $\Theta(1/\log^2 n)$. From internally vertex-disjointedness of the potential connector paths,

and since there are $\Omega(k)$ of them, it follows that with probability at least $1/2$, component $\mathcal{C}$ has at least $\Omega(\frac{k}{\log^2 n})$ internally vertex-disjoint connector paths. □

4.5 Bad Graphs with No Large Fractional CDS Packing

In this section, we prove Theorem 4.1.3, which shows that in some graphs, the maximum CDS packing size is a $\Theta(\log n)$ factor smaller than the vertex connectivity.

Theorem 4.1.3 *For any sufficiently large n, and any $k \in [1, n/4]$, there exist n-vertex graphs with vertex connectivity k where the maximum CDS packing (or partition) size is $O\big(\frac{k}{\log n}+1\big)$.*

First, in Lemma 4.5.1 we present a graph H with vertex connectivity k, size between 2^k and 4^k, and $K'_{CDS}(\mathsf{H}) < 2$. This lemma proves the theorem for $k = O(\log n)$. To prove the theorem for the case of larger vertex-connectivity compared to n, in Lemma 4.5.2, we look at randomly chosen sub-graphs of H and apply the probabilistic method [AS04].

Lemma 4.5.1. *For any k, there exists an n-node graph H with vertex connectivity k and $n \in [2^k, 4^k]$ such that $K'_{CDS}(\mathsf{H}) < 2$.*

Proof. We obtain graph H by simple modifications to the graph presented by Sanders et al. [SET03] for proving an $\Omega(\log n)$ network coding gap in the model where network is directed and each node can send *distinct* unit-size messages to its different outgoing neighbors.

The graph H has two layers. The first layer is a clique of $2k$ nodes. The second layer has $\binom{2k}{k}$ nodes, one for each subset of size k of the nodes of the first layer. Each second layer node is connected to the k first-layer nodes of the corresponding subset. Note that the total number of nodes is $\binom{2k}{k} + 2k \in [2^k, 4^k]$. Let $\mathcal{A}$ and $\mathcal{B}$ denote the set of nodes in the first and second layer, respectively.

First, we show that H has vertex connectivity k. Since the degree of each second layer node is exactly k, it is clear that the vertex connectivity of H is at most k. To prove that the vertex connectivity of H is at least k, let u and v be two arbitrary nodes of H. We show that there are at least k internally vertex disjoint paths between u and v. If u and v are both in $\mathcal{A}$, then there is one direct edge between v and u and there are $2k-2$ paths of length 2 between them. If exactly one of v and u is in $\mathcal{A}$, say $u \in \mathcal{A}$ and $v \in \mathcal{B}$, then u is directly connected to k neighbors of v. Otherwise, if both u and v are in $\mathcal{B}$, then let p be the size of the intersection of the neighbors of v and u. Note that these neighbors are all in $\mathcal{A}$. It is clear that u and v have exactly p paths of length 2 between themselves and $k-p$ paths of lengths 3, and that these paths are internally vertex disjoint.

To see that $K'_{CDS}(H) < 2$, first note that each CDS τ must include at least $k+1$ nodes of $\mathcal{A}$. This is because, otherwise, there are at least k nodes of $\mathcal{A}$ that are not included in τ and thus, there is a node in $\mathcal{B}$—corresponding to a subset of size k of these uncovered nodes

of $\mathcal{A}$—which is not dominated by τ. Thus we have,

$$\sum_{v\in\mathcal{A}}\sum_{\substack{\tau\in CDS(\mathsf{H})\\ s.t.\ v\in\tau}} x_\tau \geq (k+1)\cdot\sum_{\tau\in CDS(\mathsf{H})} x_\tau.$$

On the other hand we have,

$$\sum_{v\in\mathcal{A}}\sum_{\substack{\tau\in CDS(\mathsf{H})\\ s.t.\ v\in\tau}} x_\tau \leq \sum_{v\in\mathcal{A}} 1 = |\mathcal{A}| = 2k,$$

and thus we can conclude that $\sum_{\tau\in CDS(\mathsf{H})} x_\tau \leq \frac{2k}{k+1} < 2$. Since this holds for any CDS-Packing of H, we get $K'_{CDS}(H) < 2$. □

Note that in the above construction, we have $K_{CDS}(\mathsf{H}) = 1$ as H is connected and $K_{CDS}(\mathsf{H})$ has to be an integer.

Lemma 4.5.2. *For each large enough k and $\eta \in [4k, 2^k]$, there exists a sub-graph $H' \subseteq \mathsf{H}$ that has η nodes and vertex connectivity k but $K'_{CDS}(H') = O(\frac{k}{\log\eta})$.*

Proof. Pick an arbitrary $k \geq 64$, fix an $\eta \in [4k, 2^k]$ and let $\beta = \frac{\log\eta}{8}$. Consider a random subset $V_z \subseteq V$, where V_z includes all nodes of $\mathcal{A}$ and for each node $u \in \mathcal{B}$, u is independently included in V_z with probability p, where

$$p = \frac{65\beta^2}{\binom{2k-\beta}{k}}.$$

We now look at the sub-graph H_z of H induced on V_z. With the same argument as for H, we get that for any such V_z, the graph H_z has vertex connectivity exactly k. We show that (a) with probability at least $\frac{1}{2}$, V_z is such that $K'_{CDS}(H_z) < \frac{2k}{\beta} = O(\frac{k}{\log\eta})$, and (b) with probability at least $\frac{3}{4}$, we have $|V_z| \leq \eta$. A union bound then completes the proof.

Property (a) We first show that with probability at least $\frac{1}{2}$, V_z is such that there does not exist a subset of size β of the nodes of $\mathcal{A}$ that dominates V_z. For each subset $W \subset \mathcal{A}$ such that $|W| = \beta$, there are $\binom{2k-\beta}{k}$ nodes in $\mathcal{B}$ which are not dominated by W. Thus, for W to dominate V_z, none of these second layer nodes should be included in V_z. The probability for this to happen is

$$(1-p)^{\binom{2k-\beta}{k}} \leq e^{-65\beta^2}$$

There are $\binom{2k}{\beta}$ possibilities for set W. Hence, using a union bound, the probability that there exists such a set W that dominates V_z is at most

$$\begin{aligned} e^{-65\beta^2}\binom{2k}{\beta} \leq e^{-65\beta^2}\cdot(\frac{2ek}{\beta})^\beta &\overset{(\dagger)}{<} e^{-65\beta^2}\cdot(\eta^2)^\beta \\ &= e^{-65\beta^2+64\beta^2} \leq \frac{1}{2}, \end{aligned}$$

where Inequality (†) follows since $64 \leq k \leq \frac{\eta}{4}$, which gives $2ek < k^2 < \eta^2$.

Thus, with probability at least $\frac{1}{2}$, V_z is such that each CDS of H_z includes at least $\beta + 1$ nodes of $\mathcal{A}$. From this, similar to the last part of the proof of Lemma 4.5.1, we have that, $\sum_{\tau \in CDS(\mathsf{H})} x_\tau \leq \frac{2k}{\beta+1} < \frac{2k}{\beta}$. Since this holds for any packing of H_z, we get that with probability at least $\frac{1}{2}$, V_z is such that $K'_{CDS}(H_z) < \frac{2k}{\beta}$.

Property (b) Note that $\mathbb{E}[|V_z|] = 2k + p \cdot \binom{2k}{k}$. Substituting $p = \frac{65\beta^2}{\binom{2k-\beta}{k}}$ and noting that $\beta \leq \frac{k}{2}$, we get

$$\begin{aligned}
\mathbb{E}[|V_z|] - 2k &= p \cdot \binom{2k}{k} = 65\beta^2 \cdot \frac{\binom{2k}{k}}{\binom{2k-\beta}{k}} \\
&= 65\beta^2 \cdot \frac{2k}{2k-\beta} \cdot \frac{2k-1}{2k-\beta-1} \cdots \frac{k+1}{k-\beta+1} \\
&\leq 65\beta^2 \cdot (1 + \frac{2\beta}{k})^k \\
&\leq \frac{65 \log^2 \eta}{64} \cdot \eta^{\frac{1}{4}} \leq \frac{\eta}{4}.
\end{aligned}$$

As the second-layer nodes are picked independently, for η sufficiently large, we can apply a Chernoff bound to get $\Pr[|V_z| - 2k > \frac{\eta}{2}] \leq \frac{1}{4}$. Since $2k \leq \frac{\eta}{2}$, we then obtain $\Pr[|V_z| > \eta] \leq \frac{1}{4}$. If desired, we can adjust the number of nodes to exactly η by adding enough nodes in the second layer which are each connected to all nodes of the first layer. □

Proof of Theorem 4.1.3. The theorem follows directly from Lemma 4.5.1 and Lemma 4.5.2, which prove it for when $k = O(\log n)$ and $k = \Omega(\log n)$, respectively. In particular, Lemma 4.5.1 presents a k-vertex connected graph H with n vertices where $n \in [2^k, 4^k]$, and $K'_{CDS}(\mathsf{H}) < 2$. This lemma proves the theorem for $k = O(\log n)$. Lemma 4.5.2 presents a k-vertex connected graph H', for large enough k, with n vertices where $n \in [4k, 2^k]$, such that $K'_{CDS}(H') = O(\frac{k}{\log n})$. This lemma proves the theorem for $k = \Omega(\log n)$. □

4.6 Vertex Connectivity Decomposition and Throughput in Network Information Dissemination

In this section, we explain how the size of vertex connectivity decompositions—particularly CDS packings—is tightly coupled with the throughput of network information dissemination procedures. In particular, we show that the size of maximum fractional CDS packing and the maximum achievable throughput in a V-CONGEST model network are equal, up to constant factors.

Theorem 4.1.5 *Consider any network in the* V-CONGEST *model. Given a fractional CDS*

packing with size K, we can construct a store-and-forward broadcast algorithm with throughput $\Omega(K)$ messages per round. Conversely, given a store-and-forward broadcast algorithm with throughput K messages per round, we can construct a fractional CDS packing of size K.

Proof. **Fractional CDS Packing $\rightarrow$ Broadcast Algorithm** First consider a CDS τ and suppose that the graph induced by τ has diameter D_τ. Using τ, we can perform p broadcasts (or multicast or unicasts) in time $O(p + D_\tau)$. This can be seen as follows: Since τ is a dominating set, we can deliver each message to a node of τ in at most p rounds. Because τ is connected, $O(p + D_\tau)$ rounds are enough to broadcast the messages to all nodes in τ. Finally, because τ is a dominating set, at most p more rounds are enough to deliver the messages to all the desired destination nodes. Hence, a CDS structure allows for performing broadcasts with an (amortized) rate of $\Omega(1)$ messages per round. In other words, a CDS can be viewed as a communication backbone with throughout $\Omega(1)$ messages per round.

Consequently, K vertex-disjoint CDS sets form a communication backbone with throughput of $\Omega(K)$ messages per round. Intuitively, we can use those K vertex-disjoint sets in parallel with each other and get throughput of $\Omega(1)$ message per round from each of them. For a more formal description, consider p broadcasts such that no more than q broadcasts have the same source node. We first deliver each messages to a randomly and uniformly chosen CDS set. This can be done in time at most q. With high probability, the number of messages in each CDS is $O(\frac{p}{K} + \log n)$ and thus, we can simultaneously broadcast messages in time $O(\frac{p}{K} + \log n + D_{\max})$ where $D_{\max}$ is the maximum diameter of the CDSs. Thus, the total time for completing all the broadcasts is $O(q + \frac{p}{K} + \log n + D_{\max})$. That is, we can perform the broadcasts with a rate (throughput) of $\Omega(\min\{K, p/q\})$. Note that since each source can send only one packet per round, if $q \leq K$, then the maximum achievable throughout with any algorithm including network coding approaches is at most q packets per round. In other words, in that case, the bottleneck is not the communication protocol but rather the sources of the messages. As long as no node is the source of more than $\Theta(p/K)$ messages, K vertex-disjoint CDS sets form a communication backbone with throughput $\Omega(K)$.

Similarly one can see that a fractional CDS packing with size K provides a backbone with a throughput of $\Omega(K)$ messages per round. The only change with respect to above description is that now each node v spends a x_τ-fraction of its time for sending the messages assigned to CDS τ for every τ such that $v \in \tau$. Further, messages are assigned to each CDS τ with probability proportional to x_τ.

We remark here that even though this scheme provides a backbone with throughput $\Omega(K)$, if the weights x_τ are too small, the outlined time sharing might impose a considerably large additive term on the overall time for completing the broadcasts. This would not impact the throughput, but it would include a high latency. In fact since the number of potential CDS sets can be exponential, the time sharing might lead to exponentially large additive terms. Note that in the fractional CDS packing we present in Theorem 4.1.1, each CDS has weight at least $\Omega(1/\log n)$ and thus using the partition also leads to an asymptotically

optimal throughput for a relatively small number of broadcast messages.

Broadcast Algorithm → Fractional CDS Packing Let us now argue that a broadcast algorithm with throughput K also leads to a fractional CDS packing of size K. Suppose that there exists a a store-and-forward algorithm that broadcasts p messages originating from potentially different sources, for a desirably large integer p, in $T \leq \frac{p}{K}$ rounds. For each message σ that is being broadcast, define set $S(\sigma)$ to be the set of nodes that send σ in some round of the algorithm. Clearly $S(\sigma)$ induces a connected sub-graph and because every node needs to receive the message $S(\sigma)$ also is a dominating set. For each node v and message σ, let $N_\sigma(v)$ be the number of rounds in which node v sends message σ and let $y_\sigma(v) = \frac{N_\sigma(v)}{T}$. Moreover, for each CDS τ such that $v \in \tau$, let

$$z_\tau(v) = \sum_{\substack{\sigma \\ S(\sigma)=\tau \wedge v \in \tau}} y_\tau(v).$$

Finally, let $x_\tau = \min_{v \in \tau}\{z_\tau(v)\}$. Given these parameters, first notice that for each node v, we have

$$\sum_{\substack{\tau \\ v \in \tau}} x_\tau \leq \sum_{\substack{\tau \\ v \in \tau}} z_\tau(v) = \sum_{\substack{\tau \\ v \in \tau}} \sum_{\substack{\sigma \\ S(\sigma)=\tau}} y_\sigma(v) = \sum_\sigma \frac{N_\sigma(v)}{T} \overset{(\dagger)}{\leq} 1.$$

Here, Inequality (†) is because in each round, node v can send at most one message and thus, $\sum_\sigma N_\sigma(v) \leq T$. On the other hand, we show that $\sum_\tau x_\tau \geq \frac{p}{T} = \Omega(K)$. For this purpose, consider a CDS τ and let u^* be a node such that $z_\tau(u^*) = x_\tau$. Since each message σ such that $S(\sigma) = \tau$ is sent at least once by u^*, we have

$$\begin{aligned} \sum_{\substack{\sigma \\ S(\sigma)=\tau}} 1 \leq \sum_{\substack{\sigma \\ S(\sigma)=\tau}} N_\sigma(u^*) = & \sum\nolimits_{\substack{\sigma \\ S(\sigma)=\tau}} y_\sigma(u^*) \cdot T \\ = & \; z_\tau(u^*) \cdot T = x_\tau \cdot T \end{aligned}$$

Moreover, we have that

$$p = \sum_\sigma 1 = \sum_\tau \sum_{\substack{\sigma \\ S(\sigma)=\tau}} 1 \leq \sum_\tau x_\tau \cdot T$$

Thus, $\sum_\tau x_\tau \geq \frac{p}{T}$. Since $T \leq \frac{p}{K}$, we get that $\sum_\tau x_\tau \geq K$. □

Remark We remark that in the general formulation of CDS packings, each node might participate in arbitrarily many CDSs. In fact, this number can be even exponential in the number of nodes n. This would make CDS packing inefficient from a practical point of view if the number of messages is small compared to the number of CDSs used. Fortunately, in our construction (cf., Theorem 4.1.1), each node participates in only $O(\log n)$ CDSs, which makes the CDS packing efficient even for a small number of messages.

Chapter 5

Vertex Connectivity Under Random Sampling

5.1 Introduction & Related Work

Consider a random process where given a base graph G, each edge or node of G is sampled randomly with some probability p, i.e., removed with probability $1-p$. Given such a random graph process, it is interesting to see how various global connectivity properties of the graph induced by the sampled edges or nodes change as a function of the sampling probability p. If G is the complete n-node graph, sampling each edge independently with probability p results in the classic Erdős-Rényi random graph $G_{n,p}$, for which exact thresholds for the formation of a giant component, global connectivity, and many other properties have been studied (e.g., in [Bol98]). Thresholds for the formation of a giant component are further studied more generally in percolation theory [BR06]—mostly for graphs G defined by some regular or random lattice. In the context of percolation theory, edge sampling is called bond percolation whereas vertex sampling is referred to as site percolation.

We are interested in how the vertex connectivity of a general graph G changes under uniform random vertex or edge sampling. This is in part because we use random sampling methods in the constructions of our vertex connectivity decomposition results, particularly in Section 4.4. But it is also motivated by understanding the reliability of a network when vertices of it fail randomly, more concretely, characterizing which networks are likely to get disconnected when each node in the network fails with some given probability $q = 1 - p$.

For edge connectivity and edge sampling, the analogous question was resolved already two decades ago. Karger's seminal result [Kar94b] showed that for any λ-edge-connected graph with n vertices, sampling edges independently at random with probability $p = \Omega(\log(n)/\lambda)$ results in an $\Omega(\lambda p)$-edge-connected subgraph, with high probability[1]. This was a strong extension of the earlier result by Lomonosov and Polesskii [LP72], which stated that sampling

[1]Recall from Chapter 2 that we use the phrase '*with high probability*' (*w.h.p.*) to indicate that some event has a probability of at least $1 - n^{-\Theta(1)}$.

each edge with probability $\Theta(\log(n)/\lambda)$ leads to a connected subgraph, w.h.p. These sampling results and their extensions were cornerstone tools for addressing various important problems such as various min-cut problems [Kar94a, Kar94b], constructing cut-preserving graph sparsifiers [BK96, ST04], max-flow problems [Kar94a, KL98], and network reliability estimations [Kar95].

As in the case of edge connectivity, studying the vertex connectivity of the subgraph obtained by independently sampling vertices or edges of a k-vertex-connected graph is of fundamental interest. However, the vertex connectivity case has been recognized as being much harder and the problem remained widely open. Let us briefly overview the challenge. We briefly explain why tools with a similar flavor to the ones used for edge connectivity do not take us far in the vertex connectivity case. The key to most results about edge sampling is the so-called *cut counting* argument introduced in [Kar93], where it is shown that in a graph of edge connectivity λ, the number of cuts of size at most $\alpha\lambda$ is at most $O(n^{2\alpha})$. Combined with a standard Chernoff argument and a union bound over all cuts, this shows that when independently sampling each edge with probability $p = \Omega(\log(n)/\lambda)$, it holds w.h.p. for the subgraph induced by the sampled edges, that the size of each cut does not deviate from its expectation by more than a constant factor [Kar94b]. Hence, in particular, the edge connectivity of the sampled subgraph is $\Omega(\lambda p)$, w.h.p. Unfortunately, the same approach cannot work for vertex connectivity under vertex or edge sampling, because in graphs with vertex connectivity k, even the number of minimum vertex cuts can be as large as $\Theta(2^k(n/k)^2)$ [Kan90].

Our Result We propose a new method which overcomes this challenge and leads to tight bounds for the threshold probability for vertex connectivity and the remaining connectivity under both vertex and edge sampling. Formally, we prove the following two theorems:

Theorem 5.1.1. *Let $G = (V, E)$ be an arbitrary k-vertex-connected n-node graph, and let S be a randomly sampled subset of V where each node $v \in V$ is included in S independently with probability $p \geq \alpha\sqrt{\log(n)/k}$, for a sufficiently large constant α. Then the subgraph $G[S]$ of G induced by S has vertex connectivity $\Omega(kp^2)$, with probability $1 - e^{-\Omega(kp^2)}$.*

Theorem 5.1.2. *Let $G = (V, E)$ be a k-vertex-connected n-node graph, and let E' be a randomly sampled subset of E where each edge $e \in E$ is included in E' independently with probability $p \geq \alpha \log(n)/k$, for a sufficiently large constant α. Then the subgraph $G' = (V, E')$ of G has vertex connectivity $\Omega(kp)$, with probability $1 - e^{-\Omega(kp)}$.*

The bounds in both of these theorem statements are tight. In Section 5.5, we discuss simple graphs that show the optimality of the bounds in Theorems 5.1.1 and 5.1.2.

Organization of the next sections: Our method for proving Theorems 5.1.1 and 5.1.2 has two parts. The main part studies the sampling process and proves tight bounds on the probability of the graph becoming disconnected. In the main regime of interest, these are

probabilities that are exponentially close to 0 and for our result to be useful, we need to even prove the tight exponents in how fast this probability vanishes to 0. We thus refer to this part as probability concentration for connectivity, which is presented in Section 5.2. The second part then uses this high probability concentration for remaining simply connected and transforms it into lower bounds for the remaining connectivity. This part is presented in Section 5.3. In Section 5.4, present simple graphs which show the optimality of the bounds claimed in Theorems 5.1.1 and 5.1.2. We conclude the chapter in Section 5.5 by presenting a discussion on open problems and furture work.

5.2 Probability Concentration for Connectivity

The most important component in our approach is proving tight bounds (on the exponent of) the probability of the graph becoming disconnected when we sample each vertex with probability p. Formally we prove the following statement.

Theorem 5.2.1. *Let $G = (V, E)$ be a k-vertex-connected n-node graph. For an arbitrary $0 < \delta < 1$, let S be a randomly sampled subset of V such that each $v \in V$ is included in S independently with probability $p \geq \beta\sqrt{\log(n/\delta)/k}$, for a sufficiently large constant β (independent of n, k and δ). Then, with probability at least $1 - e^{-\Omega(kp^2)} \geq 1 - \delta$, the set S is a connected dominating set of G (and thus graph $G[S]$ is connected).*

Remark Before diving into the proof, we note that the power of Theorem 5.2.1 comes from the ability to plug exponentially small values into the error probability δ.

5.2.1 Warm Up

Instead of directly diving into the full proof of Theorem 5.2.1, we start with a much simpler argument which shows that that the remaining the graph is connected with probability at least $1 - \Omega(kp^2/\log n)$. In the next subsection, we explain how to tighten the bound to $1 - \Omega(kp^2)$.

Lemma 5.2.2. *Let G be a k-vertex-connected graph with n vertices. Suppose we sample each vertex independently with probability $p \geq \frac{c \log n}{\sqrt{k}}$ for a sufficiently large constant $c > 0$. The sampled vertices induce a connected graph, with probability at least $1 - e^{-\Omega(kp^2/\log n)}$.*

Proof Idea We consider the vertex sampling as a slowed-down gradual process that happens in phases, each being a sampling process with a smaller probability, and we analyze the set of sampled vertices as they are gradually added throughout these phases. More concretely, we show that after a few phases, the sampled vertices form a dominating set, with very high probability. From that point onward, we focus on the number of connected components of the sampled vertices and show that each new phase of sampling is likely to reduce

the number of connected components by a constant factor. Hence, after $O(\log n)$ phases, the subgraph induced by the sampled nodes becomes connected, with high probability.

Proof Sketch. We view the sampling in $L = \Theta(\log n)$ layers, where now in each layer, each vertex that is not sampled in the previous layers gets sampled with probability q, such that we have $1 - (1-q)^L = p$. It is easy to see that $q = \Omega(p/L) = \Omega(p/\log n)$. One can see that the sampled vertices of the first $L/2$ layers are a dominating set of the graph, with probability at least $1 - e^{-\Omega(kp)}$. This is because, in a k-vertex-connected graph, each vertex has at least k neighbors and the probability of each vertex to be sampled in one of the first $L/2$ layers is at least $\Omega(p)$. Hence, the probability of each node not to have a sampled neighbor in these layers is at most $(1-\Omega(p))^k = e^{-\Omega(kp)}$. Taking a union bound over all nodes, we get that with probability at least $1 - e^{-\Omega(kp)} \geq 1 - e^{-\Omega(kp^2/\log n)}$, all nodes are dominated.

Consider a layer $\ell > L/2$ and consider the subgraph induced by the sampled vertices of layers 1 to ℓ. From Menger's theorem, we know that each connected component of this subgraph is connected to other components via at least k vertex-disjoint paths (in total), that are made of non-sampled internal nodes. Thanks to the domination provided by the first $L/2$ layers, we get a stronger statement: each such component is connected to other components via at least k vertex-disjoint paths, each *with at most* 2 *non-sampled internal nodes* (thus the length of each of these paths is at most 3). We call these *connector paths*. With the arrival of the sampled vertices of layer $\ell + 1$—which are sampled with probability $\Theta(p/\log n)$—each of these short connector paths has (both of) its middle nodes sampled with probability at least $q^2 = \Theta((p/\log n)^2)$. Hence, the probability that at least one of these paths has its middle nodes sampled is at least $1 - (1-q^2)^k = 1 - e^{-\Omega(kp^2/\log^2 n)}$. Thus, the probability that this component is bad meaning that is does not get connected to another component is at most $e^{-\Omega(kp^2/\log^2 n)}$. Call this layer bad if more than half of its components are bad. Using Markov's inequality, we get that the probability that the layer is bad is at most $e^{-\Omega(kp^2/\log^2 n)}$. Now, we overall have $L/2 = \Theta(\log n)$ layers after domination and each is bad with probability at most $e^{-\Omega(kp^2/\log^2 n)}$. Therefore, the probability that more than $L/4$ of these layers are bad is at most $\binom{L/2}{L/4} e^{-\Omega(kp^2/\log^2 n)\cdot L/4} \leq e^{-\Omega(kp^2/\log n)}$. That is, with probability at least $1 - e^{-\Omega(kp^2/\log n)}$, at least $L/4$ of the layers are good. Since in each good layer the number of connected components goes down by a constant factor, the latter would mean that after $L/4 = \Theta(\log n)$ good layers, the sampled vertices induce a connected graph. Hence, taking a union bound over the $e^{-\Omega(kp)}$ failure probability of domination and the $e^{-\Omega(kp^2/\log n)}$ failure probability of connectivity, we get that the sampled vertices induce a connected graph with probability at least $1 - e^{-\Omega(kp^2/\log n)}$. □

5.2.2 Intuition and Challenges in Proving Theorem 5.2.1

In this subsection, we provide some intuition for the proof of Theorem 5.2.1.

One of the contributions of this theorem is that we can plug very small values of the error probability δ into it. For explaining the intuition, we just focus on the specific case of

$\delta = n^{-\Theta(1)}$. That is, we explain why a sampling probability of $p = \Theta(\sqrt{\log(n)/k})$ leads to a connected sampled subgraph *w.h.p.* As a comparison, the warm up result we proved in the previous subsection applies only to this case and needed the stronger assumption that $p = \Omega(\log(n)/\sqrt{k})$.

We again look at the sampling process as slowly adding nodes over time. In particular, instead of sampling nodes with probability p at once, one samples nodes over multiple, $T = \Omega(\log n)$, rounds, where in each round nodes are sampled with some smaller probability $q \approx p/T$. This allows to study and analyze the emergence and merging of connected components, as time progresses and more and more nodes are sampled.

Next, let us take a look at a single edge cut, the canonical bad cut consisting of a k-edge matching as will be discussed in Observation 5.4.1. We emphasize that understanding the behavior of *all* cuts simultaneously is the part that makes the problem challenging, but focusing on this single cut should be sufficient for delivering the right intuition about the key new element in our analysis.

In the cut consisting of a k-edge-matching, in any round, both endpoints of an edge will become sampled with probability q^2. Since there are k such edges, the probability that at least one edge gets sampled in a round is bounded by kq^2. Now, in order for a cut to merge w.h.p. in this way over the course of T rounds, we need that $Tkq^2 = kp^2/T = \Omega(\log n)$. Since we assumed $T = \Omega(\log n)$, this results in $p = \Omega(\log(n)/\sqrt{k})$ being a necessary condition. This explains in a very simplified manner why the argument of the previous warm up section does not work for $p = o(\log(n)/\sqrt{k})$.

Here, we refine this layer-by-layer sampling by further exploiting that connectivity evolves gradually. In particular, while the probability of obtaining one complete edge in one round is only q^2, and thus quite small, the number of sampled nodes on each side of the cut grows by roughly kq in each round. Thus, after λ/kq rounds for some $\lambda = \Omega(\log n)$, the number of such nodes is at least λ w.h.p. Each of these nodes intuitively already goes half way in crossing the cut. In particular, with λ such nodes, there is a chance of λq per each of the next rounds to complete such a semi-sampled edge into a fully sampled edge that crosses the cut. This means that after such λ-"semi-connectivity" is achieved, $\log(n)/\lambda q$ further rounds suffice to get an edge crossing the cut to be fully sampled, with high probability. The optimal value for λ is now chosen to balance between the λ/kq rounds to achieve λ-semi-connectivity and the $\log(n)/\lambda q$ additional rounds required to lead to connectivity. This leads to $\lambda = \sqrt{\log(n)/k}$, and results in $T = \Theta\big(\sqrt{\log(n)/k}/q\big)$ rounds and a sampling probability of $p = \Theta\big(\sqrt{\log(n)/k}\big)$ being sufficient for a single cut.

In the above description we focused on a single cut. However, understanding the behavior of all (the exponentially many) cuts together turns out to be significantly more complex. Overall, the main technical challenge is to develop notions, definitions and arguments to prove that semi-connectivity indeed gets established quickly, for all cuts.

5.2.3 Proof of Theorem 5.2.1 via Semi-Connectivity

In this section, we present the formal definition of semi-connectivity, and explain how the proof of Theorem 5.2.1 incorporates the analysis of semi-connectivity. At a high level, the process of sampling consists of three parts for obtaining (i) domination, (ii) λ-semi-connectivity for a $\lambda = \Theta(\sqrt{k \log(n/\delta)})$, and (iii) connectivity. Establishing domination is trivial, and the proof of connectivity after having λ-semi-connectivity follows easily a layer-by-layer analysis, as done in the warm up result. The key challenge is to prove λ-semi-connectivity. Precisely, we show that sampling with probability $\Theta(\lambda/k)$ suffices for increasing the semi-connectivity of a dominating set by an additive term of λ, for $\lambda = \Omega(\log n)$.

We start by fixing some basic notation. We say that a node $u \in V$ is a *neighbor of* $S \subseteq V$ or is *adjacent to* S if u is adjacent to some node $v \in S$ and $u \notin S$. The set of neighbors of S is denoted by ∂S. An edge or path *between two sets* S *and* S' is one with endpoints $u \in S$ and $u' \in S'$.

We formally define the notion of semi-connectivity as follows.

Definition 5.2.3 (λ-Semi-Connected Set)**.** *A node set $S \subseteq V$ is* λ-semi-connected*, for some $\lambda \geq 0$, if for any partition of S into two sets T and $S \setminus T$ with no edges between them, T and $S \setminus T$ have at least λ common neighbors, i.e., $|\partial T \cap \partial(S \setminus T)| \geq \lambda$.*

If a set is $(\lambda+1)$-semi-connected, then it is λ-semi-connected, as well. Also, any connected set is λ-semi-connected for any $\lambda \geq 0$, as the condition in Definition 5.2.3 is vacuously true in this case.

Next we observe that adding a node from $V \setminus S$ to a λ-semi-connected set S does not break semi-connectivity, provided that S is a dominating set.

Claim 5.2.4. *If $S \subseteq V$ is a λ-semi-connected dominating set, then for any node $u \in V \setminus S$, the set $S \cup \{u\}$ is also a λ-semi-connected dominating set.*

Proof. Since S is a dominating set, so is the set $S' = S \cup \{u\}$. Next, we show that S' is λ-semi-connected. Consider any partition of S' into two sets T' and $S' \setminus T'$, such that these sets have no edges between them. We show that T' and $S' \setminus T'$ have at least λ common neighbors. We assume w.l.o.g. that $u \in T'$. We observe that $T' \neq \{u\}$, because S is a dominating set and thus if $T' = \{u\}$ then there would be an edge between T' and $S' \setminus T'$. Thus, the set $T = T' \setminus \{u\}$ is non-empty, and the two sets T and $S \setminus T = S' \setminus T'$ constitute a partition of S. We have that T and $S \setminus T$ have no edges between them, for otherwise the same edge would also connect T' and $S' \setminus T'$. Since S is λ-semi-connected, there are at least λ common neighbors for T and $S \setminus T$. Each of these nodes is also a common neighbor for T' and $S' \setminus T'$, because it cannot be equal to u (otherwise there is an edge between T' and $S' \setminus T'$). Since this holds for any such partition, this implies that S' is λ-semi-connected. □

We now show that, if we start with a set S of nodes that is a λ-semi-connected dominating set, then it suffices to sample the remaining nodes with probability $\Theta(\log(n/\delta)/\lambda)$ to end up with a connected dominating set with probability $1 - \delta$.

Lemma 5.2.5. *Let $S \subseteq V$ be a λ-semi-connected dominating set. Sampling each remaining node $u \in V \setminus S$ with probability $\log_\gamma(n/\delta)/\lambda$, where $\gamma = \frac{2e}{e+1}$, yields a set S' such that $S \cup S'$ is a connected dominating set with probability at least $1 - \delta$.*

Proof. We perform sampling in rounds, where in each round every node that has not been sampled yet is sampled with probability $1/\lambda$. The total number of rounds is $r = \log_\gamma(n/\delta)$, thus the probability for any given node $u \in V \setminus S$ to get sampled in one of those rounds is at most $r/\lambda = \log_\gamma(n/\delta)/\lambda$, as required by the lemma statement. Let S_i, for $0 \leq i \leq r$, denote the set consisting of all nodes sampled in the first i rounds and of all nodes $u \in S$ (so $S_0 = S$). Further, let X_i denote the number of connected components of the induced subgraph $G[S_i]$. We bound $\mathbb{E}[X_i]$ next.

Fix a set S_i and suppose that $G[S_i]$ is disconnected, i.e., $X_i > 1$. Since S is a λ-semi-connected dominating set, S_i is also a λ-semi-connected dominating set, by Claim 5.2.4. Hence, each connected component C of $G[S_i]$ has at least λ common neighbors with other connected components. If any of those common neighbors gets sampled in round $i+1$, then C gets merged with another component. Then the probability of C to get merged in round $i+1$ is at least $1-(1-1/\lambda)^\lambda \geq 1-1/e$. Since the drop $X_i - X_{i+1}$ in the number of connected components in round $i+1$ is at least half the total number of connected components that get merged, it follows that

$$\mathbb{E}[X_i - X_{i+1} \mid S_i] \geq \frac{1-1/e}{2} X_i = (1-1/\gamma)X_i.$$

This inequality assumes that $X_i > 1$ (notice that X_i is fixed because S_i is fixed). To lift this assumption we define the random variables $Y_i = X_i - 1$ and consider them instead. We have

$$\mathbb{E}[Y_i - Y_{i+1} \mid S_i] = \mathbb{E}[X_i - X_{i+1} \mid S_i] \geq (1-1/\gamma)X_i \geq (1-1/\gamma)Y_i.$$

The above inequality $\mathbb{E}[Y_i - Y_{i+1} \mid S_i] \geq (1-1/\gamma)Y_i$ also holds (trivially) when $X_i = 1$, since then $Y_i = 0$. Taking now the unconditional expectation yields $\mathbb{E}[Y_i - Y_{i+1}] \geq (1-1/\gamma)\,\mathbb{E}[Y_i]$, which implies $\mathbb{E}[Y_{i+1}] \leq \mathbb{E}[Y_i]/\gamma$. Applying this inequality repeatedly gives

$$\mathbb{E}[Y_r] \leq \mathbb{E}[Y_0]/\gamma^r \leq n/\gamma^r,$$

since $Y_0 < n$. This yields $\mathbb{E}[Y_r] \leq n/\gamma^r = \delta$, as $r = \log_\gamma(n/\delta)$. By Markov's inequality then we obtain $\Pr(Y_r > 0) = \Pr(Y_r \geq 1) \leq \mathbb{E}[Y_r]/1 \leq \delta$. Therefore, the probability that there is only one connected component by the end of the last round is at least $1 - \delta$. □

Lemma 5.2.5 requires that we start with a set S of (already sampled) nodes which is a λ-semi-connected dominating set. To achieve domination (but not λ-semi-connectivity) with probability at least $1 - \delta$, it suffices to sample nodes with probability $\Theta(\log(n/\delta)/k)$. Recall that k is the vertex connectivity of the graph and thus also a lower bound on the minim degree of the graph.

Lemma 5.2.6. *Sampling each node with probability* $\ln(n/\delta)/k$ *yields a dominating set with probability at least* $1-\delta$.

Proof. From the k-vertex-connectivity of the graph, it follows that each node has degree at least k. Thus the probability for a given node that none of its neighbors gets sampled is at most $\left(1-\frac{\ln(n/\delta)}{k}\right)^k \leq e^{-\frac{\ln(n/\delta)}{k}\cdot k} = \frac{\delta}{n}$. By the union bound, the probability that this happens for at least one of the n nodes is at most δ. □

It remains to bound the sampling probability needed to achieve λ-semi-connectivity. This is the key part in our analysis. In particular, we show that a sampling probability of $\Theta((\lambda+\log n)/k)$ suffices to achieve λ-semi-connectivity. Section 5.2.4 is dedicated to the proof of this result, which is formally stated as follows.

Lemma 5.2.7 (Key Semi-Connectivity Claim)**.** *Let* $S \subseteq V$ *be a dominating set. Sampling each remaining node* $u \in V \setminus S$ *with probability* $16\lambda/k$ *yields a set* S' *such that* $S \cup S'$ *is a* λ*-semi-connected dominating set with probability at least* $1 - n/2^{\lambda}$.

We now have all the ingredients to prove Theorem 5.2.1.

Proof of Theorem 5.2.1. If $k = O(\log(n/\delta))$ then the theorem holds trivially by choosing the constant β such that $\beta\sqrt{\log(n/\delta)/k} \geq 1$. Below we assume that $k > \log(3n/\delta)$.

We consider three phases. First, we sample nodes with probability $\ln(3n/\delta)/k$, and from Lemma 5.2.6 we have that the resulting set, denoted S_1, is a dominating set with probability $1-\delta/3$.

In the next phase, we sample the remaining nodes $u \in V \setminus S_1$ with probability $16\lambda/k$, for $\lambda = \sqrt{k\log(3n/\delta)}$. From Lemma 5.2.7 it follows that if S_1 is a dominating set, then the set S_2 of all nodes sampled in the first two phases is a λ-semi-connected dominating set with probability $1-n/2^{\lambda}$. Note that $1-n/2^{\lambda} \geq 1-\delta/3$, because $\lambda = \sqrt{k\log(3n/\delta)} \geq \log(3n/\delta)$, as we have assumed $k > \log(3n/\delta)$.

In the last phase, we sample the remaining nodes $u \in V \setminus S_2$ with probability $\log_\gamma(n/\delta)/\lambda$, and obtain from Lemma 5.2.5 that the probability for the set S_3 of nodes sampled in the three phases to be a connected dominating set is at least $1-\delta/3$, provided that S_2 is a λ-semi-connected dominating set.

A union bound over all three phases shows that the probability of ending up with a connected dominating set S_3 is indeed $1-\delta$, and the total sampling probability is at most

$$\frac{\ln(3n/\delta)}{k} + \frac{16\sqrt{k\log(3n/\delta)}}{k} + \frac{\log_\gamma(n/\delta)}{\sqrt{k\log(3n/\delta)}},$$

which is $O\left(\sqrt{\log(n/\delta)/k}\right)$. □

5.2.4 Proof of Lemma 5.2.7: Sampling for λ-Semi-Connectivity

We assume that sampling is performed in rounds. In each round, each node that has not been sampled yet is sampled with probability $1/k$. Within a round, the sampling of nodes is done sequentially, in *steps*, with a single node considered for sampling at each step (the order in which nodes are considered in a round can be arbitrary). We will denote by S_t the set containing all nodes sampled in the first t steps and all nodes $u \in S$ (so $S_0 = S$).

Along with the sampling process, we consider a procedure that colors the *edges* of the graph. We describe this procedure and the related notion of *novo-connectivity* next.

Edge-Coloring Procedure. At any point in time, each edge has a color from the set {black, gray, white, color-1, ..., color-λ}. The same color can be used for more than one edge, and the color of an edge may change during the sampling process.

We have the following coloring initially: Edges with both endpoints in $S_0 = S$ are black; the edges between S and $V \setminus S$ are gray; and all remaining edges (between nodes from $V \setminus S$) are white. There are no color-i edges initially, for any $1 \leq i \leq \lambda$.

In each step of the sampling process, some edges may change color. The possible changes are that white edges may switch to color-i, for some i, and edges of any color may switch to black. At any point in time we have the following invariants:

- An edge is black iff both its endpoints belong to the set S_t of nodes sampled up to that point.
- If an edge is gray or of color-i, for some i, then exactly one of its endpoints is in S_t and the other in $V \setminus S_t$.
- If both endpoints of an edge are in $V \setminus S_t$ then this edge is white. (But it is possible for a white edge to have one endpoint in S_t and the other in $V \setminus S_t$.)

Before we describe precisely the color changes that take place in each step we must introduce the key concept of *novo-connectivity*.

Definition 5.2.8 (i-Novo-Connectivity)**.** *A simple path between two* sampled *nodes is an* i-novo-path*, for some $1 \leq i \leq \lambda$, if (1) each edge in the path has a color from the set {black, gray, color-i}, and (2) for any two consecutive edges whose common endpoint is not sampled, at least one of them is a color-i edge.*[2] *Two sampled vertices are* i-novo-connected *if there is an i-novo-path between them. Finally, an* i-novo-connected component*, or simply* i-novo-component*, is a maximal subset of the sampled nodes such that any two nodes in that set are i-novo-connected.*[3]

We describe now the color changes that take place during step $t \geq 1$. Suppose that node $u \notin S_{t-1}$ is considered for sampling in step t. If u is not sampled in that step, i.e.,

[2] The other edge is then either a color-i or a gray edge.

[3] We will see, in Claim 5.2.9, that i-novo-connectivity is an equivalence relation between sampled nodes.

$S_t = S_{t-1}$, then there are no changes. If u is sampled, i.e., $S_t = S_{t-1} \cup \{u\}$, all edges uv with $v \in S_{t-1}$ become black, and then the following λ *sub-steps* are performed. In each sub-step $i = 1, \ldots, \lambda$, some edges incident to u may switch from white to color-i. Precisely, an edge uv switches to color-i in sub-step i of step t if all the conditions below hold simultaneously:

1. uv is white before sub-step i.
2. v is adjacent to only one i-novo-component before step t—we say v is an *exclusive* neighbor of that component.
3. u is not adjacent to the same i-novo-component as v before step t.

We also have the additional rule:

4. If there are more than one node v that satisfy the three conditions above and are all adjacent to the *same* i-novo-component before step t, then only one edge uv is colored with color-i (choosing an arbitrary one among those nodes v).

Our proof of Lemma 5.2.7 relies on analyzing how i-novo-components evolve over time as the sampling proceeds. Intuitively, we will show that in the end, for each color i, all the connected components of the subgraph induced by the sampled nodes are connected by length-2 paths consisting of color-i edges and gray edges. Moreover, these paths can be chosen in such a way that the connector paths for different colors are internally vertex-disjoint so that together, they imply that the sampled set is λ-semi-connected.

Roadmap of the Rest of the Proof. The remainder of the proof unfolds in a series of claims. Claim 5.2.9 shows that i-novo-connectivity is an equivalence relation; this is required for our definition of i-novo-components. In Claim 5.2.10, we identify the setting under which two i-novo-components *merge* into a single component, and in Claim 5.2.11, we prove that if two nodes are i-novo-connected, they are also $(i-1)$-novo-connected. Next, we introduce the notion of a *critical node* of an i-novo-component (Definition 5.2.12). Roughly speaking, such a node, when sampled, causes the i-novo-component to merge with another component or, if not, it causes some j-novo-component, for $j < i$, to merge (Claim 5.2.14). We also show, in Claim 5.2.13, that the number of critical nodes for each i-novo-component is at least equal to the vertex connectivity k. These two results are used in Claims 5.2.15 and 5.2.16 to compute the expected drop in a round, of the number of distinct novo-components. This expected drop is used then in Claim 5.2.17 to bound by $O(\lambda)$ the number of rounds before there is just a single λ-novo-component (at that time, from Claim 5.2.11, it follows that for any i there is just a single i-novo-component). Finally, we show that having just a single i-novo-component, for each $1 \leq i \leq \lambda$, implies λ-semi-connectivity (Claim 5.2.18), concluding the proof of Lemma 5.2.7.

We first show that i-novo-connectivity is indeed an equivalence relation between sampled nodes, making the notion of an i-novo-component well defined.

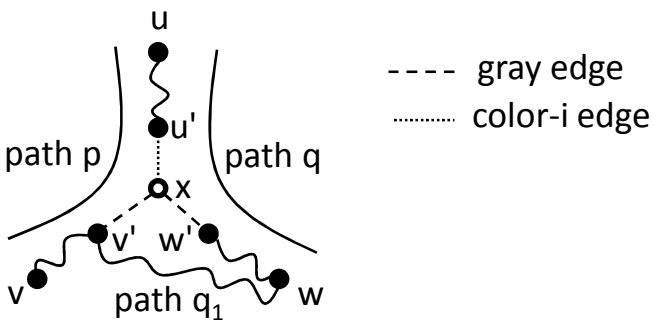


Figure 5-1: Illustrating the proof of Claim 5.2.9

Claim 5.2.9. *i-novo-connectivity is an equivalence relation between sampled nodes.*

Proof. It is straightforward to see that i-novo-connectivity is reflexive and symmetric. It remains to show transitivity, i.e., if a node u is i-novo-connected with nodes v and w, then v and w are i-novo-connected with each other.

Suppose, for the sake of contradiction, that the transitivity property is violated at some point, and let t be the earliest step when this happens. That is, at some point during step t, there is some i and nodes u, v, w such that u is i-novo-connected with both v and w, but v and w are not i-novo-connected with each other. Recall that before the first step there are no color-i edges, so at that time two nodes are i-novo-connected iff they are connected (by a path of black edges), and thus transitivity holds.

Let p be an i-novo-path between v and u, and q an i-novo-path between w and u (see Figure 5-1). Let x be the first node where the two paths intersect when going from w towards u on path q. We define r to be the concatenation of the subpath of p connecting v and x and of the subpath of q connecting x and w. Note that r is a simple path connecting v and w, and node x is the only node of r that is in the intersection of paths p and q (otherwise at least one of these paths is not a simple path). Further note that x cannot be a sampled node because in that case the path r is an i-novo-path connecting nodes v and w and thus v and w are i-novo-connected. Hence, in particular, $x \notin \{u, v, w\}$. Let v' and w' be the neighbors of x in path r towards v and w, respectively, and let u' be the neighbor of x in p towards u. Note that it is possible that $u = u'$, $v = v'$, or $w = w'$. We also observe that both edges xv' and xw' must be gray, because if at least one of them is color-i then r is an i-novo-path.

We have thus established that node x is not sampled and both edges xv' and xw' are gray. Since xv' and xu' are consecutive edges in i-novo-path p and x is not sampled, it follows that xu' must be color-i. Consider the step $t' \leq t$ at which this edge changed from white to color-i, when u' was sampled. It must be the case that before step t', and thus before step t, x was an exclusive neighbor to a single i-novo-component. We stress here that at any point before step t, i-novo-components are well defined as the transitivity property holds for i-novo-connectivity up to that step, because of the minimality of t. Since xv' and

xw' are both gray and since no edge becomes gray at any step, these edges were also gray before step t', which means that v' and w' were in the same i-novo-component before step t.

We now argue that at least one of the nodes v and w is also in the same i-novo-component as v' and w' before step t: The subpath of r between v and v' and the subpath between w and w' are both i-novo-paths and they do not intersect. We also have that in step t, as in any step, only edges incident to the node sampled in that step (if one is indeed sampled) may change color. Since the subpaths above do not share a common node, at least one of them does not change in step t. Suppose, w.l.o.g., that the subpath between w and w' does not change. It follows then that w is in the same i-novo-component as v' and w' before step t. Thus there is an i-novo-path q_1 between w and v' before step t. We further denote the subpath of p between v and v' by p_1.

We now apply a very similar argument as above using paths p_1 and q_1 in place of p and q. Two key observations here are that (1) p_1 is a proper subpath of p, and thus p_1 is strictly shorter than p; and (2) the i-novo-path q_1 between w and v' existed before step t. Similarly to before, we let x_1 be the first node in the intersection of p_1 and q_1 when going from node v towards node v' on path p_1 and we denote r_1 the concatenation of the subpaths of p_1 and q_1 connecting x_1 with v and w, respectively. Again x_1 cannot be sampled as otherwise r_1 is an i-novo-path. Defining v'_1 and w'_1 in a similar manner as before and using an analogous argument, we get that v'_1 and w'_1 are in the same i-novo-component before step t. Observe that w is also in that i-novo-component, because w and w'_1 are connected by an i-novo-path before step t, namely the subpath of q_1 between w and w'_1 (see observation (2) made earlier). Thus there is an i-novo-path q_2 between w and v'_1 before step t. We then define p_2 to be the subpath of p_1 between v and v'_1, and repeat the exact same argument for p_2 and q_2, and so on. Since the length of the paths $p_1, p_2, \ldots$ strictly decreases, it follows that for some s we will have $v'_s = v$, and from this we obtain that v and w are in the same i-novo-component before step t—a contradiction. □

Two distinct i-novo-components *merge* into a single component, if an i-novo-path is created between them. According to our method of changing edge colors, there are only two possible types of color changes that can cause a merge of i-novo-components. The first is if a common neighbor of the components is sampled and thus a black path is created between them. The second is if a neighbor u of the one component is sampled and then an edge uv to a neighbor v of the other component switches to color-i. Notice that any other case of an edge changing color to black or to color-i does not merge the two components because it does not create an i-novo-path between them, and that any other color change of an edge is from white to color-j for some $j \neq i$, and therefore also cannot cause a merge of i-novo-components.

The next claim talks about the latter case.

Claim 5.2.10. *Suppose that node u is sampled in step t, and edge uv is colored with color-i in sub-step i of that step. If u belongs to i-novo-component C before sub-step i of step t, and v is adjacent to i-novo-component C' before step t, then C and C' merge.*

Proof. We need to show that an i-novo-path is created between C and C'. Let w be a node in $C' \cap S$ to which v is connected (recall that S is the set of nodes we start with, before the first step). The node w exists, because S is a dominating set so there must be a node in S which is connected to v, and that node must belong to C' because v is an exclusive neighbor of C' (otherwise, uv would not be colored with color-i). Therefore, the edge wv must be gray since $w \in S$ and $v \in V \setminus S_t$. Since the edge vu is colored with color-i, this implies an i-novo-path between w and u, completing the proof. □

The next claim says that at any point, the partition of sampled nodes into i-novo-components is a refinement of the partition into $(i-1)$-novo-components.

Claim 5.2.11. *At any point in time, for any $2 \leq i \leq \lambda$, each i-novo-component is a subset of some $(i-1)$-novo-component.*

Proof. The proof is by induction on the number of steps t. The base case holds since when $t = 0$ there are only white, black, and gray edges, implying that any i-novo-component is also a j-novo-component for every $1 \leq i, j \leq \lambda$. Assume the claim holds after the first $t-1$ steps and consider step t.

Suppose node u is sampled at step t, and let $v_1, \ldots, v_\ell \in S_{t-1}$ be the neighbors of u that are already sampled. Since $S_{t-1} \supseteq S$ is a dominating set, we have that $\ell \geq 1$. Let $C_1, \ldots, C_\ell$ denote the i-novo-components to which $v_1, \ldots, v_\ell$, respectively, belong to before step t (these components are not necessarily distinct). When u is sampled, all edges uv_j, for $1 \leq j \leq \ell$, become black, and a new i-novo-component $C = \{u\} \cup C_1 \cup \cdots \cup C_\ell$ is formed, replacing $C_1, \ldots, C_\ell$. Similarly, if $C'_1, \ldots, C'_\ell$ are the $(i-1)$-novo-components of $v_1, \ldots, v_\ell$, respectively, before step t, then a new $(i-1)$-novo-component $C' = \{u\} \cup C'_1 \cup \cdots \cup C'_\ell$ replaces $C'_1, \ldots, C'_\ell$. From the induction hypothesis it follows that $C_j \subseteq C'_j$ for any $1 \leq j \leq \ell$, and thus $C \subseteq C'$. Also, any other i-novo-component which was a subset of one of the C'_j before step t, is now a subset of C. This proves that the claim holds before the first sub-step of step t.

It is also immediate that the claim holds before sub-step i of step t, because in the first $i-1$ sub-steps i-novo-components do not change, and in step $i-1$, $(i-1)$-novo-components may merge, but merging existing $(i-1)$-novo-components cannot invalidate the claim.

Consider now sub-step i. Let uv be an edge that is white and turns color-i at this sub-step, and let w be a neighbor of v in the i-novo-component for which v is an exclusive neighbor before step t. We need to show that u and w are $(i-1)$-novo-connected at this time. We assume otherwise, towards a contradiction, and show that uv fulfilled all the requirements for becoming color-$(i-1)$ at sub-step $i-1$. First, uv was white at sub-step $i-1$ since it is white before sub-step i. Second, u and v were not adjacent to the same $(i-1)$-novo-component before step t, otherwise u and w belong to the same $(i-1)$-novo-component, which we assumed to be false. Finally, v was adjacent to only one $(i-1)$-novo-component before step t, since otherwise, by the induction hypothesis, it is also adjacent to more than one i-novo-component. Since uv remained white after sub-step $i-1$, it must be that u and

w were already $(i-1)$-novo-connected, or became $(i-1)$-novo-connected at this sub-step by a different common neighbor v', for which uv' became color-$(i-1)$.

Finally, in the remaining sub-steps of step t after sub-step i, i-novo-components and $(i-1)$-novo-components do not change and thus the claim holds. □

Next, we define the notion of *critical nodes* of an i-novo-component, and show that each i-novo-component has at least k such critical nodes, where k is the vertex connectivity of the graph. Further, we show that the total number of j-novo-components for any $j \leq i$ that get merged in a round is bounded from below by the number of i-novo components for which some critical node gets sampled in that round.

Definition 5.2.12 (Critical Nodes)**.** *Let C be an i-novo-component before round r. A node u is* critical *for C in round r, if one of the following two conditions holds before round r: (1) u is a non-exclusive neighbor of C, i.e., it is adjacent to C and also to some i-novo-component $C' \neq C$; or (2) u is not in C or adjacent to C, but is adjacent to some exclusive neighbor of C.*

Claim 5.2.13. *Let C be an i-novo-component before round r. If there is more than one i-novo-components at that time, then there are at least k critical nodes for C in round r.*

Proof. Since the graph is k-vertex-connected, there are k internally-disjoint paths connecting C to other i-novo-components. We show that there is a critical node on each such path. First, we argue that there are k such paths of length 2 or 3. Then, we consider these two cases separately. For paths of length 2 we have that the internal node is critical for C, following directly from Definition 5.2.12. For paths of length 3, we show that the internal node that is not a neighbor of C is critical for C, by arguing that the other internal node must be an exclusive neighbor of C.

Formally, since S is a dominating set, there are k such paths of length 2 or 3. This is because, from a longer path p, we can obtain a path p' of length 2 or 3 whose internal nodes are also internal nodes of p: If p contains a node x that is a non-exclusive neighbor of C then we let x be the internal node of p', and p' has length 2. If no such node x exists, we let $y, z \notin C$ be a pair of consecutive nodes in p such that y is a neighbor of C and z is not, and we let these two nodes be the internal nodes of p'—p' has length 3 in this case. Since z is not adjacent to C it must be adjacent to another i-novo-component, as S is a dominating set.

Consider now the internal nodes of all paths of length 2 between C and other i-novo-components. These nodes are critical for C, by the first condition of Definition 5.2.12. If the number ℓ of such nodes is at least k then we are done. So, suppose that $\ell < k$, and consider the paths of length 3 that are internally-disjoint to all paths of length 2. Each internal node y on such a path p that is adjacent to C is an exclusive neighbor of C, otherwise it would have been on a path of length 2. Therefore, the other internal node z on p will be adjacent to y and to a component other than C, thus it is critical for C by the second condition of Definition 5.2.12. Since there are at least $k - \ell$ such disjoint paths of length 3 with internal

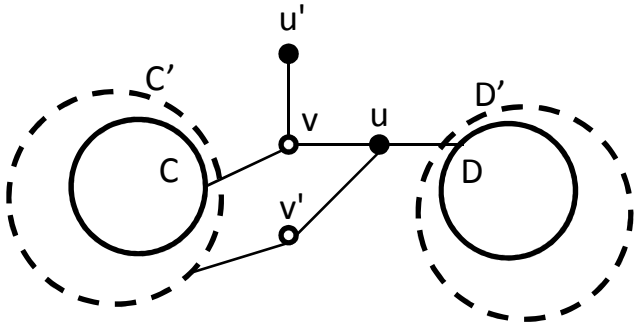


Figure 5-2: Illustrating the proof of Claim 5.2.14

nodes that are not on paths of length 2, we have at least $k - \ell$ additional critical nodes for C. □

Claim 5.2.14. *The total number of j-novo-components, summed over all $j \leq i$, that get merged in round r is at least equal to the number of i-novo-components before round r, for which some critical node gets sampled in a step of round r.*

Proof. Let C be an i-novo-component before round r, for which a critical node gets sampled in some step of round r. Let t be the earliest step when this happens, and let u be the critical node of C sampled at that step. Suppose that C does not merge with any other i-novo-component in round r. Precisely, there is no i-novo-component $D \neq C$ before round r, such that an i-novo-path is created between C and D during round r. We prove that a merge occurs between two j-novo-components in step t, for some $j < i$, and moreover, we have a distinct such merge for each such C.

First we observe that, from the assumption that C does not merge with another i-novo-component in round r, u cannot be a non-exclusive neighbor of C before round r, as this would imply that u was also adjacent to some other i-novo-component $D \neq C$ before round r, and thus sampling u creates a black path between C and D. Hence, from the definition of a critical node, we know that before the start of round r, u is not adjacent to C, but it is adjacent to some exclusive neighbor v of C, and it is also adjacent to some i-novo-component $D \neq C$ since S is a dominating set (see Figure 5-2). Consider now the graph before step t. Let $C' \supseteq C$ and $D' \supseteq D$ denote the current i-novo-components at that time, containing respectively C and D. (Note that C' may be a proper superset of C even though C does not merge, e.g., if some exclusive neighbor of C was sampled in a previous step of the round.) Then u is a neighbor of D', and we claim that it is not a neighbor of C', otherwise sampleing u would create an i-novo-path between C' and D', and thus one between C and D. We also claim that v is an exclusive neighbor of C': First, v cannot have been sampled in an earlier step, because then sampleing u would create a black path between C' and D'. Second, v cannot be a non-exclusive neighbor of C', otherwise some neighbor u' of v was sampled before step t in round r, and u' does not belong to C', implying that u' is not a neighbor of C, and

thus u' must be a critical node for C for round r. But this contradicts the assumption that u is the first critical node of C to get sampled in round r.

We have thus established that before step t, u is adjacent to D' and not adjacent to C', and v is an exclusive neighbor of C'. We now argue that uv cannot be white before sub-step i: If uv was white, then Conditions 1–3 required for uv to switch to color-i would be satisfied. From this and the last condition, 4, it would follow that either uv or some other edge uv' would become color-i, for some exclusive neighbor $v' \neq v$ of C'. From Claim 5.2.10 then, this would create an i-novo-path between C' and D' (because there is a black edge between u and D'), and thus between C and D.

Since uv is not white before sub-step i (of step t), and it is clearly white at the beginning of step t as none of its endpoints is sampled at the time, it follows that uv switches from white to color-j, for some $j < i$, in sub-step j of step t. Claim 5.2.10 then implies that two j-novo-components R and R', such that $u \in R$ and v is an exclusive neighbor of R', get merged by coloring uv.

Finally, suppose that u is also a critical node for some i-novo-component $C'' \neq C'$, and that, like C', C'' does not merge with D'. As before this implies that a pair of j'-novo-components Q and Q' get merged instead, for some $j' < i$. We observe that $(j, R, R') \neq (j', Q, Q')$, i.e, the merges of C' with D' and of C'' with D' are prevented by distinct merges of j,j'-novo-components for smaller j and j'. This is clear if $j' \neq j$, and if $j' = j$ it follows from Condition 4 for coloring an edge color-j.

We have thus shown that a distinct merge of two j-novo-components occurs, for some $j \leq i$, for each i-novo-component C before round r for which a critical node gets sampled in round r. □

We will use Claims 5.2.13 and 5.2.14 to show next that the expected drop in a round of the total number of j-novo-components for all $j \leq i$ is at least a linear function in the expected number of i-novo-components before the round.

For $1 \leq i \leq \lambda$ and $r \geq 0$, let $X_{i,r}$ denote the number of i-novo-components after the first r rounds. Also let $Y_{i,r} = X_{i,r} - 1$, and let $y_{i,r} = \mathbb{E}[Y_{i,r}]$.

Claim 5.2.15. *For any $1 \leq i \leq \lambda$ and $r \geq 1$, and for $\rho = \frac{e-1}{2e}$, we have $\sum_{j=1}^{i}(y_{j,r-1} - y_{j,r}) \geq \rho y_{i,r-1}$.*

Proof. By Claim 5.2.13 we have that every i-novo-component C has at least k critical nodes. The probability that a node gets sampled in a given round is $1/k$, thus the probability at least one of C's critical nodes gets sampled in round r is at least $1-(1-1/k)^k \geq 1-1/e = 2\rho$. If this happens we say that C *causes a merge*.

By Claim 5.2.14 we have that the total number of j-novo-components over all $j \leq i$ that get merged in round r is at least equal to the number of i-novo-components that cause a merge. The decrease in the number of j-novo-components is at least half the number of those merges.

Let X_C be an indicator random variable that is 1 if component C causes a merge and 0 otherwise. Given the number $X_{i,r-1}$ of i-novo-components before round r, the expected number of such components that cause a merge is $\mathbb{E}[\sum X_C | X_{i,r-1}] = \sum \mathbb{E}[X_C | X_{i,r-1}] \geq 2\rho Y_{i,r-1}$. Then

$$\mathbb{E}\left[\sum_{j=1}^{i}(X_{j,r-1} - X_{j,r}) \,\middle|\, X_{i,r-1}\right] \geq \frac{1}{2}\,\mathbb{E}\left[\sum X_C | X_{i,r-1}\right] \geq \rho Y_{i,r-1},$$

and by the law of total expectation

$$\mathbb{E}\left[\sum_{j=1}^{i}(X_{j,r-1} - X_{j,r})\right] \geq \rho\,\mathbb{E}[Y_{i,r-1}].$$

We therefore obtain

$$\begin{aligned}\sum_{j=1}^{i}(y_{j,r-1} - y_{j,r}) &= \mathbb{E}\left[\sum_{j=1}^{i}(Y_{j,r-1} - Y_{j,r})\right] \\ &= \mathbb{E}\left[\sum_{j=1}^{i}(X_{j,r-1} - X_{j,r})\right] \geq \rho\,\mathbb{E}[Y_{i,r-1}] \;=\; \rho y_{i,r-1},\end{aligned}$$

which concludes the proof. □

Using Claim 5.2.15 we establish the following upper bound on $y_{i,r}$.

Claim 5.2.16. *For any $1 \leq i \leq \lambda$ and $r \geq 0$, and for $\rho = \frac{e-1}{2e}$ as in Claim 5.2.15, we have*

$$y_{i,r} \leq n\left(1 - \frac{\rho}{2}\right)^r \left(1 + \frac{2}{\rho}\right)^{i-1}.$$

Proof. We prove the statement by induction on r. For $r = 0$, we have $y_{i,r} \leq n$ and thus the claimed inequality clearly holds for all $i \in \{1, \ldots, \lambda\}$.

For the induction step, we assume that the inequality holds for $y_{i,r-1}$ for all $i \in \{1, \ldots, \lambda\}$, for some $r \geq 1$, and bound $y_{i,r}$. Solving the inequality in Claim 5.2.15 for $y_{i,r}$ and using the trivial lower bound $y_{j,r} \geq 0$ for all $j \leq i - 1$, gives

$$y_{i,r} \leq (1 - \rho) y_{i,r-1} + \sum_{j=1}^{i-1} y_{j,r-1}.$$

Applying the induction hypothesis to all terms on the right-hand side, we obtain

$$\begin{aligned} y_{i,r} &\overset{\text{(I.H.)}}{\leq} n\left(1-\frac{\rho}{2}\right)^{r-1} \cdot \left[(1-\rho)\left(1+\frac{2}{\rho}\right)^{i-1} + \sum_{j=1}^{i-1}\left(1+\frac{2}{\rho}\right)^{j-1}\right] \\ &< n\left(1-\frac{\rho}{2}\right)^{r-1} \cdot \left[\left(1+\frac{2}{\rho}\right)^{i-1}\left(-\rho+\sum_{h=0}^{\infty}\left(1+\frac{2}{\rho}\right)^{-h}\right)\right] \\ &= n\left(1-\frac{\rho}{2}\right)^{r-1}\left[\left(1+\frac{2}{\rho}\right)^{i-1}\left(1-\frac{\rho}{2}\right)\right], \end{aligned}$$

and thus the claim follows. □

Using Claim 5.2.16 and Markov's inequality we bound the number of rounds before there is just a single λ-novo-component left.

Claim 5.2.17. *All λ-novo-components have merged into a single component after 16λ rounds, with probability at least $1 - n/2^{\lambda}$.*

Proof. The probability there is more than one λ-novo-component after the first r rounds is $\Pr(X_{\lambda,r} > 1) = \Pr(Y_{\lambda,r} > 0) = \Pr(Y_{\lambda,r} \geq 1) \leq \mathbb{E}[Y_{\lambda,r}]/1$, by Markov's inequality. Also from Claim 5.2.16,

$$\mathbb{E}[Y_{\lambda,r}] \leq n\left(1-\frac{\rho}{2}\right)^{r}\left(1+\frac{2}{\rho}\right)^{\lambda}.$$

Thus, in order to have $\Pr(X_{\lambda,r} > 1) \leq n/2^{\lambda}$, it suffices that

$$n\left(1-\frac{\rho}{2}\right)^{r}\left(1+\frac{2}{\rho}\right)^{\lambda} \leq n/2^{\lambda}.$$

Solving for r and substituting ρ's value, $\rho = \frac{e-1}{2e}$, we obtain $r \geq \lambda \ln\left(\frac{\rho}{2\rho+4}\right)/\ln\left(1-\frac{\rho}{2}\right) \approx 15.6085 \cdot \lambda$. □

We now show that if there is just one i-novo-component after step t, then the set S_t of nodes that have been sampled by that time is i-semi-connected, i.e., for any partition of S_t into two sets T and $S_t \setminus T$ with no edges between them, the two sets have at least i common neighbors.

Claim 5.2.18. *If there is only one i-novo-component after step t, then S_t is i-semi-connected.*

Proof. We show that if there is only one i-novo-component after step t, then for any partition of S_t into two sets T and $S_t \setminus T$ with no edges between them, there is a j-novo-path between T and $S_t \setminus T$ for each $1 \leq j \leq i$, such that all these paths have length 2 and are internally disjoint. In particular, this means that S_t is i-semi-connected.

For this, we show by induction on j that for each $1 \leq j \leq i$ we can identify a j-novo-path of length 2 between T and $S_t \setminus T$ which is internally disjoint from all previous paths.

We first note that for every j, at any point during the random process, and for any subset T' of the set S' of nodes sampled by that time, if T' and $S' \setminus T'$ are not connected by an edge and if there exists a j-novo-path between T' and $S' \setminus T'$, then there also exists such a j-novo-path of length 2. That is, there exists a path uwv, where $u \in T'$, $v \in S' \setminus T'$, and $w \notin S'$, and where at least one of the edges uw and vw is color-j and the other edge is either color-j or gray. This follows directly from the definition of j-novo-paths. As j-novo-paths only consist of color-j edges and of gray edges, at least one of the nodes of each edge has to be sampled and therefore on a j-novo-path, at least every second node has to be sampled. Any minimal j-novo-path connecting T' and $S' \setminus T'$ therefore has to be of length 2.

Since there is just a single i-novo-component after step t, Claim 5.2.11 gives that there is also just a single j-novo-component. Hence, there must be at least one j-novo-path connecting T and $S_t \setminus T$. In particular, using the above observation, there must be such a j-novo-path of length 2.

We need to show that among these length 2 j-novo-paths there is at least one that is internally disjoint from all of the ℓ-novo-paths given by the induction hypothesis, for all $1 \leq \ell < j$. Consider the first time in which such a length 2 j-novo-path is created. At this time one of the two edges of the path becomes color-j (the other edge is gray or became color-j in an earlier step). Let $u \in T$ and $v \in S_t \setminus T$ be the two endpoints of the path and $w \in V \setminus S_t$ be its internal node. Assume w.l.o.g., edge wv is the one that becomes color-j.

First we argue that there is no gray edge wv' between w and some node $v' \in S_t \setminus T$: Suppose there is such a gray edge wv'. Then edge uw cannot be color-j, because then uwv' is a j-novo-path created before uwv. Thus edge uw is gray. However, it must be the case that before wv became color-j, node w was an exclusive neighbor of a j-novo-component, and since both uw and wv' are gray, it follows that u and v' belonged to the same j-novo-component. Thus before wv became color-j there was already a j-novo-path between nodes $u \in T$ and $v' \in S_t \setminus T$, and thus there was also a j-novo-path of length 2 (see above) between two nodes from these two sets. This contradicts that uwv was the first such j-novo-path.

We now show that path uwv is internally disjoint from the ℓ-novo-paths given by the induction hypothesis for all $1 \leq \ell < j$. Fix some $\ell \in \{1, \ldots, j-1\}$. Consider an ℓ-novo-path of length 2 between T and $S_t \setminus T$ whose internal node is w; let $u''wv''$ denote that path where $u'' \in T$ and $v'' \in S_t \setminus T$. Since we have shown that there is no gray edge between w and some node from $S_t \setminus T$, it must be that edge wv'' is color-ℓ. We argue that v'' is sampled *after* v: Suppose, for contradiction, that v'' is sampled before v. Then when v is sampled, v'' and u must be in the same j-novo-component, otherwise w is adjacent to two distinct j-novo-components, preventing wv from becoming color-j. Thus before wv became color-j there was a j-novo-path between nodes $u \in T$ and $v'' \in S_t \setminus T$, and thus there was also a j-novo-path of length 2 between two nodes from these two sets. This contradicts that uwv was the first such j-novo-path. We conclude that v'' was sampled after v. This means that the ℓ-novo-path created when edge wv'' was colored, cannot be the first one created between T and $S_t \setminus T$, because that first path must have been created no later than the first j-novo-path, by Claim 5.2.11. □

By Claim 5.2.17, the sampling procedure results in a single λ-novo-component after at most 16λ rounds, with probability at least $1-n/2^{\lambda}$. In each round the sampling probability is $1/k$, thus the total sampling probability is at most $16\lambda/k$. Once there is just a single λ-novo-component, by Claim 5.2.18 we have that the set S_t of sampled nodes is λ-semi-connected. This concludes the proof of Lemma 5.2.7.

5.3 From Concentration for Simple Connectivity to High Connectivity

In this section, we prove Theorem 5.1.1 and Theorem 5.1.2, assuming that Theorem 5.2.1 holds. In simple words, the arguments presented in this section allow us to turn a "*very high probability of remaining (simply) connected after sampling*" to a "*very likely high vertex connectivity after sampling.*" In each part, we first provide an informal sketch of the idea and then proceed to the formal proof.

5.3.1 Vertex Connectivity under Vertex Sampling

Proof Idea: To show the lower bound of Theorem 5.1.1 on the remaining vertex connectivity after vertex sampling, we view the sampling with probability p as a two-step process: first sampling with probability $2p$, and then sub-sampling with probability $1/2$. We argue that if the set sampled in the first step did not have sufficiently high vertex connectivity (with sufficiently high probability), then the two-step sampling would not result in a connected set with sufficiently high probability, thus contradicting Theorem 5.2.1.

Proof of Theorem 5.1.1. Let $\alpha = 2\beta\sqrt{3}$, where β is the constant in the statement of Theorem 5.2.1, and let $\gamma = \alpha^{-2}$. We show that for vertex-sampling probability $p \geq \alpha\sqrt{\log(n)/k}$, the resulting graph has vertex connectivity at least γkp^2, with probability at least $1-e^{-\gamma kp^2}$.

Assume, towards a contradiction, that the above is not true. That is, for some sampling probability $p \geq \alpha\sqrt{\log(n)/k}$, with probability greater than $e^{-\gamma kp^2}$ the sampled graph has vertex connectivity less than γkp^2. We show that this contradicts Theorem 5.2.1.

We denote by $q = p/2$ a sampling probability for using in Theorem 5.2.1. We view the process of vertex sampling with probability q as a two-stage sampling process: one sampling stage with probability $p = 2q$, and afterwards an independent subsampling with probability $1/2$.

Assume that the vertex connectivity after sampling with probability p is less than γkp^2. This event happens with probability greater than $e^{-\gamma kp^2}$, by our assumption above. Then, conditioned on this event, if we zoom in on a single vertex cut of size at most γkp^2 in this graph, during the further $1/2$-sampling, all nodes of this cut are removed with probability at least $2^{-\gamma kp^2}$. Overall this means that sampling nodes with probability q has a probability of at least $e^{-2\gamma kp^2}$ to sample a set S that is not a connected dominating set.

On the other hand, applying now Theorem 5.2.1 with failure probability $\delta = e^{-2\gamma kp^2}$, we obtain that when sampling with probability at least $\beta\sqrt{\frac{\log(n/\delta)}{k}}$, we obtain a connected dominating set with probability at least $1 - \delta = 1 - e^{-2\gamma kp^2}$. However, since

$$\beta\sqrt{\frac{\log(n/\delta)}{k}} = \beta\sqrt{\frac{\log n}{k} + 2\gamma p^2} = \beta\sqrt{\gamma}p\sqrt{\frac{\log n}{k\gamma p^2} + 2}$$
$$= \beta\alpha^{-1}p\sqrt{\frac{\log n}{k\alpha^{-2}p^2} + 2} \leq \beta\alpha^{-1}p\sqrt{3} = \frac{p}{2} = q,$$

this contradicts the result we showed just before. (The inequality in the expression above arises from the assumption on p.) □

5.3.2 Vertex Connectivity under Edge Sampling

Proof Idea: We consider a two-phase sampling process: in one phase *edges* are sampled with probability p, and in the other phase *vertices* are sampled with probability 1/2. We can analyze the process in two ways. One way is first to argue that vertex sampling with probability 1/2 reduces vertex connectivity by at most a constant factor, by Theorem 5.1.1, implying the same lower bound on edge connectivity, and then to apply an edge sampling result by Karger [Kar94b] to conclude that the graph remains connected with very high probability. The second way of analyzing the process is first to bound from below the remaining vertex connectivity after edge sampling, as in the statement of Theorem 5.1.2, and to combine that with the probability that a minimum edge cut in the sampled graph survives the subsequent vertex sampling. Comparing the results of the above two approaches yields the bound of Theorem 5.1.2.

Proof of Theorem 5.1.2. We use the following result from [Kar94b]: There are constants $\zeta, \eta > 0$, such that for any n-node λ-edge-connected graph, independent edge sampling with probability $p \geq \zeta \log(n)/\lambda$ yields a spanning subgraph with edge connectivity at least $\eta\lambda p$, with probability at least $1 - e^{-\eta\lambda p}$.

We also use the following statement for vertex sampling with probability 1/2 which follows from Theorem 5.1.1: There are constants $g, h > 0$, such that for any n-node graph with vertex connectivity $k \geq g \log n$, vertex sampling with probability 1/2 yields a dominating set which induces a subgraph with edge connectivity at least hk, with probability at least $1 - e^{-hk}$ (the domination is immediate from the proof of Theorem 5.1.1).

Let $\alpha = \zeta/h$ and $\gamma = \eta h/2$. Suppose also that $k \geq g \log n$ (otherwise the statement of Theorem 5.1.2 holds trivially). We show that for edge-sampling probability $p \geq \alpha \log(n)/k$, the sampled graph has vertex connectivity at least γkp, with probability at least $1 - e^{-\gamma kp}$.

Assume, towards a contradiction, that this is not true. Then for some edge-sampling probability $p \geq \alpha \log(n)/k$, with probability greater than $e^{-\gamma kp}$ the sampled graph has vertex connectivity less than γkp.

Consider the following two-phase sampling process: First we sample *edges* with probability p, and then in the resulting graph we sample *vertices* with probability $1/2$. Suppose that the vertex connectivity after the edge-sampling phase is less than γkp. This event happens with probability greater than $e^{-\gamma kp}$, by our assumption of Theorem 5.1.2 not holding. Then, conditioned on this event, if we look at a single edge cut of size at most γkp in this graph, during the vertex-sampling phase, the event that no edge in this cut has both its endpoints sampled, and thus no edge in the cut survives the sampling, has probability at least $(3/4)^{\gamma kp}$. Overall this means that the two-phase sampling process has a probability of at least $2^{-2\gamma kp}$ to result in a graph that is not connected or its vertex set is not a dominating set of G.

Consider now the sampling process that is similar to the above, but with the two phases executed in reverse order. As explained above, it follows from Theorem 5.1.1 that after the vertex-sampling phase with probability $1/2$, the set of sampled vertices is a dominating set of G and induces a subgraph H with vertex connectivity at least hk, with probability at least $1 - e^{-hk}$. It follows that the *edge* connectivity of the induced subgraph H is at least hk with probability at least $1 - e^{-hk}$, as well. Further, from the result on edge sampling mentioned at the beginning, it follows that after the edge-sampling phase, the probability that the graph is a connected spanning subgraph of H, given that H has edge-connectivity at least $\lambda = hk$, is at least $1 - e^{-\eta hkp}$ (here we used the assumption $\alpha = \zeta/h$, which implies $p \geq \alpha \log(n)/k = \zeta \log(n)/\lambda$, and thus we can use the result on edge sampling). Overall this means that the two-phase sampling process has a probability of at least $1 - e^{-hk} - e^{-\eta hkp}$ to result in a connected subgraph of G whose vertex set is a dominating set of G.

Observe that the outcome of the two-phase sampling process should not depend on the order in which the phases are executed. Comparing the results above for the two different orders of the phases, we obtain the desired contradiction, because

$$e^{-hk} + e^{-\eta hkp} < 2^{-2\gamma kp}.$$

This inequality is obtained by using that $\gamma = \eta h/2$ and $kp = \Omega(\log n)$, and by assuming w.l.o.g. that $\eta < 1$.

□

5.4 Optimality of Our Sampling Results

We here argue that the bounds in Theorem 5.1.1 and Theorem 5.1.2 are existentially optimal.

The bound of Theorem 5.1.1 is existentially tight up to constant factors, as demonstrated in the following simple example.

Observation 5.4.1. *Let G be a $2n$-node graph consisting of two disjoint n-node cliques connected via a matching of $k \leq n$ edges. The vertex connectivity of G is k, and when each node is sampled with probability $p \geq 2 \ln n/n$, the expected vertex connectivity of the subgraph induced by the sampled nodes is at most $kp^2 + o(kp^2)$. If the sampling probability*

is $p = o(\sqrt{\log(n)/k})$*, then the subgraph is disconnected*[4] *with probability at least* $n^{-o(1)}$*.*

Proof of Observation 5.4.1. The edge connectivity of G is at most k as it contains an edge-cut of size k, and thus its vertex connectivity is also at most k. On the other hand, it is easy to verify that the removal of any $k-1$ vertices does not disconnect G. Therefore G has vertex connectivity exactly k.

Let K denote the number of edges in the matching that survive after sampling (i.e., both their endpoint nodes are sampled). The expected value of K is $\mathbb{E}[K] = kp^2$, since each edge survives with probability p^2. If $K \neq 0$ then K is an upper bound on the edge connectivity and thus on the vertex connectivity of the sampled subgraph. If $K = 0$ then it is still possible for the vertex connectivity to be positive, if no nodes are sampled from the one clique and at least one is sampled from the other. Let N_i, for $i = 1, 2$, denote the number of nodes sampled in each of the two cliques respectively, and let Z_i be the indicator random variable with $Z_i = 1$ if $N_i = 0$ and $Z_i = 0$ otherwise. Then $\mathbb{E}[N_i] = pn$, and $\mathbb{E}[Z_i] = \Pr(N_i = 0) = (1-p)^n$. From the discussion above it follows that the vertex expansion of the sampled subgraph is at most $K + Z_2N_1 + Z_1N_2$, and thus the expected vertex expansion is at most

$$\begin{aligned}\mathbb{E}[K + Z_2N_1 + Z_1N_2] &= \mathbb{E}[K] + 2\,\mathbb{E}[Z_2N_1] = \mathbb{E}[K] + 2\,\mathbb{E}[Z_2] \cdot \mathbb{E}[N_1] \\ &= kp^2 + 2np(1-p)^n \leq kp^2 + 2npe^{-np}.\end{aligned}$$

If $p \geq 2\ln n/n$, then the second term in the last line above is $kp^2 \cdot (2n/kp)e^{-np} \leq kp^2 \cdot (1/k\ln n) = o(kp^2)$; thus the expected vertex connectivity is at most $kp^2 + o(kp^2)$.

For the probability that the sampled subgraph is disconnected, we first observe that if $p = O(1/n)$ then the subgraph is empty (and thus by convention disconnected) with constant probability. Thus, below we assume that $p \geq 2/n$. The probability that the sampled subgraph is disconnected is bounded from below by

$$\begin{aligned}\Pr(K = 0 \wedge N_1 \neq 0 \wedge N_2 \neq 0) &\geq 1 - (\Pr(K \neq 0) + \Pr(N_1 = 0) + \Pr(N_2 = 0)) \\ &= \Pr(K = 0) - 2\Pr(N_1 = 0) \\ &= (1-p^2)^k - 2(1-p)^n.\end{aligned}$$

The second term in the last line is at most $(1-p^2)^k/2$, as

$$\frac{2(1-p)^n}{(1-p^2)^k} \leq \frac{2(1-p)^n}{(1-p^2)^n} = \frac{2}{(1+p)^n} \leq \frac{2}{(1+2/n)^n} \leq \frac{1}{2}.$$

It follows that the probability of the sampled subgraph to be disconnected is at least $(1-p^2)^k/2$, and this is at least $1/n^{o(1)}$ if $p = o(\sqrt{\log(n)/k})$. □

Even if one desires the sampled subgraph to be connected with merely a *constant* probability, our vertex sampling threshold $p = \Omega(\sqrt{\log(n)/k})$ is essentially tight as shown by the

[4] For the purposes of this statement, we consider the empty graph disconnected.

next simple example.

Observation 5.4.2. *Let G be an n-node graph consisting of n/k k-cliques ordered 1 to n/k, where each two consecutive cliques are connected via a k-edge matching. We assume that n is a multiple of k, and $k < n$. Graph G has vertex connectivity k, and when sampling nodes with probability $\omega(1/n) < p < o(\sqrt{\log(n/k)/k})$, the subgraph induced by the sampled nodes is disconnected with probability $1 - o(1)$.*

Proof Sketch of Observation 5.4.2. Since $p = \omega(1/n)$, we have with probability $1-o(1)$ that at least one node gets sampled from the first $n/3k$ cliques, and at least one gets sampled from the last $n/3k$ cliques. The probability that no edge survives in the cut between two given consecutive cliques is $(1-p^2)^k = e^{-o(\log(n/k))} = \omega(k/n)$, as $p = o(\sqrt{\log(n/k)/k})$. Thus, the probability that at least one of the cuts between the middle $n/3k$ cliques gets disconnected is at least

$$1 - (1 - \omega(k/n))^{k/6k} = 1 - o(1),$$

where for this computation we just considered every second cut, i.e., $n/6k$ cuts in total, and used the fact that these cuts are vertex-disjoint. Combining the above yields the claim. □

The bound of Theorem 5.1.2 is also existentially optimal. The reason is as follows: Karger's result from [Kar94b] states that the remaining edge connectivity is $\Theta(\lambda p)$ w.h.p. after edge sampling with probability p, when the initial edge connectivity is λ. Since the vertex connectivity k of a graph is upper bounded by its edge connectivity λ, but there are also graphs with $k = \lambda$, Karger's result implies, for such graphs, that the remaining vertex connectivity is $O(\lambda p)$ w.h.p. after edge sampling with probability p.

5.5 Discussion and Open Problems

In this chapter, we showed two main results: (1) When independently sampling *vertices* of a k-vertex-connected n-node graph with probability $p = \Omega(\sqrt{\log(n)/k})$, the sampled subgraph has a vertex connectivity of $\Omega(kp^2)$, with high probability; and (2) When independently sampling *edges* of a k-vertex-connected n-node graph with probability $p = \Omega(\log(n)/k)$, the sampled subgraph has a vertex connectivity of $\Omega(kp)$, with high probability. The core technical part, for both results, is to prove that vertex sampling with probability $p = \Omega(\sqrt{\log(n)/k})$ yields a subgraph that is (just) connected with sufficiently high probability. This is achieved by considering sampling as a gradual random process, and carefully analyzing the growth of the connected components.

The constant factors in our results are much smaller than 1; it would be interesting to identify the correct constants. Most importantly, we leave open whether the remaining vertex connectivity under vertex sampling is in fact at least $kp^2(1-\epsilon)$, for an arbitrary small $\varepsilon > 0$, assuming kp^2 is large enough, e.g., $kp^2 = \Omega(\log n/\operatorname{poly}(\varepsilon))$. In particular, for a sampling probability of $p = 1 - o(1)$, or equivalently a sub-constant deletion probability, this

would imply a remaining connectivity of $k - o(k)$ instead of just $O(k)$. The same question can also be asked for the remaining vertex connectivity under *edge* sampling.

Our results show only *lower* bounds on the remaining vertex connectivity. There are k-vertex-connected graphs which, under the same sampling processes, would retain a much higher vertex connectivity, e.g., up to kp when sampling vertices, and up to k when sampling edges. It would be interesting to see if one can tightly characterize the (e.g., expected) remaining vertex connectivity under sampling of a given graph as a simple and natural function of it. Alternatively, is there a variant of these random sampling processes, which in effect sparsifies the graph, but for which we can tightly characterize the remaining vertex connectivity?

Part III

Congestion—Distributed Algorithms

Chapter 6

Low-Congestion Shortcuts, and Minimum Spanning Tree

6.1 Introduction

In this chapter, as well as in the next chapters of this part, we present near-optimal distributed algorithms for various fundamental network optimization problems.

In this chapter, and mainly as a warm up for the next chapters, we present a near-optimal algorithm for the problem of computing a minimum spanning tree (MST). The reason that this can be viewed as a warm up is twofold: (1) the problem of computing an MST has turned out to have a central role in distributed algorithms for network optimization problems, significantly more central than its role in the centralized algorithms domain. The upper and lower bound techniques for the MST problem are used frequently in solving other distributed network optimization problems. (2) We use the MST problem here to introduce the concept of *low-congestion shortcuts*, which gives rise to a combinatorial framework that unifies the solutions of many of these problems, and also serves as a tool for obtaining better complexities in special families of network graphs.

Distributed algorithms for global network optimization problems have a long and rich history. The standard model in studying global network optimization problems is the CONGEST model, as explained in Chapter 2. A first-order summary of the state of the art is that, for many of the basic problems, including MST, Minimum Cut Approximation, Maximum Flow, and Shortest-Paths, the best-known upper bound is $\tilde{O}(D + \sqrt{n})$ rounds, or close to it, where D denotes the network diameter. See for instance [KP95, Elk04b, LPS13, GK13, Nan14, LPS14, CHGK14a, NS14, GL14, Gha14, GKK+15]. Furthermore, it is known that this round complexity is essentially the best-possible in general graphs, that is, there are graphs in which any (non-trivial approximation) algorithm for these problems requires $\tilde{\Omega}(D + \sqrt{n})$ rounds [PR99, Elk04c, DSHK+11].

We introduce the concept of *low-congestion shortcuts* and exhibit their utility, by using this concept to derive a very simple distributed algorithm for MST. Low-congestion shortcuts

lead to a nice unification of algorithmic approaches for network optimizations listed above, and particularly MST. On one hand, they can be used to obtain $\tilde{O}(D+\sqrt{n})$-round algorithms in general graphs, this is a bound that is nearly optimal in comparison the best possible on worst case networks. On the other hand, they provide a simple means for obtaining more efficient algorithms in special network graph families.

We next motivate and introduce low-congestion shortcuts. We then explain how they lead to an extremely simple $\tilde{O}(D+\sqrt{n})$ round algorithm for MST in general graphs, and faster algorithms in some special graph families.

6.1.1 The Motivation and Definition of Low-Congestion Shortcuts

Consider the following scenario, which is a recurring theme throughout distributed approaches for many network optimization problems:

> *The graph is partitioned into a number of disjoint parts, each of which induces a connected subgraph, and we need to compute a (typically simple) function for each part, e.g., the minimum of the values held by the nodes in the part.*

This scenario typically appears when the algorithmic approach works via (iterations of) merging solutions of smaller subproblems. This includes various methods based on *divide and conquer*. There are many examples, for instance [GHS83, GKP93, KP95, Elk04b, DSHK+11, GK13, NS14, Gha14, GKK+15]. Perhaps the most prominent example is the 1926 algorithm of Boruvka[1] [NMN01] for computing a Minimum Spanning Tree. This algorithm works as follows: We start with the trivial partition of each node being its own part. Then, in each iteration, each part computes its *minimum-weight outgoing edge*, and adds it to the current partial solution. This added edge makes the parts incident on it merge. After $O(\log n)$ iterations, we arrive at the MST.

Typically, since the number of parts can be large, we need to solve the problems of all parts in parallel. Most naturally, this would be by solving each part's problem using communication only inside the part. However, this would take a long time as the diameter of these parts can be large, much larger than the diameter D of the graph. It is often hard to enforce small part-wise diameters because the structure of the parts (i.e., subproblems) is usually dictated by the problem itself. For instance, think of the MST problem and Boruvka's method, discussed above.

To overcome this generic issue, we introduce the notion of *low-congestion shortcuts*: Intuitively, we want to augment each part with some extra edges, taken from the network G, so that we effectively reduce the diameter of the part. These shortcutting edges are to be used for communication purposes. To be able to use these shortcuts without creating

[1] We note that, within the distributed literature, this method is more often called the *GHS approach*, after Gallagher, Humblet and Spira who rediscovered it in 1983 [GHS83], and extended it to a distributed asynchronous setting.

communication bottlenecks, we need to ensure that each edge is not used in too many shortcuts. We next formalize this intuition:

Definition 6.1.1. *Given a graph $G = (V, E)$ and a partition of V into disjoint subsets $S_1, \ldots, S_N \subset V$, each inducing a connected subgraph $G[S_i]$, we define an α-congestion shortcut with dilation β to be a set of subgraphs $H_1, \ldots, H_N \subset G$, one for each set S_i, such that:*

(1) For each i, the diameter of the subgraph $G[S_i] + H_i$ is at most β.

(2) For each $e \in E$, the number of subgraphs $G[S_i] + H_i$ containing e is at most α.

As we see later, given such a low-congestion shortcut, one can solve the common scenario problem stated above in $\tilde{O}(\alpha + \beta)$ rounds, using standard *random delay* techniques for pipelining messages [LMR94].

6.1.2 Our Results

As the main application of low-congestion shortcuts, we provide the following unified distributed algorithm for computing an MST. As evident from the statement, the round complexity of this algorithm can be optimized easily as a function of the graph family on which we work on, by finding the best possible low-congestion shortcut for that family.

Theorem 6.1.2. *Suppose that the graph family $\mathcal{G}$ is such that for each graph $G \in \mathcal{G}$, and any partition of G into vertex-disjoint connected subsets $S_1, \ldots, S_N$, we can find an α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} \leq K$. Then, there is a randomized distributed MST algorithm that computes an MST in $O(\alpha \log n + \beta \log^2 n) = \tilde{O}(K)$ rounds, with high probability, in any graph from the family $\mathcal{G}$. Here, K can be a function of the family $\mathcal{G}$, and it can depend on n and D.*

In particular, as we discuss next, general graphs easily admit shortcuts with congestion and dilation in $O(D + \sqrt{n})$. Thus, we get a very simple and clean way to obtain the near-optimal $\tilde{O}(D + \sqrt{n})$ round complexity for computing an MST in general graphs.

Corollary 6.1.3. *For any n-node D-diameter graph, any partition of its vertices admits an α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} \leq O(D + \sqrt{n})$. Thus, there is a distributed algorithm that computes MST in general graphs in $\tilde{O}(D + \sqrt{n})$ rounds.*

Proof. Simply set the shortcut $H_i = G$ for each part S_i where $|S_i| \geq \sqrt{n}$, and set $H_i = \emptyset$ for any other part. This ensures that each shortcutted part has diameter at most $\max\{D, \sqrt{n}\} = O(D + \sqrt{n})$ and each edge is used in at most $n/\sqrt{n} + 1 = O(\sqrt{n})$ shortcutted parts. □

We note that a slightly more improved bound, with less logarithmic factors, is explained at the end of this chapter. Moreover, as a consequence of Theorem 6.1.2, we get more

optimized results for a few special graph families: for planar networks by using results of Ghaffari and Haeupler [GH16b] on shortcuts in planar graphs, and for near-expanders by using results of Ghaffari, Kuhn, and Su [GKS16] on shortcuts in near-expander graphs. In the following, we state these corollaries.

Theorem 6.1.4 (Ghaffari & Haeupler [GH16b]). *For any planar graph with n nodes and diameter D, any partition of the vertices admits an α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} = O(D \log D)$. Moreover, such a shortcut can be computed distributedly in $\tilde{O}(D)$ rounds.*

Corollary 6.1.5. *There is a randomized distributed algorithm that computes an MST in planar graphs in $\tilde{O}(D)$ rounds, with high probability.*

Theorem 6.1.6 (Ghaffari, Kuhn, & Su [GKS16]). *In any graph graph $G = (V, E)$ with n nodes, edge expansion at least $\frac{1}{\text{poly}(\log n)}$, and maximum degree at most $\text{poly}(\log n)$, any partition of the vertices admits an α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} = 2^{O(\sqrt{\log n \log\log n})}$. Moreover, such a shortcut can be computed distributedly in $2^{O(\sqrt{\log n \log\log n})}$ rounds. Here, the* edge expansion *of graph $G = (V, E)$ is defined as $\mathsf{expansion}(G) := \min_{S \subset V, |S| \leq n/2} \frac{e(S, V \setminus S)}{|S|}$, where $e(S, V \setminus S)$ denotes the number of G-edges between S and $V \setminus S$.*

Corollary 6.1.7. *There is a randomized distributed algorithm that computes an MST in any graph with edge-expansion at least $\frac{1}{\text{poly}(\log n)}$ and maximum degree at most $\text{poly}(\log n)$ in $2^{O(\sqrt{\log n \log\log n})}$ rounds, with high probability.*

6.2 Shortcuts in Action: Minimum Spanning Tree

In this section, we explain how to use low-congestion shortcuts to solve the common scenario problem described above. In particular, we explain how to compute an MST using low-congestion shortcuts. The method can be easily extended to solving a few of the other network optimization problems, for instance it can be used to compute a $(1+\varepsilon)$ approximation of the minimum cut, as we explain in the next chapter.

6.2.1 The Generic Method For Using Shortcuts

Before starting the MST algorithm, we explain the generic method for using low-congestion shortcuts, in the context of a simple example of simultaneously growing BFSs in the shortcutted parts.

Growing BFSs on Shortcuts via Random Delay: Suppose that we are given a partition of the graph $G = (V, E)$ into parts $S_1, S_2, \ldots, S_N$, each inducing a connected subgraph and having a leader $s_i \in S_i$. Assume that we are also given the corresponding shortcuts $H_1, S_2, \ldots, S_N$, which form an α-congestion β-dilation shortcut. We explain how to simultaneously grow N BFSs, one BFS for each $i \in [1, N]$ which is a BFS in the graph $G[S_i] + H_i$ rooted in s_i. All these BFSs will be constructed in $O(\alpha + \beta \log n)$ rounds.

Claim 6.2.1. *There is a randomized distributed algorithm that simultaneously constructs all the N BFSs, one for each subgraph $G[S_i] + H_i$ rooted in the leader s_i of the corresponding part S_i, all in $O(\alpha + \beta \log n)$ rounds, with high probability.*

Proof. To construct these BFSs, we make use of the *random delays* technique [LMR94]. Divide time into phases, each having $O(\log n)$ rounds. Delay the start of each BFS by a random delay, a uniformly chosen number of phases in $[0, \alpha/\log n]$. Once A BFS starts, it proceeds at a speed of one hop per phase. If a node v receives the BFS token of a leader s_i in phase j for the first time, it forwards this token to all its neighbors in shortcut subgraph $G[S_i] + H_i$ in phase $j+1$. We claim that over all BFSs, with high probability, there is a total of at most $O(\log n)$ tokens that need to go through each edge e in each phase, and thus, since a phase has $O(\log n)$ rounds, there is time for sending all of these $O(\log n)$ tokens along the edge. For a given BFS, the probability that it has to go through a particular edge e in a given phase is at most $\frac{\log n}{\alpha}$. Hence, over all the at most α BFSs that can go through e—because of the definition of the shortcuts—we expect at most $\log n$ of them to be scheduled to go through it in any particular phase. Since the random delays of different BFSs are chosen independently, we can apply a Chernoff bound and conclude that no more than $O(\log n)$ BFSs need to cross the edge in each phase, w.h.p. Hence, each phase has enough time for all the messages that need to go through an edge e in this phase. This allows us to run all the BFS simultaneously. We need only $\alpha/\log n + \beta$ phases, because each BFS starts by phase $\alpha/\log n$ and each BFS runs for at most β phases. Therefore, all BFSs finish in $O(\alpha + \beta \log n)$ rounds. Each node remembers for each BFS its parent from whom it received the related BFS token for the first time. The nodes can also easily known their depth in the BFS trees, simply by adding a standard hop-counter to each BFS token, which starts equal to 0 at the root and gets incremented before being passed on to the neighbors. □

6.2.2 Computing an MST via Shortcuts

In this subsection, we present the proof of Theorem 6.1.2, which uses shortcuts to compute an MST. For simplicity, we first restate the theorem here.

Theorem 6.1.2 *Suppose that the graph family $\mathcal{G}$ is such that for each graph $G \in \mathcal{G}$, and any partition of G into vertex-disjoint connected subsets $S_1, \ldots, S_N$, we can find an α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} \leq K$. Then, there is a randomized distributed MST algorithm that computes an MST in $O(\alpha \log n + \beta \log^2 n) = \tilde{O}(K)$ rounds, with high probability, in any graph from the family $\mathcal{G}$. Here, K can be a function of the family $\mathcal{G}$, and it can depend on n and D.*

Proof of Theorem 6.1.2. The high-level idea is to incorporate low-congestion shortcuts into (a variant of) the classic 1926 approach of Boruvka [NMN01]. Let us first recall this variant of Boruvka's approach.

The Algorithm Outline: We have $O(\log n)$ iterations where we gradually grow a forest until we reach a spanning tree. We start with the trivial forest where each node forms its own component of the forest, that is, each node is one separate part in our partition of G.

In each iteration, each part S_i will have a leader node $s_i \in S_i$. Each part S_i suggests a merge along the edge with exactly one endpoint in S_i that has the smallest weight among such edges. This is called the *minimum weight outgoing edge (MWOE)*. It is well-known that all such edges belong to MST[2]. We soon explain how to compute these min-weight outgoing edges, one per part, using shortcuts. Let us for now continue with the high-level explanation of how to use these edges to merge parts. We restrict the merge shapes to be *star* shapes, using a simple random coin idea: toss a random coin per part and then allow only merges centered on head-parts, each accepting incoming suggested merge-edges from tail-parts. The leader of this head-part becomes the leader of the merged new part.

Let N be the current number of connected components of the forest. If $N = 1$, we are done already. Otherwise, each component suggests one merge edge. Each suggested merge edge is accepted for a merge with probability $1/4$. Thus, we expect to have $N/4$ accepted merge edges. Hence, the number of (excess) parts shrinks in expectation by a constant factor. Since we start with at most n components, after $O(\log n)$ iterations, the expected number of (excess) parts is down to $\frac{1}{n^2}$. Using Markov's inequality, this implies that, with high probability, we have at most one component, that is, we have reached a spanning tree.

The Computational Steps of Each Iteration What remains is to explain how to find the merge edges, and how to perform the merges. Both of these steps make use of low-congestion shortcuts. More concretely, we use the same procedure as the method for simultaneously growing BFSs, as explained in the proof of Claim 6.2.1, to do communications inside each part. In particular, using this method, we make all nodes learn (1) the coin tossed by their leader, (2) the minimum-weight outgoing edge of their part, and finally (3) the ID of their new part leader. We next explain how to perform each of these steps in $O(\alpha + \beta \log n)$ rounds. Thus, each iteration takes $O(\alpha + \beta \log n)$ rounds. Since the MST algorithm uses $O(\log n)$ iterations, the overall complexity becomes $O(\alpha \log n + \beta \log^2 n) = \tilde{O}(K)$.

- Item (1)—which is to let each node know the coin toss of its part leader—is by a simple repetition of the messages sent in growing the BFS, while now the message starting at the root also carries the random bit flipped by the leader.

- Item (2)—which is to let each node know the the minimum weight outgoing edge of its component—can be also computed by two iterations of the same procedure. Let each node v set $c(v)$ to be the minimum-weight outgoing edge among edges incident

[2]This assumes that the edge-weights are unique, which is a common assumption in this area. Furthermore, if the goal is to compute one MST, assuming unique edge weights is without loss of generality. The reason is as follows: we can append the identifier of the edge—composed of the identifiers of its two endpoints—to its weight in a manner that makes the edge weights unique, and guarantees that the MST according to the new weights is one of the MSTs according to the original weights.

to v. The objective is that each node $v \in S_i$ learns the weight of the minimum-weight outgoing edge among edges all edges incident on part S_i. Each node v starts with its own smallest weight-outgoing edge and its weight $c(v)$. Notice that node v can easily find $c(v)$ by first receiving from all neighbors the part leader IDs of their parts and then only considering the smallest of those edges having the other endpoint in a different part. Then, we perform a convergecast on the BFS trees, by a simple reversal of the schedule of the growth of the BFSs. This convergecast goes from the leaves to the root, maintaining the minimum value seen, and thus eventually delivering the minimum-weight outgoing edge to the part leader. The information about this edge can be delivered to all nodes of the part by repeating the BFS schedule in the forward direction, from the root to the leaves. At the end of these steps, each edge chosen for merge can be identified by its endpoint in the part holding a tail coin.

- Item (3)—which is to let each node knows its new part leader ID—is performed as follows. We define the part leader ID to be the leader of the center part of the merge, who had a head coin. This ID is already delivered to the physical endpoint of the merge edge in the tail part. We perform a convergecast and then a broadcast along the BFS tree, similar to above, to deliver this ID first to the root of the part BFS, and then to all its nodes. This ensures that all nodes know the ID of their new leader. □

Better MST for General Graphs As stated in Corollary 6.1.3, the above approach leads to an $O((D + \sqrt{n}) \log^2 n)$-round distributed MST algorithm in general graphs. We next explain how to improve this bound to $O((D + \sqrt{n}) \log n)$, by leveraging the special structure of the shortcuts in Corollary 6.1.3. This results in a simple MST algorithm that is only a $\log n$ factor slower than the $O((D + \sqrt{n} \log^* n)$ algorithm of Kutten and Peleg [KP95].

Notice that the choice of the shortcut in the proof of Corollary 6.1.3 treats parts in two different categories: (1) large parts S_i for which $|S_i| \geq \sqrt{n}$ and thus we set $H_i = G$, and (2) small parts S_i for which $|S_i| \leq \sqrt{n}$ and thus we set $H_i = \emptyset$. For the latter category of small parts, the BFS-growth, and BFS-based broadcast and converge-cast all happen inside the part and we can perform these all in parallel in $\sqrt{n}$ rounds, as each of these parts has diameter at most $\sqrt{n}$ and they are disjoint. What takes somewhat more care is the communications of the large parts. However, there are at most $n/\sqrt{n} = \sqrt{n}$ large parts. Hence, we can let the broadcast and convergecast of these parts happen on the BFS of the whole graph G, simultaneously. Using standard pipelining techniques [Pel00], we can see that we can perform one convergecast per each large part, all in parallel, in $O(D + \sqrt{n})$ rounds. The same holds also for their broadcasts. Therefore, we conclude that we can perform one iteration of Boruvka in general graphs in $O(D + \sqrt{n})$ rounds. This leads to overall round complexity of $O((D + \sqrt{n}) \log n)$.

We note that this is slower than the $O(D + \sqrt{n} \log^* n)$ algorithm of Kutten and Peleg [KP95]. However, we believe that this approach is considerably simpler than that of [KP95], and it perhaps provides a more suitable choice for teaching distributed MST algorithms.

THIS PAGE IS LEFT BLANK INTENTIONALLY.

Chapter 7

Minimum Edge Cut Approximation

7.1 Introduction & Related Work

Finding minimum cuts or approximately minimum cuts are classical and fundamental algorithmic graph problems with many important applications. In particular, minimum edge cuts and their size (i.e., the edge connectivity) are relevant in the context of networks, where edge weights might represent link capacities and therefore edge connectivity can be interpreted as the throughput capacity of the network. Decomposing a network using small cuts helps designing efficient communication strategies and finding communication bottlenecks (see, e.g., [PQ82, KS93]). Both the exact and approximate variants of the minimum cut problem have received extensive attention in the domain of centralized algorithms (cf. Section 7.1.2 for a brief review of the results in the centralized setting). This line of research has led to (almost) optimal centralized algorithms with running times $\tilde{O}(m+n)$ [Kar00] for the exact version and $O(m+n)$ [Mat93] for constant-factor approximations, where n and m are the numbers of nodes and edges, respectively.

As indicated by Elkin [Elk04a] and Das Sarma et al. [DHK+12], the problem has remained essentially open in the distributed setting. In the LOCAL model [Pel00] where in each round, a message of unbounded size can be sent over each edge, the problem has a trivial time complexity of $\Theta(D)$ rounds, where D is the (unweighted) diameter of the network. The problem is therefore more interesting and also practically more relevant in models where messages are of some bounded size B. The standard model incorporating this restriction is the CONGEST model [Pel00], a synchronous message passing model where in each time unit, B bits can be sent over every link (in each direction). It is often assumed that $B = \Theta(\log n)$. The only known non-trivial result is an elegant lower bound by Das Sarma et al. [DHK+12] showing that any α-approximation of the minimum cut in weighted graphs requires at least $\Omega(D + \sqrt{n/(B \log n)})$ rounds.

7.1.1 Our Contribution

We present three distributed minimum edge cut approximation algorithms for undirected weighted graphs, with successively better approximations but increasing round-complexities. The complexities of all these algorithms match the lower bound of [DHK+12] up to at most logarithmic factors. We also extend the lower bound of [DHK+12] to unweighted graphs and multigraphs.

Algorithms

Our first algorithm, presented in Section 7.4, with high probability[1] finds a cut of size at most $O(\varepsilon^{-1}\lambda)$, for any $\epsilon \in (0,1)$ and where λ is the edge connectivity, i.e., the size of the minimum cut in the network. The time complexity of this algorithm is $O(D) + O(n^{1/2+\epsilon}\log^3 n \log\log n \log^* n)$. The algorithm is based on a simple and novel approach for analyzing random edge sampling, a tool that has proven extremely successful also for studying the minimum cut problem in the centralized setting (see, e.g., [KS93]). Our analysis is based on *random layering*, and we believe that the approach might also be useful for studying other connectivity-related questions. Assume that each edge $e \in E$ of an unweighted multigraph $G = (V, E)$ is independently sampled and added to a subset $E' \subset E$ with probability p. For $p \leq \frac{1}{\lambda}$, the graph $G' = (V, E')$ induced by the sampled edges is disconnected with at least a constant probability (just consider one min-cut). In Section 7.3, we use random layering to show that if $p = \Omega(\frac{\log n}{\lambda})$, the sampled graph G' is connected w.h.p. This bound is optimal and was known previously, with two elegant proofs: [LP72] and [Kar94b]. Our proof is simple and self-contained and it serves as a basis for our algorithm in Section 7.4.

The second algorithm, presented in Section 7.5, finds a cut with size at most $(2+\varepsilon)\lambda$, for any constant $\varepsilon > 0$, in time $O((D + \sqrt{n}\log^* n)\log^2 n \log\log n \cdot \frac{1}{\varepsilon^5})$. This algorithm combines the general approach of Matula's centralized $(2+\varepsilon)$-approximation algorithm [Mat93] with Thurimella's algorithm for sparse edge-connectivity certificates [Thu97] and with the famous random edge sparsification technique of Karger (see e.g., [Kar94a]).

The third algorithm, presented in Section 7.6, is somewhat slower but it still runs in $\tilde{O}(D + \sqrt{n})$ in general graphs and obtains a better approximation factor of $1 + \varepsilon$. It uses a mix of a number of technical ingredients: the *tree-packing* approach of Thorup [Tho01], the sampling idea of Karger [Kar94b], our MST algorithm from the previous chapter, and some further applications of low-congestion shortcuts, and finally some *sketching* ideas that we present. We note that the distributed min-cut $(1+\varepsilon)$-approximation of Nanongkai and Su [NS14] also uses the first two of these ingredients to achieve a $(1+\varepsilon)$-approximation of minimum-cut in $\tilde{O}(D + \sqrt{n})$ rounds. We believe that our result is more modular and more general. This is especially true because it easily fits the framework of low-congestion shortcuts, and thus extends to more efficient algorithms for special graph families such as

[1] Recall from Chapter 2 that we use the phrase *with high probability* (w.h.p.) to indicate probabilities greater than $1 - \frac{1}{n^c}$ for a desirably large constant $c \geq 2$.

planar networks or near-expanders, as we point out in Section 7.6.

Lower Bound:

To complement our upper bounds, we also extend the lower bound of Das Sarma et al. [DHK+12] to unweighted graphs and multigraphs. When the minimum cut problem (or more generally problems related to small edge cuts and edge connectivity) are in a distributed context, often the weights of the edges correspond to their capacities. It therefore seems reasonable to assume that over a link of twice the capacity, we can also transmit twice the amount of data in a single time unit. Consequently, it makes sense to assume that over an edge of weight (or capacity) $w \geq 1$, $O(w \log n)$ bits can be transmitted per round (or equivalently that such a link corresponds to w parallel links of unit capacity). The lower bound of [DHK+12] critically depends on having links with (very) large weight over which in each round only $O(\log n)$ bits can be transmitted. We generalize the approach of [DHK+12] and obtain the same lower bound result as in [DHK+12] for the weaker setting where edge weights correspond to edge capacities (i.e., the setting that can be modeled using unweighted multigraphs). Formally, we show that if Bw bits can be transmitted over every edge of weight $w \geq 1$, for every $\alpha \geq 1$ and sufficiently large λ, there are λ-edge-connected networks with diameter $O(\log n)$ on which computing an α-approximate minimum cut requires time at least $\Omega\big(\sqrt{n/(B \log n)}\big)$. Further, for unweighted simple graphs with edge connectivity λ, we show that for diameter at most $D = O\big(\frac{1}{\lambda} \cdot \sqrt{n/(\alpha\lambda B \log n)}\big)$ finding an α-approximate minimum cut or approximating the edge connectivity by a factor of α requires at least time $\Omega\big(\sqrt{n/(\alpha\lambda B \log n)}\big)$.

7.1.2 Related Work in the Centralized Setting

Starting in the 1950s [FF56, EFS56], the traditional approach to the minimum cut problem was to use max-flow algorithms (cf. [FF62] and [KS93, Section 1.3]). In the 1990s, three new approaches were introduced which go away from the flow-based method and provide faster algorithms: The first method, presented by Gabow [Gab91], is based on a matroid characterization of the min-cut and it finds a min-cut in $O(m + \lambda^2 n \log \frac{n}{m})$ steps, for any unweighted (but possibly directed) graph with edge connectivity λ. The second approach applies to (possibly) weighted but undirected graphs and is based on repeatedly identifying and contracting edges outside a min-cut until a min-cut becomes apparent (e.g., [NI92, Kar93, KS93]). The beautiful *random contraction algorithm* (RCA) of Karger [Kar93] falls into this category. In the basic version of RCA, the following procedure is repeated $O(n^2 \log n)$ times: contract uniform random edges one by one until only two nodes remain. The edges between these two nodes correspond to a cut in the original graph, which is a min-cut with probability at least $1/O(n^2)$. Karger and Stein [KS93] also present a more efficient implementation of the same basic idea, leading to total running time of $O(n^2 \log^3 n)$. The third method, which again applies to (possibly) weighted but undirected graphs, is due to Karger [Kar96] and is based on a "semiduality" between minimum cuts and maximum spanning tree packings.

This third method leads to the best known centralized minimum-cut algorithm [Kar00] with running time $O(m \log^3 n)$.

For the approximation version of the problem (in undirected graphs), the main known results are as follows. Matula [Mat93] presents an algorithm that finds a $(2+\varepsilon)$-minimum cut for any constant $\varepsilon > 0$ in time $O((m+n)/\varepsilon)$. This algorithm is based on a graph search procedure called *maximum adjacency search*. Based on a modified version of the random contraction algorithm, Karger [Kar94b] presents an algorithm that finds a $(1+\varepsilon)$-minimum cut in time $O(m + n \log^3 n/\varepsilon^4)$.

7.2 Preliminaries

Notations and Definitions

We usually work with an undirected weighted graph $G = (V, E, w)$, where V is a set of n vertices, E is a set of (undirected) edges $e = \{v, u\}$ for $u, v \in V$, and $w : E \to \mathbb{R}^+$ is a mapping from edges E to positive real numbers. For each edge $e \in E$, $w(e)$ denotes the weight of edge e. In the special case of unweighted graphs, we simply assume $w(e) = 1$ for each edge $e \in E$.

For a given non-empty proper subset $C \subset V$, we define the cut $(C, V \setminus C)$ as the set of edges in E with exactly one endpoint in set C. The size of this cut, denoted by $w(C)$ is the sum of the weights of the edges in set $(C, V \setminus C)$. The edge-connectivity $\lambda(G)$ of the graph is defined as the minimum size of $w(C)$ as C ranges over all nonempty proper subsets of V. A cut $(C, V \setminus C)$ is called *α-minimum*, for an $\alpha \geq 1$, if $w(C) \leq \alpha\lambda(G)$. When clear from the context, we sometimes use λ to refer to $\lambda(G)$.

Model

Throughout this chapter, we work with the E-CONGEST model. Recall that here, the execution proceeds in synchronous rounds and in each round, each node can send a message of size B bits to each of its neighbors. A typically standard case is $B = \Theta(\log n)$.

For upper bounds, for simplicity we assume[2] that $B = \Theta(\log n)$. For upper bounds, we further assume that B is large enough so that a constant number of node identifiers and edge weights can be packed into a single message. For $B = \Theta(\log n)$, this implies that each edge weight $w(e)$ is at most (and at least) polynomial in n. W.l.o.g., we further assume that edge weights are normalized and each edge weight is an integer in the range $\{1, \ldots, n^{\Theta(1)}\}$. Thus, we can also view a weighted graph as a multi-graph in which all edges have unit weight and multiplicity at most $n^{\Theta(1)}$ (but still only $O(\log n)$ bits can be transmitted over all these parallel edges together).

[2]Note that by choosing $B = b \log n$ for some $b \geq 1$, in all our upper bounds, the term that does not depend on D could be improved by a factor $\sqrt{b}$.

For lower bounds, we assume a weaker model where $B \cdot w(e)$ bits can be sent in each round over each edge e. To ensure that at least B bits can be transmitted over each edge, we assume that the weights are scaled such that $w(e) \geq 1$ for all edges. For integer weights, this is equivalent to assuming that the network graph is an unweighted multigraph where each edge e corresponds to $w(e)$ parallel unit-weight edges.

Problem Statement

In the problem of computing an *α-approximation of the minimum cut*, the goal is to find a cut $(C^*, V \setminus C^*)$ that is α-minimum. To indicate this cut in the distributed setting, each node v should know whether $v \in C^*$. In the problem of *α-approximation of the edge-connectivity*, all nodes must output an estimate $\tilde{\lambda}$ of λ such that $\tilde{\lambda} \in [\lambda, \lambda\alpha]$. In randomized algorithms for these problems, the time complexity and correctness guarantees are required to hold with high probability.

Black-Box Algorithms

We again make frequent use of Thurimella's *connected component identification* algorithm [Thu97] as a subroutine. This algorithm itself builds on the MST algorithm of Kutten and Peleg [KP95]. We next state the guarantee that this subroutine provides.

Thurimella's Algorithm: Given a graph $G = (V, E)$ and a subgraph $H = (V, E')$ such that $E' \subseteq E$, Thurimella's algorithm identifies the connected components of H by assigning a label $\ell(v)$ to each node $v \in V$ such that two nodes get the same label iff they are in the same connected component of H. The time complexity of the algorithm is $O(D + \sqrt{n}\log^* n)$ rounds, where D is the (unweighted) diameter of G. Moreover, it is easy to see that the algorithm can be made to produce labels $\ell(v)$ such that $\ell(v)$ is equal to the smallest (or the largest) ID in the connected component of H that contains v. Furthermore, the connected component identification algorithm can also be used to test whether the graph H is connected (assuming that G is connected). H is not connected if and only if there is an edge $\{u, v\} \in E$ such that $\ell(u) \neq \ell(v)$. If some node u detects that for some neighbor v (in G), $\ell(u) \neq \ell(v)$, u broadcasts *not connected*. Connectivity of H can therefore be tested in $O(D)$ additional rounds. We refer to this as Thurimella's *connectivity-tester* algorithm. Finally, we remark that the same algorithms can also be used to solve k independent instances of the connected component identification problem or k independent instances of the connectivity-testing problem in $O(D + k\sqrt{n}\log^* n)$ rounds. This is achieved by pipelining the messages of the broadcast parts of different instances.

7.3 Edge Sampling and The Random Layering Technique

Here, we study the process of random edge-sampling and present a simple technique, which we call *random layering*, for analyzing the connectivity of the graph obtained through sampling. This technique also forms the basis of our min-cut approximation algorithm presented in the next section.

Edge Sampling Consider an arbitrary unweighted multigraph $G = (V, E)$. Given a probability $p \in [0, 1]$, we define an *edge sampling experiment* as follows: choose a subset $S \subseteq E$ by including each edge $e \in E$ in set S independently with probability p. We call the graph $G' = (V, S)$ *the sampled subgraph.*

We use the *random layering technique* to answer the following *network reliability* question: "How large should p be, as a function of the minimum cut size λ, so that the sampled graph is connected w.h.p.?"[3] Considering just one cut of size λ we see that if $p \leq \frac{1}{\lambda}$, then the probability that the sampled subgraph is connected is at most $\frac{1}{e}$. We show that $p \geq \frac{20 \log n}{\lambda}$ suffices so that the sampled subgraph is connected w.h.p. Note that this is non-trivial as a graph has exponentially many cuts. It is easy to see that this bound is asymptotically optimal [LP72].

Theorem 7.3.1. *Consider an arbitrary unweighted multigraph $G = (V, E)$ with edge connectivity λ and choose subset $S \subseteq E$ by including each edge $e \in E$ in set S independently with probability p. If $p \geq \frac{20 \log n}{\lambda}$, then the sampled subgraph $G' = (V, S)$ is connected with probability at least $1 - \frac{1}{n}$.*

We remark that this result was known previously, via two different proofs by Lomonosov and Polesskii [LP72] and Karger [Kar94b]. The Lomonosov-Polesskii proof [LP72] uses an interesting coupling argument and shows that among the graphs of a given edge-connectivity λ, a cycle of length n with edges of multiplicity $\lambda/2$ has the smallest probability of remaining connected under random sampling. Karger's proof [Kar94b] uses the powerful fact that the number of α-minimum cuts is at most $O(n^{2\alpha})$ and then uses basic probability concentration arguments (Chernoff and union bounds) to show that, w.h.p., each cut has at least one sampled edge. There are many known proofs for the $O(n^{2\alpha})$ upper bound on the number of α-minimum cuts (see [Kar00]); an elegant argument follows from Karger's *random contraction algorithm* [Kar93].

Our proof of Theorem 7.3.1 is simple and self-contained, and it is the only one of the three approaches that extends to the case of random vertex failures[4] [CHGK13, Theorem 1.5].

[3] A rephrased version is, how large should the edge-connectivity λ of a network be such that it remains connected w.h.p. if each edge fails with probability $1 - p$.

[4] There, the question is, how large the vertex sampling probability p has to be chosen, as a function of vertex connectivity k, so that the vertex-sampled graph is connected, w.h.p. The extension to the vertex version requires important modifications and leads to $p = \Omega(\frac{\log n}{\sqrt{k}})$ being a sufficient condition. Refer to [GK13, Section 3] for details.

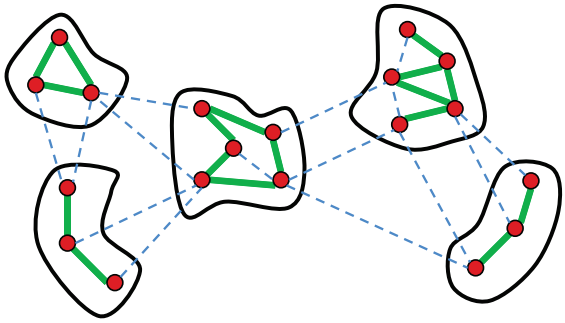


Figure 7-1: Graph G_i and its connected components. The green solid links represent edges in S_{i-} and the blue dashed links represent $E \setminus S_{i-}$.

Proof of Theorem 7.3.1. Let $L = 20 \log n$. For each edge $e \in E$, we independently choose a uniform random *layer number* from the set $\{1, 2, \ldots, L\}$. Intuitively, we add the sampled edges layer by layer and show that with the addition of the sampled edges of each layer, the number of connected components goes down by at least a constant factor, with at least a constant probability, and independently of the previous layers. After $L = \Theta(\log n)$ layers, connectivity is achieved w.h.p.

We start by presenting some notations. For each $i \in \{1, \ldots, L\}$, let S_i be the set of sampled edges with layer number i and let $S_{i-} = \bigcup_{j=1}^{i} S_j$, i.e., the set of all sampled edges in layers $\{1, \ldots, i\}$. Let $G_i = (V, S_{i-})$ and let M_i be the number of connected components of the graph G_i. We show that $M_L = 1$, w.h.p.

For any $i \in [1, L-1]$, since $S_{i-} \subseteq S_{(i+1)-}$, we have $M_{i+1} \leq M_i$. Consider the indicator variable X_i such that $X_i = 1$ iff $M_{i+1} \leq 0.87 M_i$ or $M_i = 1$. We show the following claim, after which, applying a Chernoff bound completes the proof.

Claim 7.3.2. *For all $i \in [1, L-1]$ and $T \subseteq E$, we have* $\Pr[X_i = 1 | S_{i-} = T] \geq 1/2$.

To prove this claim, we use *the principle of deferred decisions* [Knu96] to view the two random processes of sampling edges and layering them. More specifically, we consider the following process: first, each edge is sampled and given layer number 1 with probability p/L. Then, each remaining edge is sampled and given layer number 2 with probability $\frac{p/L}{1-p/L} \geq p/L$. Similarly, after determining the sampled edges of layers 1 to i, each remaining edge is sampled and given layer number $i+1$ with probability $\frac{p/L}{1-(i\,p)/L} \geq p/L$. After doing this for L layers, any remaining edge is considered not sampled and it receives a random layer number from $\{1, 2, \ldots, L\}$. It is easy to see that in this process, each edge is independently sampled with probability exactly p and each edge e gets a uniform random layer number from $\{1, 2, \ldots, L\}$, chosen independently of the other edges and also independently of whether e is sampled or not.

Fix a layer $i \in [1, \ldots, L-1]$ and a subset $T \subseteq E$. Let $S_{i-} = T$ and consider graph $G_i = (V, S_{i-})$. Section 7.3 presents an example graph G_i and its connected components. If $M_i = 1$ meaning that G_i is connected, then $X_i = 1$. Otherwise, suppose that $M_i \geq 2$. For each component $\mathcal{C}$ of G_i, call the component *bad* if $(\mathcal{C}, V \setminus \mathcal{C}) \cap S_{i+1} = \emptyset$. That is, $\mathcal{C}$ is

bad if after adding the sampled edges of layer $i+1$, $\mathcal{C}$ does not get connected to any other component. We show that $\Pr[\mathcal{C} \textit{ is bad}] \leq \frac{1}{e}$.

Since G is λ-edge connected, we have $w(C) \geq \lambda$. Moreover, none of the edges in $(\mathcal{C}, V \setminus \mathcal{C})$ is in S_{i-}. Thus, using the principle of deferred decisions as described, each of the edges of the cut $(\mathcal{C}, V \setminus \mathcal{C})$ has probability $\frac{p/L}{1-(i\,p)/L} \geq p/L$ to be sampled and given layer number $i+1$, i.e., to be in S_{i+1}. Since $p \geq \frac{20 \log n}{\lambda}$, the probability that none of the edges $(\mathcal{C}, V \setminus \mathcal{C})$ is in set S_{i+1} is at most $(1-p/L)^{\lambda} \leq 1/e$. Thus, $\Pr[\mathcal{C} \textit{ is bad}] \leq 1/e$.

Now let Z_i be the number of bad components of G_i. Since each component is bad with probability at most $1/e$, we have $\mathbb{E}[Z_i] \leq M_i/e$. Using Markov's inequality, we get $\Pr[Z_i \geq 2M_i/e] \leq 1/2$. Since each component that is not bad gets connected to at least one other component (when we look at graph G_{i+1}), we have $M_{i+1} \leq Z_i + \frac{(M_i - Z_i)}{2} = \frac{M_i + Z_i}{2}$. Therefore, with probability at least $1/2$, we have $M_{i+1} \leq \frac{1+2/e}{2} M_i < 0.87 M_i$. This means that $\Pr[X_i = 1] \geq 1/2$, which concludes the proof of the claim.

Now using the claim, we get that $\mathbb{E}[\sum_{i=1}^{L-1} X_i] \geq 10 \log n$. A Chernoff bound then shows that $\Pr[\sum_{i=1}^{L-1} X_i \geq 5 \log n] \geq 1 - \frac{1}{n}$. This means that w.h.p, $M_L \leq \frac{n}{2^{\log n}} = 1$. That is, w.h.p, $G_L = (V, S) = (V, S_{L-}) = G'$ is connected. □

Theorem 7.3.1 provides a very simple approach for finding an $O(\log n)$-approximation of the edge connectivity of a network graph G in $O(D + \sqrt{n} \log^2 n \log^* n)$ rounds, simply by trying exponentially growing sampling probabilities and checking the connectivity.

Corollary 7.3.3. *There exists a distributed algorithm that for any unweighted multi-graph $G = (V, E)$, in $O(D + \sqrt{n} \log^2 n \log^* n)$ rounds, finds an approximation $\tilde{\lambda}$ of the edge connectivity such that $\tilde{\lambda} \in [\lambda, \lambda \cdot \Theta(\log n)]$ with high probability.*

Proof. We run $\Theta(\log^2 n)$ edge-sampling experiments: $\Theta(\log n)$ experiments for each sampling probability $p_j = 2^{-j}$ where $j \in [1, \Theta(\log n)]$. From Theorem 7.3.1, we know that, if $p_j \geq \Omega\big(\frac{\log n}{\lambda}\big)$, the sampled graph is connected with high probability. On the other hand, by focusing on just one minimum cut, we see that if $p_j \leq \frac{1}{\lambda}$, then the probability that the sampled graph is connected is at most $3/4$. Let p^* be the smallest sampling probability p_j such that at least $9/10$ of the sampling experiments with probability p_j lead to sampled graph being connected. With high probability, $\tilde{\lambda} := \frac{1}{p^*}$ is an $O(\log n)$-approximation of the edge-connectivity. To check whether each sampled graph is connected, we use Thurimella's connectivity-tester (refer to Section 7.2), and doing that for $\Theta(\log^2 n)$ different sampled graphs requires $O(D + \sqrt{n} \log^2 n \log^* n)$ rounds. □

7.4 A Constant Approximation of Minimum Edge Cut

Now we use random layering to design a min-cut approximation algorithm. We first present the outline of the algorithm and its major ideas. Then, Section 7.4.3 explains how to put the pieces together to prove Theorem 7.4.1.

Algorithm 1 An $O(\log n)$ Approximation Algorithm for the Edge-Connectivity

for $i = 1$ to $\log n$ **do**
  for $j = 1$ to $4 \log n$ **do**
    Choose subset $E_i^j \subseteq E$ by adding each edge $e \in E$ to E_i^j independently with probability 2^{-i}

Run Thurimella's connectivity-tester on graph G with $\Theta(\log^2 n)$ subgraphs $H_i^j = (V, E_i^j)$, in $O(D + \sqrt{n} \log^2 n \log^* n)$ rounds. ▷ Refer to Section 7.2 for Thurimella's connectivity-tester algorithm.

for $i = 1$ to $\Theta(\log n)$ **do**
  for $j = 1$ to $c \log n$ **do**
    if graph $G = (V, E_i^j)$ is connected **then**
      $X_i^j \leftarrow 1$
    else
      $X_i^j \leftarrow 0$
    $X_i \leftarrow \sum_{j=1}^{c \log n} X_i^j$
$i^* \leftarrow \arg\max_{i \in [1, \Theta(\log n)]} (X_i \geq 9c \log n/10)$
$\tilde{\lambda} \leftarrow 2^{i^*}$

Return $\tilde{\lambda}$

Theorem 7.4.1. *There is a distributed algorithm that, for any $\epsilon \in (0, 1)$, finds an $O(\epsilon^{-1})$-minimum cut in $O(D) + O(n^{0.5+\epsilon} \log^3 n \log \log n \log^* n)$ rounds, w.h.p.*

7.4.1 Algorithm Outline

The algorithm is based on closely studying the sampled graph when the edge-sampling probability is between the two extremes of $\frac{1}{\lambda}$ and $\frac{\Theta(\log n)}{\lambda}$. Throughout this process, we identify a set $\mathcal{F}$ of $O(n \log n)$ cuts such that, with at least a 'reasonably large probability', $\mathcal{F}$ contains at least one 'small' cut.

The Crux of the Algorithm: Sample edges with probability $p = \frac{\epsilon \log n}{2\lambda}$ for a small $\epsilon \in (0, 1)$. Also, assign each edge to a random layer in $[1, \ldots, L]$, where $L = 20 \log n$. For each layer $i \in [1, \ldots, L-1]$, let S_i be the set of sampled edges of layer i and let $S_{i-} = \bigcup_{j=1}^{i} S_j$. For each layer $i \in [1, \ldots, L-1]$, for each component $\mathcal{C}$ of graph $G_i = (V, S_{i-})$, add the cut $(\mathcal{C}, V \setminus \mathcal{C})$ to the collection $\mathcal{F}$. Since in each layer we add at most n new cuts and there are $L = O(\log n)$ layers, we collect $O(n \log n)$ cuts in total.

We show that with probability at least $n^{-\epsilon}/2$, at least one of the cuts in $\mathcal{F}$ is an $O(\epsilon^{-1})$-minimum cut. Note that thus repeating the experiment for $\Theta(n^{\epsilon} \log n)$ times is enough to get that an $O(\epsilon^{-1})$-minimum cut is found w.h.p.

Theorem 7.4.2. *Consider performing the above sampling and layering experiment with edge*

sampling probability $p = \frac{\epsilon \log n}{2\lambda}$ *for* $\epsilon \in (0,1)$ *and* $L = 20 \log n$ *layers. Then,*

$$\Pr[\mathcal{F} \text{ contains an } O(\epsilon^{-1})\text{-minimum cut}] \geq n^{-\epsilon}/2.$$

Proof. Fix an edge sampling probability $p = \frac{\epsilon \log n}{2\lambda}$ for an $\epsilon \in (0,1)$ and let $\alpha = 40\epsilon^{-1}$. We say that a sampling and layering experiment is *successful* if $\mathcal{F}$ contains an α-minimum cut or if the sampled graph $G_L = (V, S_{L-})$ is connected. We first show that each experiment is *successful* with probability at least $1 - \frac{1}{n}$. The proof of this part is very similar to that of Theorem 7.3.1.

For an arbitrary layer number $1 \leq i \leq L-1$, consider graph $G_i = (V, S_{i-})$. If $M_i = 1$ meaning that G_i is connected, then G_L is also connected. Thus, in that case, the experiment is successful and we are done. In the more interesting case, suppose $M_i \geq 2$. For each component $\mathcal{C}$ of G_i, consider the cut $(\mathcal{C}, V \setminus \mathcal{C})$. If any of these cuts is α-minimum, then the experiment is successful as then, set $\mathcal{F}$ contains an α-minimum cut. On the other hand, suppose that for each component $\mathcal{C}$ of G_i, we have $w(C) \geq \alpha\lambda$. Then, for each such component $\mathcal{C}$, each of the edges of the cut $(\mathcal{C}, V \setminus \mathcal{C})$ has probability $\frac{p/L}{1-(ip)/L} \geq p/L$ to be in set S_{i+1} and since $w(\mathcal{C}) \geq \alpha\lambda$, where $\alpha = 20\epsilon^{-1}$, the probability that none of the edges of this cut in set S_{i+1} is at most $(1-p/L)^{\alpha\lambda} \leq e^{\frac{p}{L}\cdot\alpha\lambda} = e^{-\frac{\epsilon\log n}{2\lambda}\cdot\frac{1}{L}\cdot\frac{40}{\epsilon}\cdot\lambda} = 1/e$. Hence, the probability that component $\mathcal{C}$ is *bad* as defined in the proof of Theorem 7.3.1 (i.e., in graph G_{i+1}, it does not get connected to any other component) is at most $1/e$. The rest of the proof can be completed exactly as the last paragraph of of the proof of Theorem 7.3.1, to show that

$$\Pr[\textit{successful experiment}] \geq 1 - 1/n.$$

Thus we have a bound on the probability that $\mathcal{F}$ contains an α-minimum cut or that the sampled graph $G = (V, S_{L-})$ is connected. However, in Theorem 7.4.2, we are only interested in the probability of $\mathcal{F}$ containing an α-minimum cut. Using a union bound, we know that

$$\Pr[\textit{successful experiment}] \leq \Pr[\mathcal{F} \textit{ contains an } \alpha\textit{-min cut}] + \Pr[G_L \textit{ is connected}].$$

On the other hand,

$$\Pr[G_L \textit{ is connected}] \leq 1 - n^{-\epsilon}.$$

This is because, considering a single mininmum cut of size λ, the probability that none of the edges of this cut are sampled, in which case the sampled subgraph is disconnected, is $(1 - \frac{\epsilon \log n}{2\lambda})^{\lambda} \geq n^{-\epsilon}$. Hence, we can conclude that

$$\Pr[\mathcal{F} \textit{ contains an } \alpha\textit{-min cut}] \geq (1 - 1/n) - (1 - n^{-\epsilon}) = n^{-\epsilon} - 1/n \geq n^{-\epsilon}/2.$$

□

Remark: It was brought to our attention that the approach of Theorem 7.4.2 bears some cosmetic resemblance to the technique of Goel, Kapralov and Khanna [GKK10]. As discussed

in personal communication with Kapralov during 2013, the approaches are fundamentally different. The only similarity is having $O(\log n)$ repetitions of sampling. In [GKK10], the objective is to estimate the *strong connectivity* of edges via a streaming algorithm. See [GKK10] for related definitions and note also that strong connectivity is (significantly) different from (standard) connectivity. In a nutshell, [GKK10] uses $O(\log n)$ iterations of sub-sampling, each time further sparsifying the graph until at the end, all edges with strong connectivity less than a threshold are removed (and identified) while edges with strong connectivity that is a $\Theta(\log n)$ factor larger than the threshold are preserved (proven via Benczur-Karger's sparsification).

7.4.2 Testing Cuts

So far we know that $\mathcal{F}$ contains an α-minimum cut with a reasonable probability. We now need to devise a distributed algorithm to read or test the sizes of the cuts in $\mathcal{F}$ and find that α-minimum cut, in $O(D) + \tilde{O}(\sqrt{n})$ rounds. In the remainder of this section, we explain our approach to this part.

Consider a layer i and the graph $G_i = (V, S_{i-})$. For each component $\mathcal{C}$ of G_i, $diam(\mathcal{C})$ rounds is enough to read the size of the cut $(\mathcal{C}, V \setminus \mathcal{C})$ such that all the nodes in component $\mathcal{C}$ know this size. However, $diam(\mathcal{C})$ can be considerably larger than $D = diam(G)$ and thus, this method would not lead to a round complexity of $\tilde{O}(D + \sqrt{n})$. To overcome this problem, notice that we do not need to read the exact size of the cut $(\mathcal{C}, V \setminus \mathcal{C})$. Instead, it is enough to devise a *test* that *passes* w.h.p. if $w(C) \leq \alpha\lambda$, and *does not pass* w.h.p. if $w(C) \geq (1+\delta)\alpha\lambda$, for a small constant $\delta \in (0, 1/4)$. In the distributed realization of such a test, it would be enough if all the nodes in $\mathcal{C}$ consistently know whether the test passed or not. Next, we explain a simple algorithm for such a test. This test itself uses random edge sampling. Given such a test, in each layer $i \in [1, \ldots, L-1]$, we can test all the cuts and if any cut passes the test, meaning that, w.h.p., it is a $((1+\delta)\alpha)$-minimum cut, then we can pick such a cut.[5]

Lemma 7.4.3. *Given a subgraph $G' = (V, E')$ of the network graph $G = (V, E)$, a threshold κ and $\delta \in (0, 1/4)$, there exists a randomized distributed cut-tester algorithm with round complexity $\Theta\big(D + \frac{1}{\delta^2}\sqrt{n}\log n \log^* n\big)$ such that, w.h.p., for each node $v \in V$, we have: Let $\mathcal{C}$ be the connected component of G' that contains v. If $w(\mathcal{C}) \leq \kappa/(1+\delta)$, the test passes at v, whereas if $w(\mathcal{C}) \geq \kappa(1+\delta)$, the test does not pass at v.*

Proof of Lemma 7.4.3. For pseudo-code, see Algorithm 2. We first run Thurimella's connected component identification algorithm (refer to Section 7.2) on graph G for subgraph G', so that each node $v \in V$ knows the smallest ID in its connected component of graph G'. Then, each node v adopts this label $componentID$ as its own ID (temporarily). Thus, nodes

[5] This can be done for example by picking the cut which passed the test and for which the related component has the smallest ID among all the cuts that passed the test.

Algorithm 2 Distributed cut tester vs. threshold κ @ node v

Given a subgraph $G' = (V, E')$ where $E' \subseteq E$, and a threshold κ

$v.componentID \leftarrow$ the smallest ID in the component of G' that contains v ▷ Using Thurimella's Component Identification Alg.

for $j = 1$ **to** $c \log(n)/\delta^2$ **do**
    Choose subset $E_i \subseteq E \setminus E'$ by adding each edge $e \in E \setminus E'$ to E_j independently with probability $1 - 2^{-\frac{1}{\kappa}}$

$\ell_j^{max}(v) \leftarrow$ the largest *componentID* in the connected component of $H_i = (V, E' \cup E_i)$ that contains v
$\ell_j^{min}(v) \leftarrow$ the smallest *componentID* in the connected component of $H_i = (V, E' \cup E_i)$ that contains v
▷ Using Thurimella's Component Identification on the $\Theta(\log n)$ values of i, simultaneously. (cf. Section 7.2)

$X_i \leftarrow 0$
for $i = 1$ **to** $\alpha \log n$ **do**
    if $\ell_j^{max}(v) \neq v.componentID$ **or** $\ell_j^{min}(v) \neq v.componentID$ **then** $X_i \leftarrow X_i + 1$

Test passes @ node v iff $X_i \leq \frac{c \log n}{2\delta^2}$

of each connected component of G' will have the same id. Now, the test runs in $\Theta(\log^2 n/\delta^2)$ *experiments*, each as follows: in the j^{th} experiment, for each edge $e \in E \setminus E'$, put edge e in set E_j with probability $p' = 1 - 2^{-\frac{1}{\kappa}}$. Then, run Thurimella's algorithm on graph G with subgraph $H_j = (V, E' \cup E_j)$ and with the new ids twice, such that at the end, each node v knows the smallest and the largest ID in its connected component of H_j. Call these new labels $\ell_j^{min}(v)$ and $\ell_j^{max}(v)$, respectively. For a node v of a component $\mathcal{C}$ of G_i, we have that $\ell_j^{min}(v) \neq v.id$ or $\ell_j^{max}(v) \neq v.id$ iff at least one of the edges of cut $(\mathcal{C}, V \setminus \mathcal{C})$ is sampled in E_j, i.e., $(\mathcal{C}, V \setminus \mathcal{C}) \cap E_j \neq \emptyset$. Thus, each node v of each component $\mathcal{C}$ knows whether $(\mathcal{C}, V \setminus \mathcal{C}) \cap E_j \neq \emptyset$ or not. Moreover, this knowledge is consistent between all the nodes of component $\mathcal{C}$. After $\Theta(\log n/\delta^2)$ experiments, each node v of component $\mathcal{C}$ considers the test *passed* iff v noticed $(\mathcal{C}, V \setminus \mathcal{C}) \cap E_j \neq \emptyset$ in at most half of the experiments. The full proof is as follows.

If a cut $(\mathcal{C}, V \setminus \mathcal{C})$ has size at most $\kappa/(1+\delta)$, then the probability that $(\mathcal{C}, V \setminus \mathcal{C}) \cap E_j \neq \emptyset$ is at most $1 - (1-p')^{\frac{\kappa}{1+\delta}} = 1 - 2^{-\frac{1}{1+\delta}} \leq 0.5 - \frac{\delta}{4}$. On the other hand, if cut $(\mathcal{C}, V \setminus \mathcal{C})$ has size at least $((1+\delta)\kappa)$, then the probability that $(\mathcal{C}, V \setminus \mathcal{C}) \cap E_j \neq \emptyset$ is at least $1 - (1-p')^{(1+\delta)\kappa} \geq 1 - 2^{-1+\delta} \geq 0.5 + \frac{\delta}{4}$. This $\Theta(\delta)$ difference between these probabilities gives us our basic tool for distinguishing the two cases. Since we repeat the experiment presented in Section 7.4.2 for $\Theta(\frac{\log n}{\delta^2})$ times, an application of Hoeffding's inequality shows that if cut $(\mathcal{C}, V \setminus \mathcal{C})$ has size at most $\kappa/(1+\delta)$, the test passes w.h.p., and if cut $(\mathcal{C}, V \setminus \mathcal{C})$ has size at least $\kappa(1+\delta)$, then, w.h.p., the test does not pass. □

7.4.3 Wrap up

Here, we explain how to put the pieces presented in Section 7.4 together to get the claim of Theorem 7.4.1.

Proof of Theorem 7.4.1. For simplicity, we first explain an $O(\varepsilon^{-1})$ minimum-cut approximation algorithm with time complexity $O((D + \sqrt{n}\log^* n \log n)n^\epsilon \log^2 n \log\log n)$. Then, we explain how to reduce it to the claimed bound of $O(D) + O(n^{0.5+\epsilon}\log^3 n \log\log n \log^* n)$ rounds.

We first find an $O(\log n)$ approximation $\tilde{\lambda}$ of λ, using Corollary 7.3.3, in time $O(D) + O(\sqrt{n}\log^*)\log^2 n)$. This complexity is subsumed by the complexity of the later parts. After this, we use $\Theta(\log\log n)$ guesses for a 2-approximation of λ in the form $\lambda'_i = \tilde{C}2^i$ where $i \in [-\Theta(\log\log n), \Theta(\log\log n)]$. For each such guess λ'_i, we have $n^\epsilon \log n$ *epochs* as follows:

In each *epoch*, we sample edges with probability $p = \frac{\epsilon \log n}{2\lambda'}$ and assign each edge to a random layer in $[1, \ldots, L]$, where $L = 20\log n$. For each layer $i \in [1, \ldots, L-1]$, we let S_i be the set of sampled edges of layer i and let $S_{i-} = \cup_{j=1}^{i} S_j$. Then, for each $i \in [1, \ldots, L]$, we use the Cut-Tester Algorithm (see Section 7.4.2) on graph G with subgraph $G_i = (V, S_{i-})$, threshold $\kappa = 50\lambda'/\epsilon$, and with parameter $\delta = 1/8$. This takes $O((D+\sqrt{n}\log n \log^* n)\log n)$ rounds (for each layer). If in a layer, a component passes the test, it means its cut has size at most $O(\lambda'/\epsilon)$, with high probability. To report the results of the test, we construct a BFS tree rooted in a leader in $O(D)$ rounds and we convergecast the minimum *componentID* that passed the test, in time $O(D)$. We then broadcast this *componentID* to all nodes and all nodes that have this *componentID* define the cut that is $O(\lambda'/\epsilon)$-minimum, with high probability.

Over all the guesses, we know that there is a guess λ'_j that is a 2-approximation of λ. In that guess, from Theorem 7.4.2 and a Chernoff bound, we know that at least one cut that is an $O(\epsilon^{-1})$-minimum cut will pass the test. We stop the process in the smallest guess for which a cut passes the test.

Finally, to reduce the time complexity to $O(D) + O(n^{0.5+\epsilon}\log^3 n \log\log n \log^* n)$ rounds, note that we can parallelize (i.e., pipeline) the $\Theta(n^\epsilon \log^2 n \log\log n)$ runs of the Cut-Testing algorithm, which come from $\Theta(\log\log n)$ guesses λ'_i, $n^\epsilon \log n$ epochs for each guess, and $\Theta(\log n)$ layers in each epoch. We can do this pipelining simply because these instances of Cut-Testing do not depend on the outcomes of each other and k instances of Thurimella's algorithms can be run together in time $O(D+k\sqrt{n}\log^* n)$ rounds (refer to Section 7.2). To output the final cut, when doing the convergecast of the Cut-Testing results on the BFS, we append the edge-connectivity guess λ'_j, epoch number, and layer number to the *componentID*. Then, instead of taking minimum on just *componentID*, we choose the *componentID* that has the smallest tuple (guess λ'_j, epoch number, layer number, *componentID*). Note that the smallest guess λ'_j translates to the smallest cut size, and the other parts are simply for tie-breaking. □

Algorithm 3 $(2+\varepsilon)$-minimum cut approximation: Matula's Approach

Given a $(1+\varepsilon/10)$-factor approximation $\tilde{\lambda}$ of λ

1: $E_c \leftarrow \emptyset$, $E^* \leftarrow E$, $\eta_{old} \leftarrow n$, $\eta_{new} \leftarrow 1$
2: **while** $(\eta \geq 2)$ & $(\eta_{new} \leq \eta_{old}(1-\varepsilon/10))$ **do**
3: $E_c \leftarrow E \setminus E^*$
4: $E^* \leftarrow$ a sparse certificate for $\tilde{\lambda}(1+\varepsilon/5)$-edge-connectivity of graph $G' = (V', E')$ obtained by contracting edges of E_c
5: $\eta_{new} \leftarrow$ number of connected components of subgraph $H = (V, E \setminus E^*)$
6: **endwhile**
7: Test cuts defined by connected components of graph $H = (V, E \setminus E^*)$ versus threshold $\kappa = \tilde{\lambda}(2+\varepsilon/3)$
8: Output the component that passes the test and contains the smallest ID between such components

7.5 A $(2+\varepsilon)$ Approximation of Minimum Edge Cut

In [Mat93], Matula presents an elegant centralized algorithm that for any constant $\varepsilon > 0$, finds a $(2+\varepsilon)$-min-cut in $O(|V| + |E|)$ steps. Here, we explain how with the help of a few additional elements, this general approach can be used in the distributed setting, to find a $(2+\varepsilon)$-minimum cut in $O\big((D + \sqrt{n}\log^* n)\log^2 n \log\log n \cdot \frac{1}{\varepsilon^5}\big)$ rounds. We first recap the concept of *sparse certificates for edge connectivity*.

Definition 7.5.1. *For a given unweighted multi-graph $H = (V_H, E_H)$ and a value $k > 0$, a set $E^* \subseteq E_H$ of edges is a* sparse certificate for k-edge-connectivity *of H if (1) $|E^*| \leq k|V_H|$, and (2) for each edge $e \in E_H$, if there exists a cut $(\mathcal{C}, V \setminus \mathcal{C})$ of H such that $|(\mathcal{C})| \leq k$ and $e \in (\mathcal{C}, V \setminus \mathcal{C})$, then we have $e \in E^*$.*

Thurimella [Thu97] presents a simple distributed algorithm that finds a sparse certificate for k-edge-connectivity of a network graph G in $O(k(D + \sqrt{n}\log^* n))$ rounds. With simple modifications, we get a generalized version, presented in Lemma 7.5.2.

Lemma 7.5.2. *Let E_c be a subset of the edges of the network graph G and define the virtual graph $G' = (V', E')$ as the multi-graph that is obtained by contracting all the edges of G that are in E_c. Using the modified version of Thurimella's certificate algorithm, we can find a set $E^* \subseteq E \setminus E_c$ that is a sparse certificate for k-edge-connectivity of G', in $O(k(D+\sqrt{n}\log^* n))$ rounds.*

Proof of Lemma 7.5.2. The idea of Thurimella's original sparse certificate-algorithm [Thu97] is relatively simple: E^* is made of the edges of k MSTs that are found in k iterations. Initially, we set $E^* = \emptyset$. In each iteration, we assign weight 0 to the edges in $E \setminus E^*$ and weight 1 to the edges in E^*. In each iteration, we find a new MST with respect to the new weights using the MST algorithm of [KP95], and add the edges of this MST to E^*. Because of the weights, each MST tries to avoid using the edges that are already in E^*. In particular, if in one iteration, there exist two edges e, e', a cut $(\mathcal{C}, V \setminus \mathcal{C})$ such that $e, e' \in (\mathcal{C}, V \setminus \mathcal{C})$ and $e \in E^*$ but $e' \notin E^*$, then the new MST will not contain e but will contain an edge

$e'' \in (E \setminus E^*) \cap (\mathcal{C}, V \setminus \mathcal{C})$. This is because, MST will prefer e'' to e and there is at least one such e'', namely edge e'. As a result, if there is a cut with size at most k, in each MST, at least one edge of the cut gets added to E^*, until all edges of the cut are in E^*.

To solve our generalized version of sparse certificate, we modify the algorithm in the following way. As before, we construct the set E^* iteratively such that at the beginning $E^* = \emptyset$. In each iteration, we give weight 0 to edges of E_c, weight 1 to edges of $E \setminus (E_c \cup E^*)$ and weight 2 to edges in E^*. Moreover, in each iteration, if the newly found MST is T, we only add edges in $T \setminus E_c$ to the set E^*. Note that if for an edge $e = \{u, v\} \in E$, nodes u and v correspond to the same node of the edge-contracted graph G', then edge e will never be added to E^* as either it is in E_c or u and v are connected via a path made of edges in E_c and thus, in each MST, that path is always preferred to e. Moreover, if there is a cut $(\mathcal{C}, V \setminus \mathcal{C})$ of G such that $(\mathcal{C}, V \setminus \mathcal{C}) \cap E_c = \emptyset$ and there are two edges $e, e' \in (\mathcal{C}, V \setminus \mathcal{C})$ such that $e \in E^*$ but $e' \notin E^*$, then the new MST will not contain e but will contain an edge $e'' \in (E \setminus E^*) \cap (\mathcal{C}, V \setminus \mathcal{C})$. □

Following the approach of Matula's centralized algorithm[6] [Mat93], and with the help of the sparse certificate algorithm of Lemma 7.5.2 and the random sparsification technique of Karger [Kar94b], we get the following result.

Theorem 7.5.3. *There is a distributed algorithm that, for any constant $\varepsilon > 0$, finds a $(2+\varepsilon)$-minimum cut in $O((D + \sqrt{n}\log^* n)\log^2 n \log\log n \cdot \frac{1}{\varepsilon^5})$ rounds.*

Proof of Theorem 7.5.3. We assume that nodes know a $(1 + \varepsilon/10)$-factor approximation $\tilde{\lambda}$ of the edge connectivity λ, and explain a distributed algorithm with round complexity $O((D + \sqrt{n}\log^* n)\log^2 n \cdot \frac{1}{\varepsilon^4})$. Note that this assumption can be removed at the cost of a $\Theta(\frac{\log\log n}{\log(1+\varepsilon/10)}) = \Theta(\log\log n \cdot \frac{1}{\varepsilon})$ factor increase in round complexity by trying $\Theta(\frac{\log\log n}{\varepsilon})$ exponential guesses $\tilde{\lambda}(1+\varepsilon/10)^i$ for $i \in [0, \Theta(\frac{\log\log n}{\varepsilon})]$ where $\tilde{\lambda}$ is an $O(\log n)$-approximation of the edge-connectivity, which can be found by Corollary 7.3.3.

For simplicity, we first explain an algorithm that finds a $(2+\varepsilon)$-minimum cut in $O(\lambda(D+\sqrt{n}\log^* n)\log n \cdot \frac{1}{\varepsilon^2})$ rounds. Then, we explain how to reduce the round complexity to $O((D + \sqrt{n}\log^* n)\log^2 n \cdot \frac{1}{\varepsilon^4})$.

First, we compute a sparse certificate E^* for $\tilde{\lambda}(1 + \varepsilon/5)$-edge-connectivity for G, using Thurimella's algorithm. Now consider the graph $H = (V, E^*)$. We have two cases: either (a) H has at most $|V|(1 - \varepsilon/10)$ connected components, or (b) there is a connected component $\mathcal{C}$ of H such that $w(\mathcal{C}) \leq \frac{2\lambda(1+\varepsilon/10)(1+\varepsilon/5)}{1-\varepsilon/10} \leq (2+\varepsilon)\lambda$. Note that if (a) does not hold, case (b) follows because H has at most $(1 + \varepsilon/5)\tilde{\lambda}|V|$ edges.

In Case (b), we can find a $(2+\varepsilon)$-minimum cut by testing the connected components of H versus threshold $\kappa = \tilde{\lambda}(2 + \varepsilon/3)$, using the Cut-Tester algorithm presented in Lemma 7.4.3. In Case (a), we can solve the problem recursively on the virtual graph $G' = (V', E')$ that is obtained by contracting all the edges of G that are in $E_c = E \setminus E^*$. Note that this contraction

[6] We remark that Matula [Mat93] never uses the name *sparse certificate* but he performs *maximum adjacency search* which indeed generates a sparse certificate.

process preserves all the cuts of size at most $\tilde{\lambda}(1+\varepsilon/5) \geq \lambda$ but reduces the number of nodes (in the virtual graph) at least by a $(1-\varepsilon/10)$-factor. Consequently, $O(\log(n)/\varepsilon)$ recursions reduce the number of components to at most 2 while preserving the minimum cut.

We now explain how to remove the dependence on λ from the time complexity. Let E_S be a subset of the edges of $G = (V, E)$ where each $e \in E$ is independently included in E_S with probability $p = \frac{100 \log n}{\varepsilon^2} \cdot \frac{1}{\lambda}$. Then, using the edge-sampling result of Karger [Kar94b, Theorem 2.1][7], we know that with high probability, for each $\mathcal{C} \subseteq V$, we have

$$(1-\varepsilon/3) \cdot |(\mathcal{C}, V \setminus \mathcal{C})| \cdot p \leq |(\mathcal{C}, V \setminus \mathcal{C}) \cap E_S| \leq (1+\varepsilon/3) \cdot |(\mathcal{C}, V \setminus \mathcal{C})| \cdot p.$$

Hence, in particular, we know that graph $G_{new} = (V, E_S)$ has edge connectivity at least $\lambda p(1-\varepsilon/3)$ and at most $\lambda p(1+\varepsilon/3)$, i.e., $\lambda_{new} = \Theta(\log n \cdot \frac{1}{\varepsilon^2})$. Moreover, for every cut $(\mathcal{C}, V \setminus \mathcal{C})$ that is a $(1+\varepsilon/3)$-minimum cut in graph G_{new}, we have that $(\mathcal{C}, V \setminus \mathcal{C})$ is a $(1+\varepsilon)$-minimum cut in graph G. We can therefore solve the cut-approximation problem in graph G_{new}, where we only need to use sparse certificates for $\Theta(\log n \cdot \frac{1}{\varepsilon^2})$ edge-connectivity[8]. The new round complexity becomes $O\big((D + \sqrt{n} \log^* n) \log^2 n \cdot \frac{1}{\varepsilon^4}\big)$ rounds.

The above round complexity is assuming a $(1+\varepsilon/10)$-approximation of edge-connectivity is known. Substituting this assumption with trying $\Theta(\log \log n/\varepsilon)$ guesses around the $O(\log n)$ approximation obtained by Corollary 7.3.3 (and outputting the smallest found cut) brings the round complexity to the claimed bound of $O((D + \sqrt{n} \log^* n) \log^2 n \log \log n \cdot \frac{1}{\varepsilon^5})$. □

7.6 A $(1+\varepsilon)$ Approximation of Minimum Edge Cut

Here, we explain a distributed algorithm which finds a $1+\varepsilon$ approximation of the minimum edge cut, with a nearly-optimal time complexity. The formal result is as follows:

Theorem 7.6.1. *There is an $\tilde{O}(D + \sqrt{n})$ round distributed algorithm that in any weighted network, computes a $(1+\varepsilon)$ approximation of the min-cut, for any constant [9] $\varepsilon > 0$.*

To present a more general result, which can also be made more efficient in special graph families, we present a modular algorithm using the framework of low-congestion shortcuts. In particular, we explain an algorithm with round complexity $\tilde{O}(K)$, where K is a given known upper bound such that for any partition of $G = (V, E)$ into vertex-disjoint individually-connected subsets $S_1, S_2, \ldots, S_N$, there is a low-congestion shortcut with congestion at most K and dilation at most K, and assuming that this shortcut can be computed in at most $\tilde{O}(K)$ rounds. Recall the definitions of low-congestion shortcuts from Chapter 6. In particular,

[7] We emphasize that this result is non-trivial. The proof follows from the powerful bound of $O(n^{2\alpha})$ on the number of α-minimum cuts [Kar93] and basic concentration arguments (Chernoff and union bounds).

[8] Note that, solving the cut approximation on the virtual graph G_{new} formally means that we set the weight of edges outside $E \setminus E_0$ equal to zero. However, we still use graph G to run the distributed algorithm and thus, the round complexity depends on $diam(G) = D$ and not on the possibly larger $diam(G_{new})$.

[9] We assume a constant ε for simplicity. The round complexity has a poly$(1/\varepsilon)$ dependency.

following Corollary 6.1.3, we can set $K = D + \sqrt{n}$ in general graphs. Moreover, following results of Ghaffari and Haeupler [GH16b], we can set $K = D \log D$ in any planar network, and following results of Ghaffari, Kuhn, and Su [GKS16], we can set $K = 2^{O(\sqrt{\log n \log\log n})}$ in any network with edge-expansion at least $\frac{1}{\text{poly}(\log n)}$ and maximum degree at most $\text{poly}(\log n)$.

7.6.1 High-Level Description of the Algorithm

Throughout, we assume that we have a 2-approximation $\tilde{\lambda}$ of the min-cut size λ. This assumption can be removed by trying $O(\log n)$ guesses of the form $\tilde{\lambda} = 2^k$ and outputting the smallest cut found overall. Having this, we use Karger's sampling [Kar94b] to reduce the min-cut size to $\lambda' = O(\log n)$, while keeping the min-cut sizes around their expectation. Then, we greedily pack $\tilde{O}(\lambda'^7) = \text{poly}(\log n)$ minimum spanning trees, one by one, using the MST algorithm of the previous subsection. Thorup's fascinating result shows that, there is going to be one of these trees $\mathcal{T}$, and specially one of its edges e^*, that if we remove e^* from $\mathcal{T}$, the remaining components define the two sides of a min-cut. We will check all the trees in our collection, and moreover, for each tree, we will check all the cuts each induced by removing one of the tree edges, and we report the smallest cut found. Doing this latter part in $\tilde{O}(K)$ rounds is where we use our new ideas.

Having this brief and very rough explanation, we now proceed to present the algorithm:

Karger's Sampling Although the communications will always be in the base graph G, for the following discussions, imagine that we replace the weighted graph G with an unweighted multi-graph G' where each edge e is replaced by $w(e)$ copies of e. Then, sample each edge with probability $\Theta(\frac{\log n}{\varepsilon^2 \tilde{\lambda}})$ and let $\mathcal{G}$ be the spanning graph with the sampled edges. By classical results of Karger [Kar94b], we get that w.h.p. $\mathcal{G}$ has min-cut size $\lambda' = \Theta(\frac{\log n}{\varepsilon^2})$, which is $\Theta(\log n)$ for constant $\varepsilon > 0$, and each cut of $\mathcal{G}$ has size within $1 \pm \frac{\varepsilon}{3}$ factor of its expectation. Hence, finding a $(1 + \frac{\varepsilon}{3})$ approximation of the min-cut on $\mathcal{G}$ gives an $1 + \varepsilon$ approximation (at most) for the min-cut of G.

Throup's Tree-Packing Now we use the tree-packing idea of Thorup [Tho01] on this graph $\mathcal{G}$. Initially, define the load of each edge to be 0. Then, for $\eta = \Theta(\lambda'^7 \log^3 n) = \Theta(\log^{10} n)$ iterations, do as follows: In iteration i, compute the minimum spanning tree where the weight/cost of each edge is simply its load, and remember this as tree $\mathcal{T}_i$. Increase the load of each of the edges in $\mathcal{T}_i$ by 1, and go to the next iteration. To compute each of these MSTs, we simply use our $\tilde{O}(K)$-round MST algorithm, presented in the previous chapter.

By Throup's results [Tho01], there is one of these trees $\mathcal{T}_i$ and one edge $e^* \in \mathcal{T}_i$ such that if we remove e^* from $\mathcal{T}_i$, we get a min-cut; more precisely, each of the two connected component of $\mathcal{T}_i \setminus e^*$ is one of the sides of a min-cut. Hence, to find the min-cut, we will work on these trees one by one, each time looking for this special edge e^*, which gives us our desired small cut.

What remains to be solved When working on each tree $\mathcal{T}_i$, even if this is the right tree, we still do not know which of its edges is that special min-cut defining edge e^*. Hence, we will need to read/approximate the sizes of all the cuts, each defined by removing a single edge of $\mathcal{T}_i$, and we will report the smallest of these, overall, and its associated cut.

Thus, the problem that remains to be solved distributedly can be recapped as follows: given an arbitrary tree $\mathcal{T}$, we want to read the size of each of the cuts defined by removing an edge of $\mathcal{T}$, and report the smallest of these cuts (smallest up to $1 + \varepsilon/3$ factor).

Reading Tree-Edge Induced Cuts Imagine that we pick an arbitrary root for $\mathcal{T}$ and orient its edges outwards from the root, i.e., from the parents to the children. Then, the weight of the cut defined by each edge $e = (u, v)$, where u is the $\mathcal{T}$-parent of v, is equal to the total summation of the weights of $\mathcal{G}$-edges that connect the subtree $\mathcal{T}_v$ below v to the rest of the tree, i.e., $\mathcal{T} \setminus \mathcal{T}_v$. To solve this problem, a basic subroutine that we will make frequent use of it is *subset sums*, where each node u starts with a value x_u, and each node v must learn the sum of the values of itself and its descendants, $y_v = \sum_{u \in \mathcal{T}_v} x_u$. In Section 7.6.2, we explain how to solve this problem in $\tilde{O}(K)$, using our low-congestion shortcuts. Then, in Section 7.6.3, we explain how by using $\text{poly}(\log n)$ iterations of this subroutine, we can (simultaneously) compute a $(1 + \varepsilon/3)$-approximation of the sizes of cuts each defined by removing one $\mathcal{T}$-edge, all in $\tilde{O}(K)$ rounds.

7.6.2 Subtree Sums

We first explain how to orient the tree from the root outwards, in $\tilde{O}(K)$ rounds, and then explain how to use this orientation to compute the subtree sums, in $\tilde{O}(K)$ rounds. Both parts make use of our low-congestion shortcuts.

Orienting A Tree in $\tilde{O}(K)$ Rounds Let us first see how to algorithmically orient $\mathcal{T}$ such that each node knows its parent, in $\tilde{O}(K)$ rounds. Note that the tree $\mathcal{T}$ might have an arbitrarily large diameter and hence, the standard approaches such as doing a flooding on $\mathcal{T}$ from the root outwards would not finish in $\tilde{O}(K)$ rounds. The remedy is in using our low-congestion shortcuts.

Consider the following fragment-merging process which has $O(\log n)$ levels: in each level, the tree $\mathcal{T}$ is partitioned into a number of fragments, each being an induced subtree. In level 1, each node is its own fragment. From that point on, in each level, each fragment of level i is formed by merging some of the fragments of the level $i - 1$, which are adjacent in $\mathcal{T}$. More precisely, in each iteration i, each fragment picks a $\mathcal{T}$-edge e that connects it to one of the other fragments, and suggests a merge along e. At the same time, each fragment tosses a coin. Then, each head-fragment accepts the proposed merge edges coming from tail-fragments. That is, each merge is a star-formation, centered at a fragment that has a head in its random coin toss, and with a number of sides which each had a tail coin. See Figure 7-2, which shows the fragments of three levels. As in the previous subsection, it is

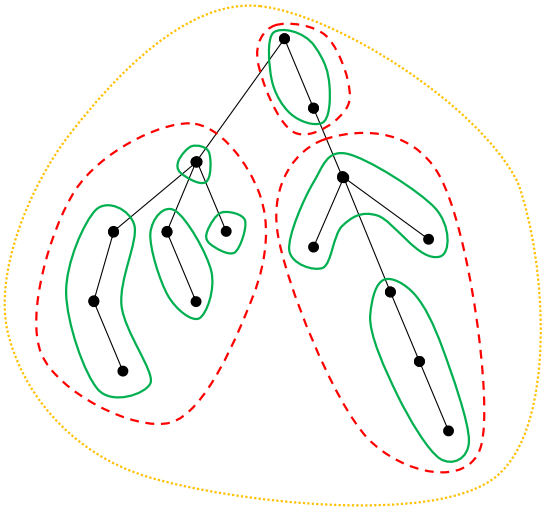


Figure 7-2: The fragmentation of a tree; each color shows the fragments of one level. The fragments of the first level, where each node is trivially its own fragment, are not depicted.

easy to see that after $O(\log n)$ such iterations, w.h.p., we have a single fragment, that is the whole tree $\mathcal{T}$. During the iterations, each of the fragments might have a large diameter. Hence, we use our low-congestion shortcuts and the random-delay based routing explained in the proof of Theorem 6.1.2, which allows each fragment to pick an edge to one of the other fragments and make this edge, as well as the outcome of the random coin toss, known to all the nodes of the fragment, and this happens for all the fragments together in $O(K \log n)$ rounds, per level.

Now let us take a look at these fragments of the $O(\log n)$ levels, from the last level backwards. In the very last level, we have (at most) a single star merge. One of the fragments in this merge contains the root r of $\mathcal{T}$. For every other fragment, we can easily find the root of it, using a few iterations of working on the low-congestion shortcuts. First, identify the fragment that contains the root. Then, let each of the nodes in this fragment send a special message to their neighbors in $\mathcal{T}$. Nodes that received this special message but were not in that root fragment are actually root of their own fragment. Via one application of low-congestion shortcuts, we can make all of the nodes in these fragments know their fragment root. Since a star has depth at most 2, with one more repetition of the same idea, we will reach the point that we have identified the root of each fragment, in the top level. Now we remove the $\mathcal{T}$-edges between these fragments, and recurse, going one level deeper. Since always the fragments are disjoint parts of the graph, and each of them induces a connected subgraph, in each level, we can use low-congestion shortcuts to identify the root of each fragment, in $O(K \log n)$ rounds. After repeating this for $O(\log n)$ levels, which takes $O(K \log^2 n)$ rounds, we have identified the roots of the fragments of each of the levels.

Now each node v can easily identify its $\mathcal{T}$-parent as follows: v considers its lowest-level fragment, say level i, in which v is not the root of this fragment. Notice that this level-i fragment is a star-shape merge of some level $i-1$ fragments, one of which contains v as

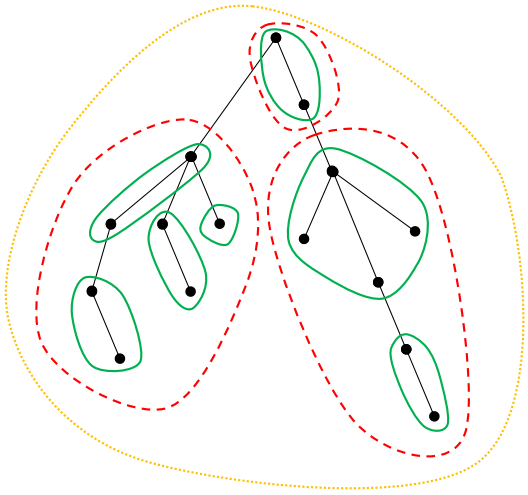


Figure 7-3: A fragmentation where each level-i fragment is made of merging a level-$(i-1)$ fragment and some of its level-$(i-1)$ fragment children.

its root. During the root-identification of level-i fragments, v received the ID of this level-i fragment's root along an edge between level-$(i-1)$ fragments, i.e., from one of its $\mathcal{T}$-neighbors which is in a different level-$(i-1)$ fragment. This neighbor is in fact the parent of v in $\mathcal{T}$.

Subtree Sums in $\tilde{O}(K)$ Rounds Having the orientation defined above, here we explain an $\tilde{O}(K)$ round algorithm which computes subtree sums. More formally, suppose each node u has a $O(\log n)$-bit value x_u. The objective is to have each node v know the summation of the values in the $\mathcal{T}$-subtree below it, i.e., $y_v = \sum_{u \in \mathcal{T}_v} x_u$, where $\mathcal{T}_v$ includes v and all its $\mathcal{T}$-descendants. In the next subsubsection, we explain how this scheme helps us to find the min-cut.

The first step, which is mainly done for simplicity, is to redefine the $O(\log n)$-level fragmentation process such that each level-i fragment is made of merging a level-$(i-1)$ fragment and some of its level-$(i-1)$ fragment children. See Figure 7-3 for an example. This fragmentation process is quite similar to the process explained above, with the exception that here, in iteration i, each fragment suggests the edge to its parent fragment for the merge and then, the head-fragments accept all their children tail fragments. Again, we easily see that after $L = O(\log n)$ levels, w.h.p., we reach a single fragment which includes the whole tree $\mathcal{T}$. As explained above, this process takes $O(K \log^2 n)$ rounds overall, thanks to using low-congestion shortcuts in each level.

Having these fragments, we are ready to compute the subtree sum for each node. We will solve the problem recursively. Note that in level-L, there is only one fragment, but this is (potentially) made of a level $L-1$ fragment and its level $L-1$ children. The first step is to separate the problems of these level $L-1$ fragments, by removing the edges between them. Notice that the children level $L-1$ fragments are ready to start the problem within their own fragment and do not need to learn any information from the parent level $L-1$ fragment. However, this parent fragment needs to learn some information from the children.

Particularly, for each of the children fragments F_c, there is one node v in the parent fragment F_p that is the parent of the root of this child fragment F_c. We need to deliver the total sum of the values in F_c to node v. If we do this for all such nodes v in F_p, those nodes can increment their value by the received amount and afterwards drop the $\mathcal{T}$-edge to F_c. At that point, the problem would be to solve the subtree sum within each level $L-1$ fragment, which means we have progressed one recursion level, and we can continue to solve the problem recursively now, for $L = O(\log n)$ recursion levels.

To compute the sum of the values in each of the (children) level $L-1$ fragments, so that we can deliver it to $v \in F_p$, we use another recursion, now going bottom-top, i.e., from level 1 to level $L-1$: That is, we walk through these $O(\log n)$ levels, and keep the variant that at each point, the root of each fragment knows the summation of the values in that fragment. At the start, this is trivially satisfied as each level-1 fragment, which is simply a node, knows its own value. In level $i \geq 2$, each level-i fragment is formed by merging one level-$(i-1)$ fragment with some of its level-$(i-1)$ fragment children. Each of the roots of these children fragments knows the total sum of its own fragment. They send these values to their parents, in one round. At this point, we have all the values in nodes of the parent level-$(i-1)$ fragment. Using a convergecast on the BFS of the shortcutted version of these parent fragments, similar to the approach explained in the proof of Theorem 6.1.2, in $O(K \log n)$ rounds, we can gather the summation of these values at the root of the parent level-$(i-1)$ fragment. Then, that parent adds the value of its own level-$(i-1)$ fragment to this sum and remembers the result as the sum of its level-i fragment. After $O(\log n)$ repetitions of all the $L-1$ levels, each level $L-1$ fragment root knows the total sum of its fragment. Hence, as described above, each of these can report the value to its parent node, which is in the parent level $L-1$ fragment, and then we are ready for the higher-level top-bottom recursion to subset sum problems confined to level $L-1$ fragments.

Notice that we are able to use low-congestion shortcuts throughout all of these recursions because the parameters of the shortcuts do not depend on how many parts there are, and only require that the parts are disjoint and each part induces a connected subgraph, and these two properties are clearly satisfied for the fragments of each level. Hence, after $O(\log n)$ levels of the top-bottom recursion as described above, each of which contains an $O(\log n)$ level bottom-top recursion, each node v knows its subtree sum $y_v = \sum_{u \in \mathcal{T}_v} x_u$.

7.6.3 Approximating Tree-Edge Induced Cuts

We are now ready to explain the approach we use for approximating the sizes of the cuts each defined by removing one $\mathcal{T}$-edge e from $\mathcal{T}$. This uses the subtree sum subroutine presented in the previous subsubsection, and a small sketching type of idea.

Let us focus on just one of these cuts; as we will see later, the proposed solution solves the problem for all these cuts simultaneously. Consider one $\mathcal{T}$-edge $e = (v, u)$ and suppose that u is the parent of v. Let us say we want to see if the size of the cut $(\mathcal{T}_v, V \setminus \mathcal{T}_v)$ is larger than some threshold $\tau = (1+\varepsilon')^k$ or not, where $\varepsilon' = \varepsilon/3$. Checking the cut versus the

$O(1/\varepsilon)$ many thresholds of the form $(1+\varepsilon')^k$ that are within a 2-factor of our estimate $\tilde{\lambda}$ of λ will suffice to get a $(1+\varepsilon')$ approximation of the size of the cut.

To compare the cut $(\mathcal{T}_v, V \setminus \mathcal{T}_v)$ versus threshold τ, what we do is based on repetitions of a simple randomized experiment. Each experiment is as follows: Mark each $\mathcal{G}$-edge as *active* with probability $1-2^{-\frac{1}{\tau}}$[10], and as *inactice* otherwise. Now for each active edge e, this edge contributes a ± 1 to the value of its endpoints, as follows: randomly select one of the endpoints of the active edge e, assign a $+1$ to this endpoint, and a -1 to the other endpoint. Define the value x_w of each node w to be the summation of all the values contributed to w by the active edges incident on w.

Now let us take a look at the subtree sum $y_v = \sum_{w \in \mathcal{T}_v} x_w$, where v is the child in the cut-defining $\mathcal{T}$-edge e under consideration. Each $\mathcal{G}$-edge that has both of its endpoints in $\mathcal{T}_v$ does not contribute anything to y_v as, either it is inactive, or the $+1$ and -1 values of its contributions are both in the subtree $\mathcal{T}_v$ and thus get canceled out. This is also clearly true for edges with both their endpoints in $\mathcal{T} \setminus \mathcal{T}_v$. Hence, the subtree sum y_v is simply the summation of the ± 1 values coming from active edges with exactly one endpoint in $\mathcal{T}_v$. Our indicator random variable for comparing the cut-size $(\mathcal{T}_v, V \setminus \mathcal{T}_v)$ versus threshold τ is whether $y_v = 0$ or not. If the number of $\mathcal{G}$-edges in the cut $(\mathcal{T}_v, V \setminus \mathcal{T}_v)$ is smaller than $\tau(1-\varepsilon')$, then we can see that, the probability that there is at least one active edge in $(\mathcal{T}_v, V \setminus \mathcal{T}_v)$ is at most $0.5 - \varepsilon'/10$. On the other hand, if the cut size is at least $\tau(1+\varepsilon')$, then the same probability is at least $0.5 + \varepsilon'/10$.

The above is already the *distinguisher* that we desired; but we still need to work a bit more. Note that even if the set of active edges across the cut is non-empty, it is still possible that we get unlucky and the contributions of the (single) $\mathcal{T}_v$-endpoints of these active cut-edges sum up to 0. However, this is easy to fix. For each random experiment defined as above, we repeat $b = \Theta(\log(1/\varepsilon))$ sub-experiments: Throughout each experiment, which has b sub-experiments, we keep the set of active edges the same, but in each sub-experiment, we re-sample the ± 1 contributions to the endpoints. That is, in each sub-experiment, using fresh randomness, we determine which end of each active edge gets a $+1$ and which endpoint gets a -1.

If the set of active cut-edges in an experiment is non-empty, in each of these sub-experiments, $y_v \neq 0$ with probability at least $1/2$. To see why, suppose we expose the randomness of the $\pm$ contributions one by one and just consider the last cut active edge that exposes its ± 1 contribution. Regardless of the outcome of the previous such edges, there is at least a $1/2$ chance that because of the randomness of this last edge, the sum becomes nonzero. We conclude that the probability that, even though the set of active cut-edges in an experiment is nonempty, all of its sub-experiments show $y_v = 0$ is at most $(1/2)^b \ll \varepsilon'/20$. Hence, overall, if $\tau(1+\varepsilon')$, the experiment will show $y_v \neq 0$, in at least one of its sub-experiments, with probability at least $0.5 + \varepsilon'/10 - \varepsilon'/20 \geq 0.5 + \varepsilon'/20$.

Hence, by Hoeffding's bound, we get that $\Theta(\frac{\log n}{\varepsilon^2})$ iterations of this experiment suffice for

[10] We note that this is roughly equal to $\frac{1}{\tau}$, but this special formula will simplify the calculations.

a high probability distinguisher. More precisely, we simply repeat the above experiment for $\Theta(\frac{\log n}{\varepsilon^2})$ iterations, and check if the majority of the experiments are showing a nonzero y_v (in at least one of their sub-experiments) or not. This w.h.p distinguishes the case where the cut size is greater than $\tau(1+\varepsilon')$ from the case that the cut size is less than $\tau(1-\varepsilon')$. Doing this for the $O(1/\varepsilon)$ many thresholds of the form $\tau = (1+\varepsilon')^k$ that are within a 2-factor of our guesstimate $\tilde{\lambda}$ of λ suffices to get a $(1+\varepsilon')$ approximation of the cut.

Finally, notice that, to run the above process for all the cuts defined each by removing a single $\mathcal{T}$-edge e, all that we need to do is as follows: sample the active edges and their ± 1 endpoint contributions and then compute the subtree sums y_v for all the nodes v of $\mathcal{T}$. The former can be done locally for each edge, say by the larger-ID endpoint of the edges picking these random values, and for the latter part, we already saw how to compute subtree sums for all nodes in $\tilde{O}(K)$ rounds, using the low-congestion shortcuts. This concludes the description of our $\tilde{O}(K)$ round min-cut $(1+\varepsilon)$-approximation algorithm.

7.7 Lower Bounds

In this section, we present our lower bounds for minimum cut approximation, which can be seen as strengthening and generalizing some of the lower bounds of Das Sarma et al. [DHK+12].

The lower bounds of [DHK+12] are based on an n-node graph G with diameter $O(\log n)$ and two distinct nodes s and r. The proof deals with distributed protocols where node s gets a b-bit input x, node r gets a b-bit input y, and apart from x and y, the initial states of all nodes are globally known. Slightly simplified, the main technical result of [DHK+12] (Simulation Theorem 3.1) states that if there is a randomized distributed protocol that correctly computes the value $f(x,y)$ of a binary function $f : \{0,1\}^b \times \{0,1\}^b \to \{0,1\}$ with probability at least $1-\varepsilon$ in time T (for sufficiently small T), then there is also a randomized ε-error two-party protocol for computing $f(x,y)$ with communication complexity $O(TB \log n)$. For small enough T and large enough b, this would be in contradiction to some known two-party communication complexity lower bounds (mainly those of set-disjointness), thus proving that there cannot be such a distributed algorithm.

For our lower bounds, we need to extend the simulation theorem of [DHK+12] to a larger family of networks and to a slightly larger class of problems. In Section 7.7.1, we present this generalized simulation theorem. Then, in Sections 7.7.2 and 7.7.3, we explain how to use the generalized simulation theorem to prove our minimum cut approximation lower bounds, in weighted and simple unweighted graphs, respectively.

7.7.1 Generalized Simulation Theorem

We start by presenting a number of definitions.

Distributed Protocols: Given a weighted network graph $G = (V, E, w)$ ($\forall e \in E : w(e) \geq 1$), we consider distributed tasks for which each node $v \in V$ gets some private input $x(v)$ and every node $v \in V$ has to compute an output $y(v)$ such that the collection of inputs and outputs satisfies some given specification. To solve a given distributed task, the nodes of G apply a distributed protocol. We assume that initially, each node $v \in V$ knows its private input $x(v)$, as well as the set of its neighbors in G. Time is divided into synchronous rounds and in each round, every node can send at most $B \cdot w(e)$ bits over each of its incident edges e. We say that a given (randomized) distributed protocol solves a given distributed task with error probability ε if the computed outputs satisfy the specification of the task with probability at least $1 - \varepsilon$.

Graph Family $\mathcal{G}(n, k, c)$: For parameters n, k, and c, we define the *family of graphs* $\mathcal{G}(n, k, c)$ as follows. A weighted graph $G = (V, E, w)$ is in the family $\mathcal{G}(n, k, c)$ iff $V = \{0, \ldots, n-1\}$ and for all $h \in \{0, \ldots, n-1\}$, the total weight of edges between nodes in $\{0, \ldots, h\}$ and nodes in $\{h + k + 1, \ldots, n-1\}$ is at most c. For an integer $\eta \geq 1$, we define $L_\eta := \{0, \ldots, \eta - 1\}$ and $R_\eta := \{n - \eta, \ldots, n - 1\}$.

Two-party η-solving: Given a parameter $\eta \geq 1$ and a network $G \in \mathcal{G}(n, k, c)$, we say that a two-party protocol between Alice and Bob *η-solves* a given distributed task for G with error probability ε if the following two conditions hold:

(a) initially Alice knows all inputs and initial states of nodes in $V \setminus R_\eta$, and Bob knows all inputs and initial states of nodes in $V \setminus L_\eta$, and

(b) in the end, Alice outputs $y(v)$ for all $v \in L_{n/2}$ and Bob outputs $y(v)$ for all $v \in R_{n/2}$ such that with probability at least $1 - \varepsilon$, these outputs satisfy the specification of the given distributed task.

A two-party protocol is said to be *public coin* if Alice and Bob have access to a common random string.

Theorem 7.7.1 (Generalized Simulation Theorem)**.** *Assume we are given positive integers n, k, and η, a parameter $c \geq 1$, as well as a subfamily $\tilde{\mathcal{G}} \subseteq \mathcal{G}(n, k, c)$. Further assume that for a given distributed task and graphs $G \in \tilde{\mathcal{G}}$, there is a randomized distributed protocol with error probability ε that runs in $T \leq (n - 2\eta)/(2k)$ rounds. Then, there exists a public-coin two-party protocol that η-solves the given distributed task on graphs $G \in \tilde{\mathcal{G}}$ with error probability ε and communication complexity at most $2BcT$.*

Proof. We show that Alice and Bob can simulate an execution of the given distributed protocol to obtain outputs that satisfy the specification of the given distributed task with probability at least $1 - \varepsilon$.

We first argue that being able to simulate deterministic distributed algorithms is enough. This is because a randomized distributed algorithm can be modeled as a deterministic algorithm where at the beginning, each node v receives a sufficiently large random string $r(v)$ as additional input. Assume that R is the concatenation of all the random strings $r(v)$. Then, a randomized distributed protocol with error probability ε can be seen as a deterministic protocol that computes outputs that satisfy the specification of the given task with probability at least $1-\varepsilon$ over all possible choices of R. (A similar argument has also been used, e.g., in [DHK+12]). Alice and Bob have access to a public coin giving them a common random string of arbitrary length. As also the set of nodes $V = \{0, \ldots, n-1\}$ of G is known, Alice and Bob can use the common random string to model R and to therefore consistently simulate all the randomness used by all n nodes in the distributed protocol. Given R, it remains for Alice and Bob to simulate a deterministic protocol. If they can (deterministically) compute the outputs of some nodes of a given deterministic protocol, they can also compute outputs for a randomized protocol with error probability ε such that the outputs are consistent with the specification of the distributed task with probability at least $1-\varepsilon$.

Given a deterministic distributed protocol for graphs $G \in \tilde{\mathcal{G}}$ with time complexity $T \leq (n-2\eta)/(2k)$, we now describe a two-party protocol with communication complexity at most $2BcT$ bits that η-solves the given distributed task on graphs $G \in \tilde{\mathcal{G}}$. Suppose that initially, Alice knows the states of node $V \setminus R_\eta$ and Bob knows the states of nodes $V \setminus L_\eta$. We show that the said two-party protocol satisfies the following two properties for each round $r \in \{0, \ldots, T\}$:

(I) Alice computes the states of all nodes $i < n - \eta - r \cdot k$ at the end of round r, and

(II) Bob computes the states of all nodes $i \geq \eta + r \cdot k$ at the end of round r.

Because the output $y(u)$ of every node u is determined by u's state after T rounds, and since $T \leq (n-2\eta)/(2k)$, conditions (I) and (II) respectively imply that Alice can compute the outputs of all nodes $v \in R_{n/2}$ and Bob can compute the outputs of all nodes $v \in L_{n/2}$. Therefore, this two-party η-solves the distributed task solved by the given distributed protocol. Hence, in order to prove the claim of the theorem, it remains to show that there exists a deterministic two-party protocol with communication complexity at most $2BcT$ satisfying (I) and (II).

We prove conditions (I) and (II) by induction on r. The base of the induction is trivial because conditions (I) and (II) are satisfied for $r = 0$ by the assumption that initially, Alice knows the initial states of all nodes $V \setminus R_\eta = \{0, \ldots, n-1-\eta\}$ and Bob knows the initial states of all nodes $V \setminus L_\eta = \{\eta, \ldots, n-1\}$.

Next, we prove the inductive step. Assume that (I) and (II) hold for some $r = r' \in \{0, \ldots, T-1\}$. Based on this, we show how to construct a protocol with communication complexity at most $2Bc$ such that (I) and (II) hold for $r = r' + 1$. We formally show how, based on assuming conditions (I) and (II) for $r = r'$, Alice can compute the states of nodes $i < n - \eta - (r'+1)k$ using only Bc bits of communication. The argument for Bob can be

done in a completely symmetric way so that we get a total communication complexity of $2Bc$.

Note that in a deterministic algorithm, the state of a node u at the end of a round t—and thus at the beginning of round $t+1$—is completely determined by the state of u at the beginning of round t and by the messages node u receives in round t from its neighbors. Thus, in order to compute the state of a node $i < n-\eta-(r'+1)k$ at the end of round $r'+1$, it is sufficient for Alice to know the state of node i at the beginning of round $r'+1$—i.e., at the end of round r'—and the message sent by each neighbor j in round $r'+1$. By induction hypothesis, Alice knows the state of i at the beginning of round r' and the messages of neighbors $j < n-\eta-r'k$. Thus, she is just missing the messages from neighbors $j \geq n-\eta-r'k$. On the other hand, by induction hypothesis for $r=r'$, Bob knows the states of all nodes $j' \geq \eta+r'k$. Since $r' \leq T$ and $T \leq (n-2\eta)/(2k)$, we get that $\eta+r'k \leq n-\eta-r'k$. Hence, at the beginning of round r', Bob knows the states of all nodes $j \geq n-\eta-r'k$. Furthermore, by the definition of the graph family $\mathcal{G}(n,k,c)$, the total weight of edges between nodes $i < n-\eta-(r'+1)k$ and nodes $j \geq n-\eta-r'k$ is at most c. The number of bits sent over these edges in round $r'+1$ is therefore at most cB. Therefore, Bob can send these cB bits to Alice. With these, Alice can can compute the states of nodes $i < n-\eta-(r'+1)k$. This completes the induction proof. □

7.7.2 Lower Bound for Approximating Minimum Cut: Weighted Graphs

In this subsection, we prove a lower bound on approximating the minimum cut in weighted graphs (or equivalently in unweighted multigraphs). The case of simple unweighted graphs is addressed in the next subsection.

Let $k \geq 1$ be an integer parameter. We first define a fixed n-node graph $H=(V,E_H)$ that we will use as the basis for our lower bound. The node set V of H is $V=\{0,\ldots,n-1\}$. For simplicity, we assume that n is an integer multiple of k and that $\ell := n/k$. The edge E_H consists of three parts $E_{H,1}$, $E_{H,2}$, and $E_{H,3}$ such that $E_H = E_{H,1} \cup E_{H,2} \cup E_{H,3}$. The three sets are defined as follows.

$$
\begin{aligned}
E_{H,1} &:= \{\{i,j\} : i,j \in \{0,\ldots,n-1\} \text{ and } j = i+k\}, \\
E_{H,2} &:= \{\{i,j\} : i,j \in \{0,\ldots,n-1\} \text{ and } \exists s \in \mathbb{N} \text{ s.t. } i \equiv 0 \pmod{k2^s} \text{ and } j = i+k2^s\}, \\
E_{H,3} &:= \{\{i,j\} : i,j \in \{0,\ldots,n-1\},\ i \equiv 0 \pmod{k},\ \text{and } 0 < j-i \leq k-1\}.
\end{aligned}
$$

Figure 7-4 shows an example graph in this family. The edges $E_{H,1}$ connect the nodes V of H to k disjoint paths of length $\ell-1$ (consisting of ℓ nodes), where for each integer $x \in \{0,\ldots,k-1\}$, the nodes $i \equiv x \pmod{k}$ form on of the paths. In Figure 7-4, these are the k horizontal paths. Using the edges of $E_{H,2}$, the nodes of the first of these paths are connected to a graph of small diameter. In Figure 7-4, these are the dotted shortcut links is at the top of the graph. Finally, using the edges $E_{H,3}$ the paths are connected to each other

in the following way. We can think of the n nodes as consisting of groups of size k, where corresponding nodes of each of the k paths form a group (for each integer $h \geq 0$, nodes $hk, \ldots, (h+1)k-1$ form a group). Using the edges of $E_{H,3}$ each such group is connected to a star, where the node of the first path is the center of the star.

Based on graph H, we define a family $\mathcal{H}(n,k)$ of weighted graphs as follows. The family $\mathcal{H}(n,k)$ contains all weighted versions of graph H, where the weights of all edges of $E_{H,2}$ are 1 and weights of all remaining edges are at least 1, but otherwise arbitrary. The following lemma shows that $\mathcal{H}(n,k)$ is a subfamily of $\mathcal{G}(n,k,c)$ for appropriate c and that graphs in $\mathcal{H}(n,k)$ have small diameter.

Lemma 7.7.2. *We have $\mathcal{H}(n,k) \subset \mathcal{G}(n,k,c)$ for $c = \log_2(n/k)$. Further, each graph in $\mathcal{H}(n,k)$ has diameter at most $O(\log(n/k))$.*

Proof. To show that $\mathcal{H}(n,k) \subset \mathcal{G}(n,k,c)$, we need to show that for each $h \in \{0,\ldots,n-1\}$, the total weight of edges between nodes in $\{0,\ldots,h\}$ and nodes in $\{h+k+1,\ldots,n-1\}$ is at most c. All edges in $E_{H,1}$ and $E_{H,3}$ are between nodes i and j for which $|j-i| \leq k$, the only contribution to the weight of edges between nodes in $\{0,\ldots,h\}$ and nodes in $\{h+k+1,\ldots,n-1\}$ thus comes from edges $E_{H,2}$. For each $h \in \{0,\ldots,n-1\}$ and for each $s \in \mathbb{N}$, there is at most one pair (i,j) such that $i \equiv j \equiv \pmod{k2^s}$ and such that $i \leq h$ and $j > h+k$. The number of edges between nodes in $\{0,\ldots,h\}$ and nodes in $\{h+k+1,\ldots,n-1\}$ therefor is at most $\log_2(n/k)$ and the first claim of the lemma therefore follows because edges in $E_{H,2}$ are required to have weight 1. The bound on the diameter follows directly from the construction: With edges $E_{H,3}$, each node is directly connected to a node of the first path and with edges $E_{H,2}$, the nodes of the first path are connected to a graph of diameter $O(\log(n/k))$. □

Based on the graph family $\mathcal{H}(n,k)$ as defined above, we can now use the basic approach of [DHK+12] to prove a lower bound for the distributed minimum cut approximation problem.

Theorem 7.7.3. *In weighted graphs (and unweighted multi-graphs), for any $\alpha \geq 1$, computing an α-approximation of the edge connectivity λ or computing an α-approximate minimum cut (even if λ is known) requires at least $\Omega\big(D + \sqrt{n/(B\log n)}\big)$ rounds, even in graphs of diameter $D = O(\log n)$.*

Proof. We prove the theorem by reducing from the two-party set disjointness problem [CP11, KS92a, Raz92]. Assume that as input, Alice gets a set X and Bob get a set Y such that the elements of X and Y are from a universe of size $O(p)$. It is known that in general, Alice and Bob need to exchange at least $\Omega(p)$ bits in order to determine whether X and Y are disjoint [KS92a, Raz92]. This lower bound holds even for public coin randomized protocols with constant error probability and it also holds if Alice and Bob are given the promise that if X and Y intersect, they intersect in exactly one element [Raz92]. As a consequence, if Alice and Bob receive sets X and Y as inputs with the promise that $|X \cap Y| = 1$, finding the element in $X \cap Y$ also requires them to exchange $\Omega(p)$ bits.

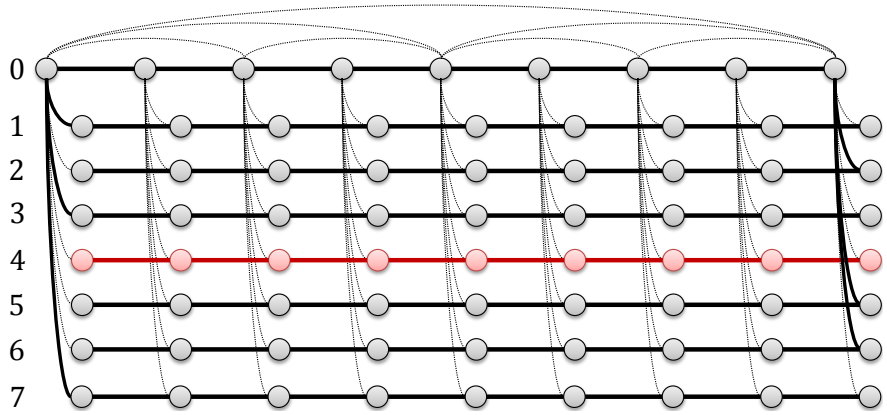


Figure 7-4: Illustration of the lower bound construction of Theorem 7.7.3 for parameters $k = 8$ and $\ell = 8$: Bold, solid edges have weight $\alpha\ell+1$, thin dotted edges have weight 1 .Each of the seven rows 1–7 corresponds to one of the elements of the set disjointness universe. Alice' input $X = \{2, 4, 5, 6\}$ is encoded by using a heavy edge of weidth $\alpha\ell + 1$ to connect the first nodes of the remaining paths $\{1, 3, 7\}$ to the first node of the top path. Bob's input $Y = \{1, 3, 4, 7\}$ in encoded by using a heavy edge to connect the last nodes of the remaining paths $\{2, 5, 6\}$ to the last node of the top path. The graph induced by the edges of weight $\alpha\ell + 1$ is not connected if and only if $X \cap Y \neq \emptyset$. In that case, the paths representing the elements in $X \cap Y$ correspond to the minimum cuts of weight ℓ. In the example, path 4 (marked in red) gives a minimum cut of the graph.

Assume that there is a protocol to find an α-minimum cut or to α-approximate the size of a minimum cut in time T with a constant error probability ε. In both cases, if T is sufficiently small, we show that Alice and Bob can use this protocol to efficiently solve set disjointness by simulating the distributed protocol on a special network from the family $\mathcal{H}(n, k)$.

We now describe the construction of this network $G \in \mathcal{H}(n, k)$. We assume that the set disjointness inputs X and Y of Alice and Bob are both of size $\Theta(k)$ and from a universe of size $k - 1$. The structure of G is already given, the edge weights of edges in $E_{H,1}$ and $E_{H,3}$ are given as follows. First, all edges $E_{H,1}$ (the edges of the paths) have weight $\alpha\ell + 1$ (recall that $\ell = n/k$ is the number of nodes of the paths). We number the paths from 0 to $k - 1$ as follows. Path $p \in \{0, \ldots, k - 1\}$ consists of all nodes i for which $i \equiv p \pmod{k}$. Note that the first node of path p is node p and the last node of path p is $n - k + p$. We encode the set disjointness inputs X and Y in the edge weights of the edges of $E_{H,3}$ as follows. For each $x \in \{0, \ldots, k - 1\} \setminus X$, the edge between node 0 and node x has weight $\alpha\ell + 1$. Further, for each $y \in \{0, \ldots, k - 1\} \setminus Y$, the edge between $n - k$ and $n - k + y$ has weight $\alpha\ell + 1$. All other edges of $E_{H,3}$ have weight 1. The construction is also shown in Figure 7-4

Hence, the graph induced by the edges with large weight $\alpha\ell + 1$ (in the following called heavy edges) looks as follows. It consists of the k paths of length $\ell - 1$. In addition for each $x \notin X$, path x is connected to node 0 and for each $y \notin Y$, path y is connected to node $n - k$. Assume that there is exactly one element $z \in X \cap Y$. Path z is not connected to path 0 through a heavy edge, all other paths are connected to each other by heavy edges. The

minimum cut $(S, V \setminus S)$ is defined by the nodes $S = \{i \in \{0, \dots, n-1\} : i \equiv z \pmod{k}\}$. As each node on path z is connected by a single weight 1 edge to a node on path 0, the size of the cut $(S, V \setminus S)$ is ℓ. There is at least one heavy edge crossing every other cut and thus, every other cut has size at least $\alpha\ell + 1$. In order to find an α-approximate minimum cut, a distributed algorithm therefore has to find path z and thus the element $z \in X \cap Y$.

Assume now that there is a distributed protocol that computes an α-approximate minimum cut in T rounds, by using messages of at most B bits. The described graph G is in $\mathcal{H}(n, k)$ and by Lemma 7.7.2, the graph therefore also is in $\mathcal{G}(n, k, \log(n/k))$ and it has diameter at most $O(\log n)$. We can therefore prove the claim of the theorem by providing an appropriate lower bound on T. We reduce the problem to the two-party set disjointness problem by describing how Alice and Bob can together simulate the given distributed protocol.

Initially, only the nodes $0, \dots, k-1$ depend on the input X of Alice and only the nodes $n-k, \dots, n-1$ depend on the input Y of Bob. The inputs of all other nodes are known. Initially, Alice therefore knows the inputs of all nodes in $\{0, \dots, n-k-1\}$ and Bob knows the inputs of all nodes in $\{k, \dots, n-1\}$. Thus, by Theorem 7.7.1, for $T \leq (n-2k)/(2k)$, there exists a $2BcT = O(TB \log n)$-bit public coin two-party protocol between Alice and Bob that k-solves the problem of finding an α-approximate minimum cut in G. However, since at the end of such a protocol, Alice and Bob know the unique minimum cut $(S, V \setminus S)$, they can also use it to find the element $z \in X \cap Y$. We have seen that this requires them to exchange at least $\Omega(k)$ bits and we thus get a lower bound of $T = \Omega(k/(B \log n))$ on T. Recall that we also need to guarantee that $T \leq (n-2k)/(2k)$. We choose $k = \Theta(\sqrt{nB \log n})$ to obtain the lower bound claimed by the theorem statement. Note that the lower bound even applies if the size ℓ of the minimum cut is known.

If the size of the minimum cut is now known and the task of an algorithm is to approximate the size of the minimum cut, we can apply exactly the same reduction. This time, we do not use the promise that $|X \cap Y| = 1$, but only that $|X \cap Y| \leq 1$. The size of the minimum cut is ℓ if X and Y intersect and it is at least $\alpha\ell + 1$ if they are disjoint. Approximating the minimum cut size therefore is exactly equivalent to solving set disjointness in this case. □

7.7.3 Lower Bound for Approximating Minimum Cut: Simple Unweighted Graphs

We next present our lower bound for approximating the minimum cut problem in unweighted simple graphs.

Theorem 7.7.4. *In unweighted simple graphs, for any $\alpha \geq 1$ and $\lambda \geq 1$, computing an α-approximation of λ or finding an α-approximate minimum cut (even if λ is known) requires at least $\Omega\big(D + \sqrt{n/(B\alpha\lambda \log n)}\big)$ rounds, even in networks of diameter $D = O(\log n + \frac{1}{\lambda} \cdot \sqrt{n/(B\alpha\lambda \log n)})$.*

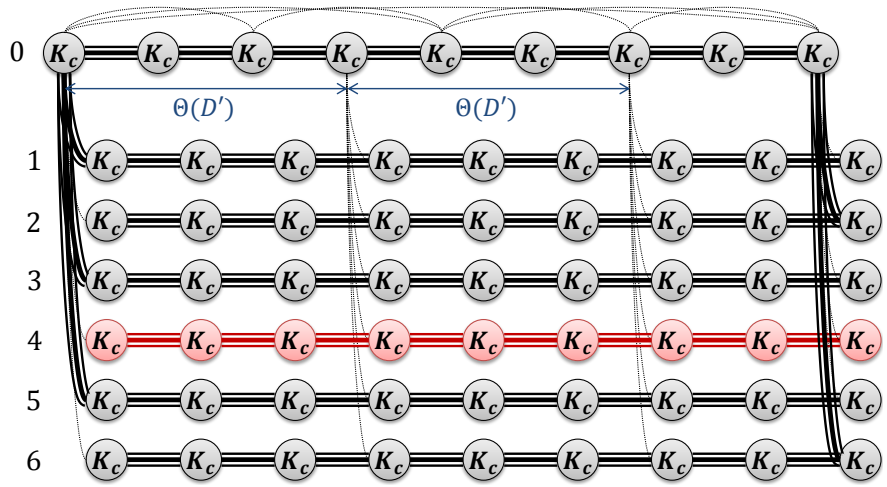


Figure 7-5: Illustration of the lower bound construction of Theorem 7.7.3 for parameters $k = 7$ and $\ell = 8$: Nodes in the construction of Figure 7-4 are replaced by cliques of size $c = \alpha\lambda + 1$. Connections marked by three lines mark complete bipartite subgraphs, thin dotted connections correspond to single edges connecting the two cliques. Similarly to Figure 7-4, each of the six rows 1–6 corresponds to one of the elements of the set disjointness universe and Alice' and Bob's inputs are encoded in the same way as before (replacing heavy edges by complete bipartite subgraphs). Vertical connections to the top path are only done every $\Theta(D')$ hops in order to keep the diameter at most $O(D' + \log n)$ and the size of the minimum cut as small as possible.

Proof Sketch. The basic proof argument is the same as the proof of Theorem 7.7.3. We therefore only describe the differences between the proofs. Because in a simple unweighted graph, we cannot add edges with different weights and we cannot add multiple edges, we have to adapt the construction. Assume that $\alpha \geq 1$ and $\lambda \geq 1$ are given. First note that for $\lambda = O(1)$, the statement of the theorem is trivial as $\Omega(D)$ clearly is a lower bound for approximating the edge connectivity or finding an approximate minimum cut. We can therefore assume that λ is sufficiently large.

We adapt the construction of the network G to get a simple graph G' as follows. First, every node of G is replaced by a clique of size $\alpha\lambda+1$. Recall that k is the number of paths and that each of these paths is of length $\ell-1$. Instead of ℓk nodes, the new graph G' therefore has $\ell k(\alpha\lambda + 1)$ nodes. All the "path" edges $e \in E_{H,1}$ are replaced by complete bipartite graphs between the cliques corresponding to the two nodes connected by e in G. For each edge $e \in E_{H,2}$—the edges that are used to reduce the diameter of the graph induced by the first path—, we add a single edge between two nodes of the corresponding cliques. Adding only one edge suffices to reduce the diameter of the graph induced by the cliques of the first path. For the edges in $E_{H,3}$, the adaptation is slightly more complex. The edges among the first k nodes and the last k nodes of G that are used to encode the set disjointness instance (X, Y) into the graph are adapted as follows. Each edge of weight $\alpha\ell + 1$ is replaced by a complete bipartite subgraph, whereas each edge of weight 1 is replaced by a single edge connecting the

corresponding cliques in G'. For the remaining edges, we introduce a parameter $D' = \ell/\lambda$. Instead of vertically connecting each of the ℓ cliques of all paths to stars (with the center in a node on path 0), we only add some of these vertical connections. We already connected the first and the last clique of each path. In addition, we add such vertical connections (a single edge between the clique on path 0 and each of the corresponding cliques on the other paths) such that: a) between two vertically connected "columns" there is a distance of at most $2D'$ and b) in total, the number of vertically connected "columns" is at most λ (including the first and the last column). Note that because the length of the paths is $\ell - 1$, the choice of D' allows to do so. We now get a graph G' with the following properties. The construction is also depicted in Figure 7-5.

- For each $x \in X \cap Y$, all the vertical connections are single edges connecting path x with path 0. The total number of edges connecting the cliques of path x with other nodes is at most λ.
- For each $x \notin X \cap Y$, path x is connected to path 0 through a complete bipartite graph $K_{\alpha\lambda+1,\alpha\lambda+1}$.
- The diameter of G' is $O(D' + \log n)$.

Let us consider the spanning subgraph G'' of G' induced by only the edges of all the complete bipartite subgraphs $K_{\alpha\lambda+1,\alpha\lambda+1}$ of our construction. If $X \cap Y = \emptyset$, G'' is connected. Therefore, in this case, the edge connectivity of G'' (and thus also of G') is at least $\alpha\lambda$. If $|X \cap Y| = 1$ and if we assume that z is the element in $X \cap Y$, G'' consists of two components. The first component is formed by all the nodes (of the cliques) of path z, whereas the second component consists of all the remaining nodes. By the above observation, the number of edges in G' between the two components of G'' is at most λ and therefore the edge connectivity of G' is at most λ. Also, every other edge cut of G' has size at least $\alpha\lambda$. Using the same reduction as in Theorem 7.7.3, we therefore obtain the following results

- If the edge connectivity of the network graph is not known, approximating it by a factor α requires $\Omega\big(\min\{k/(B\log n), \ell\}\big)$ rounds.
- If the edge connectivity λ is known, then finding a cut of size at most $\alpha\lambda$ requires at least $\Omega\big(\min\{k/(B\log n), \ell\}\big)$ rounds.

The lower bound then follows by setting $k/(B\log n) = \ell$. Together with $n = k\ell(\alpha\lambda + 1)$, we get $\ell = \Theta(\sqrt{n/(B\alpha\lambda\log n)})$. □

THIS PAGE IS LEFT BLANK INTENTIONALLY.

Chapter 8

Minimum-Weight Connected Dominating Set

8.1 Introduction & Related Work

Connected dominating set (CDS) is one of the classical structures studied in graph optimization problems which also has deep roots in networked computation. For instance, CDSs have been used rather extensively in distributed algorithms for wireless networks (see e.g. [CL02, AWF02a, WGS01, CWD08, DB97, MDJ+06, BDTC05, CHL+03, AWF02b, DW04, WAF02]), typically as a global-connectivity backbone.

In this chapter, we present distributed algorithms for approximating *minimum-weight connected dominating set* (MCDS) in the CONGEST model. We first take a closer look at the MCDS problem.

A Closeup of MCDS, in Contrast with MST: Given a graph $G = (V, E)$, a set $S \subseteq V$ is called a *dominating set* if each node $v \notin S$ has a neighbor in S, and it is called a *connected dominating set* (CDS) if the subgraph induced by S is connected. Figure 8-1 shows an example. In the *minim-weight CDS* (MCDS) problem, each node has a weight and the objective is to find a CDS with the minimum total weight.

The MCDS problem is often viewed as the node-weighted analogue of the *minimum-weight spanning tree* (MST) problem. Here, we recap this connection. The natural interpretation of the definition of CDS is that a CDS is a selection of *nodes* that provides *global-connectivity*—that is, any two nodes of the graph are connected via a path that its internal nodes are in the CDS. On the counterpart, a *spanning tree* is a (minimal) selection of *edges* that provides global-connectivity. In both cases, the problem of interest is to minimize the total weight needed for global-connectivity. In one case, each edge has a weight and the problem becomes MST; in the other, each node has a weight and the problem becomes MCDS.

Despite the seemingly analogous nature of the two problems, MCDS turns out to be a significantly harder problem: The MST problem can be computed sequentially in (almost) $O(m)$ time, where m is the number of edges. On the other hand, MCDS is NP-hard [GJ90],

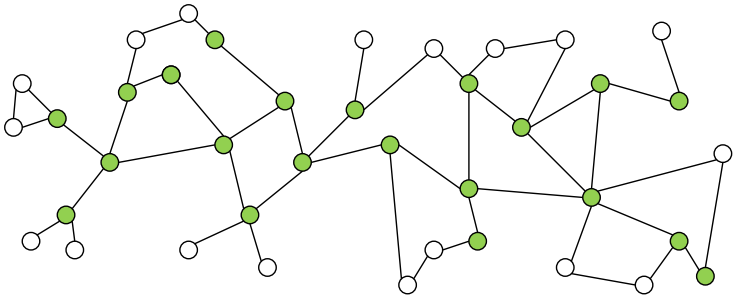


Figure 8-1: The green nodes represent a connected dominating set (CDS) of the graph.

and in fact, unless P = NP, no polynomial time algorithm can find any approximation better than $\Theta(\log n)$-factor for it (see [Fei96, RS97, AMS06]). Furthermore, the known sequential algorithms for $O(\log n)$ approximation of MCDS (see [GK98, GK99]) have unspecified polynomial time complexity, which are at least $\Theta(n^3)$.

The MST Problem in the Distributed Setting Given the aforementioned analogy between MCDS and MST, it is relevant to briefly recall the state of the art for MST in the distributed setting. A beautiful line of work shows that MST can be solved in $O(D + \sqrt{n}\log^* n)$ rounds [GKP93, KP95] and that this is (existentially) optimal modulo logarithmic factors [DSHK+11, Elk04c, PR99], and a similar lower bound also applies to many other distributed graph problems [DSHK+11][1]. By now, achieving an $\tilde{O}(D+\sqrt{n})$ round complexity is viewed as sort of a golden standard for (global) network optimization problems in the CONGEST model.

8.1.1 Result

Our contribution is to show that in the CONGEST model, MCDS can be solved—that is, approximated optimally—in a time close to that of MST.

Theorem 8.1.1. *There is a randomized distributed algorithm in the* CONGEST *model that, with high probability, finds an $O(\log n)$ approximation of the minimum-weight connected dominating set, using $\tilde{O}(D + \sqrt{n})$ rounds.*

This algorithm is (near) optimal in both round complexity and approximation factor: Using techniques of [DSHK+11], one can reduce the *two-party set-disjointness communication complexity* problem on $\Theta(\sqrt{n})$-bit inputs to MCDS, proving that the round complexity is optimal, up to logarithmic factors, for any approximation (see Section 8.5). As mentioned above, the $O(\log n)$ approximation factor is known to be optimal up to a constant factor, unless P = NP, assuming that nodes can only perform polynomial-time computations. Note that this assumption is usual, see e.g. [DMP+03, JRS01, KW03].

[1] For the reader interested in distributed (approximation) algorithms while considering congestion, the author recommends reading [DSHK+11] and the prior work on that thread, e.g., [Elk04c, PR99].

8.1.2 Other Related Work

To the best of our knowledge, no efficient algorithm was known before for MCDS in the CONGEST model. Notice that in the LOCAL model, MCDS boils down to a triviality and is thus never addressed in the literature: it is folklore[2] that in this model, D rounds is both necessary and sufficient for any approximation of MCDS. However, a special case of MCDS is interesting in the LOCAL model; the so-called "unweighted case" where all nodes have equal weight. Although, the unweighted-case has a significantly different nature as it makes the problem "*local*": Dubhashi et al. [DMP+03] present a nice and simple $O(\log n)$ approximation for the unweighted-case algorithm which uses $O(\log^2 n)$ rounds of the LOCAL model. To our knowledge, the unweighted case has not been addressed in the CONGEST model. Another problem which has a name resembling MCDS is the *minimum-weight dominating set* (MDS) problem. However, MDS is also quite different from MCDS as the former is "*local*", even in the weighted case and the CONGEST model: an $O(\log n)$ factor approximation can be found in $O(\log^2 n)$ rounds [JRS01, KW03].

8.2 Preliminaries

We assume that all nodes know an upper bound $N = \text{poly}(n)$ on n. We assume each node has a unique id with $O(\log n)$ bits, although this is not critical as each node simply picking a random id in $\{0,1\}^{4\log N}$ would ensure uniqueness of ids, with high probability. Also, recall from Chapter 2 that we use the phrase *with high probability* (w.h.p.) to indicate a probability being at least $1 - \frac{1}{n^c}$, for a desirably large constant $c \geq 2$.

Notations and basic definitions We work with an undirected graph $G = (V, E)$, $n = |V|$, and for each vertex $v \in V$, $c(v)$ denotes the weight (i.e., cost) of node v. Throughout the paper, we will use the words *cost* and *weight* interchangeably. For each subset $T \subseteq V$, we define $\text{cost}(T) = \sum_{v \in T} c(v)$. We assume the weights are at most polynomial in n, so each weight can fit in one message (such assumptions are usual, e.g. [GKP93]). We use notation OPT to denote the CDS with the minimum cost. Also, for convenience and when it does not lead to any ambiguity, we sometimes use OPT to refer to the cost of the optimal CDS.

Problem Statement Initially, each node v knows only its own weight $c(v)$. The objective is to find a set S in a distributed fashion—that is, each node v will need to output whether $v \in S$ or not—such that $\text{cost}(S) = O(\mathsf{OPT} \cdot \log n)$.

[2] On one hand, D rounds is enough for learning the whole graph. On the other, D rounds is necessary for guaranteeing any approximation factor α. Consider a cycle with with $2D$ nodes where two nodes v and u are at distance D. For each of v and u, assign a random weight in $\{n^2, n^2\alpha + 1\}$ and give weight 1 to each other node. For the CDS to α-factor optimal, the following should hold: if one of v and u has cost $n^2\alpha + 1$, then before joining the CDS, it needs to make sure that the other does not have weight n^2. This requires D rounds.

A Basic Tool (Thurimella's algorithm) A basic tool that we frequently use in this chapter is a *connected component identification algorithm* presented by Thurimella [Thu95], which itself is a simple application of the MST algorithm of Kutten and Peleg [KP95]. Given a subgraph $H = (V, E')$ of the main network graph $G = (V, E)$, this algorithm identifies the connected components of H by giving a label $\ell(v)$ to each v such that $\ell(v) = \ell(u)$ if an only if v and u are in the same connected component of H. This algorithm uses $O(D + \sqrt{n}\log^* n)$ rounds of the CONGEST model. It is easy to see that the same strategy can be adapted to solve the following problems also in $O(D + \sqrt{n}\log^* n)$ rounds. Suppose each node v has an input $x(v)$. For each node v, which is in a component $\mathcal{C}$ of H, we can make $\ell(v)$ be equal to: (A) the *maximum* value $x(u)$ for nodes $u \in \mathcal{C}$ in the connected component of v, or (B) the *list of* $k = O(1)$ *largest* values $x(u)$ for nodes $u \in \mathcal{C}$, or (C) the *summation* of values $x(u)$ for nodes $u \in \mathcal{C}$.

8.3 The Algorithm for MCDS

8.3.1 The Outline

The top-level view of the approach is as follows: We start by using the $O(\log^2 n)$ rounds algorithm of [JRS01] to find a dominating set S with cost $O(\log n \cdot \mathsf{OPT})$. The challenge is in adding enough nodes to connect the dominating set, while spending extra cost of $O(\log n \cdot \mathsf{OPT})$. We achieve connectivity in $O(\log n)$ phases. In each phase, we add some nodes to set S so that we reduce the number of connected components of S by a constant factor, while spending a cost of $O(\mathsf{OPT})$. After $O(\log n)$ phases, the number of connected components goes down to 1, meaning that we have achieved connectivity. Each phase uses $\tilde{O}(D + \sqrt{n})$ rounds of the CONGEST model. What remains is to explain how a phase works.

The reader might recall that such "component-growing" approaches are typical in the MST algorithms, e.g., [KP95, GKP93]. While in MST, the choice of the edge to be added to each component is clear (*the lightest outgoing edge*), the choice of the nodes to be added in MCDS is not clear (and in fact can be shown to be an NP-hard problem, itself).

The problem addressed in one phase can be formally recapped as follows (the reader might find the illustration in Figure 8-2 helpful here): We are given a dominating subset $S \subseteq V$ and the objective is to find a subset $S' \subseteq V \setminus S$ with $\text{cost}(S') = O(\mathsf{OPT})$ such that the following condition is satisfied. Let $\mathcal{F}$ be the set of subsets of S such that each $\mathcal{C} \in \mathcal{F}$ is a connected component of $G[S]$. Call a connected component $\mathcal{C} \in \mathcal{F}$ *satisfied* if in $G[S \cup S']$, $\mathcal{C}$ is connected to at least one other component $\mathcal{C}' \in \mathcal{F}$. We want S' to be such that at least half of the connected components of $G[S]$ are satisfied. Note that if this happens, then the number of connected components goes down by a 3/4 factor. To refer to the nodes easier, we assume that all nodes that are in S at the start of the phase are colored *green* and all the other nodes are *white*, initially. During the phase, some white nodes will become gray meaning that they joined S'.

Before moving on to the algorithm, we emphasize two key points:

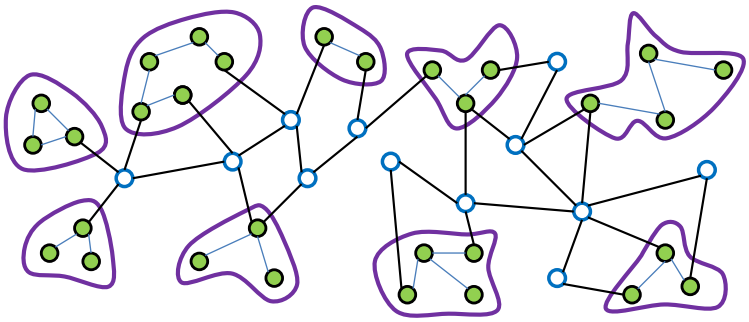


Figure 8-2: An example scenario at the start of a phase. Green nodes indicate those in S and white nodes are $V \setminus S$. Unrelated nodes and edges are discarded from the picture.

(1) It is critical to seek satisfying only a constant fraction of the components of $G[S]$. Using a simple reduction from the set cover problem, it can be shown that satisfying all components might require a cost $O(\mathsf{OPT} \log n)$ for a phase. Then, at least in the straightforward analysis, the overall approximation factor would become $O(\log^2 n)$.

(2) In each phase, we *freeze* the set of components $\mathcal{F}$ of $G[S]$. That is, although we continuously add nodes to the CDS and thus the components grow, we will not try to satisfy the newly formed components. We keep track of whether a component $\mathcal{C} \in \mathcal{F}$ is satisfied and the satisfied ones become "inactive" for the rest of the phase, meaning that we will not try to satisfy them again. However, satisfied components will be used in satisfying the others.

8.3.2 A High-level View of the Algorithm for One Phase

Note that since S is a dominating set, $\mathcal{C} \in \mathcal{F}$ is satisfied iff there exist one or two nodes that connect $\mathcal{C}$ to another component $\mathcal{C}' \in \mathcal{F}$. That is, either there is a node v such that path $\mathcal{C}$-v-$\mathcal{C}'$ connects component $\mathcal{C}$ to component $\mathcal{C}'$ or there are two adjacent nodes v and w such that path $\mathcal{C}$-v-w-$\mathcal{C}'$ does that. Having this in mind, and motivated by the solution for the unweighted case [DMP+03], a naive approach would be that, for each component $\mathcal{C}$, we pick one or two nodes—with smallest total weight—that connect $\mathcal{C}$ to another component, and we do this for each component $\mathcal{C}$ independently. However, in the weighted case, this naive idea would perform terribly. To see why, let us consider a simple example (see Figure 8-3): take a cycle with $n-1$ nodes where every other node has weight 1 and the others have weight $\sqrt{n}$, and then add one additional node at the center with weight n, which is connected to all weight-1 nodes. Clearly, the set of weight-1 nodes gives us an optimal dominating set. However, naively connecting this

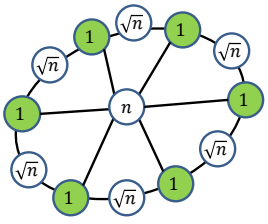


Figure 8-3: The naive approach

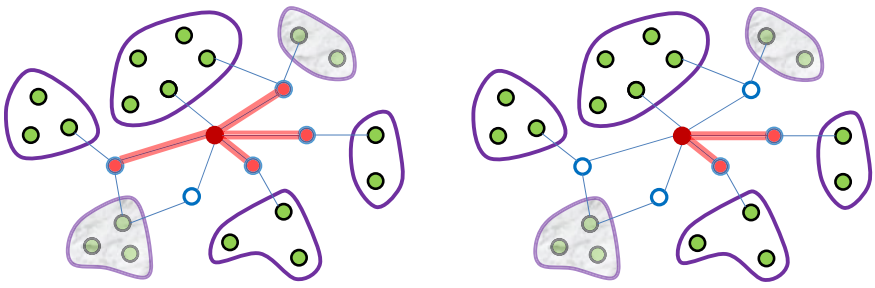


Figure 8-4: A basic-star. The opaque components indicate those that are already satisfied and thus deactivated. Two legs of the general star (colored red, on the left) are discarded in the basic-star (colored red, on the right), as each of them forms a useful star, meaning that the leg itself can satisfy at least one active component.

dominating set following the above approach would make us include at least half of the $\sqrt{n}$-weight nodes, leading to overall weight of $\Theta(n\sqrt{n})$. On the other hand, simply adding the center node s to the dominating set would provide us with a CDS of weight $O(n)$.

Inspired by this simple example, we view *stars* as the key elements of optimization (instead of 2 or 3 hop paths). We next define what we mean by a star and outline how we use it. We note that the concept is also similar to the notion of *spiders* used in [KR95] for the node-weighted Steiner trees problem.

Definition 8.3.1. ***(Stars)*** *A star X is simply a set of white nodes with a* center *$s \in X$ such that each non-center node in the star is connected to the center s. Naturally, we say a star X* satisfies *an active component $\mathcal{C} \in \mathcal{F}$ if adding this star to S'—that is, coloring its nodes gray—would connect $\mathcal{C}$ to some other component and thus make it satisfied. Let $\Phi(X)$ be the set of unsatisfied components in $\mathcal{F}$ that would be satisfied by X. We say a star is useless if $\Phi(X) = \emptyset$. The cost of a star X is $\text{cost}(X) = \sum_{w \in X} c(w)$ and its efficiency is $\rho(X) = \frac{|\Phi(X)|}{\text{cost}(X)}$. We say X is ρ'-efficient if $\rho(X) \geq \rho'$.*

In Figure 8-3, each white node is one star, the center has efficiency $\Theta(1)$ and every other star has efficiency $\Theta(1/\sqrt{n})$. Notice that in general, different stars might intersect and even a white node v might be the center of up to $2^{\Theta(n)}$ different stars.

The general plan (while ignoring some difficulties) We greedily[3] add stars to the gray nodes. That is, we pick a star that has the maximum efficiency and color its nodes gray. It can be shown that this greedy idea would satisfy half of components using cost only $\mathcal{O}(\mathsf{OPT})$. However, clearly adding stars one by one would be too slow. Instead we adopt a nice and natural technique due to Berger et al. [BRS94] which by now has become a standard trick for speeding up greedy approaches via parallelizing their steps. The key point is, stars that have efficiency within a constant factor of the max-efficiency are essentially

[3]The greedy approaches are typically standard in solving MCDS or other problems similar in nature. Furthermore, often the notion of *efficiency* as explained above or some variant of it is the base of picking the next good move, in these greedy approaches. See e.g. [GK98, GK99, KR95, BRS94].

as good as the max-efficient star and hence, we can add those as well. The only catch is, one needs to make sure that adding many stars simultaneously does not lead to (too much) *double counting* in the efficiency calculations. In other words, if there are many stars that try to satisfy the same small set of components, even if each of these stars is very efficient, adding all of them is not a good idea. The remedy is to probabilistically add stars while the probabilities are chosen such that not too many selected stars try to satisfy one component.

While this general outline roughly explains what we will do, the plan faces a number of critical issues. We next briefly hint at two of these challenges and present the definitions that we use in handling them.

Challenge 1 The first step in the above outline is to compute (or approximate) the efficiency of the max-efficient star. Doing this for the general class of stars turns out the be a hard problem in the CONGEST model. Note that for a white node v to find (or approximate) the most-efficient star centered on it, v would need to know which components are adjacent to each of its white neighbors. As each white node might be adjacent to many components, this is like learning the 2-neighborhood of v and appears to be intrinsically slow in the CONGEST model. Instead, we will focus on a special form of stars, which we call *basic-stars* and explain next. Figure 8-4 shows an example.

Definition 8.3.2. ***(Basic-Stars)*** *Call a white node u* self-sufficient *if u is adjacent to two or more components, at least one of which is not satisfied. A star X is called* basic *if for each non-center node $w \in X$, w is not self-sufficient. That is, the star $X' = \{w\}$ is useless.*

We argue later that, considering only the basic-stars will be sufficient for our purposes (sacrificing only a constant factor in the approximation quality) and that we can indeed evaluate the max-efficiency of the basic-stars.

Challenge 2 The other issue, which is a bit more subtle but in fact significantly more problematic, is as follows: as we color some white nodes gray, some components grow and thus, the efficiencies of the stars change. For instances, a useless star $X = \{v\}$ might now become useful–e.g., it gets connected to a satisfied component $\mathcal{C}'$ via a node u that just got colored gray, and X can now satisfy an adjacent unsatisfied component $\mathcal{C}$ by connecting it to $\mathcal{C}'$. Another example, which is rooted also in the congestion related issues, is as follows: During our algorithm, to be able to cope with communication issues, each white node v will work actively on only one max-efficient basic-star centered on v. But, v might be the center of many such stars and even if one of them looses the efficiency after this iteration, another max-efficient star which existed before might be now considered actively by v.

We note that, if there were no such *"new-stars"* issues, we could use here standard methods such as (a modification of) the LP relaxation based technique of Kuhn and Wattenhofer [KW03]. However, these changes break that approach and it is not even clear how to formulate the problem as an LP (or even a convex optimization problem, for that matter).

If not controlled, these changes in the stars can slow down our plan significantly. For example, if for a given almost-maximum efficiency $\tilde{\rho}$, in each iteration a small number of $\tilde{\rho}$-efficient new basic-stars are considered actively, we will have to spend some time on these stars but as the result, we would satisfy only very few components, which would become prohibitively slow. To remedy this, when coloring stars gray, we will do it for certain types of $\tilde{\rho}$-efficient basic-stars, which we define next, and after that, we do some clean up work to remove the new $\tilde{\rho}$-efficient basic-stars that would be considered actively later on.

Definition 8.3.3. *(ρ^*-**Augmented Basic-Stars**) A ρ^*-efficient basic-star X centered on node $v \in X$ is called ρ^*-minimal if for any other star $X' \subset X$ centered on v, we have, $\rho(X') < \rho^*$. For a ρ^*-minimal basic-star X centered on v, a* good auxiliary-leg *is a white node $u \notin X$ that is adjacent to v and furthermore, the following conditions are satisfied: u is adjacent to only one component $\mathcal{C} \in \mathcal{F}$, component $\mathcal{C}$ is not satisfied and it is not adjacent to X, and we have $\text{cost}(u) \leq 2/\rho^*$. A* ρ^*-Augmented Basic-Star *X' is one that can be derived by (one-by-one)[4] adding to ρ^*-minimal basic-star X all good auxiliary-legs adjacent to its center.*

An example is shown in Figure 8-5. The actual reasoning for why this definition is good is somewhat subtle to be explained intuitively. A very rough version is as follows: after coloring some ρ^*-augmented basic-stars gray, by just handling the nodes which each have cost at most $1/\rho^*$ (in a step we call clean up), we will be able to remove any new ρ^*-augmented basic-star. The point should become clear after seeing the algorithm (and Lemma 8.4.2 and Lemma 8.4.3).

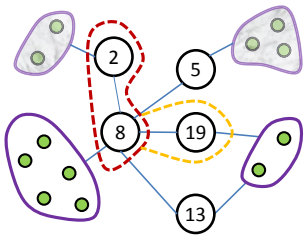


Figure 8-5: A 0.1-augmented basic-star is indicated with the dashed lines; the red part is a minimal 0.1-efficient basic-star and the orange part is a good auxiliary leg.

Observation 8.3.4. *Each ρ^*-Augmented Basic-Star X has efficiency $\rho(X) \geq \frac{\rho^*}{2}$. Furthermore, if a ρ^*-Augmented Basic-Stars X contains a white node w, then all unsatisfied components adjacent to w get satisfied by X.*

8.3.3 The Algorithm For One Phase

The objective of the algorithm is to satisfy at least half of the components, using a cost $O(\mathsf{OPT})$, and in $O((D + \sqrt{n}\log^* n)\log^3 n)$ rounds. Throughout the phase, each non-white node will keep track of whether its component in $\mathcal{F}$ is satisfied or not. Let $N = |\mathcal{F}|$ and also, make all nodes know N by running Thurimella's connected component identification at the start of the phase and then globally gathering the number of components.

While at least $\lfloor N/2 \rfloor$ components in $\mathcal{F}$ remain unsatisfied, we repeat the following iteration, which has 8 steps—$\mathcal{S}1$ to $\mathcal{S}8$—and each step uses $O(D + \sqrt{n}\log^* n)$ rounds:

[4] This has to be done one-by-one as adding one good auxiliary leg might make the star adjacent to a component $\mathcal{C}$ and then, no other white node adjacent to $\mathcal{C}$ can be a good auxiliary-leg.

($\mathcal{S}1$) We first use Thurimella's algorithm (see Section 8.2) to identify the connected components of non-white nodes and also to find out whether each component is satisfied (i.e. if it contains a gray node). These take $O(D+\sqrt{n}\log^* n)$ rounds. Each non-white node broadcasts its component id and whether its component is satisfied to all neighbors. We also find the total number of unsatisfied connected components and if it is less than $N/2$, we call this phase finished and start the next phase.

($\mathcal{S}2$) We now find the globally-maximum efficiency ρ^* of the basic-stars.

They key part is to compute the efficiency of the most-efficient basic-star centered on each white node. After that, the global-maximum can be found in $O(D)$ rounds easily. We first use one round of message exchanges between the white nodes so that each white node knows all the basic-stars it centers.

Each white node v does as follows: if v is adjacent to only one component (satisfied or unsatisfied), it sends the id of this component, its satisfied/unsatisfied status and v_{id} to its neighbors. If v is adjacent to two or more components, but all of them are satisfied, then v sends a message to its neighbors containing v_{id} and an indicator message "*all-satisfied*". If v is adjacent to two or more components, at least one of which is unsatisfied, then v does not send any message. This is because, by Definition 8.3.2, node v is *self-sufficient* and it thus can be only in basic-stars centered on v. At the end of this round, each white node v has received some messages from its white neighbors. These messages contain all the information needed for forming all the basic-stars centered on v and calculating their efficiency. Node v finds the most-efficient of these basic-stars. It is easy to see that this can indeed be done in polynomial-time local computation[5]. We emphasize that the basic-stars found in this step are not important and the only thing that we want is to find the globally-maximum efficiency ρ^*.

($\mathcal{S}3$) Let $\tilde{\rho} = 2^{\lfloor \log_2 \rho^* \rfloor}$, i.e., $\tilde{\rho}$ is equal to ρ^* rounded down to the closest power of 2. We pick at most one $\tilde{\rho}$-augmented basic-star X_v^i (see Definition 8.3.3) centered on each white node v, where i is the iteration number.

We reuse the messages exchanged in the previous step. First, each white node v finds a minimal $\tilde{\rho}$-efficient basic-star centered on v, if there is one. Call this the *core-star of* v. Then, v adds to this core-star any good auxiliary-legs available (one by one), to find its ρ^*-augmented basic-star X_v^i. This is the only star centered on v that will be considered for the rest of this iteration. Thus, at most one star X_v^i centered on each white node v remains active for the rest of iteration i. Note that all active remaining stars are $\tilde{\rho}/2$-efficient.

[5]For that, node v first adds itself to the basic-star. Then, it discards any adjacent white node u for which the only unsatisfied component adjacent to u is also adjacent to v. Then, v sorts all the remaining white-neighbors $u_1, u_2, ..., u_\ell$—from which it received a message—by increasing cost order. It then adds u_i-s one by one to its basic-star, as long as each new addition increases the efficiency. Since each white-neighbor is adjacent to at most one unsatisfied component, it is easy to see that this indeed finds the maximum efficiency.

For each active-remaining star X_v^i and each unsatisfied component $\mathcal{C}$ it satisfies, the center v elects one of the white nodes of the star to be *responsible for communicating*[6] with $\mathcal{C}$. If $\mathcal{C}$ has at least one non-center neighbor in X_v^i, then one such non-center node u (selected arbitrarily) is called *responsible for communicating with* $\mathcal{C}$. Otherwise, the center v is responsible[7] for communicating with $\mathcal{C}$.

($\mathcal{S}4$) For each unsatisfied component $\mathcal{C} \in \mathcal{F}$, we find the number of active stars that satisfy $\mathcal{C}$. The objective is to find the maximum such number $\Delta_{\tilde{\rho}}^*$, over all unsatisfied components. First, each white node v that centers an active star X_v^i reports this star to each non-center node u of it, by just sending v_{id}, special message *active-star*, and the id of the component $\mathcal{C}$ for which u is responsible for communicating with (if there is one). Then, for each white node w and each unsatisfied component $\mathcal{C}$ that w is responsible for communicating with it in any star, node u sends to one of the nodes of $\mathcal{C}$ the number of stars in which u is responsible for communicating with $\mathcal{C}$. These counts are summed up in each component $\mathcal{C}$ via Thurimella's algorithm, and it is called the *active-degree* of $\mathcal{C}$. The maximum active-degree is found globally and called $\Delta_{\tilde{\rho}}^*$.

($\mathcal{S}5$) Next, some active stars propose to their adjacent unsatisfied components.

We mark each active star with probability $\frac{1}{5\Delta_{\tilde{\rho}}^*}$, where the decision is made randomly by the center of the star and sent to the other nodes of the star (if there is any). Then, these marks are sent to the components that get satisfied by the marked stars, as proposals, via the white nodes that are responsible for communicating with the components. If v is self-sufficient, it would need to send at most one proposal to each adjacent component (it would be to those components for which v is responsible for communicating with them in X_v^i). However, if v is not self-sufficient, then v might want to send many proposals to an unsatisfied component adjacent to it (there is at most one such component). This is not feasible in the CONGEST model. Instead, v selects at most 3 of these proposals (arbitrarily) and just submits these 3 proposals.

($\mathcal{S}6$) Each component grants at most 3 of the proposals it receives. This is done via Thurimella's algorithm, where 3 proposals with largest center ids are granted. Finally components report the granted proposals to the adjacent white nodes.

($\mathcal{S}7$) Each marked star collects how many of its proposals are granted. If at least 1/3 of the proposals of this star were granted, then all nodes of this marked star become gray. After that, we use Thurimella's algorithm again to identify the green nodes which their component (in $\mathcal{F}$) is satisfied (by checking if their component has a gray node).

[6]Note that each star might have many nodes that are adjacent to an unsatisfied component $\mathcal{C}$. As this would be problematic for our communication purposes, we avoid this by making only one node in the star responsible for each unsatisfied adjacent component.

[7]Since any white node u that is not self-sufficient is adjacent to at most one unsatisfied component, in any basic-star that contains u, node u can be responsible only for this one unsatisfied adjacent component. On the other hand, if v is self-sufficient, it will be only in one star X_v^i.

($\mathcal{S}8$) Finally, we have a *clean up* step, which removes the newly-formed $\tilde{\rho}$-augmented basic-stars that if not removed now, might be active in the next iterations. Temporarily (just for this clean up step) color each white node *blue* if its cost is at most $1/\tilde{\rho}$. For each unsatisfied component $\mathcal{C} \in \mathcal{F}$ that can be satisfied using only blue nodes, we find one or two blue nodes that connect $\mathcal{C}$ to some other component in $\mathcal{F}$ and we color these blue nodes gray, thus making $\mathcal{C}$ satisfied. In the first round, for each blue node v, if v is adjacent to only one component, it sends the id of this component and its own id v_{id}. If v is adjacent to two or more components, it just sends its own id with an indicator symbol *"two-or-more"*. In the second round, for each blue node u, if u is adjacent to an unsatisfied component $\mathcal{C}$, node u creates a proposal for $\mathcal{C}$ as follows: if u is adjacent to at least one other component $\mathcal{C}' \in \mathcal{F}$, then the proposal is simply the id of u. If u is not adjacent to any other component $\mathcal{C}'$ but there is a blue neighbor w of u such that in the first round, w sent the id of a component $\mathcal{C}'' \neq \mathcal{C}$ or w sent the *"two-or-more"* indicator symbol, then the proposal contains the ids of u and v. Otherwise, the proposal is empty. Each unsatisfied component picks one (nonempty) proposal, if it receives any, and grants it. The granted proposal is reported to all nodes adjacent to the component and if the proposal of u is granted, it becomes gray and if this granted proposal contained a blue neighbor w, then u informs w about the granted proposal which means that w also becomes gray.

A remark about the time complexity In the above algorithm, each phase takes $O(\log^3 n)$ iterations, w.h.p., which leads to $O((D + \sqrt{n}\log^* n)\log^3 n)$ rounds for each phase, and thus $O((D + \sqrt{n}\log^* n)\log^4 n)$ rounds for the whole algorithm. One can remove one logarithmic factor off of this complexity by (further) leveraging the fact that in each phase, we need to satisfy only half of the components. To do that, if for a max-efficiency level $\tilde{\rho}$ and the respective max-component-degree $\Delta^*_{\tilde{\rho}}$, we have satisfied at least half of the components with degree at least $\Delta^*_{\tilde{\rho}}/2$, we can *excuse* the other half from needing to be satisfied in this phase. This way, with constant probability, after just a constant number of iterations, we are done with components of degree at least $\Delta^*_{\tilde{\rho}}/2$. A standard concentration bound then shows that w.h.p. $O(\log n)$ iterations are enough for all degree levels (with respect to efficiency $\tilde{\rho}$). We leave formalizing the details of this $\log n$ factor optimization to the interested reader.

8.4 Analysis

For the analysis, we need to establish two facts, (1) that the cost of each phase is $O(\mathsf{OPT})$, and (2) that each phase takes only $O(\log^3 n)$ iterations, w.h.p. As each iteration is implemented in $O(D + \sqrt{n}\log^* n)$ rounds, these prove the desired properties of each phase.

8.4.1 Cost Related Analysis

In each iteration, we color some white nodes gray and thus satisfy some components. This is done in a way that the overall efficiency of the nodes added in this iteration is within a constant factor of the best basic-star. That is, the number of components satisfied in this iteration is $\Theta(\rho^*)$ times the total cost of the nodes colored gray in this iteration. As the heart of cost analysis, we show that in each iteration, as long as at least $N/2$ unsatisfied components exist, $\rho^* \geq \frac{N}{4\mathsf{OPT}}$. This will be done by showing that one can cover the (white nodes of) OPT with basic-stars, such that each white node is in at most 2 basic-stars.

Lemma 8.4.1. *In each phase, we spend a cost of at most $O(\mathsf{OPT})$.*

Proof. First note that, in steps $\mathcal{S}6$ and $\mathcal{S}7$, each active star has efficiency at least $\tilde{\rho}/2$, and an active star becomes gray if at least $1/3$ of its proposals are granted and each component grants at most 3 proposals. Thus, the efficiency of the whole set of white nodes colored gray in step $\mathcal{S}7$ is $\Theta(\tilde{\rho})$. Moreover, in the clean up step (step $\mathcal{S}8$), each component grants at most one proposal and each proposal contains at most two blue nodes, each of which has weight at most $1/\tilde{\rho}$. Hence, the efficiency in the clean up step is also $\Theta(\tilde{\rho})$.

Now as the key part of the proof, we claim that in each iteration in which at least $N/2$ unsatisfied components remain, there is at least one basic-star that has efficiency of at least $\frac{N}{4\mathsf{OPT}}$. This claim implies that in this iteration, $\tilde{\rho} \geq \rho^*/2 \geq \frac{N}{8\mathsf{OPT}}$. Over all iterations of this phase, we satisfy at most N components, always with an efficiency $\Theta(\tilde{\rho})$, which means that we spend a cost of $O(\mathsf{OPT})$ over the whole phase. Recall that if the number of the unsatisfied components drops below $N/2$, we call the phase finished, and move to the next phase.

Now to prove the claim, consider one iteration and assume that at least $N/2$ unsatisfied components remain. For the sake of contradiction, suppose that each basic-star has efficiency strictly less than $\frac{N}{4\mathsf{OPT}}$. Consider the minimum-cost CDS OPT. Let T be the set of white nodes in OPT. We cover T with a number of basic-stars $X_1, X_2, \ldots, X_\ell$ such that each node of T is in at most two of these basic-stars and each unsatisfied component can be satisfied by at least one of these basic stars. Then, for each X_i, define $C'(X_i) = \text{cost}(X_i)/2$. Note that $\sum_{i=1}^{\ell} C'(X_i) \leq \sum_{v \in T} c(v) = \mathsf{OPT}$. Each basic-star X_i splits cost $C'(X_i)$ equally between the unsatisfied components $\Phi(X)$ that get satisfied by X. That is, each such component gets cost $\frac{C'(X_i)}{|\Phi(X_i)|} > \frac{2\mathsf{OPT}}{N}$ from star X_i. Hence, each unsatisfied component gets a cost strictly greater than $\frac{2\mathsf{OPT}}{N}$ and summed up over all the unsatisfied components—which are at least $N/2$ many—, we get that $\sum_{i=1}^{\ell} C'(X_i) > \mathsf{OPT}$, which is a contradiction.

What is left is thus to show that we can cover T with a number of basic-stars X_1, X_2, $\ldots$, X_ℓ such that each node of T is in at most two of these basic-stars and each unsatisfied component can be satisfied by at least one of these basic stars. We give a simple sequential procedure which produces such basic-stars. During this procedure, each node $v \in T$ keeps a Boolean variable hit_v which is false initially. For each node $v \in T$, call v *lonely* if it is adjacent to exactly one component and that component is not satisfied.

Sequentially, go over the nodes in T one by one and for each $v \in T$, do as follows: consider the star X_v made of v and all lonely neighbors w of v that are not hit so far, i.e., those such that $hit_w = false$. Add X_v to the collection if it satisfies at least one component, and if this happens, also for each $w \in X_v \setminus \{v\}$, set $hit_w = true$. Note that if a lonely node w gets hit, then the single unsatisfied component $\mathcal{C}$ adjacent to w gets satisfied by X_v.

Now note that in this algorithm, each node u will be in at most two stars, one star X_u that is centered on u, and one star X_v that is centered on a neighbor v of u and such that in the iteration in which we consider w, we set $hit_u = true$.

On the other hand, consider an unsatisfied component $\mathcal{C}$. We show that $\mathcal{C}$ gets satisfied by one of the basic-stars produced by the above algorithm. Note that OPT satisfies $\mathcal{C}$. Therefore, there is a white node $v \in T$ that is adjacent to $\mathcal{C}$ and either v is adjacent to a different component $\mathcal{C}' \neq \mathcal{C}$ or v has another white neighbor $w \in T$ and w is adjacent to a different component $\mathcal{C}' \neq \mathcal{C}$. Now if the node v is not lonely, it is adjacent to at least two components, and hence X_v satisfies $\mathcal{C}$ and we are done. Otherwise, suppose v is lonely. Since v is lonely, when we consider w in the loop, either v is already hit by some other basic-star $X_{w'}$, or the basic-star X_w hits v. In either case, $\mathcal{C}$ gets satisfied. This finishes the proof. □

8.4.2 Speed Related Analysis

The speed analysis has more subtle points. We show that after $O(\log^3 n)$ iterations, at least half of the components would be satisfied and thus this phase ends. A critical point for establishing this is to show that, thanks to the clean up step (analyzed in Lemma 8.4.2), for each unsatisfied component, the number of active $\tilde{\rho}$-augmented basic-stars X_v^i that satisfy this component is monotonically non-increasing when viewed as a function of the iteration number i. This part will be our main tool for managing the issue of "*new stars*" (discussed in Challenge 2 above), and is proven in Lemma 8.4.3. Furthermore, in Lemma 8.4.4, we show that with at least a constant probability, a constant fraction of the components that are now each in at least $\Delta^*_{\tilde{\rho}}/2$ active stars X_v^i of iteration i get satisfied. Hence, it will follow that in iteration $j = i + O(\log n)$, there remains no unsatisfied component that can be satisfied by at least $\Delta^*_{\tilde{\rho}}/2$ many active stars X_v^j, w.h.p. Thus, in each $O(\log n)$ iterations, $\Delta^*_{\tilde{\rho}}$ decreases by a factor of 2, w.h.p. After $O(\log^2 n)$ iterations of the loop, there will be no $\tilde{\rho}$-augmented basic-star, and hence, no basic-star with efficiency $\tilde{\rho}$. Then we move to the next efficiency level, which is at most $\tilde{\rho}/2$. After $O(\log^3 n)$ iterations, more than half of the components would be satisfied and we stop the phase.

Lemma 8.4.2. *In the clean up step, if a unsatisfied component $\mathcal{C}$ could have been satisfied using only blue nodes, it indeed gets satisfied.*

Proof. Suppose that unsatisfied component $\mathcal{C}$ can be satisfied using only blue nodes. Then, as green-or-gray nodes dominate the graph, there is a component $\mathcal{C}'$ and either one blue node v such that $\mathcal{C}$-v-$\mathcal{C}'$ connects $\mathcal{C}$ to $\mathcal{C}'$or two adjacent blue nodes v and w such that $\mathcal{C}$-v-w-$\mathcal{C}'$ is a path connecting $\mathcal{C}$ to $\mathcal{C}'$. In the former case, v clearly proposes to $\mathcal{C}$. In the latter case, v

will receive either the id of $\mathcal{C}'$ or the special symbol "*two-or-more*" from w. And thus again, in either case, v proposes to $\mathcal{C}$. Therefore, component $\mathcal{C}$ will receive at least one proposal. Component $\mathcal{C}$ will accept one proposal, and this will make it connect to one other component $\mathcal{C}''$ (which might be equal to $\mathcal{C}'$) and hence satisfied. □

Lemma 8.4.3. *For each unsatisfied component $\mathcal{C} \in \mathcal{F}$ and the almost-max-efficiency $\tilde{\rho}$, the number of $\tilde{\rho}$-augmented basic-stars X_v^i selected in* step $\mathcal{S}3$ *that satisfy $\mathcal{C}$ does not increase from one iteration to the next.*

Proof. Fix an iteration $i \geq 2$. We claim that if an unsatisfied component $\mathcal{C} \in \mathcal{F}$ can be satisfied by a $\tilde{\rho}$-augmented basic-star X_v^i centered on v that was selected in step $\mathcal{S}3$ of iteration i, then there was one $\tilde{\rho}$-augmented basic-star X_v^{i-1} centered on v that was selected in step $\mathcal{S}3$ of iteration $i-1$ and $\mathcal{C}$ could be satisfied by X_v^{i-1} as well. This then directly leads to the lemma.

We first show that it cannot be the case that the core X'' of X_v^i (see step $\mathcal{S}3$ and Definition 8.3.3 for definition of core) was a useless star in iteration $i-1$. Having this established, we then show that X_v^{i-1} could satisfy $\mathcal{C}$, as well.

First, for the sake of contradiction, suppose that the core basic-star $X'' \subseteq X_v^i$ was useless in iteration $i-1$. Suppose that X'' could satisfy unsatisfied component $\mathcal{C}'$ in iteration i. Component $\mathcal{C}'$ might be equal to $\mathcal{C}$ or not. As X'' is useful in iteration i but not in iteration $i-1$, it means that there is a node u that was white at the start of iteration $i-1$ and it became gray at the end of that iteration and such that u connects X'' to a now satisfied component $\mathcal{C}'' \neq \mathcal{C}'$. In iteration i, X'' cannot be adjacent to two unsatisfied components as then it would be useful in iteration $i-1$. As in iteration i basic-star X'' is $\tilde{\rho}$-efficient and it satisfies only one unsatisfied component, we get that the total cost of nodes in X'' is at most $\frac{1}{\tilde{\rho}}$. Hence, all nodes of X'' were blue in the clean up step of iteration $i-1$. Furthermore, u was either gray at the start of the clean up step of iteration $i-1$ or it was a blue node in that step and then it became gray. We know that u is not adjacent to $\mathcal{C}'$ (otherwise $\mathcal{C}'$ would be satisfied). But, we know that each node of X'' must be adjacent to at least one green node, and in iteration $i-1$, X'' could not have been adjacent to more than one component (otherwise it would be useful for satisfying $\mathcal{C}'$). Thus, we get that each node of X'' is adjacent to component $\mathcal{C}'$. Therefore, $\mathcal{C}'$ could have been satisfied using only one or two blue nodes: either with one blue node of X'' connecting it to u which was gray then, or with one blue node of X'' and node u which was blue then. Hence, Lemma 8.4.2 gives that $\mathcal{C}'$ must have been satisfied at the end of iteration $i-1$ (perhaps through a different path). This is in contradiction with X'' having $\mathcal{C}'$ as its unsatisfied adjacent component in iteration i. Thus, we conclude that the core basic-star X'' was useful in iteration $i-1$.

Note that if an unsatisfied component is adjacent to X'' in iteration i, it was adjacent to X'' in iteration $i-1$ as well. Hence, the number of unsatisfied components that could be satisfied by X'' in iteration $i-1$ is at least as many as those that could be satisfied in iteration i. This establishes that X'' was at least $\tilde{\rho}$-efficient in iteration $i-1$. Thus, indeed

there was a $\tilde{\rho}$-augmented basic-star X_v^{i-1} centered on v and selected in step $\mathcal{S}3$ of iteration $i-1$. It remains to show that X_v^{i-1} could satisfy $\mathcal{C}$.

For the sake of contradiction, suppose that X_v^{i-1} was not adjacent to $\mathcal{C}$ (as otherwise we would be done). It means that there is another white node $w \in X_v^i \setminus X_v^{i-1}$ that connects $\mathcal{C}$ to v. Also, $\mathcal{C}$ is the only component in $\mathcal{F}$ that is adjacent to w as otherwise, w would have been self-sufficient and hence it would not report $\mathcal{C}$ to v in iteration i and thus it would not be in X_v^i (recall the definition of basic-star). Therefore, we know that in iteration $i-1$, $\{w\}$ could have potentially been a good auxiliary-leg for the core of X_v^{i-1}. As $\{w\}$ was not included in X_v^{i-1}, we know $\{w\}$ was not a good auxiliary-leg. As X_v^{i-1} is not adjacent to $\mathcal{C}$, from Definition 8.3.3 we can infer it must have been the case that $\text{cost}(w) > 2/\tilde{\rho}$. But now in iteration i, basic-star X_v^i which includes w and thus has cost strictly greater than $2/\tilde{\rho}$ has efficiency $\tilde{\rho}$. So, X_v^i must satisfy at least 3 components. But then, even if we discard w from star X_v^i, we get a smaller $\tilde{\rho}$-efficient basic-star. Hence, w was not included in the core X'' of X_v^i, which means that w was included in X_v^i as a good auxiliary-leg, showing that $\text{cost}(w) \leq 2/\tilde{\rho}$, which is a contradiction. Having arrived at the contradiction from the assumption that X_v^{i-1} was not adjacent to $\mathcal{C}$, we get that X_v^{i-1} must indeed have been adjacent to $\mathcal{C}$. That is, the $\tilde{\rho}$-augmented basic-star centered on v in iteration $i-1$ could have satisfied $\mathcal{C}$. This completes the proof. □

Lemma 8.4.4. *In each iteration i, the set of grayed stars has efficiency within a constant factor of the max-efficient basic-star. Furthermore, with at least a constant probability, a constant fraction of the components that can be satisfied by at least $\Delta^*_{\tilde{\rho}}/2$ many $\tilde{\rho}$-augmented basic-stars X_v^i get satisfied.*

Proof. For the first part, note that a basic-star joins if it is almost max-efficient and at least 1/3 of its proposals are granted, and each component grants at most 3 proposals. Thus, the set of grayed stars has efficiency within a constant factor of the max-efficient basic-star.

For the second part, first note that the probability that an almost-max-efficient star X is marked active and at least 1/3 of its proposals are accepted is $\Theta(1/\Delta^*)$. This is because, X is marked active with probability $1/(5\Delta^*)$ and then, for each unsatisfied component $\mathcal{C}$ that gets satisfied by X, the probability that more than 3 stars satisfying $\mathcal{C}$ are marked is at most $\binom{\Delta^*_{\tilde{\rho}}}{4}(\frac{1}{5\Delta^*_{\tilde{\rho}}})^4 \leq (\frac{e}{5})^4 < 1/10$. Hence, the expected fraction of the unaccepted proposals of X is at most 1/10, which using Markov's inequality means that the probability that more than 2/3 are unaccepted is at most 3/20. Therefore, the probability that X is marked active and at least 1/3 of its proposals are accepted is at least $0.03/\Delta^*$.

Call a component *large-degree* if it can be satisfied by at least $\Delta^*_{\tilde{\rho}}/2$ many almost-max-efficient star stars. We get that for each large-degree unsatisfied component $\mathcal{C}$, the expected number of stars that satisfy $\mathcal{C}$ and get colored gray is at least 1/100. On the other hand, the probability that there are z stars that satisfy $\mathcal{C}$ and are colored gray (which shows that they are marked active) decays exponentially with z, as it is at most $\binom{\Delta^*_{\tilde{\rho}}}{z}(\frac{1}{5\Delta^*_{\tilde{\rho}}})^z \leq (\frac{e}{5})^z$. It follows that with at least a constant probability, one or more of stars that satisfy $\mathcal{C}$ gets colored gray. This is because otherwise, only an ε of the total probability mass is on $z \geq 1$,

for a sub-constant ε, which given the exponentially decaying tail, it would contradict with the expectation being at least constant $1/100$. Hence, we get that $\mathcal{C}$ gets satisfied with at least a constant probability.

It follows from Markov's inequality that with at least a constant probability, at least a constant fraction of large-degree components get satisfied, finishing the proof. □

8.5 Round Complexity Lower Bound

Here, we mention the simple observation that the techniques of Das Sarma et al. [DSHK+11] imply a $\tilde{\Omega}(D + \sqrt{n})$ rounds lower bound for any approximation of MCDS. For simplicity, we only explain an $\Omega(\sqrt{n}/\log n)$-round lower bound for the case where $D = O(\log n)$ and in fact we will just sketch the changes. We encourage the interested reader to see [DSHK+11] for the details and generalization to other diameter values.

Observation 8.5.1. *For any polynomial $\alpha(n)$, there is a constant $\varepsilon > 0$ such that any $\alpha(n)$-approximation algorithm for the minimum-weight connected dominating set that has error-probability at most ε requires at least $\Omega(\sqrt{n}/\log n)$ rounds of the* CONGEST *model, on a graph that has diameter $D = O(\log n)$.*

The general approach (following [DSHK+11]) is that we present a graph and show that one can encode instances of two-party set disjointness on $\sqrt{n}$-bits in the node-weights of this graph such that the following holds: if there is an $\alpha(n)$-approximation algorithm $\mathcal{A}$ for the minimum-weight connected dominating set problem that has error-probability at most ε and uses $T \leq \Omega(\sqrt{n}/\log n)$ rounds, then there is a randomized algorithm for two party set on instances with $\sqrt{n}$-bits inputs with error-probability at most ε that uses less than $\Theta(T \log n)$ communication rounds, which would be a contradiction. Without loss of generality, enhance $\mathcal{A}$ so that each node knows the total weight of final CDS, note that this can be done in additional $O(D)$ rounds and is thus without loss of generality.

Consider a graph made of three parts: $\sqrt{n}$ aligned parallel paths of length $\sqrt{n}$ each, a tree of depth $\log n - 1$ on top of these trees such that each of its leaves is aligned with one column of the nodes of the paths, and finally, for each leaf of the tree, edges from this leaf to all the nodes in the paths that are in the same column. Figure 8-6 shows an example.

Call the left-most leaf Alice and the rightmost leaf Bob and that they are given an instance of set-disjointness, where Alice and Bob respectively get inputs $\mathcal{X}$ and $\mathcal{Y}$ that are subsets of $\{1, 2, \ldots, \sqrt{n}\}$. We next describe how to encode these inputs in the weights of the MCDS problem. Give a weight of 1 to each non-leaf node of the tree, the nodes held by Alice and Bob, and also all nodes on the paths except the leftmost and the rightmost ones on each path. These weight-1 nodes are indicated with a light green color in Figure 8-6. Then, for each other leaf node, give a weight $M = \alpha(n) \cdot n + 1$ (light red color). Finally, for each $i \in \{1, \ldots, \sqrt{n}\}$, for the leftmost node of the i^{th} path, give a weight of 1 if $i \notin \mathcal{X}$ (dark green color) and give a weight M if $i \in \mathcal{X}$ (dark red). Similarly, for the rightmost node of

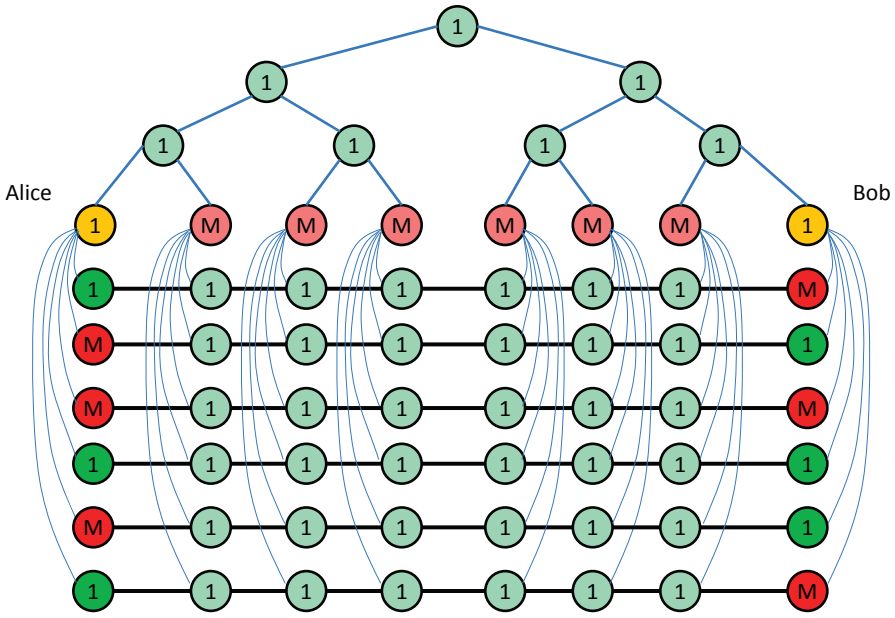


Figure 8-6: The round complexity lower bound graph

the i^{th} path, give a weight of 1 if $i \notin \mathcal{Y}$ and give a weight M if $i \in \mathcal{Y}$. Note that Alice and Bob can indeed put these weights as inputs to the MCDS problem in just one round, as all the nodes know the fixed part of the weight, and the variable part which depends on the set disjointness inputs is on the neighbors of Alice and Bob, and thus, they can lean their weights in just one round.

Now notice that if $i \in \mathcal{X} \cap \mathcal{Y}$, then any CDS must contain at least one node of weight M. On the other hand, if $\mathcal{X}$ and $\mathcal{Y}$ are disjoint, then there is an CDS with weight (less than) n, which includes all weight-1 nodes. Since $\frac{M}{n} > \alpha$, and as $\mathcal{A}$ finds an $\alpha(n)$-approximation of MCDS, the weight output by $\mathcal{A}$ lets the nodes distinguish the case where the sets are disjoint from the case where they are not (with error-probability being the same as in $\mathcal{A}$). The final piece, which is the key technical part, is to show that Alice and Bob can indeed simulate $\mathcal{A}$ being run over the whole graph, using only $\Theta(T \log n)$ communication rounds between themselves. This follows exactly from [DSHK+11, Simulation Theorem].

8.6 Open Problems and Future Work

We present a distributed $O(\log n)$ approximation algorithm for the MCDS problem in $\tilde{O}(D + \sqrt{n})$ rounds of the CONGEST model.

As mentioned before, MCDS is NP-hard and if one assumes that nodes can only perform polynomial-time computations (which is a practically reasonable assumption and also a usual one [DMP+03, JRS01, KW03]), the $O(\log n)$ approximation factor is optimal up to a constant factor, unless P = NP. The author finds it quite an intriguing question to see if one can get an $o(\log n)$ approximation in a non-trivial number of rounds, by relaxing this assumption

about local computations. However, this question might be only of theoretical interest.

In the current presentation of the algorithm, we have not tried to optimize the constant in the approximation factor. However, it is not clear how to get a $(1+o(1))\log n$ approximation and that is another interesting question.

The author started looking into the MCDS problem with the hope of solving it—i.e., finding an $O(\log n)$ approximation for it—in $\tilde{O}(D+\sqrt{n})$ rounds of CONGEST, which is a more restricted version of the CONGEST model where in each round, each node can send one $O(\log n)$-bits message to all of its neighbors. Notice that the same message has to be sent to all neighbors. This model is called VCONGEST as the congestion is on vertices, rather than on edges. Note that this restriction is natural in node-capacitated networks, and MCDS is also more important in such settings. It would be interesting to see if an $\tilde{O}(D+\sqrt{n})$-rounds $O(\log n)$-approximation for MCDS can be found in VCONGEST.

Chapter 9

Distributed Connectivity Decomposition

9.1 Introduction & Related Work

In this chapter, we present our distributed constructions for the connectivity decompositions that were introduced and proven to exist in Chapter 4. In particular, we present distributed algorithms that construct edge and vertex connectivity decompositions, which are comparable to their existential/centralized counterparts, while having near-optimal round complexity.

We note that, although our edge-connectivity decomposition builds on a number of standard techniques and known results, our vertex connectivity decomposition algorithm is the main technical novelty of this chapter. In particular, it achieves near-optimal time complexities in both distributed and centralized settings (Theorem 9.1.1 and Theorem 9.1.2), and it leads to a number of interesting algorithmic implications in both of these settings.

Throughout, we assume a connected undirected network $G = (V, E)$ with $n = |V|$ nodes, $m = |E|$ edges, and diameter D. Moreover, we use k and λ to denote the vertex connectivity and edge connectivity, respectively.

Theorem 9.1.1. *There is an $\tilde{O}(\min\{D+\sqrt{n}, \frac{n}{k}\})$-rounds randomized distributed algorithm in the* V-CONGEST *model that w.h.p. finds a fractional dominating tree packing of size $\Omega(\frac{k}{\log n})$, where k is the vertex connectivity of the graph. More specifically, this algorithm finds $\Omega(k)$ dominating trees, each of diameter $\tilde{O}(\frac{n}{k})$, such that each node is included in $O(\log n)$ trees.*

Note that diameter $\Omega(n/k)$ for all except very few of the dominating trees is unavoidable, even in graphs of diameter 2. Just consider a graph consisting of sequence of n/k cliques of size k, where subsequent cliques are connected by matchings of size k. Adding a single node u and connecting u to all other nodes creates a graph with diameter 2 and vertex connectivity $k + 1$, where any dominating tree that does not include u needs to have diameter at least n/k.

Theorem 9.1.2. *There is an $\tilde{O}(m)$ time randomized centralized algorithm that w.h.p. finds a fractional dominating tree packing of size $\Omega(\frac{k}{\log n})$, where k is the vertex connectivity of the*

graph. More specifically, this algorithm finds $\Omega(k)$ dominating trees, each of diameter $\tilde{O}(\frac{n}{k})$, such that each node is included in $O(\log n)$ trees.

Theorem 9.1.2 improves over the $\Omega(n^3)$ algorithms that can be obtained from the approach we explained in Chapter 4 (as presented in [CHGK14a]) and also on the polynomial time method of [EKV13]. Regarding Theorem 9.1.1, we note that the algorithm of [EKV13] does not appear to admit an efficient distributed implementation as it is based on a number of centralized tools and techniques such as the ellipsoid method for linear programming, the meta-rounding of [CV00], and the Min-Cost-CDS approximation result of [GK98]. The algorithm of [CHGK14a], which we presented in Chapter 4 as a proof of existence, has a similar problem which seems essential and unavoidable. See the last part of Section 9.3 for an explanation of why the algorithm of [CHGK14a] does not extend to an efficient algorithm in the distributed setting and for how it compares with the approach in this chapter.

Theorem 9.1.3. *There is an $\tilde{O}(D + \sqrt{\lambda n})$-rounds randomized distributed algorithm in the* E-CONGEST *model that w.h.p. finds a fractional spanning tree packing of size $\lceil \frac{\lambda-1}{2} \rceil (1-\varepsilon)$, where λ is the edge connectivity of the graph. Furthermore, each edge is included in at most $O(\log^3 n)$ trees.*

Remark on Integral Tree Packings Both algorithms of Theorem 9.1.1 and Theorem 9.1.2 can be adapted to produce a dominating tree packing of size $\Omega(\frac{k}{\log^2 n})$, in similar time-complexities, using the random layering technique of Section 4.4.

Moreover, we note that a considerably simpler variant of the algorithm of Theorem 9.1.3 can be adapted to produce an integral spanning tree packing of size $\Omega(\frac{\lambda}{\log n})$, with a round complexity of $\tilde{O}(D + \sqrt{\lambda n})$.

For information dissemination purposes, which comprise the primary application of our decompositions, fractional packings usually are as useful as integral packings. We can typically timeshare each node or edge among the trees that use it (with a time share proportional to the weights of the tree). Since our fractional packing results have better sizes, our main focus will be on describing the fractional versions.

Lower Bounds While $\Omega(m)$ is a trivial lower bound on the time complexity of the centralized decompositions, by extending results of [DHK+12], we show lower bounds of $\tilde{\Omega}(D+\sqrt{\frac{n}{k}})$ and $\tilde{\Omega}(D+\sqrt{\frac{n}{\lambda}})$ on the round complexities of the distributed decompositions of Theorem 9.1.1 and Theorem 9.1.3. See Section 9.10 for the formal statements and the proofs.

9.1.1 Applications

Information Dissemination

As their primary application, our decompositions provide time-efficient distributed constructions for broadcast algorithms (based on routing, i.e., without using network coding) with

existentially optimal throughput. See Section 9.8 for a simple example. Note that in the V-CONGEST model, vertex cuts characterize the main limits on the information flow, and k messages per round is the clear information-theoretic limit on the broadcast throughput (even with network coding) in each graph with vertex connectivity k. Similarly, in the E-CONGEST model, edge cuts characterize the main limits on the information flow, and λ messages per round is the information-theoretic limit on the broadcast throughput (even with network coding) in each graph with edge connectivity λ. Our optimal-throughput broadcast algorithms are as follows:

Corollary 9.1.4. *In the* V-CONGEST *model, using the $\tilde{O}(D+\sqrt{n})$-rounds construction of Theorem 9.1.1, and then broadcasting each message along a random tree, we get a broadcast algorithm with throughput of $\Omega(\frac{k}{\log n})$ messages per round. [CHGK14a] shows this throughput to be existentially optimal.*

See Section 9.8 for an explanation and a simple example of how one can use dominating tree packings to broadcast messages by routing them along different trees.

Corollary 9.1.5. *In the* E-CONGEST *model, using the $\tilde{O}(D+\sqrt{\lambda n})$-rounds construction of Theorem 9.1.3, and then broadcasting each message along a random tree, we get a broadcast algorithm with throughput of $\lceil\frac{\lambda-1}{2}\rceil(1-\varepsilon)$ messages per round.*

We emphasize that the above broadcast algorithms provide *oblivious broadcast routing* (see [Rac02]). In an *oblivious routing* algorithm, the path taken by each message is determined (deterministically or probabilistically) independent of the current load on the graph; that is, particularly independent of how many other messages exist in the graph and how they are routed. Note that this is in stark contrast to adaptive algorithms which can tailor the route of each message, while knowing the current (or future) load on the graph, in order to minimize congestion. Oblivious routing algorithms appear to be a much more restricted family and a priori, it is somewhat unclear whether they can have a performance close to that of the optimal adaptive algorithms. Interestingly, a beautiful line of work [Rac02, ACF+03, HHR03, BKR03, Rac08] shows this to be possible: These papers present centralized oblivious routing algorithms with successively improved edge-congestion competitiveness ratios. This line ends in the algorithm of Räcke [Rac08] which achieves the optimal $O(\log n)$-competitive edge-congestion. That is, in this algorithm, the expected maximum congestion over all edges is at most $O(\log n)$ times the maximum congestion of the offline optimal algorithm. The problem of designing distributed oblivious routing algorithms achieving this performance remains open. Furthermore, it is known that no *point-to-point* oblivious routing can have vertex-congestion competitiveness better than $\Theta(\sqrt{n})$ [HKRL07].

Our results address *oblivious routing for broadcast*:

Corollary 9.1.6. *By routing each message along a random one of the trees generated by Theorem 9.1.1 and Theorem 9.1.3, we get distributed oblivious routing broadcast algorithms that respectively have vertex-congestion competitiveness of $O(\log n)$ and edge-congestion competitiveness of $O(1)$.*

Vertex Connectivity Approximation

Vertex connectivity is a central concept in graph theory and extensive attention has been paid to developing algorithms that compute or approximate it. In 1974, Aho, Hopcraft and Ulman [AHU74, Problem 5.30] conjectured that there should be an $O(m)$ time algorithm for computing the vertex connectivity. Despite many interesting works in this direction—e.g., [Tar74, Eve75, Gal80, Hen97, HRG96, Gab00]—finding $O(m)$ time algorithms for vertex connectivity has yet to succeed. The current state of the art is an $O(\min\{n^2k + nk^{3.5}, n^2k + n^{1.75}k^2\})$ time exact algorithm by Gabow [Gab00] and an $O(\min\{n^{2.5}, n^2k\})$ time 2-approximation by Henzinger [Hen97]. The situation is considerably worse in distributed settings and the problem of upper bounds has remained widely open, while we show in Section 9.10 that an $\Omega(D + \sqrt{\frac{n}{k}})$ round complexity lower bound follows from techniques of [DHK+12].

Since Theorem 9.1.1 and Theorem 9.1.2 work without a priori knowledge of vertex connectivity and as the size of the achieved dominating trees packing is in the range $[\Omega(\frac{k}{\log n}), k]$, our dominating tree packing algorithm provides the following implication:

Corollary 9.1.7. *We can compute an $O(\log n)$ approximation of vertex connectivity, centralized in $\tilde{O}(m)$ time, and distributed in $\tilde{O}(D + \sqrt{n})$ rounds of the* V-CONGEST *model.*

Note that it is widely known that in undirected graphs, vertex connectivity and vertex cuts are significantly more complex than edge connectivity and edge cuts, for which currently the following result are known: an $\tilde{O}(m)$ time centralized exact algorithm [Kar96], an $O(m)$ time centralized $(1+\varepsilon)$-approximation [Kar94b], and recent $\tilde{O}(D + \sqrt{n})$ rounds distributed algorithms for $(2+\varepsilon)$-approximation [GK13] and $(1+\varepsilon)$-approximation [NS14].

9.1.2 Other Related Work

Vertex-Independent Trees

Dominating tree packings have some resemblance to *vertex independent trees* [ZI89, KS92b] and are in fact a strictly stronger concept. In a graph $G = (V, E)$, k' trees are called *vertex independent* if they are spanning trees all rooted in a node $r \in V$ and for each vertex $v \in V$, the paths between r and v in different trees are internally vertex-disjoint. We emphasize that these trees are not vertex disjoint.

Zehavi and Itai conjectured in 1989 [ZI89] that each graph with vertex connectivity k contains k vertex independent trees. Finding such trees, if they exist, is also of interest. The conjecture remains open and is confirmed only for cases $k \in \{2, 3\}$. Itai and Rodeh [IR88] present an $O(m)$ time centralized algorithm for finding 2 vertex independent trees, when the graph is 2-vertex-connected and Cheriyan and Maheshvari [CM88] present an $O(n^2)$ time centralized algorithm for finding 3 vertex independent trees, when the graph is 3-vertex-connected.

Vertex disjoint dominating trees are a strictly stronger notion[1]: Given k' vertex-disjoint dominating trees, we get k' vertex independent trees, for *any* root $r \in V$. This is by adding all the other nodes to each dominating tree as leaves to make it spanning. Then, for each vertex $v \in V$, the path from r to v in each (now spanning) tree uses only internal vertices from the related dominating tree.

In this regard, one can view [CHGK14a, Theorem 1.2] as providing a poly-logarithmic approximation of the Zehavi and Itai's conjecture. Furthermore, the vertex connectivity algorithm presented here (formally, its extension to integral dominating tree packing, mentioned in Section 4.1.1) makes this approximation algorithmic with near-optimal complexities $\tilde{O}(m)$ centralized and $\tilde{O}(D + \sqrt{n})$ distributed.

A Review of Centralized Connectivity Decompositions

Edge Connectivity Edge connectivity decompositions into spanning tree packings of $\lceil \frac{\lambda-1}{2} \rceil$ have been known due to results of Tutte [Tut61] and Nash-Williams [NW61] from 1960, and they have found many important applications: the best known centralized minimum edge cut algorithm [Kar96], the network coding advantage in edge-capactitated networks [LLL09], and tight analysis of the number of minimum cuts of a graph and random edge-sampling [Kar96]. Centralized algorithms for finding such a spanning tree packing include: an $\tilde{O}(\min\{mn, \frac{m^2}{\sqrt{n}}\})$ time algorithm for unweighted graphs by Gabow and Westermann [GW88], an $\tilde{O}(mn)$ time algorithm for weighted graphs by Barahona [Bar95], and an $\tilde{O}(m\lambda)$ time algorithm for a fractional packing via the general technique of Plotkin et al. [PST91] (see [Kar96]).

Vertex Connectivity As mentioned before, vertex connectivity decompositions were introduced only recently by Censor-Hillel, Ghaffari, and Kuhn [CHGK14a]. As discussed in Chapter 4, the problem also has connections to analyzing vertex connectivity under vertex sampling and also network coding gap in node-capacitated networks. Consequent to (a preliminary version of) [CHGK14a], Ene et al. [EKV13] presented a nice alternative proof for obtaining fractional dominating tree packing of size $\Omega(\frac{k}{\log n})$, which uses metarounding results of Carr and Vempala [CV00] and the Min-Cost-CDS approximation result of Guha and Khuller [GK98]. That proof does not extend to integral packing. Even though the proofs presented in [CHGK14a] and [EKV13] are based on polynomial time algorithms, neither of the algorithms seems to admit a distributed implementation and even their centralized complexities are at least $\Omega(n^3)$.

We also note that finding many *dominating sets* (particularly algorithmic inapproximability of it) was addressed in the beautiful work of Feige et al. [FHK00]. Although, as we

[1]The relation is strict as e.g., the following graph with vertex connectivity 3 does not admit more than 1 vertex-disjoint dominating trees while [CM88] implies that this graph has 3 vertex independent trees: A graph with a clique of size $n^{1/3}$, plus one additional node for each subset of three nodes of the clique, connected exactly to those three clique nodes.

will see in Section 9.3, from an algorithmic viewpoint, finding many dominating sets is significantly simpler than finding many connected dominating sets, and in fact, getting just the domination part in our results is rather a triviality (see Lemma 4.3.2).

9.2 Preliminaries

We here recall some of the basic notations and definitions and then explain the related distributed representation aspects.

Given an undirected graph $G = (V, E)$ and a set $S \subseteq V$, we use the notation $G[S]$ to indicate the subgraph of G induced by S. A set $S \subseteq V$ is called a *connected dominating set (CDS)* iff $G[S]$ is connected and for each node $u \in V \setminus S$, u has a neighbor $v \in S$. A subgraph $T = (V_T, E_T)$ of graph $G = (V_G, E_G)$ is a *dominating tree* of graph G if T is a tree and V_T is a dominating subset of V_G.

Dominating Tree Packing Let $DT(G)$ be the set of all dominating trees of G. A κ-size dominating tree packing of G is a collection of κ *vertex-disjoint dominating* trees in G. A κ-size fractional dominating tree packing of G assigns a weight $x_\tau \in [0, 1]$ to each $\tau \in DT(G)$ s.t. $\sum_{\tau \in DT(G)} x_\tau = \kappa$ and $\forall v \in V$, $\sum_{\tau, v \in \tau} x_\tau \leq 1$.

Spanning Tree Packing Let $ST(G)$ be the set of all spanning trees of G. A κ-size spanning tree packing of G is a collection of κ *edge-disjoint spanning* trees in G. A κ-size fractional spanning tree packing of G assigns a weight $x_\tau \in [0, 1]$ to each $\tau \in ST(G)$ s.t. $\sum_{\tau \in ST(G)} x_\tau = \kappa$ and $\forall e \in E$, $\sum_{\tau, e \in \tau} x_\tau \leq 1$.

Distributed Problem Requirements In the distributed versions of dominating or spanning tree packing problems, we consider each tree τ as one class with a unique identifier ID_τ and a weight x_τ. In the spanning tree packing problem, for each node v and each edge e incident to v, for each spanning tree τ that contains e, node v should know ID_τ and x_τ. In the dominating tree packing problem, for each node v and each dominating tree τ that contains v, node v should know ID_τ, x_τ, and the edges of τ incident to v.

Some Details of the Distributed Model See Chapter 2 for model definitions. Since our focus is on randomized algorithms, nodes can generate random ids by each taking random binary strings of $4 \log n$ bits and delivering it to their neighbors. We assume no initial knowledge about the graph. By using a simple and standard BFS tree approach, nodes can learn the number of nodes in the network n, and also a 2-approximation of the diameter of the graph D, in $O(D)$ rounds. Our algorithms assume this knowledge to be ready for them. Note that these $O(D)$ rounds do not affect the asymptotic bounds of our round complexities.

9.3 Dominating Tree Packing Algorithm

In this section, we present the main technical contribution of the paper, which is introducing a new algorithm for vertex connectivity decomposition that has near optimal time complexities for both centralized and distributed implementations. In this decomposition, we construct a collection of $\Omega(k)$ classes, each of which is a dominating tree w.h.p., such that each vertex is included in at most $O(\log n)$ classes. This gives a fractional dominating tree packing[2] of size $\Omega(\frac{k}{\log n})$ and lets us achieve Theorems 9.1.1 and 9.1.2. The analysis is presented in Section 9.4.

The Algorithm's Outline For the construction, we first assume that we have a 2-approximation of k, and then explain how to remove this assumption. We construct $t = \Theta(k)$ connected dominating sets (CDS) such that each node is included in $O(\log n)$ CDSs. We work with CDSs, since it is sufficient to determine their vertices. To get dominating trees, at the end of the CDS packing algorithm, we remove the cycles in each CDS using a simple application of a minimum spanning tree algorithm.

We transform the graph $G = (V, E)$ into a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which is called the *virtual graph* [3], and is constructed as follows: Each node $v \in V$ simulates $\Theta(\log n)$ *virtual* nodes $\nu_1, \nu_2, \ldots, \nu_{\Theta(\log n)} \in \mathcal{V}$ and two virtual nodes are connected if they are simulated by the same real node or by two G-adjacent real nodes. To get the promised CDS Packing, we partition the virtual nodes $\mathcal{V}$ into t disjoint *classes*, each of which is a CDS of $\mathcal{G}$, w.h.p. Each CDS $\mathcal{S}$ on $\mathcal{G}$ defines a CDS S on G in a natural way: S includes all real nodes v for which at least one virtual node of v is in $\mathcal{S}$. Thus, the t classes of virtual nodes w.h.p. give t CDSs on G and clearly each real node is included in $O(\log n)$ CDSs.

For the construction, we organize the virtual nodes by giving them two attributes: each virtual node has a *layer number* in $\{1, 2, \ldots, L\}$, where $L = \Theta(\log n)$, and a *type number* in $\{1, 2, 3\}$. For each real node $v \in V$, the $3L = \Theta(\log n)$ virtual nodes simulated by v are divided such that, for each layer number in $\{1, 2, \ldots, L\}$ and each *type number* in $\{1, 2, 3\}$, there is exactly one virtual node. For the communication purposes in the distributed setting, note that each communication round on $\mathcal{G}$ can be simulated via $\Theta(\log n)$ communication rounds on G. Thus, to simplify discussions, we divide the rounds into groups of $\Theta(\log n)$ consecutive rounds and call each group one *meta-round*.

As explained, we assign each virtual node to a class. This class assignment is performed in a recursive manner based on the layer numbers. First, with a jump-start, we assign

[2] The approach of this algorithm can be also used to get an $\Omega(\frac{\kappa}{\log^2 n})$ dominating tree packing, where κ is the remaining vertex-connectivity when each node is sampled with probability 1/2. This is a $\Omega(\frac{k}{\log^2 n})$-szie dominating tree packing, as [CGG+15] proves that $\kappa = \Omega(k)$. See [CHGK14a, Section 4] for the simple extension to dominating tree packing.

[3] The virtual graph $\mathcal{G}$ is nothing but using $\Theta(\log n)$ copies of G, or simply reusing each node of G for $\Theta(\log n)$ times, $\Theta(1)$ times per *layer* (described later). We find it more formal to use $\mathcal{G}$ instead of directly talking about G.

each virtual node of layers 1 to $L/2$ to a random class in classes 1 to t. This step gives us that each class dominates $\mathcal{G}$, w.h.p. The interesting and challenging part is to achieve connectivity for all classes. For this purpose, we go over the layers one by one and for each layer $\ell \in [L/2, L-1]$, we assign class numbers to the virtual nodes of layer $\ell+1$ based on the assignments to the virtual nodes of layers 1 to ℓ. The goal is to connect the components of each class such that the total number of connected components (summed up over all classes) decreases (in expectation) by a constant factor, with the addition of each layer. This would give us that after $\Theta(\log n)$ layers, all classes are connected, w.h.p. We next explain the outline of this step, after presenting some notations.

Let $\mathcal{V}_\ell^i$ be the set of virtual nodes of layers 1 to ℓ assigned to class i (note that $\mathcal{V}_\ell^i \subseteq \mathcal{V}_{\ell+1}^i$). Let N_ℓ^i be the number of connected components of $\mathcal{G}[\mathcal{V}_\ell^i]$ and let $M_\ell := \sum_{i=1}^{t}(N_\ell^i - 1)$ be the total number of excess components after considering layers $1, \ldots, \ell$, compared to the ideal case where each class is connected. Initially $M_1 \leq n - t$, and as soon as $M_\ell = 0$, each class induces a connected subgraph.

Recursive Class Assignment Suppose that we are at the step of assigning classes to virtual nodes of layer $\ell + 1$. We call virtual nodes of layer $\ell + 1$ *new nodes* and the virtual nodes of layers 1 to ℓ are called *old nodes*. Also, in the sequel, our focus is on the virtual nodes and thus, unless we specifically use the phrase "real node", we are talking about a virtual node. First, each new node of type 1 or type 3 joins a random class. It then remains to assign classes to type-2 new nodes, which is the key part of the algorithm. The outline of this procedure is as follows:

Recursive Class Assignment Outline:

(1) ***Identify the connected components of old nodes***, i.e., those of $\mathcal{G}[\mathcal{V}_\ell^i]$ for each class i.

(2) ***Create the bridging graph***, a bipartite graph between the connected components of old nodes and type-2 new nodes, which is defined as follows: We view each connected component of old nodes as one node on one side of the bipartite graph, by assuming all its vertices are contracted into one node. Each type-2 new node v is adjacent to a connected component $\mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$ if all the following conditions hold: (a) v has a neighbor in $\mathcal{C}$, (b) $\mathcal{C}$ does not have a type-1 new node neighbor u such that u joined class i and that u has a neighbor in a component $\mathcal{C}' \neq \mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$, and (c) v has a type-3 new node neighbor w such that w has joined class i and w has a neighbor in a connected component $\mathcal{C}'' \neq \mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$.

(3) ***Find a maximal matching $\mathcal{M}$ in the bridging graph***, i.e., between components and type-2 new nodes. For each type-2 new node v, if it is matched in $\mathcal{M}$, then it joins the class of its matched component and otherwise, it joins a random class.

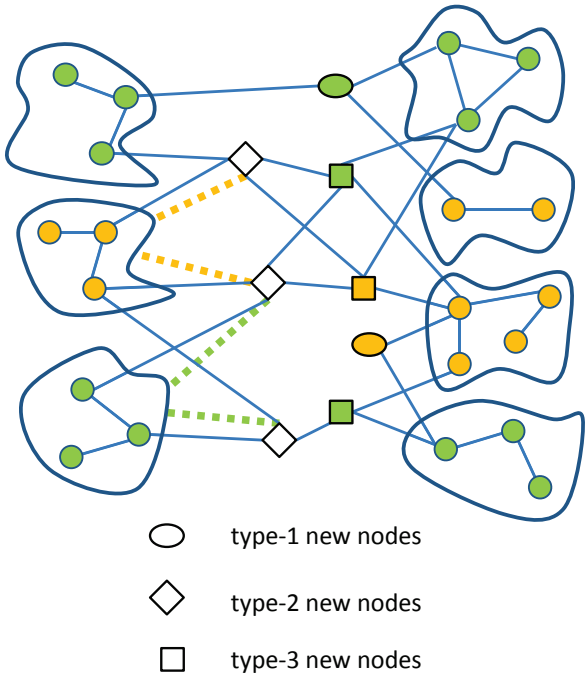


Figure 9-1: Bridging graph edges are presented by dotted lines; each color is one class.

Intuitively, the rule described in step (2) means the following: v is a neighbor of $\mathcal{C}$ in the bridging graph if component $\mathcal{C}$ is not (already) connected to another component of $\mathcal{G}[\mathcal{V}_\ell^i]$ via one type-1 new node, but if v joins class i, then with the help of v and w, the component $\mathcal{C}$ will be merged with some other component $\mathcal{C}'' \neq \mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$. See Figure 9-1.

This recursive class assignment outline can be implemented in a distributed manner in $\tilde{O}(\min\{D+\sqrt{n}, \frac{n}{k}\})$ rounds of the V-CONGEST model, and in a centralized manner in $\tilde{O}(m)$ steps, thus proving respectively Theorems 9.1.1 and 9.1.2. We present the details of the distributed implementation in Section 9.5. The centralized implementation is presented in Section 9.6.

We show in the analysis that the above algorithm indeed constructs k CDSs, w.h.p., and clearly each real node is contained in at most $O(\log n)$ CDSs, at most one for each of its virtual nodes. To turn these CDSs into dominating trees, we simply remove some of the edges of each class so as to make it a tree. We do this by an application of a MST algorithm on the virtual graph $\mathcal{G}$: We give weight of 0 to the edges between virtual nodes of the same class and weight 1 to other edges. Then, the edges with weight 0 that are included in the MST of $\mathcal{G}$ form our dominating trees, exactly one for each CDS.

Remark 9.3.1. *The assumption of knowing a 2-approximation of k can be removed at the cost of at most an $O(\log n)$ increase in the time complexities.*

To remove the assumption, we use a classical try and error approach: we simply try exponentially decreasing guesses about k, in the form $\frac{n}{2^j}$, and we test the outcome of the dominating tree packing obtained for each guess (particularly its domination and connectivity) using a

randomized testing algorithm , presented in Section 9.7, on the virtual graph $\mathcal{G}$. For this case, this test runs in a distributed setting in $O(\min\{\frac{n \log^2 n}{k}, (D + \sqrt{n \log n} \log^* n) \log^2 n\})$ round of the V-CONGEST model, and in a centralized setting with step complexity of $O(m \log^3 n)$.

An intuitive comparison with the approach of Chapter 4 We note that the approach of the above algorithm is significantly different than that of our simpler existence proof in Chapter 4, which was first presented in [CHGK14a]. Mainly, the key part in our existential proof (Chapter 4) is that it finds short paths between connected components of the same class, called *connector paths*. The high-level idea is that, by adding the nodes on the connector paths of a class to this class, we can merge the connected components at the endpoints of this path. However, unavoidably, each node of each path might be on connector paths of many classes. Thus, the class assignment of this node is not clear. Our existential proof carefully allocates the nodes on the connector paths to different classes so as to make sure that the number of connected components goes down by a constant factor in each layer (in expectation).

Finding the connector paths does not seem to admit an efficient distributed algorithm and it is also slow in a centralized setting. The algorithm presented in this chapter does not find connector paths or use them explicitly. However, it is designed such that it enjoys the existence of connector paths and its performance gains implicitly from the abundance of the connector paths. While this is the key part that allows us to make the algorithm distributed and also makes it simpler and faster centralized, the analysis becomes more involved. The main challenging part in the analysis is to show that the the size of the maximal matching found in the bridging graph is large enough so that in each layer, the number of connected components (summed up over all classes) goes down by a constant factor, with at least a constant probability. This is addressed in Section 9.4.2.

9.4 Dominating Tree Packing Analysis

In this section, we present the analysis for the algorithm explained in Section 9.3. We note that this analysis is regardless of whether we implement the algorithm in a distributed or a centralized setting. In a first simple step, we show that each class is a dominating set. Then, proving the connectivity of all classes, which is the core technical part, is divided into two subsections: We first present the concept of connector paths in Section 9.4.1 and then use this concept in Section 9.4.2 to achieve the key point of the connectivity analysis, i.e., the Fast Merger Lemma (Lemma 9.4.4).

Lemma 9.4.1 (Domination Lemma). *W.h.p., for each class i, $\mathcal{V}^i_{L/2}$ is a dominating set.*

Proof of Lemma 9.4.1. Each virtual node $v \in \mathcal{V}$ has in expectation $\frac{k \log n}{2t} = \Omega(\log n)$ virtual neighbors in $\mathcal{V}^i_{L/2}$. Choosing constants properly, the claim follows from a standard Chernoff argument combined with a union bound over all choices of v and over all classes. □

Since $\mathcal{V}_\ell^i \subseteq \mathcal{V}_{\ell'}^i$ for $\ell \leq \ell'$, the domination of each class follows directly from this lemma. For the rest of this section, we assume that for each class i, $\mathcal{V}_{L/2}^i$ is a dominating set.

9.4.1 Connector Paths

The concept of connector paths is a simple toolbox that we developed in [CHGK14a]. For completeness, we present a considerably simpler version here:

Consider a class i, suppose $N_\ell^i \geq 2$, and consider a component $\mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$. For each set of virtual vertices $\mathcal{W} \subseteq \mathcal{V}$, define the projection $\Psi(\mathcal{W})$ of $\mathcal{W}$ onto G as the set $W \subseteq V$ of real vertices w, for which at least one virtual node of w is in $\mathcal{W}$.

A path P in the real graph G is called a *potential connector* for $\mathcal{C}$ if it satisfies the following three conditions: (A) P has one endpoint in $\Psi(\mathcal{C})$ and the other in $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$, (B) P has at most two internal vertices, (C) if P has exactly two internal vertices and has the form s, u, w, t where $s \in \Psi(\mathcal{C})$ and $t \in \Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$, then w does not have a neighbor in $\Psi(\mathcal{C})$ and u does not have a neighbor in $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$.

Intuitively, condition (C) requires *minimality* of each potential connector path. That is, there is no potential connector path connecting $\Psi(\mathcal{C})$ to another component of $\Psi(\mathcal{V}_\ell^i)$ via only u or only w.

From a potential connector path P on graph G, we derive a connector path $\mathcal{P}$ on virtual graph $\mathcal{G}$ by determining the types of the related internal virtual vertices as follows: (D) If P has one internal real vertex w, then for $\mathcal{P}$ we choose the virtual vertex of w in layer $\ell+1$ in $\mathcal{G}$ with type 1. (E) If P has two internal real vertices w_1 and w_2, where w_1 is adjacent to $\Psi(\mathcal{C})$ and w_2 is adjacent to $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$, then for $\mathcal{P}$ we choose the virtual vertices of w_1 and w_2 in layer $\ell+1$ with types 2 and 3, respectively. Finally, for each endpoint w of P we add the copy of w in $\mathcal{V}_\ell^i$ to $\mathcal{P}$. We call a connector path that has one internal vertex a *short connector path*, whereas a connector path with two internal vertices is called a *long connector path*.

Figure 9-2, demonstrates an example of potential connector paths for a component $\mathcal{C}_1 \in \mathcal{G}[\mathcal{V}_\ell^i]$ (see Section 9.4.1). The figure on the left shows the graph G, where the projection $\Psi(\mathcal{V}_\ell^i)$ is indicated via green vertices, and the green paths are potential connector paths of $\Psi(\mathcal{C}_1)$. On the right side, the same potential connector paths are shown, where the type of the related internal vertices are determined according to rules (D) and (E) above, and vertices of different types are distinguished via different shapes (for clarity, virtual vertices of other types are omitted).

Because of condition (C), and rules (D) and (E) above, we get the following fact:

Proposition 9.4.2. *For each class i, each type-2 virtual vertex u of layer $\ell+1$ is on connector paths of at most one connected component of $\mathcal{G}[\mathcal{V}_\ell^i]$.*

We next show that each component that is not single in its class has k connector paths.

Lemma 9.4.3 (Connector Abundance Lemma). *Consider a layer $\ell \geq L/2+1$ and a class i such that $\mathcal{V}_{L/2}^i \subseteq \mathcal{V}_\ell^i$ is a dominating set of $\mathcal{G}$ and $N_\ell^i \geq 2$. Further consider an*

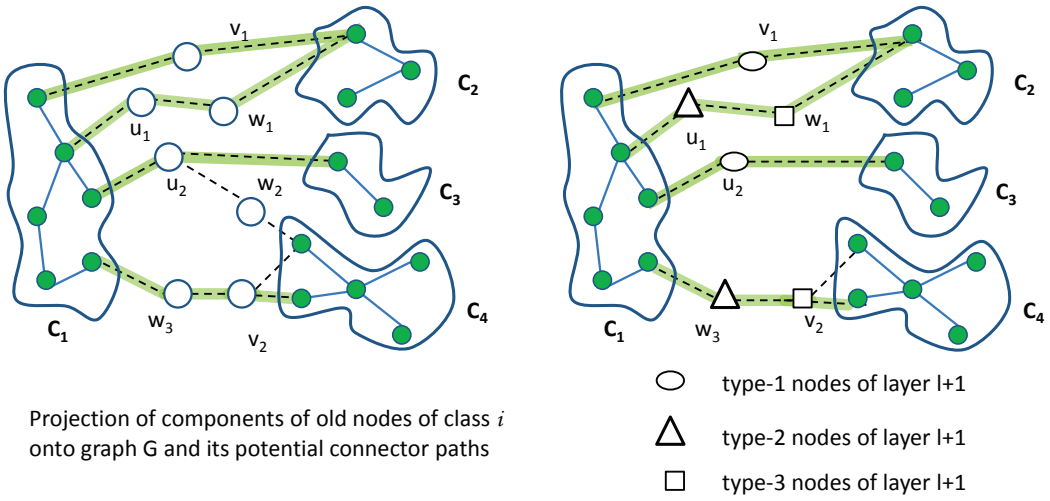


Figure 9-2: Connector Paths for component $\mathcal{C}_1$ in layer $\ell+1$ copies of G

arbitrary connected component $\mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$. Then, $\mathcal{C}$ has at least k internally vertex-disjoint connector paths.

The proof is based on a simple application of Menger's theorem [BM08, Theorem 9.1] and crucially uses the domination of each class, Lemma 9.4.1. The proof can also be viewed as a simplified version of that of [CHGK14a, Lemma 3.4].

Proof of Lemma 9.4.3. Fix a layer $\ell \in [L/2, L-1]$. Fix $\mathcal{V}_\ell^i$ and suppose it is a dominating set of $\mathcal{G}$.

Consider the projection $\Psi(\mathcal{V}_\ell^i)$ onto G and recall Menger's theorem: Between any pair (u, v) of non-adjacent nodes of a k-vertex connected graph, there are k internally vertex-disjoint paths connecting u and v. Applying Menger's theorem to a node in $\Psi(\mathcal{C})$ and a node in $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$, we obtain at least k internally vertex-disjoint paths between $\Psi(\mathcal{C})$ and $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$ in G.

We first show that these paths can be shortened so that they satisfy conditions (B) and (C) of potential connector paths, stated in Section 9.4.1.

Pick an arbitrary one of these k paths and denote it $P = v_1, v_2, ..., v_r$, where $v_1 \in \Psi(\mathcal{C})$ and $v_r \in \Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$. By the assumption that $\mathcal{V}_\ell^i$ dominates $\mathcal{G}$, since $v_1 \in \Psi(\mathcal{C})$ and $v_r \in \Psi(\mathcal{S}_\ell^i \setminus \mathcal{C})$, either there is a node v_i along P that is connected to both $\Psi(\mathcal{C})$ and $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$, or there must exist two consecutive nodes v_i, v_{i+1} along P, such that one of them is connected to $\Psi(\mathcal{C})$ and the other is connected to $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$. In either case, we can derive a new path P' which has at most 2 internal nodes, i.e., satisfies (B), is internally vertex-disjoint from the other $k-1$ paths since its internal nodes are a subset of the internal nodes of P and are not in $\Psi(\mathcal{V}_\ell^i)$.

If in this path with length 2, the node closer to $\mathcal{C}$ has a neighbor in $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$, or if the node closer to the $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$ side has a neighbor in $\mathcal{C}$, then we can further shorten the path and get a path with only 1 internal node. Note that this path would still remain internally vertex-disjoint from the other $k-1$ paths since its internal nodes are a subset of the internal nodes of P and are not in $\Psi(\mathcal{V}_\ell^i)$.

After shortening all the k internally vertex-disjoint paths, we get k internally vertex-disjoint paths in graph G that satisfy conditions (A), (B), and (C).

Now using rules (D) and (E) in Section 9.4.1, we can transform these k internally vertex-disjoint *potential* connector paths in G into k internally vertex-disjoint connector paths on the virtual nodes of layer $l+1$. It is clear that during the transition from the real nodes to the virtual nodes, the connector paths remain internally vertex-disjoint. □

9.4.2 The Fast Merger Lemma

We next show that the described algorithms will make the number of connected components go down by a constant factor in each layer. The formal statement is as follows:

Lemma 9.4.4 (Fast Merger Lemma). *For each layer $\ell \in [\frac{L}{2}, L-1]$, $M_{\ell+1} \leq M_\ell$, and moreover, there are constants $\delta, \rho > 0$ such that $\Pr[M_{\ell+1} \leq (1-\delta)\cdot M_\ell] \geq \rho$ with independence between layers.*

Proof. Let ℓ be a layer in $[\frac{L}{2}, L-1]$. For the first part note that since from Lemma 9.4.1 we know that $\mathcal{V}^i_{L/2}$ is a dominating, and as $\mathcal{V}^i_\ell \subseteq \mathcal{V}^i_{\ell'}$ for $\ell \leq \ell'$, each new node of layer $\ell+1$ that is added to class i has a neighbor in the old components of this class and thus, the new nodes do not increase the number of connected components.

For the second part, let i be a class for which $N^i_\ell \geq 2$ and consider a component $\mathcal{C}$ of $\mathcal{G}[\mathcal{V}^i_\ell]$. We say that $\mathcal{C}$ is *good* if one of the following two conditions is satisfied: (I) There is a type-1 new node v that has a neighbor in $\mathcal{C}$ and a neighbor in a component $\mathcal{C}' \neq \mathcal{C}$ of G^i_ℓ and v joins class i. (II) There are two neighboring new nodes, w and u, with types 2 and 3, respectively, such that w has a neighbor in $\mathcal{C}$, u has a neighbor in a component $\mathcal{C}' \neq \mathcal{C}$ of $\mathcal{G}[\mathcal{V}^i_\ell]$, and both u and w join class i. Otherwise, we say that $\mathcal{C}$ is *bad*. By definition, if a connected component of old nodes is good, then at the next layer it is merged with another component of its class.

Let X^i_ℓ be the number of bad connected components of class i if $N^i_\ell \geq 2$ and $X^i_\ell = 0$ otherwise. Also, define $Y_\ell = \sum_{i=1}^t X^i_\ell$, which gives $M_{\ell+1} \leq \frac{M_\ell - Y_\ell}{2} + Y_\ell$. To prove the lemma, we show that $\mathbb{E}[Y_\ell] \leq (1-3\delta)\cdot M_\ell$ for some constant $\delta > 0$. Then, using Markov's inequality we get that $\Pr[Y_\ell \leq (1-2\delta)\cdot M_\ell] \geq 1 - \frac{1-3\delta}{1-2\delta}$ and therefore $\Pr[M_{\ell+1} \leq (1-\delta)M_\ell] \geq 1 - \frac{1-3\delta}{1-2\delta}$ and thus the lemma follows.

Hence, it remains to prove that $\mathbb{E}[Y_\ell] \leq (1-3\delta)\cdot M_\ell$ for some $\delta > 0$. For this, we divide the connected components of old nodes into two groups of *fast* and *slow* components, as follows: Consider a class i such that $N^i_\ell \geq 2$. A connected component $\mathcal{C}$ of $\mathcal{G}[\mathcal{V}^i_\ell]$ is called *fast* if it has at least $\Omega(k)$ short connector paths, and *slow* otherwise. Note that by Lemma 9.4.3, w.h.p., each slow component has at least $\Omega(k)$ long connector paths.

Let M^F_ℓ and M^S_ℓ be the total number of fast and slow connected components, respectively, of graphs $\mathcal{G}[\mathcal{V}^i_\ell]$ as i ranges over all classes. Note that $M^F_\ell + M^S_\ell = M_\ell$. We say that a short connector path p for $\mathcal{C}$ is *good* if its internal node is in the same class as $\mathcal{C}$. Let Y^F_ℓ be the total number of fast connected components for which none of the short paths is good.

Because every type-1 new node picks its class number randomly, each of the $\Omega(k)$ short paths (independently) has probability at least $1/k$ to be in the right class. The expected number of short paths in the right class is therefore constant and hence, there exists a constant $\delta > 0$ such that $\mathbb{E}[Y_\ell^F] \leq (1 - 3\delta) \cdot M_\ell^F$.

Moreover, let $\mathcal{K}$ be set of the slow connected components of the graphs $\mathcal{G}[\mathcal{V}_\ell^i]$ (for all classes $i \in [1, t]$) for which none of the short paths is good and let $K := |\mathcal{K}|$. Let $\mathcal{M}$ be the maximal matching the algorithm computes for the bridging graph. In order to complete the proof, we show that the expected size of $\mathcal{M}$ is at least $3\delta \cdot K$ for some $\delta > 0$. Given this, we get that

$$\begin{aligned}\mathbb{E}[Y_\ell] &= \mathbb{E}[Y_\ell^F] + \big(\mathbb{E}[K] - \mathbb{E}[|\mathcal{M}|]\big) \\ &\leq (1 - 3\delta) \cdot (M_\ell^F + K) \\ \leq (1 - 3\delta) \cdot (M_\ell^F + M_\ell^S) &= (1 - 3\delta) \cdot M_\ell,\end{aligned}$$

which would complete the proof.

To show that the expected size of the maximal matching is at least $3\delta \cdot K$ for some $\delta > 0$, it is sufficient to prove that the expected size of a maximum matching is at least $\Omega(K)$. It is easy to see that the size of any maximal matching is at least half of the size of a maximum matching. Hence, we just need to show that the expected size of the maximum matching is at least $\Omega(K)$. We do this in Lemma 9.4.5. □

Lemma 9.4.5. *The expected size of the maximum matching in the bridging graph is at least* $\Omega(K)$.

This lemma is the key part of the analysis. Roughly speaking, the bridging graph might have a complex structure and thus, we do not know how to work with it directly. However, the saving grace is that we know more about the connector paths, thanks to Lemma 9.4.3. Using long connector paths, we algorithmically identify a (random) subgraph $\mathcal{H}$ of the bridging graph and show that just this subgraph $\mathcal{H}$ contains a matching of size $\Omega(K)$. The analysis of this algorithm uses a simple probability tail bound inequality that we develop for our specific problem.

Proof of Lemma 9.4.5. In order to prove that the expected size of the maximum matching is at least $\Omega(K)$, we focus on a special sub-graph $\mathcal{H}$ of the bridging graph (to be described next). We show that in expectation, $\mathcal{H}$ has a matching of size $\Omega(K)$. Since $\mathcal{H}$ is a sub-graph of the bridging graph, this proves that the expected size of the maximum matching in the bridging graph is at least $\Omega(K)$ and thus completes the proof of Lemma 9.4.4.

Subgraph $\mathcal{H}$ This graph is obtained from the connector paths of components in $\mathcal{K}$. First, *discard* each new node of type 3 with probability 1/2. This is done for cleaner dependency arguments. For each type-2 new node v, we determine components that are neighbors of v in $\mathcal{H}$ as follows: Consider all the long connector paths for components in $\mathcal{K}$ that go through v.

Pick an arbitrary one of these long connector paths and assume that it belongs to component $\mathcal{C} \in \mathcal{K}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$. Suppose that the path goes from $\mathcal{C}$ to v, then to a type-3 node u, and then finally to a component $\mathcal{C}' \neq \mathcal{C}$ of the graph $\mathcal{G}[\mathcal{V}_\ell^i]$. Mark component $\mathcal{C}$ as a potential neighbor for v in $\mathcal{H}$ if and only if w is not discarded and w has joined class i. Go over all long connector paths of v and mark the related potential component neighbors of v accordingly. If at the end, v has exactly one potential component neighbor, then we include that one as the neighbor of v in $\mathcal{H}$. Otherwise, v does not have any neighbor in $\mathcal{H}$.

It is easy to see see that $\mathcal{H}$ is a sub-graph of the bridging graph. Moreover, the degree of each type-2 new node in $\mathcal{H}$ is at most 1. However, we remark that it is possible that a component has degree greater than one in $\mathcal{H}$. Thus, $\mathcal{H}$ is not necessarily a matching.

To complete the proof, in the following, we show that the expected size of the maximum matching of $\mathcal{H}$ is at least $\Omega(K)$.

More specifically, we show that there is a constant $\delta > 0$ such that for each component $\mathcal{C}$ in $\mathcal{K}$, with probability at least δ, $\mathcal{C}$ has at least one long connector path p that satisfies the following condition:

- **($\star$)** The long connector path p has internal type-2 new node v, and in $\mathcal{H}$, node v has $\mathcal{C}$ as its only neighbor.

If a component $\mathcal{C}$ has at least one long connector path that satisfies ($\star$), then we pick exactly one such long connector path and we match the type-2 node of that path to $\mathcal{C}$.

Once we show that each component $\mathcal{C}$ in $\mathcal{K}$ with probability at least δ has a long connector path satisfying ($\star$), then the proof can be completed by linearity of expectation since the number of components in $\mathcal{K}$ is K and each components in $\mathcal{K}$ gets matched with probability at least δ.

We first study each long connector path p of $\mathcal{C}$ separately and show that p satisfies ($\star$) with probability at least $\frac{1}{4t}$. Moreover, we show that regardless of what happens for other long connector paths of $\mathcal{C}$, the probability that p satisfies ($\star$) is at most $\frac{1}{2t}$.

Suppose that p is composed of type-2 new node v and type-3 new node w. Suppose that other than class i, v is also on long connector paths of classes i'_1, i'_2, ..., i'_z where $z < t$. By Proposition 9.4.2, for each other class i'_j, v is on a connector path of at most one component of class i'_j. Let u_1 to $u_{z'}$ be the type-3 nodes on the long connector paths related to these classes. Note that z' might be smaller than z as it is possible that the long connector paths of the z classes share some of the type 3 nodes. Path p satisfies condition ($\star$) if and only if the following two conditions hold: (a) w is not discarded and it joins class i, (b) for each class i'_j, the type-3 node on the long connector path related to class i'_j that goes through v is either discarded or it does not join class i'_j. The probability that (a) is satisfied is exactly $\frac{1}{2t}$. On the other hand, since different classes might have common type-3 nodes on their paths, the events of different classes i'_j satisfying the condition (b) are not independent. However, for each type-3 new node $u_{j'}$, suppose that $x_{j'}$ is the number of classes other than i which have long connector paths through $u_{j'}$. Then, the probability that $u_{j'}$ is discarded or that it

does not join any of these $x_{j'}$ classes is $1 - \frac{x_{j'}}{2t} \geq 4^{-\frac{x_{j'}}{2t}}$, where the inequality follows because $\frac{x_{j'}}{2t} \leq \frac{1}{2}$. The probability that the above condition is satisfied for all choices of $u_{j'}$ is at least $4^{-\sum_{j'=1}^{z} \frac{x_{j'}}{2t}}$. Since $\sum_{j'=1}^{z'} x_{j'} = z \leq t$, we get that the probability that (b) holds is at least $4^{-\frac{1}{2}} = \frac{1}{2}$. Hence, the probability that both (a) and (b) happen is at least $\frac{1}{4t}$. This proves that p satisfies $(\star)$ with probability at least $\frac{1}{4t}$. To show that this probability is at most $\frac{1}{2t}$, regardless of what happens with other paths, it is sufficient to notice that w satisfies (a) with probability at most $\frac{1}{2t}$.

We now look over all long connector paths of component $\mathcal{C}$ together. Let Z be the number of long connector paths of $\mathcal{C}$ which satisfy condition $(\star)$. To conclude the proof, we need to show that $\Pr[Z \geq 1] \geq \delta$ for some constant $\delta > 0$. Note that the events of satisfying this condition for different long connector paths are not independent. In fact, they are positively correlated and thus we can not use standard concentration bounds like a Chernoff bound. Markov's inequality does not give a sufficiently strong result either. To prove the claim, we use an approach which has a spirit similar to the proof of Markov's inequality but is tailored to this particular case.

We know that w.h.p., $\mathcal{C}$ has at least $k' = \Omega(k)$ long connector paths. Let us assume that this holds. Using linearity of expectation, we have $\mathbb{E}[Z] \geq \frac{k'}{4t}$. Since $t = \Theta(k)$, by choosing a small enough constant in definition of $t = \Theta(k)$, we get that $\mathbb{E}[Z] \geq \frac{k'}{4t} = z_0$ for some constant $z_0 > 1$. Given this, we want to show that $\Pr[Z \geq 1] \geq \delta$ for some constant $\delta > 0$.

Because of the upper bound on the probability for a path p to satisfy condition $(\star)$ which holds independently of what happens for other connector paths, we have

$$\Pr[Z = \zeta] \leq \binom{k'}{\zeta} (\frac{1}{2t})^{\zeta} \leq \left(\frac{2ek'}{2t\zeta}\right)^{\zeta} = \left(\frac{4ez_0}{\zeta}\right)^{\zeta}.$$

Following the above equation, intuitively, for some constant threshold ζ_0 and a variable $\zeta \geq \zeta_0$, $\Pr[Z = \zeta]$ decreases exponentially. This happens for example if we set $\zeta_0 = 20z_0$. Hence, the contribution of the part where $Z > \zeta_0$ to the expectation $\mathbb{E}[Z]$ is very small and essentially negligible. This means that to have $\mathbb{E}[Z] \geq z_0$, a constant part of the probability mass should be on values $Z \in [1, \zeta_0]$, which completes the proof.

Having this intuition, the formal argument is as follows. Let $\beta = \Pr[Z \geq 1]$. Then we have

$$\begin{aligned}
z_0 \leq \mathbb{E}[Z] &= \sum_{\zeta=0}^{\infty} \zeta \Pr[Z = \zeta] = \sum_{\zeta=1}^{\zeta_0} \zeta \Pr[Z = \zeta] + \sum_{\zeta=\zeta_0+1}^{\infty} \zeta \Pr[Z = \zeta] \\
&\leq \beta \cdot \zeta_0 + \sum_{\zeta=\zeta_0+1}^{\infty} \zeta \binom{k'}{\zeta} \left(\frac{1}{2t}\right)^{\zeta} \leq \beta \cdot \zeta_0 + \sum_{\zeta=\zeta_0+1}^{\infty} \zeta \left(\frac{2ek'}{2t\zeta}\right)^{\zeta} \\
&\leq \beta \cdot \zeta_0 + \sum_{\zeta=\zeta_0+1}^{\infty} \zeta \left(\frac{4ez_0}{\zeta}\right)^{\zeta} < \zeta_0(\beta + \frac{1}{2^{10}}),
\end{aligned}$$

where the last inequality holds if constant ζ_0 is chosen sufficiently large—e.g. $\zeta_0 = 20z_0$. We get that $\beta = \Pr[Z \geq 1] \geq \frac{1}{20} - \frac{1}{2^{10}}$. This shows that $\Pr[Z \geq 1] \geq \delta$ for some constant $\delta > 0$ and thus completes the proof.

□

Lemma 9.4.6. *W.h.p., for each i, the number of virtual nodes in class i is $O(\frac{n \log n}{k})$.*

proof of Lemma 9.4.6. Consider an arbitrary virtual node v. Either (a) v chooses its class number randomly, or (b) v is a type-2 node and it chooses its class number based on the maximal matching. Using a Chernoff bound, the total number of virtual nodes that join class i randomly—following condition (a)—is $O(\frac{n \log n}{t}) = O(\frac{n \log n}{k})$ w.h.p. The number of virtual nodes that join class i following condition (b) is at most equal to the number of connected components of $G^i_{L/2}$. Since virtual nodes of layers 1 to $L/2$ choose their classes following condition (a), we get that the number of virtual nodes that join class i following condition (b) is also $O(\frac{n \log n}{k})$ w.h.p. □

9.5 Distributed Dominating Tree Packing

Here we present the distributed implementation of the outline of Section 9.3:

Theorem 9.5.1. *There is distributed implementation of the fractional dominating tree packing algorithm of Section 9.3 in $O(\min\{\frac{n \log n}{k}, D + \sqrt{n \log n} \log^* n\} \log^3 n)$ rounds of the* V-CONGEST *model.*

Throughout the implementation, we make frequent use of the following protocol, which is an easy extension of the connected component identification algorithm of Thurimella [Thu95, Algorithm 5]. Recalled that we used this algorithm as a subroutine also in the previous two chapters. For simplicity, we recall the functionality provided by this subroutine.

Theorem 9.5.2. [4] *Suppose that we are given a connected network $G = (V, E)$ with n nodes and diameter D and a subgraph $G_{sub} = (V, E')$ where $E' \subseteq E$ and each network node knows the edges incident on v in graphs G and G_{sub}. Moreover, suppose that each network node has a value $x_v \in \{0,1\}^{O(\log n)}$. There is a distributed algorithm in the* V-CONGEST *model with round complexity of $O(\min\{D', D + \sqrt{n} \log^* n\})$ which lets each node v know the smallest (or largest) x_u for nodes u that are in the connected component of G_{sub} that contains v. Here, D' is the largest strong diameter among connected components of G_{sub}.*

[4] We note that the MST algorithm of [KP95] and the component identification algorithm of [Thu95] were originally expressed in the CONGEST model but it is easy to check that the algorithms actually work in the more restricted V-CONGEST model. In particular, [Thu95, Algorithm 5] finds the smallest preorder rank x_v (for node v) in the connected component of each node u but the same scheme works with any other inputs $y_v \in \{0,1\}^{O(\log n)}$ such that $y_v \neq y_u$ for $v \neq u$. To satisfy this condition, we simply set $y_v = (x_v, id_v)$, that is, we append the id of each node to its variable. As a side note, we remark that computing a preorder in the V-CONGEST model requires $\Omega(n)$ rounds but we never use that.

9.5.1 Identifying the Connected Components of Old Nodes

To identify connected components of old nodes, each old node—i.e., those in layers 1 to ℓ—sends a message to all its $\mathcal{G}$-neighbors declaring its class number. We put each $\mathcal{G}$-edge that connects two virtual nodes of the same class in a subgraph $\mathcal{G}_{old}$. Moreover, each virtual node ν sets it id $ID_\nu = (ID_v, layer_\nu, type_\nu)$ where v is the real node that contains ν. Then, using Theorem 9.5.2, each virtual old node learns the smallest ID in its $\mathcal{G}_{old}$-component and remembers this id as its $componentID_\nu$. Running the algorithm of Theorem 9.5.2 takes $O(\min\{\frac{n \log n}{k}, D + \sqrt{n \log n} \log^* n\})$ meta-rounds as the diameter of $\mathcal{G}$ is $D = diam(G)$ and since each component of $\mathcal{G}_{old}$ contains at most $O(\frac{n \log n}{k})$ nodes (Lemma 9.4.6) and thus has strong diameter $O(\frac{n \log n}{k})$.

9.5.2 Creating the Bridging Graph

In the bridging graph, a type-2 new node v is connected to a connected component $\mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$ if and only if the following condition holds: if v joins class i, then component $\mathcal{C}$ becomes connected to some other component $\mathcal{C}'$ of $\mathcal{G}[\mathcal{V}_\ell^i]$ through a type-3 new node, and $\mathcal{C}$ is not already connected to another component by type-1 new nodes.

We first deactivate the components of old nodes that are connected to another component of the same class by type-1 new nodes which chose the same class. This is because in this layer we do not need to spend a type-2 new node to connect these components to other components of their class. To find components that are already connected through type-1 new nodes, first, every old node v sends its class number and $componentID_\nu$ to all its neighbors. Let u be a type-1 new node that has joined class i. If u receives component IDs of two or more components of class i, then u sends a message containing i and a special connector symbol "CONNECTOR" to its neighbors. Each component of class i that receives a message from a type-1 new node containing class i and the special connector symbol gets deactivates, that is, the node sets it local variable $activity = false$. This deactivation decision can be disseminated inside components of $\mathcal{G}_{old}$ in $O(\min\{\frac{n \log n}{k}, D + \sqrt{n \log n} \log^* n\})$ meta-rounds using Theorem 9.5.2.

Now, we start forming the bridging graph. Each old node v (even if v is in a deactivated component) sends its $componentID_v$ and its *activity* status to all neighbors. For a type-3 new node w, let C_w be the set of component IDs w receives in this meta-round. Assume that w joined class i. The node w creates a message m_w using the following rule: If C_w does not contain the component ID of a component of class i, then the message m_w is empty. If C_w contains exactly one component ID of class i, m_w contains the class number i and this component ID. Finally, if C_w contains at least two component IDs of class i, m_w contains the class number i and a special indicator symbol "CONNECTOR". We use this symbol instead of the full list of component IDs, due to message size considerations. Each type-3 new node w sends m_w to all its neighbors.

To form the bridging graph, each type-2 new node v creates a *neighbors list* $List_v$ of active

components which are its neighbors in the bridging graph, as follows: Consider a component $\mathcal{C} \in \mathcal{G}[\mathcal{V}_\ell^i]$. Node v adds $\mathcal{C}$ to $List_v$ if v has a neighbor in active component $\mathcal{C}$ and v received a message m_w from a type-3 new neighbor such that m_w is for class number i and m_w either contains the ID of a component $\mathcal{C}' \neq \mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$, or m_w contains the special "CONNECTOR" symbol.

9.5.3 Maximal Matching in the Bridging Graph

To select a maximal matching in the bridging graph, we simulate Luby's well-known distributed maximal independent set algorithm [Lub86]. Applied to computing a maximal matching of a graph H, the variant of the algorithm we use works as follows: The algorithm runs in $O(\log |H|)$ phases. Initially all edges of H are active. In each phase, each edge picks a random number from a large enough domain such that the numbers picked by edges are distinct with at least a constant probability. An edge that picks a number larger than all adjacent edges joins the matching. Then, matching edges and their adjacent edges become inactive. It follows from [Lub86] that this algorithm produces a maximal matching in $O(\log |H|)$ phases, with high probability.

We adapt this approach to our case as follows. We have $O(\log n)$ stages, one for each phase of Luby's algorithm. Throughout these stages, each type-2 new node v that is still unmatched keeps track of the active components that are still available for being matched to it. This can be done by updating the neighbors list $List_v$ to the remaining matching options. In each stage, each unmatched type-2 new node v chooses a random value of $\Theta(\log n)$ bits for each component in $List_v$. Then, v picks the component $\mathcal{C} \in List_v$ with the largest random value and proposes a matching to $\mathcal{C}$ by sending a proposal message m_v that contains (a) the ID of v, (b) the component ID of $\mathcal{C}$ and (c) the random value chosen for $\mathcal{C}$ by v.

Nodes inside an active connected component may receive a number of proposals and their goal is to select the type-2 new node which proposed the largest random value to any node of this component. Each old node u has a variable named $acceptedProposal_u$, which is initialized to the proposal received by u with the largest random value (if any). We use algorithm of Theorem 9.5.2 with subgraph $\mathcal{G}_{old}$ (described in Section 9.5.2) and with initial value x_u of each node u being its $acceptedProposal_u$. Hence, in $O(\min\{\frac{n \log n}{k}, D + \sqrt{n \log n} \log^* n)$ meta-rounds, each old node u learns the largest $acceptedProposal_v$ amongst nodes v which are in the same $\mathcal{G}_{old}$component as u. Then, u sets its $acceptedProposal_u$ equal to this largest proposal and also sends this $acceptedProposal_u$ to all its neighbors. If a type-2 new node v has its proposal accepted, then v joins the class of that component. Otherwise, v remains unmatched at this stage and updates its neighbors list $List_v$ by removing the components in $List_v$ that accepted proposals of other type-2 new nodes (those from which v received an *acceptedProposal* message). This process is repeated for $O(\log n)$ stages. Each type-2 new node that remains unmatched after these stages joins a random class.

9.5.4 Wrap Up

Now that we have explained the implementation details of each of the steps of the recursive class assignment, we get back to concluding the proof of Theorem 9.5.1.

Proof of Theorem 9.5.1. Since in the matching part of the algorithm each component accepts at most one proposal from a type-2 new node, the described algorithm computes a matching between type-2 new nodes and components. From the analysis of Luby's algorithm [ABI86, Lub85], it follows that after $O(\log n)$ stages, the selected matching is maximal w.h.p. Note that in some cases, the described algorithm might match a type-2 node v and a component $\mathcal{C}$ even if the corresponding edge in the bridging graph does not get the maximal random value among all the edges of v and $\mathcal{C}$ in the bridging graph. However, it is straightforward to see in the analysis of [ABI86, Lub85] that this can only speed up the process.

Regarding the time complexity, for each layer, the identification of the connected components on the old nodes and also creating the bridging graph take $O(\min\{\frac{n \log n}{k}, D + \sqrt{n} \log^* n\})$ meta-rounds. Then, for each layer we have $O(\log n)$ stages and each stage is implemented in $O(\min\{\frac{n \log n}{k}, D + \sqrt{n \log n} \log^* n\})$ meta-rounds. Thus, the time complexity for the each layer is $O(\min\{\frac{n \log n}{k}, D + \sqrt{n \log n} \log^* n\} \log^2 n)$ rounds, which accumulates to $O(\min\{\frac{n \log n}{k}, D + \sqrt{n \log n} \log^* n\} \log^3 n)$ rounds over $L = \Theta(\log n)$ layers.

Furthermore, at the end of the CDS packing construction, in order to turn the CDSs into dominating trees, we simply use a linear time minimum spanning tree algorithm of Kutten and Peleg [KP95] on virtual graph $\mathcal{G}$ with weight 0 for edges between nodes of the same class and weight 1 for other edges. Then, the 0-weight edges included in the MST identify our dominating trees. Running the MST algorithm of [KP95] on virtual graph $\mathcal{G}$ takes at most $O(D + \sqrt{n \log n} \log^* n)$ meta-rounds, or simply $O((D + \sqrt{n \log n} \log^* n) \log n)$ rounds. This MST can be performed also in $O(\frac{n \log n}{k})$ meta-rounds just by solving the problem of each class inside its own graph, which has diameter at most $O(\frac{n \log n}{k})$. In either case, both of these time complexities are subsumed by the other parts. □

9.6 Centralized Dominating Tree Packing

In this section, we explain the details of a centralized implementation of the CDS-Packing algorithm presented in Section 9.3.

Theorem 9.6.1. *There is centralized implementation of the fractional dominating tree packing algorithm of Section 9.3 with time complexity of $O(m \log^2 n)$.*

Proof. We use *disjoint-set data structures* for keeping track of the connected components of the graphs of different classes. Initially, we have one set for each virtual node, and as the algorithm continues, we union some of these sets. We use a simple version of this data structure that takes $O(1)$ steps for find operations and $O(\eta \log \eta)$ steps for union, where η is

the number of elements. Moreover, for each layer ℓ and each type r, we have one linked list which keeps the list of virtual nodes of layer ℓ and type r.

We start with going over real nodes and choosing the layer numbers and type numbers of their virtual nodes. Simultaneously, we also add these virtual nodes to their respective linked list, the linked list related to their layer number and type. This part takes $O(n \log n)$ time in total over all virtual nodes.

To keep the union-set data structures up to date, at the end of the class assignment of each layer ℓ, we go over the edges of virtual nodes of this layer—by going over the virtual nodes of the related linked lists, and their edges—and we update the disjoint-set data structures. That is, for each virtual node v in these linked lists, we check all the edges of v. If the other end of the edge, say u, has the same class as v, then we union the disjoint-set data structures of v and u. Since throughout these steps over all layers, each edge of the virtual graph is checked for union at most twice—once from each side—there are at most $O(m \log^2 n)$ checking steps for union operations. Moreover, the cost of all union operations summed up over all layers is at most $O(n \log^2 n)$. Since $m \geq \frac{nk}{2}$ and $k = \Omega(\log n)$, the cost of unions is dominated by the $O(m \log^2 n)$ cost of checking.

Now we study the recursive class assignment process and its step complexity. For the base case of layers 1 to $L/2$, we go over the linked lists related to layers 1 to $L/2$, one by one, and set the class number for each virtual node in these lists randomly.

In the recursive step, for each layer $j+1$ we do as follows: We first go over the linked list of type-1 virtual nodes of layer $j+1$ and the linked list of type-3 virtual nodes of layer $j+1$ and for each node v in these lists, we select the class number of v randomly. Over all layers, these operations take time $O(n \log n)$. Now we get to the more interesting part, choosing the class numbers of type-2 nodes of layer $j+1$. Recall that this is done via finding a maximal matching in the bridging graph.

We first go over the linked list of type-1 virtual nodes of layer $j+1$ and for each node v in this list, we do as follows. Suppose that v has joined class i. We go over edges of v and find the number of connected components of class i that are adjacent to v. Then if this number is greater than or equal to two, we go over those connected components and mark them as *deactivated* for matching.

Next, for each type-2 virtual node of layer $j+1$, we have one array of size L, called *potential-matches* array. Each entry of this array keeps a linked list of component ids. We moreover assume that we can read the size of this linked list in $O(1)$ time. Note that this can be easily implemented by having a length variable for each linked list.

Now we begin the matching process. For this, we start by going over the linked list of type-3 virtual nodes of layer $j+1$. For each virtual node u in this list, we go over the edges of u and do as follows: if there is a neighbor w of u which is in a layer in $[1, j]$ and is in the same class as u, then u remembers the component id of w. This component id is obtained by performing a find operation on the disjoint-set data structure of w. After going over all edges, u has a list of neighboring connected component ids of the same class as u. Let us call these *suitable components* for u. Then, we go over all the edge of u for one more time

and for each type-2 virtual neighbor v of u that is in layer $j+1$, we add the list of suitable components of u into the entry of the potential-matches array of v which is related to the class of u.

After doing as above for the whole linked list of type-3 virtual nodes of layer $j+1$, we now find the maximal matching. For this purpose, for each component of nodes of layers 1 to j, we have one Boolean flag variable which keeps track of whether this component is matched or not in layer $j+1$. We go over the linked list of type-2 virtual nodes of layer $j+1$ and for each virtual node v in this list, we go over the edges of v and do as follows (until v gets its class number): for each virtual neighbor w' of v, if w' is in a layer in $[1, j]$, we check the component of w'. If this component is unmatched and is not deactivated for matching, we check for possibility of matching v to this component. Let i' be the class of this component and let $CID_{w'}$ be the component id of w'. Note that we find $CID_{w'}$ using a find operation on the disjoint-set data structure of w'. We look in the potential-matches array of v in the entry related to class i'. If this linked list has length greater than 1, or if it has length exactly 1 and the component id in it is different from $CID_{w'}$, then node v chooses class i'. In that case, we also set the matched flag of component of w' to indicate that it is matched now. If v is matched, we are done with v and we go to the next type-2 node in the linked list. However, if v does not get matched after checking all of its neighbors, then we choose a random class number for v.

It is clear that in the above steps, each edge of the virtual graph that has at least one endpoint in layer $j+1$ is worked on for $O(1)$ times. In each such time, we access $O(1)$ variables and we perform at most one find operation on a disjoint-set data structure. Since each find operation costs $O(1)$ time, the overall cost of these class assignment steps over all the layers becomes $O(m \log^2 n)$.

At the end of the CDS packing construction, in order to turn the CDSs into dominating trees, we simply use a linear time minimum spanning tree algorithm, e.g., [FW90], which on the virtual graph takes $O(m \log^2 n)$ steps. □

9.7 Testing A Dominating Tree Packing

Lemma 9.7.1. *A dominating tree packing of a connected undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be tested, using a distributed algorithm in $\tilde{O}(\min\{d', diam(\mathcal{G}) + \sqrt{|\mathcal{V}|}\})$ rounds of the* V-CONGEST *model, where d' is an upper bound on the diameter of each dominating tree, or using a centralized algorithm in $\tilde{O}(\mathcal{E})$ steps,*

More specifically, the lemma states the following: Suppose that we are given a partition of vertices $\mathcal{V}$ into disjoint classes $V_1, V_2, \ldots, V_t$ where each node knows its class number and the value of t. We can simultaneously test whether it is true for all classes $i \in [1, t]$ that $\mathcal{G}[V_i]$ is a CDS, or not. If each class is a CDS, then the test passes and otherwise—i.e., if there is even one class that is not a CDS—then the test fails with high probability. Moreover, the

outputs of all nodes are consistent in that either all the nodes declare a failure or the test passes in all the nodes.

Proof. We first explain the distributed algorithm. The centralized algorithm is a simpler variant of the same approach. The general idea is to first check connectivity of all classes, and then check whether there is any disconnected class or not. Let $D = diam(\mathcal{G})$, $n' = |\mathcal{V}|$, and $m' = |\mathcal{E}|$.

Distributed Domination Test We first check if each class is a dominating set. For this, each node sends its class number to its neighbors. If a node v is not dominated by a class i, that is if v does not receive any message from a node in class i, then v initiates a *'domination-failure'* message and sends it to its neighbors. We use $\Theta(D)$ rounds to propagate these *'domination-failure'* messages: in each round, each node sends the *'domination-failure'* message to its neighbors if it received *'domination-failure'* message in one of the previous rounds. After these $\Theta(D)$ rounds, if the domination part of the test passes, we check for connectivity.

Distributed Connectivity Test We first use $O(\min\{d', Diam(\mathcal{G}) + \sqrt{n'}\log^* n\})$ rounds to identify the connected components of each class, using Theorem 9.5.2 where each node v starts with its own id as its variable x_v and only edges between the nodes of the same class are included in the subgraph $\mathcal{G}_{sub}$. Hence, at the end of this part, each connected component has a leader and every node u knows the id of the leader of its connected component, which is recorded as the component id of u.

Given these component ids, to test connectivity, we check if there exist two nodes in the same class with different component ids. Suppose that there exists a nonempty set of classes I which each have two or more connected components. We show a protocol such that w.h.p., at least one node v receives two different component ids related to a class in $i \in I$. We call this a *"disconnect detection"* as it indicates that class i is disconnected. If this happens, then v initiates a *'connectivity-failure'* message. $\Theta(D)$ rounds are used to propagate these *'connectivity-failure'* messages.

In the first round of the connectivity test, each node sends its class number and its component id to all of its neighbors. Since each class is dominating (already tested), each node receives at least one component id for each class. If a disconnect is detected at this point, we are done. Suppose that this is not the case. Note that this is possible because connected components of each class $i \in I$ can be at distance more than 1 from each other.

However, using Menger's theorem along with vertex connectivity k of the graph and since the domination part of the test has passed, with an argument as in the proof of Lemma 9.4.3, we get that for each class $i \in I$ and each component $\mathcal{C}$ of class i, there are k internally vertex-disjoint paths of length exactly 3 connecting $\mathcal{C}$ with other components of class i. Note that the length is exactly 3 because length-2 would lead to detection of inconsistency in the first part of the connectivity test. Let us call these *detector paths* of class i.

The algorithm is as follows: in each round, each node v chooses a random class i' and sends the component ID related to class i' (the component ID related to class i' that v has heard so far). In order for the inconsistency to be detected, it is enough that one of the internal nodes on the (at least) k detector paths related to a class $i \in I$ sends the component ID of class i that it knows. This is because, if that happens, then the other internal node on that path would detect the disconnect.

For each node v, let x_v be the number of disconnected classes i for which v is an internal node on one of the detector paths of class i. Then, in each round, with probability $\frac{x_v}{t}$, node v sends a component ID which leads to disconnect detection. Hence, for each round, the probability that no such ID is sent is

$$\prod_{v \in V} \left(1 - \frac{x_v}{t}\right) \leq \quad e^{-\sum_{v \in V} \frac{x_v}{t}} \overset{(\dagger)}{\leq} e^{-\frac{2k \cdot |I|}{t}} \overset{(*)}{\leq} \quad e^{-\frac{2k \cdot \max\{1, t-k\}}{t}} < e^{-\frac{1}{2}}.$$

Here, Inequality $(\dagger)$ holds because there are $|I|$ disconnected classes and each disconnected class has at least $2k$ internal nodes on its detector paths. Inequality $(*)$ holds because a graph with vertex connectivity k can have at most k vertex-disjoint CDS sets and thus $|I| \geq t - k$, and we have assumed that $I \neq \emptyset$. Since in each round there is a constant probability for disconnect detection, after $\Theta(\log n')$ rounds, at least one node will detect it with high probability, and thus after additional $\Theta(D)$ rounds, all nodes know that at least one class is not connected. If no such disconnect is detected in initial $\Theta(\log n')$ rounds (thus not reported by the end of $\Theta(D + \log n')$ rounds), the connectivity test also passes and thus, the complete CDS partition test passes claiming that w.h.p., each class is a CDS.

The Centralized Tests Now we turn to explaining the centralized counterpart of the above algorithm: Testing domination in $O(m')$ time is easy: we go over the nodes one by one, for each node, we read the class number of its neighbors and record which classes are dominating this node. After that, if there is any class left out, we have found 'domination-failure'. This way, we work on each edge at most twice, once from each side, and thus the whole domination testing finishes in $O(m')$ steps. For testing connectivity, the general approach remains the same as in the distributed setting, but we change the component identification part. Note that in the centralized setting, one can identify the connected components of a subgraph of the graph $\mathcal{G}$ in $O(m')$ rounds, using disjoint-set data structures (see Section 9.6). After identifying the components, we can deliver the component id of each node to its neighbors in a total of $O(m')$ rounds. Then, we simply run the $\Theta(\log n')$ rounds of the distributed algorithm where each node sends the id of a random class to its neighbors in a centralized manner. Each round can be clearly simulated in $O(m')$ steps of the centralized setting. Hence, $O(\log n')$ rounds can be simulated in $O(m' \log n')$ rounds and after that, if there is any disconnected class, with high probability a disconnect detection has happened. This concludes the centralized test. □

9.8 A Simple Application Example: Gossiping

Here, we explain a simple and crisp example which shows how our connectivity decompositions can be used in information dissemination. We study the classical *gossiping* problem (aka *all-to-all broadcast*): Each node of the network has one $O(\log n)$-bit message and the goal is for each node to receive all the messages. In the following, we study this problem in the V-CONGEST model. To make the discussion more intuitive, we first explain the approach for a particular value of connectivity and then state how it extends to other connectivity values.

If the network is merely connected, solving the gossiping problem in $O(n)$ rounds is trivial. Now suppose that the network has in fact a vertex connectivity of $\sqrt{n}$. Despite this extremely good connectivity, prior to this work, the aforementioned $O(n)$ rounds solution remained the best known bound. The main difficulty is that, even though we know that each vertex cut of the network admits a flow of $\sqrt{n}$ messages per round, it is not clear how to organize the transmissions such that a flow of $\Omega(\sqrt{n})$ (distinct) messages per round passes through each cut. As a side note, it is worth mentioning that a graph with vertex connectivity k can have up to $\Theta(2^k \cdot (\frac{n}{k})^2)$ vertex-cuts of size k [Kan90].

Our vertex connectivity decomposition, claimed in Theorem 9.1.1, runs in $\tilde{O}(\sqrt{n})$ rounds in this example. Note that in a graph with vertex connectivity k, the diameter is at most $O(n/k)$ and thus here we have $D = O(\sqrt{n})$. This construction generates $O(\sqrt{n})$ dominating trees, each of diameter $\tilde{O}(\sqrt{n})$, where each node is contained in $O(\log n)$ trees. Then, to use this decomposition for gossiping, we do as follows: first each node gives its message to (one node in) a random one of the trees. Note that this is easy as each node has neighbors in all of the trees and it can easily learn the ids of those trees in just one round. Then, w.h.p, we have $O(\sqrt{n})$ messages in each tree, ready to be broadcast. We can broadcast all the messages just inside the dominating trees in $\tilde{O}(\sqrt{n})$ rounds. Furthermore, by a small change, we can make sure that each node transmits the messages assigned to its dominating trees and thus, each node in the network receives all the messages (because of domination). Overall, this method solves the problem in $\tilde{O}(\sqrt{n})$ rounds.

We now state how this bound generalizes:

Corollary 9.8.1. *Suppose that there are N messages in arbitrary nodes of the network such that each node has at most η messages. Using our vertex connectivity decomposition, we can broadcast all messages to all nodes in $\tilde{O}(\eta + \frac{N+n}{k})$ rounds of the* V-CONGEST *model.*

Proof Sketch. This approach is exactly as explained above. Each node gives its messages to random dominating trees of the decomposition and then we broadcast each message only using the nodes in its designated dominating tree. The vertex connectivity decomposition runs in $\tilde{O}(\min\{D + \sqrt{n}, \frac{n}{k}\})$. Then, delivering messages to the trees takes at most η rounds. Finally, broadcasting messages using their designated trees takes $\tilde{O}(\frac{n}{k} + \frac{N}{k})$ rounds because the diameter of each tree is $O(\frac{n \log n}{k})$ and each tree w.h.p is responsible for broadcasting at most $O(\frac{N}{k} + \log n)$ messages. □

Note that the bound in Corollary 9.8.1 is optimal, modulo logarithmic factors, because, (1) $\frac{N}{k}$ is a clear information theoretic lower bound as per round only $O(k \log n)$ bits can cross each vertex cut of size k, (2) similarly, if a node has η messages, it takes at least η rounds to send them, (3) in graphs with vertex connectivity k, the diameter can be up to $\frac{n}{k}$. In fact, the diameter of the original graph is a measure that is rather irrelevant because even if the diameter is smaller, achieving a flow of size $\tilde{\Theta}(k)$ messages per round unavoidably requires routing messages along routes that are longer than the shortest path (for any algorithm). See Section 9.10 for more discussion about this last point.

9.9 Distributed Fractional Spanning-Tree Packing

Recall that the celebrated results of Tutte [Tut61] and Nash-Williams [NW61] show that each graph with edge connectivity λ contains $\lceil \frac{\lambda-1}{2} \rceil$ edge-disjoint spanning trees. In this section, we prove Theorem 9.1.3 which achieves a fractional spanning tree packing with almost the same size. In Section 9.9.1, we explain the algorithm for the case where $\lambda = O(\log n)$. We later explain in Section 9.9.2 how to extend this algorithm to the general case.

9.9.1 Fractional Spanning Tree Packing for Small Edge Connectivity

We follow a classical and generic approach (see e.g. [PST91, SM90, KPST94, Kar96]) which can be viewed as an adaptation of the Lagrangian relaxation method of optimization theory. Tailored to our problem, this approach means we always maintain a collection of weighted trees which might have a large weight going through one edge, but we iteratively improve this collection by penalizing the edges with large load, which incentivizes the collection to take some weight away from the edges with larger load and distribute it over the edges with smaller loads. We next present the formal realization of this idea.

Algorithm Outline We will always maintain a collection T of weighted trees—where each tree $\tau \in T$ has weight $w_\tau \in [0, 1]$—such that the total weight of the trees in the collection is 1. That is $\sum_{\tau \in T} w_\tau = 1$. We start with a collection containing only one (arbitrary) tree with initial weight 1 and iteratively improve this collection for $\Theta(\log^3 n)$ iterations: During each iteration, for each edge $e \in E$, let x_e be the *weighted load* on edge e, that is $x_e = \sum_{\tau, e \in \tau} w_\tau$ and also, let $z_e = x_e \lceil \frac{\lambda-1}{2} \rceil$. Our goal is that at the end, we have $\max_{e \in E} z_e \leq 1 + O(\varepsilon)$.

In each iteration, for each edge e, we define a cost $c_e = exp(\alpha \cdot z_e)$, where $\alpha = \Theta(\log n)$. Then, we find the Minimum Spanning Tree (MST) with respect to these costs. If $Cost(MST) = \sum_{e \in MST} c_e > (1 - \varepsilon) \sum_{e \in E} c_e \cdot x_e$, then the algorithm terminates. On the other hand, if $Cost(MST) = \sum_{e \in MST} c_e \leq (1 - \varepsilon) \sum_{e \in E} c_e \cdot x_e$, then we add this MST to our weighted tree collection T, with weight $\beta = \Theta(\frac{1}{\alpha \log n})$, and to maintain condition $\sum_{\tau \in T} w_\tau = 1$, we multiplying the weight of the old trees in T by $1 - \beta$.

Analysis for the Algorithm of Section 9.9.1 First, in Lemma 9.9.1 we show that if in some iteration we stop because of the condition $\sum_{e\in MST} c_e > (1-\varepsilon)\sum_{e\in E} c_e \cdot x_e$, then $\max_{e\in E} z_e \leq 1+\varepsilon$. Then, in Lemma 9.9.2, we show that if throughout $\Theta(\log^3 n)$ iterations, the condition $\sum_{e\in MST} c_e > (1-\varepsilon)\sum_{e\in E} c_e \cdot x_e$ is never satisfied, then the collection attained at the end of $\Theta(\log^3 n)$ iterations has the property that $\max_{e\in E} z_e \leq 1+\varepsilon$.

Lemma 9.9.1. *If in some iteration* $\sum_{e\in MST} c_e > (1-\varepsilon)\sum_{e\in E} c_e \cdot x_e$, *then* $\max_{e\in E} z_e \leq 1+6\varepsilon$.

Proof. Let $Z = \max_{e\in E} z_e$. First note that

$$\begin{aligned}\sum_{e\in E \text{ and } z_e\leq(1-\varepsilon)Z} c_e \leq & \sum_{e\in E} exp(\alpha(1-\varepsilon)Z) \\ \leq & \; m\cdot exp(-\alpha\varepsilon Z)\cdot exp(\alpha Z) \leq m\cdot exp(-\alpha\varepsilon Z)\sum_{e\in E} c_e \leq (\varepsilon/2)\cdot\sum_{e\in E} c_e.\end{aligned}$$

Thus, we have

$$\sum_{e\in E} c_e\cdot x_e \geq \sum_{e\in E \text{ and } z_e\geq(1-\varepsilon)Z} c_e\cdot x_e \geq (1-\varepsilon)\frac{Z}{\lceil\frac{\lambda-1}{2}\rceil}\sum_{e\in E \text{ and } z_e\geq(1-\varepsilon)Z} c_e \geq (1-\varepsilon)^2\frac{Z}{\lceil\frac{\lambda-1}{2}\rceil}\sum_{e\in E} c_e,$$

and hence

$$\sum_{e\in MST} c_e > (1-\varepsilon)\sum_{e\in E} c_e\cdot x_e > (1-\varepsilon)^3\frac{Z}{\lceil\frac{\lambda-1}{2}\rceil}\sum_{e\in E} c_e \geq (1-\varepsilon)^3 Z\sum_{e\in MST} c_e,$$

where the last inequality follows from the results of Tutte and NashWillams, which show that E contains at least $\lceil\frac{\lambda-1}{2}\rceil$ edge-disjoint spanning trees and clearly each of these trees has cost at least equal to that of the MST. Comparing the two sides of the above inequality, we get $Z \leq (1-\varepsilon)^{-3} \leq 1+6\varepsilon$. □

Lemma 9.9.2. *If the condition* $\sum_{e\in MST} c_e > (1-\varepsilon)\sum_{e\in E} c_e\cdot x_e$ *is never satisfied in* $\Theta(\log^3 n)$ *iterations of the algorithm, then for the collection attained at the end of* $\Theta(\log^3 n)$ *iterations, we have* $\max_{e\in E} z_e \leq 1+\varepsilon$.

Proof. Consider the potential function $\Phi = \sum_{e\in E} c_e = \sum_{e\in E} exp(\alpha z_e)$. We first show that, if in an iteration we have $\sum_{e\in MST} c_e \leq (1-\varepsilon)\sum_{e\in E} c_e\cdot x_e$, then with the update of this

iteration, the potential function decreases at least by a factor of $1 - \Theta(\varepsilon/\log n)$.

$$\begin{aligned}
\Delta\Phi = \quad & \Phi_{old} - \Phi_{new} \\
= \quad & \sum_{e\in E} exp(\alpha z_e^{old}) - exp(\alpha z_e^{new}) = \sum_{e\in E} exp(\alpha z_e^{old}) \cdot (1 - exp(\alpha\beta\lceil\frac{\lambda-1}{2}\rceil \cdot (1_e^{MST} - x_e^{old}))) \\
\geq \quad & \alpha\beta\lceil\frac{\lambda-1}{2}\rceil \sum_{e\in E} exp(\alpha z_e^{old}) \cdot (x_e^{old} - 1_e^{MST}) = \alpha\beta\lceil\frac{\lambda-1}{2}\rceil(\sum_{e\in E} c_e \cdot x_e - \sum_{e\in MST} c_e) \\
\geq \quad & \alpha\beta\lceil\frac{\lambda-1}{2}\rceil\varepsilon \sum_{e\in E} c_e \cdot x_e \geq \alpha\beta\lceil\frac{\lambda-1}{2}\rceil\varepsilon \sum_{e\in E \text{ and } z_e\geq(1-\varepsilon)Z} c_e \cdot x_e \\
\geq \quad & \alpha\beta\varepsilon(1-\varepsilon)\cdot Z \sum_{e\in E \text{ and } z_e\geq(1-\varepsilon)Z} c_e \geq \alpha\beta\varepsilon(1-\varepsilon)^2 Z \sum_{e\in E} c_e \geq \Theta(\frac{\varepsilon}{\log n})\Phi_{old}.
\end{aligned}$$

Now note that the starting potential is at most $m \cdot exp(\alpha\lceil\frac{\lambda-1}{2}\rceil)$. When the potential falls below $exp(\alpha(1+\varepsilon))$, all edges have $z_e \leq 1+\varepsilon$ which means we have found the desired packing. Since in each iteration that condition $\sum_{e\in MST} c_e \leq (1-\varepsilon)\sum_{e\in E} c_e \cdot x_e$ holds, the potential decreases by a factor of $1 - \Theta(\varepsilon/\log n)$, we get that after at most $\Theta(\frac{\log n}{\varepsilon} \cdot (\alpha\lambda + \log m))$ iterations, it falls below $exp(\alpha(1+\varepsilon))$. Noting that $\alpha = O(\log n)$, $\lambda = O(\log n)$ and $\varepsilon = \Theta(1)$, we can infer that this happens after at most $\Theta(\log^3 n)$ iterations. □

Distributed Implementation Using the distributed minimum spanning tree algorithm of Kutten and Peleg [KP95], we can perform one iteration in $O(D + \sqrt{n}\log^* n)$ rounds of the V-CONGEST model [5]. Hence, the $\Theta(\log^3 n)$ iterations of the above algorithm can be performed in $O((D + \sqrt{n}\log^* n)\log^3 n)$ rounds. Note that in these iterations, each node v simply needs to know the weight on edges incident on v and whether another iteration will be used or not. The latter decision can be made centrally—in a leader node, e.g., the node with the largest id—by gathering the total cost of the minimum spanning tree over a breadth first search tree rooted at this leader and then propagating the decision of whether to continue to next iteration or not to all nodes.

9.9.2 Generalized Fractional Spanning Tree Packing

The key idea for addressing the general case of λ—specially when $\lambda = \Omega(\log n)$—is that we randomly decompose the graph into spanning subgraphs each with connectivity $\min\{\lambda, \Theta(\log n/\varepsilon^2)\}$ using random edge-sampling and then we run the edge-connectivity decomposition in each subgraph.

[5] For communication purposes, [KP95] assumes that the weight of each edge can be described in $O(\log n)$ bits. In our algorithm, the weight of each edge e is in the form $c_e = exp(\alpha \cdot z_e)$ and can be potentially super-polynomial, which means the naive way of sending it would require $\omega(\log n)$ bits. However, it simply is enough to send z_e instead of c_e as then the receiving side can compute c_e. Fortunately, the maximum value that z_e can obtain is $\Theta(\log^3 n)$ and we can always round it to multiples of e.g. $\Theta(\frac{1}{n})$ with negligible $o(1)$ effect on the collection's final load on each edge.

The famous random edge-sampling technique of Karger [Kar94a, Theorem 2.1] gives us that, if we randomly put each edge of the graph in one of η subgraphs H_1 to H_η, where η is such that $\frac{\lambda}{\eta} \geq \frac{10 \log n}{\varepsilon^2}$, then each subgraph has edge-connectivity in $[\frac{\lambda}{\eta}(1-\varepsilon), \frac{\lambda}{\eta}(1+\varepsilon)]$ with high probability. Having this, we first find a 3-approximation $\tilde{\lambda}$ of λ using the distributed minimum edge cut presented in the previous section in $O((D+\sqrt{n}\log^* n)\log^2 n \log\log n)$ rounds. Then, using $\tilde{\lambda}$, we choose η such that we are sure that $\frac{\lambda}{\eta} \in [\frac{20 \log n}{\varepsilon^2}, \frac{60 \log n}{\varepsilon^2}]$ and then put each edge in a random subgraph H_1 to H_η. This way, each subgraph has edge-connectivity in the range $[\frac{10 \log n}{\varepsilon^2}, \frac{100 \log n}{\varepsilon^2}]$, which as ε is a constant, fits the setting of Section 9.9.1. On the other hand, the summation of the edge-connectivities λ_1 to λ_η of subgraphs H_1 to H_η is at least $\lambda(1-\varepsilon)$.

The remaining problem is to solve the spanning tree packing problem in each subgraph H_i, all in parallel. Recall that the algorithm explained in Section 9.9.1 for the case of $O(\log n)$ edge connectivity, requires solving $O(\log^3 n)$ minimum spanning tree problems. Hence, if we solve the MSTs of different subgraphs naively with repetitive black-box usage of the MST algorithm of Kutten and Peleg [KP95], this would take $O((D+\sqrt{n}\log^* n)\lambda \log^2 n)$ rounds. To obtain the round complexity $\tilde{O}(D+\sqrt{n\lambda})$, instead of a simple black-box usage, we do a few simple modifications.

Lemma 9.9.3. *The fractional spanning tree packing of all subgraphs can be implemented simultaneously, all in $O((D+\sqrt{\frac{n\lambda}{\log n}}\log^* n)\log^3 n)$ rounds of the* E-CONGEST *model.*

Proof of Lemma 9.9.3. We first briefly review the the general approach of [KP95]. Their algorithm uses $O(d \log^*)$ rounds to get a d-dominating set T with size at most $O(\frac{n}{d})$ and a partition of the graph into clusters of radius at most d around each node of T, where also each of these clusters is spanned by a fragment of the minimum spanning tree. Thus, the part of the minimum spanning tree that is completely inside one fragment is already determined. It then remains to determine the MST edges between different fragments. This part is performed by a pipe-lined upcast of the inter-fragment edges on a breadth first search and it is shown that this upcast takes at most $O(D+\frac{n}{d})$ rounds, where $O(\frac{n}{d})$ is the number of the inter-fragment edges in the MST. At the end, $O(\frac{n}{d})$ inter-fragment edges are broadcast to all nodes. Choosing $d=\sqrt{n}$ then leads to time complexity of [KP95].

In our problem, we solve η MSTs of subgraphs H_1 to H_η in parallel. The first part of creating the local fragments of MST is done in each subgraph independently, as they are edge-disjoint, in $O(d \log^* n)$ rounds. However, we must not do the upcasts on the BFS trees of subgraphs H_1 to H_η as each of these subgraphs might have a large diameter. Instead, we perform all the upcasts on the same BFS tree of the whole graph. It is easy to see that we can pipe-line the inter-fragment edges of different MSTs so that they all arrive at the root of this BFS after at most $O(D+\eta\frac{n}{d})$ rounds. Choosing $d=\sqrt{n\eta}$ gives us that we can simultaneously run one iteration of the fractional spanning tree packing of each subgraph, all together in time $O(D+\sqrt{n\eta}\log^* n)$. Since we have at most $\Theta(\log^3 n)$ iterations in the fractional spanning tree packing, and as $\eta = \Theta(\frac{\lambda}{\log n})$, the total round complexity becomes

at most $O((D + \sqrt{\frac{n\lambda}{\log n}} \log^* n) \log^3 n)$. □

9.10 Lower Bounds

In this section, we present the distributed lower bounds on finding fractional dominating tree packings or fractional spanning tree packings with size approximately equal to connectivity. Formally, we give lower bounds on approximating the value of the vertex or edge connectivity of a graph. The lower bounds about tree packings are then obtained because given a (fractional) dominating tree or spanning tree packing of a certain size—which is promised to be an approximation of connectivity—all nodes can immediately obtain an approximation of vertex or edge connectivity.

The lower bound for approximating the edge connectivity of a graph in the E-CONGEST model was presented in Section 7.7.2. We just restate it here together with the implication on computing (fractional) spanning tree packings.

Theorem 9.10.1. *[Rephrasing of Theorem 7.7.4] For any $\alpha > 1$ and $\lambda \geq 1$, even for diameter $D = O\big(\frac{1}{\lambda \log n} \cdot \sqrt{\frac{n}{\alpha\lambda}}\big)$, distinguishing networks with edge connectivity at most λ from networks with edge connectivity at least $\alpha\lambda$ requires at least $\Omega\big(D + \frac{1}{\log n}\sqrt{\frac{n}{\alpha\lambda}}\big)$ rounds in the* E-CONGEST *model. The same lower bound applies to computing (fractional) spanning tree packings of size larger than $n/(\alpha\lambda)$, where λ is the edge connectivity of the network.*

For vertex connectivity, we even get the following stronger lower bound.

Theorem 9.10.2. *For any $\alpha > 1$ and $k \geq 4$, even in networks of diameter 3, distinguishing networks with vertex connectivity at most k from networks with vertex connectivity at least αk requires at least $\Omega\big(\sqrt{n/(\alpha k \log n)}\big)$ rounds in the* V-CONGEST *model. The same lower bound also applies to computing (fraction) dominating tree packings of size larger than $n/(k\alpha)$ or for finding a vertex cut of size at most $\min\Big\{\delta \cdot \sqrt{n/(\alpha k \log n)}, \alpha \cdot k\Big\}$, for some constant $\delta > 0$ and where k is the vertex connectivity of the network.*

In the remainder of the section, we prove Theorem 9.10.2. Both lower bounds (Theorems 9.10.1 and 9.10.2) are based on the approach used in [DHK+12]. However, since all the lower bounds in [DHK+12] are for the E-CONGEST model, in order to get the slightly stronger bound of Theorem 9.10.2, we need to adapt to the node capacitated V-CONGEST model.

The lower bound is proven by a reduction from the 2-party set disjointness problem. Assume that two players Alice and Bob get two sets X and Y as inputs. If the elements of sets are from a universe of size N, it is well known that determining whether X and Y are disjoint requires Alice and Bob to exchange $\Omega(N)$ bits [KS92a, Raz92]. This lower bound even holds if Alice and Bob are promised that $|X \cap Y| \leq 1$ [Raz92], it even holds for randomized protocols with constant error probability and also if Alice and Bob only have access to public randomness (i.e., to a common random source). Note that this immediately also implies an $\Omega(N)$ lower bound on the problem of finding $X \cap Y$, even if Alice and Bob

know that X and Y intersect in exactly one element. In fact, if Alice and Bob even need to exchange $\Omega(N)$ bits in order to solve the following problem. Alice is given a set X as her input and Bob is given a set Y as his input, with the promise that $|X \cap Y| = 1$. Alice needs to output a set $X' \subseteq X$ and Bob needs to output a set $Y' \subseteq Y$ such that $X \cap Y \subseteq X' \cup Y'$ and such that $|X' \cup Y'| \leq cN/\log_2 n$ for an appropriate constant $c > 0$. Given such sets X' and Y', Alice can just send X' to Bob using $|X'| \cdot \log_2 N \leq cN$ bits. For a sufficiently small constant $c > 0$, that is at most a constant fraction of the bits that are needed to find $X \cap Y$.

9.10.1 Lower Bound Construction

We next describe the construction of a family $\mathcal{G}$ of networks that we use for our reductions from the above variants of the set disjointness problem. Instead of directly defining $\mathcal{G}$, it is slightly easier to first introduce a construction $\mathcal{H}$ for weighted graphs. Eventually, nodes of weight $w \geq 1$ will be replaced by cliques of size 2 and edges are replaced by complete bipartite subgraphs. The weighted graph family $\mathcal{H}$ is based on two integer parameters $h \geq 2$ and $\ell \geq 1$ and a positive (integer) weight $w > 1$. The family contains a graph $H(X, Y) \in \mathcal{H}$ for every set $X \subseteq [h]$ and for every $Y \subseteq [h]$ (i.e., for every possible set disjointness input for sets over the universe $[h]$). The node set $V_H(X, Y)$ of $H(X, Y)$ is defined as

$$V_H(X, Y) := \{0, \ldots, h\} \times [2\ell] \cup \{a, b\} \cup V_X \cup V_Y,$$

where $V_X := \{u_x : x \in X\}$ and $V_Y := \{v_y : y \in Y\}$. Hence, $V_H(X, Y)$ contains a node (p, q) for every $q \in \{0, \ldots, h\}$ and every $p \in [2\ell]$, a node u_x for each $x \in X$, a node v_y for each $y \in Y$, and two additional nodes a and b. All the nodes (p, q) (for $(p, q) \in \{0, \ldots, h\} \times [2\ell]$) have weight w, all other nodes have weight 1. The edges of $H(X, Y)$ are defined as follows. First, the "heavy" nodes (p, q) are connected to $h+1$ disjoint paths by adding an edge between (p, q) and $(p, q + 1)$ for each $p \in \{0, \ldots, h\}$ and each $q \in \{1, \ldots, 2\ell - 1\}$. The nodes u_x and v_y are used to encode a set disjointness instance (X, Y) into the graph $H(X, Y)$. For every $x \in X$, node u_x is connected to node $(0, 1)$ (the first node of path 0) and to node $(x, 1)$ (the first node of path x). In addition, for all $x' \notin X$, node $(0, 1)$ is directly connected to node $(x', 1)$ (the first node of path x'). We proceed similarly with the nodes $v_y \in V_Y$. For every $y \in Y$, node u_y is connected to node $(0, 2\ell)$ (the last node of path 0) and to node $(y, 2\ell)$ (the last node of path y). In addition, for all $y' \notin Y$, node $(0, 2\ell)$ is directly connected to node $(y', 1)$ (the last node of path y'). Finally, we use the nodes a and b in order to get a graph with small diameter. The two nodes are connected by an edge and every other node of the graph is either connected to a or to b. Basically, the left half of the graph is connected to node a and the right half of the graph is connected to b. Formally, all nodes $u_x \in V_X$ and all nodes (p, q) for all $q \leq \ell$ are connected to node a. Symmetrically, all nodes $v_y \in V_Y$ and all nodes (p, q) for $q > \ell$ are connected to node b. An illustration of $H(X, Y)$ is given in Figure 9-3.

We first state an important structural property of graph $H(X, Y)$. In the following, the

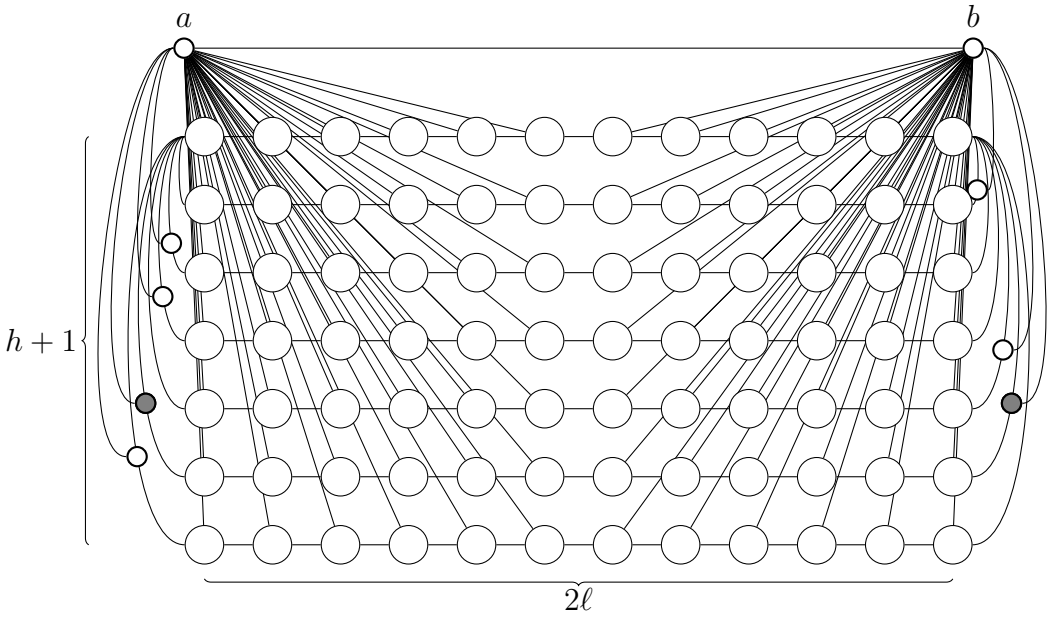


Figure 9-3: Lower bound construction: Nodes depicted by large circles have weight w (heavy nodes), nodes depicted by small circles have weight 1 (light nodes). The graph consists of $h+1$ paths, each consisting of 2ℓ heavy nodes ($h = \ell = 6$ in the example). Assuming that paths are numbered from 0 to h from top to down. Then, the left-most node on path 0 is directly connected to the left-most node of path x for every $x \notin X$. For $x \in X$, the left-most node of path 0 is connected to the left-most node of path x through an intermediate node of weight 1. The right-most nodes are connected in the same way by using the set Y. In the figure, we have $X = \{2, 3, 5, 6\}$ and $Y = \{1, 4, 5\}$. The node corresponding to element 5 in the intersection is marked in grey. In addition, nodes a and b are used to obtain a network with small diameter.

size of a vertex cut S of the weighted graph $H(X, Y)$ is the total weight of the nodes in S.

Lemma 9.10.3. *Consider the graph $H(X, Y)$ and assume that $|X \cap Y| \leq 1$. Then, if X and Y are disjoint, every vertex cut of graph $H(X, Y)$ contains a node of weight w (and thus has size at least w) and if $X \cap Y = \{z\}$ for some $z \in [h]$, the smallest vertex cut of $H(X, Y)$ has size 4 and it consists of the nodes a, b, u_z, and v_z. In addition, in the second case, every vertex cut of $H(X, Y)$ that does not contain a, b, u_z, and v_z contains a node of weight w. Further, the diameter of $H(X, Y)$ is at most 3.*

Proof. Let us first consider the case $X \cap Y = \emptyset$. In that case, for every $z \in [h]$, we either have $z \notin X$ or $z \notin Y$. If $z \notin X$, node $(0, 1)$ is directly connected to node $(z, 1)$, if $z \notin Y$, node $(0, 2\ell)$ is directly connected to node $(z, 2\ell)$. In both cases the path consisting of the nodes (z, p) for $p \in [2\ell]$ is directly connected to the top path. As this is the case for every $z \in [h]$, all $h + 1$ paths are directly connected to each other and therefore all the nodes of weight w induce a connected subgraph. As all other nodes are connected to some node of weight w, every vertex cut has to contain at least one node of weight w and thus, the claim

for the case where X and Y are disjoint follows.

For the case, where X and Y intersect in a single element z, let us consider the path consisting of the nodes (z, p) for $p \in [2\ell]$. All the nodes of the path are either connected to node a or to node b. In addition to this, only the first node $(z, 1)$ and the last node $(z, 2\ell)$ of the path are connected to additional nodes. As $z \in X$ and $z \in Y$, node $(z, 1)$ is connected to $(0, 1)$ through node u_z and node $(z, 2\ell)$ is connected to node $(0, 2\ell)$ through node v_z. Consequently, by removing nodes a, b, u_z, and v_z, path z (consisting of the nodes (z, p)) is disconnected from the rest of the graph. The four nodes therefore form a vertex cut of size 4.

Now, let us consider any other vertex cut $S \subseteq V_H(X, Y)$ that does not contain all of these four nodes. We want to show that S needs to contain at least one node of weight w. For contradiction, assume that S contains only nodes of weight 1. Because for every $z' \in [h] \setminus \{z\}$, $z' \notin X$ or $z' \notin Y$, the same argument as in the $X \cap Y = \emptyset$ case shows that every path $z' \in [h] \setminus \{z\}$ is directly connected to path 0. As by assumption also one of the nodes a, b, u_z, or v_z is not in S, also path z is still connected to the other paths. Again since all weight 1 nodes are directly connected to a weight w node, this implies that the nodes $V_H(X, Y) \setminus S$ induce a connected subgraph, a contradiction to the assumption that S contains only nodes of weight 1.

It remains to show that the diameter of $H(X, Y)$ is 3. This follows because every node is either directly connected to node a or to node b and there also is an edge between nodes a and b. □

We conclude the discussion on the lower bound construction by finally also introducing a family $\mathcal{G}$ of unweighted graphs. Given the three integer parameters h, ℓ, and w, there is a one-to-one correspondence between the graphs of $\mathcal{H}$ and $\mathcal{G}$. Also in $\mathcal{G}$, there is a graph $G(X, Y)$ for every possible set disjointness input $(X, Y) \in [h]^2$. Given $H(X, Y)$, $G(X, Y)$ is obtained by using the following transformation:

1. Each node of weight w in $H(X, Y)$ is replaced by a clique of size w.
2. Each edge of $H(X, Y)$ is replaced by a complete bipartite subgraph.[6]

Note that while graphs in $\mathcal{H}$ have $\Theta(h\ell)$ nodes, graphs in $\mathcal{G}$ have $\Theta(h\ell w)$ nodes. The statements of Lemma 9.10.3 hold in exactly the same way for graphs of $\mathcal{G}$.

Lemma 9.10.4. *Consider the graph $G(X, Y)$ and assume that $|X \cap Y| \leq 1$. Then, if X and Y are disjoint, every vertex cut of graph $G(X, Y)$ has size at least w and if $X \cap Y = \{z\}$ for some $z \in [h]$, the smallest vertex cut of $G(X, Y)$ has size 4 and it consists of the nodes a, b, u_z, and v_z. In addition, in the second case, every vertex cut of $G(X, Y)$ that does not contain a, b, u_z, and v_z has size at least w. Further, the diameter of $G(X, Y)$ is at most 3.*

[6] Hence, edges between two nodes of weight w are replaced by a subgraph isomorphic to $K_{w,w}$ and edges between a node of weight 1 and a node of weight w are replaced by a subgraph isomorphic to $K_{1,w}$.

Proof. Let $V(X,Y)$ be the set of nodes of $G(X,Y)$ and consider a vertex cut $S \subseteq V(X,Y)$ of $G(X,Y)$. Hence, removing the nodes of S disconnects the remainder of $G(X,Y)$ into at least 2 components. Let $A \subseteq V(X,Y)$ be the w nodes of a clique of size w corresponding to one of the weight w nodes in $H(X,Y)$ and assume that $|S \cap A| \in \{1,\dots,w-1\}$ (i.e., S contains some, but not all the nodes of A). We first observe that if S is a vertex cut, the set $S \setminus A$ is also a vertex cut. Because all edges of $H(X,Y)$ are replaced by complete bipartite subgraphs in $G(X,Y)$, a single node of A connects the same nodes to each other as all the nodes of A do. Given a vertex cut S of $G(X,Y)$, we can therefore always find a vertex cut $S' \subseteq S$ of $G(X,Y)$ such that S' contains either none or all the nodes of each of the cliques of size w corresponding to the weight w nodes of $H(X,Y)$. Let us call such a vertex cut S', a reduced vertex cut. Note that there is a one-to-one correspondence between the vertex cuts of $H(X,Y)$ and the reduced vertex cuts of $G(X,Y)$.

The first part of Lemma 9.10.3 therefore implies that if $X \cap Y = \emptyset$, every reduced vertex cut of $G(X,Y)$ contains at least one complete clique of size w and it therefore has size at least w. Hence, using the above observation, we also get that every vertex cut of $G(X,Y)$ has size at least w.

If X and Y intersect in a single element $z \in [h]$, Lemma 9.10.3 implies that nodes a, b, u_z, and v_z form a (reduced) vertex cut of size 4 (note that the four nodes all have weight 1 in $H(X,Y)$). Also, if a reduced vertex cut S of $G(X,Y)$ does not contain all the four nodes, Lemma 9.10.3 implies that contains at least one complete clique of size w and thus every vertex cut that does not contain all the four nodes has size at least w.

Finally, we get that graph $G(X,Y)$ has diameter 3 by using exactly the same argument as for $H(X,Y)$. □

9.10.2 Reduction

We next show how an efficient distributed algorithm to approximate the vertex connectivity or find a small vertex cut in networks of the family $\mathcal{G}$ can be used to get a two-party set disjointness protocol with low communication complexity. We first show that for $T < \ell$, any T-round distributed protocol on a graph $G(X,Y) \in \mathcal{G}$ can be simulated in a low communication public-coin two-party protocol by Alice and Bob, assuming that Alice knows the inputs of all except the right-most nodes of $G(X,Y)$ and Bob knows the inputs of all except the left-most nodes of $G(X,Y)$. Because only these nodes are used to encode the set disjointness instance (X,Y) into $G(X,Y)$, together with Lemma 9.10.4, this allows to derive a lower bound on the time to approximate the vertex connectivity or finding small vertex cuts. For convenience, we again first state the simulation result for graphs $H(X,Y) \in \mathcal{H}$. The proof of the following lemma is done in a similar way as the corresponding simulation in [DHK+12]. For all $r \in \{0,\dots,\ell-1\}$, we define set $V_A(r)$ and $V_B(r)$ as follows.

$$\begin{aligned} V_A(r) &:= \{a\} \cup V_X \cup \{(p,q) \in \{0,\dots,h\} \times [2\ell] : q < 2\ell - r\}, \\ V_B(r) &:= \{b\} \cup V_Y \cup \{(p,q) \in \{0,\dots,h\} \times [2\ell] : q > r+1\}. \end{aligned}$$

Lemma 9.10.5. *Let $T \leq \ell$ be an integer and let $\mathcal{A}$ be a T-round randomized distributed algorithm on graphs $H(X,Y) \in \mathcal{H}$. Assume that in each round, nodes a and b locally broadcast a message of at most B bits to their neighbors (other nodes are not restricted). Further, assume that Alice knows the initial states of nodes $V_A(0)$ and Bob knows the initial states of nodes $V_B(0)$. Then, Alice and Bob can simulate $\mathcal{A}$ using a randomized public-coin protocol such that:*

1. *At the end, Alice knows the states of nodes $V_A(T)$ and Bob knows the states of nodes $V_B(T)$*
2. *Alice and Bob need to exchange at most $2B \cdot T$ bits.*

Proof. First note that we can use the public randomness to model the randomness used by all the nodes of $H(X,Y)$. Hence, the random bits used by the nodes in the distributed protocol $\mathcal{A}$ is publicly known. We next describe a two-party protocol in which Alice and Bob simulate $\mathcal{A}$ in a round-by-round manner such that for all rounds $0 \leq r < \ell$, after simulating round r (or initially for $r = 0$),

(I) Alice knows the states of nodes in $V_A(r)$.

(II) Bob knows the states of nodes in $V_B(r)$.

(III) Alice and Bob have exchanged at most $2B \cdot r$ bits.

We prove (I), (II), and (III) by induction on r.

Induction Base For $r = 0$, statements (I)–(III) follow directly from the assumptions about the initial knowledge of Alice and Bob.

Induction Step For $r \geq 1$, assume that (I)–(III) hold for $r < r'$, where $r' \in \{0, \ldots, T-1\}$ so that we need to show that it also holds for $r = r'$. We need to show how Alice an Bob can simulate round r. In order for (III) to hold, Alice an Bob can exchange at most $2B$ bits for the simulation of round r. In order to satisfy (I), observe the following. We need to show that after the simulation of round r, Alice knows the states of all nodes in $V_A(r)$. By the induction hypothesis, we know that Alice knows the states of the nodes $V_A(r-1) \supset V_A(r)$ after round $r-1$. Hence, in addition, in order to be able to compute the states of the nodes $V_A(r)$ after round r, Alice needs to know all the messages that nodes in $V_A(r)$ receive in round r. She therefore needs to know all the messages that are sent by neighbors of nodes in $V_A(r)$ in round r. The set of neighbors of nodes in $V_A(r)$ consists of the nodes $V_A(r-1)$ and of node b. Note that in particular, because $T \leq \ell$, $V_A(r-1)$ also contains all the neighbors of node $a \in V_A(r)$. Except for node b, Alice thus knows the state of all neighbors of node in $V_A(r)$ at the beginning of round r and she therefore also knows the messages sent by these nodes in round r. In order complete her simulation of round r, she therefore only needs to

learn the message (of at most B bits) sent by node b in round r. By the induction hypothesis, Bob knows the content of this message and can send it to Alice. Similarly, Bob can also compute the states of all nodes in $V_B(r)$ at the end of round r if Alice sends the round r message of node a to Bob. This completes the proof of the induction step and thus also the proof of the lemma. □

An analogous lemma can also be shown for graphs $G(X,Y) \in \mathcal{G}$. Here, we define $V'_A(r)$ and $V'_B(r)$ to be the node sets corresponding to $V_A(r)$ and $V_B(r)$. That is, $V'_A(r)$ contains all weight 1 nodes of $V_A(r)$ and all the w nodes of each clique of size w corresponding to a weight w node in $V_A(r)$. The set $V'_B(r)$ is defined analogously. Based on the argument for $\mathcal{H}$, we then directly obtain the following statement for graphs in $\mathcal{G}$.

Lemma 9.10.6. *Let $T \leq \ell$ be an integer and let $\mathcal{A}$ be a T-round randomized distributed algorithm on graphs $G(X,Y) \in \mathcal{G}$. Assume that in each round, nodes a and b locally broadcast a message of at most B bits to their neighbors (other nodes are not restricted). Further, assume that Alice knows the initial states of nodes $V'_A(0)$ and Bob knows the initial states of nodes $V'_B(0)$. Then, Alice and Bob can simulate $\mathcal{A}$ using a randomized public-coin protocol such that:*

1. *At the end, Alice knows the states of nodes $V'_A(T)$ and Bob knows the states of nodes $V'_B(T)$*
2. *Alice and Bob need to exchange at most $2B \cdot T$ bits.*

Proof. The proof is done in the same way as for Lemma 9.10.5. □

We are now ready to prove the lower bound Theorem 9.10.2.

Proof of Theorem 9.10.2. Let us first assume that there is a randomized T-round V-CONGEST model protocol $\mathcal{A}$ that allows distinguish graphs of vertex connectivity at most k from graphs of vertex connectivity at least $k\alpha$. Alice and Bob can use protocol $\mathcal{A}$ to solve the set disjointness problem as follows. Assume that Alice and Bob are given inputs $X \subseteq [h]$ and $Y \subseteq [h]$ for some positive integer h with the promise that X and Y intersect in at most 1 value. We pick $\ell = h/\log n$ and $w = \alpha k + 1$ and we consider the graph $G(X,Y)$ with parameters h, ℓ, and w. Assume that $T < \ell$. Note that except for the very first cliques of each of the paths of $G(X,Y)$ and the very last cliques of each of the paths of $G(X,Y)$, the graph $G(X,Y)$ does not depend on X and Y. Hence, Alice knows the initial states of all nodes in $V'_A(0)$ and Bob knows the initial states of all nodes in $V'_B(0)$. Using Lemma 9.10.6, Alice and Bob can therefore simulate the T rounds of $\mathcal{A}$ by exchanging at most $2BT$ bits such that in the end for all nodes v of $G(X,Y)$, either Alice or Bob knows the final state of v. Alice and Bob therefore definitely learn the approximation of the vertex connectivity computed by $\mathcal{A}$. By Lemma 9.10.4, if $X \cap Y = \emptyset$, the vertex connectivity of $G(X,Y)$ is at least $w \geq \alpha k + 1$ and if $X \cap Y \neq \emptyset$, the vertex connectivity of $G(X,Y) = 4 \leq k$. An α-approximation of the vertex connectivity therefore allows Alice and Bob to solve the set

disjointness instance (X, Y). As by the set disjointness lower bound of [Raz92], solving set disjointness of sets from the universe $[h]$ requires Alice and Bob to exchange at least $\Omega(h)$ bits, we get that $2TB = \Omega(h)$ and thus $T = \Omega(h/B) = \Omega(h/\log n)$. Together with $n = \Theta(h\ell\alpha k)$, the claimed lower bound follows.

We directly also get a lower bound on computing a fractional dominating tree packing (or a fractional connected dominating set packing) of size at least k/α because the size of such a packing leads to the corresponding approximation of the vertex connectivity.

To prove the lower bound on finding small vertex cuts, we consider instances (X, Y) for which $|X \cap Y| = 1$. Let the element in the intersection $X \cap Y$ be z. Note that by Lemma 9.10.4, in that case the vertex connectivity of $G(X, Y)$ is 4 and every vertex cut of size at most $\alpha k < w$ needs to contain the nodes u_z, v_z, a, and b. Hence, an algorithm that outputs a vertex cut of size $s \leq \min\left\{\delta\sqrt{n/(\alpha k \log n)}, \alpha k\right\}$ has to output a node set S of size s such that in particular $u_z, v_z \in S$. Since S contains at most $s-2$ other nodes $u_x \in V_X$ or $v_y \in V_Y$, the same reduction as above allows Alice and Bob to output a set of at most $s-1$ elements from $[h]$ such that z is contained in this set. For a sufficiently small constant $\delta > 0$, we have seen that for this, Alice and Bob also need to exchange at least $\Omega(h)$ bits. □

Part IV

Congestion—Scheduling Distributed Protocols

Chapter 10

Scheduling Distributed Protocols

10.1 Introduction & Related Work

10.1.1 General Motivation and Background

Computer networks are constantly running many applications at the same time and because of the bandwidth limitations, each application gets slowed down due to the activities of the others. Despite that, for the vast majority of the distributed protocols introduced in theoretical distributed computing, the initial design and analysis have been carried out with the assumption that each protocol uses the network alone. In this chapter, we investigate the issue of what happens when many distributed protocols are to be run together.

Specifically, we study the questions of *how to run these distributed protocols simultaneously as fast as possible* and *what are the limitations on how fast that can be done*. While being arguably a fundamental issue, to the best of our knowledge, these questions have not been investigated in their full generality. However, we describe a number of special cases that have been studied in the past. Let us first make these questions somewhat more concrete by fixing the model and the problem statement.

Model Throughout this chapter, we work with the CONGEST model [Pel00], as explained in Chapter 2. Recall that CONGEST is a standard distributed model that takes bandwidth limitations into account. The communication network is represented by an undirected graph $G = (V, E)$ where $|V| = n$. Communications occur in synchronous rounds and in each round each node can send one B-bit message to each of its neighbors, where $B = O(\log n)$. In this context, we use the term *protocol* to refer to any distributed algorithm in the CONGEST model. For instance, a distributed algorithm that computes a Breadth First Search tree (rooted in a given node) is a protocol.

Scheduling Distributed Protocols In an informal sense, the general scenario that we consider is as follows: We want to run many independent distributed protocols $\mathcal{P}_1$, $\mathcal{P}_2$, ...,

$\mathcal{P}_k$ together. However, we do not know what problem is being solved by each protocol. Hence, we must run each protocol essentially in a black-box manner without altering the content of its messages, except for potentially adding a small amount of auxiliary information to each message.

As mentioned before, we are not aware of a prior work that studied this problem of scheduling distributed protocols in its full generality. However, we now mention some prior work that can be viewed as studying (very) special cases of this scheduling problem:

(I) The first example is broadcasting k messages from different sources, each to the h-hop neighborhood of its source. We can naturally view this as k simultaneous problems, each of which requires sending a given message from its source to the h-hop neighborhood of the source. Classical analysis [Top85] shows that the natural method in which at each round each node sends one message that it has not sent before (and has traveled less than h hops so far) solves the problem in $O(k+h)$ rounds. The significant aspect in this bound is the additive appearance of k. This in a sense implies a perfect pipelining between the k broadcast protocols.

(II) The second example is running breadth-first search protocols from different sources. Holzer and Wattenhofer [HW12] show that one can run n BFSs starting in different nodes all together in $O(n)$ rounds. This is done by delaying BFSs carefully in a way that they do not interfere (once started). More generally, Lenzen and Peleg [LP13] show that k many h-hop BFSs from different sources can be performed in $O(k+h)$ rounds. This is in a sense a strengthening of the broadcast result of [Top85], as it shows that in fact each BFS-token (or equivalently, broadcast message) will be delivered to each related destination along a shortest path.

(III) The third example is routing many packets, each from a source to a destination, along a given path. This problem has received the most attention [LMR94, Rot13, Sch98, LMR99, BG99, Wie11, RT96, OR97, AdHV99, PW11] among these special cases. Viewing our distributed protocol scheduling problem as a generalization of this packet routing problem, we adopt a terminology close to the terminology introduced for packet routing in [LMR94]. We discuss some known results for packet routing after introducing this (generalized) terminology.

10.1.2 A Zoomed-In View of Scheduling and Our Terminology

Recall from above that we are primarily interested in running distributed protocols $\mathcal{P}_1$, $\mathcal{P}_2$, ..., $\mathcal{P}_k$ concurrently. Having seen the above examples, we are now ready to investigate this in more depth. In particular, we now discuss simple lower bounds on the *schedule length, which informally means how long it takes to run all the protocols together*[1]. In the course of this discussion, we fix some terminology, which is an immediate generalization of

[1] See Section 10.2.1 for the formal definitions.

some terminology of Leighton, Maggs, and Rao [LMR94], and will be in use throughout the chapter. We note that the definitions presented here are informal and are presented mainly for introductory purposes. See Section 10.2.1 for the formal definitions.

Dilation: If one of the protocols $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ that are to be run together takes d rounds, running all of them together will clearly require at least d rounds. We refer to the maximum running time of the protocols as $\mathsf{dilation}$. Note that $\mathsf{dilation}$ is thus a trivial lower bound on the length of schedule that runs all the protocols.

Congestion: There is another lower bound due to the bandwidth limitations. For each edge e, let $c_i(e)$ be the number of rounds in which protocol $\mathcal{P}_i$ sends a message over e. Then, running all the protocols together requires at least $\mathsf{congestion} = \max_{e \in E} \mathsf{congestion}(e)$ rounds where $\mathsf{congestion}(e) = \sum_{i=1}^{k} c_i(e)$. Thus, $\mathsf{congestion}$ is another trivial lower bound on the length of schedule that runs all the protocols.

From the above discussions, we can conclude that

Observation 10.1.1. *Any schedule for running protocols $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ together requires at least* $\max\{\mathsf{congestion}, \mathsf{dilation}\} \geq (\mathsf{congestion} + \mathsf{dilation})/2$ *rounds.*

In light of this simple observation, the key question of interest is:

> **Question:** Can we always find a schedule that has a length close to the aforementioned $\Omega(\mathsf{congestion} + \mathsf{dilation})$ lower bound?

A simple yet quite powerful technique that proves helpful in this regard is *random delays*. This technique was first introduced by Leighton, Maggs, and Rao [LMR94] in the context of the *packet routing* problem. In packet routing, $\mathsf{dilation}$ is the length of longest path and $\mathsf{congestion}$ is the maximum number of paths that go through an edge. The random delays method achieves an upper bound within an $O(\log n)$ factor of the lower bound; more precisely, a schedule of size $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$. Improving this simple $O(\log n)$-approximation to an $O(1)$-approximation received quite an extensive amount of attention and by now there are many existence proofs and also algorithmic constructions that give schedules of length $O(\mathsf{congestion} + \mathsf{dilation})$ for packet routing. That is, schedules within an $O(1)$ factor from the trivial lower bound. See for instance [LMR94, Rot13, Sch98, LMR99, BG99, Wie11, RT96, OR97]. The classical method [LMR94] is based on $\log^* n$ levels of recursively applying the Lovasz's Local Lemma[2], each time reducing the parameter $\mathsf{congestion} + \mathsf{dilation}$ of the new problem to a polylogarithmic function of the same parameter in the problem of the previous level.

Going back to the question of scheduling general distributed protocols, the random delays technique proves useful here as well. If nodes have access to shared randomness, the same simple random delays technique as in [LMR94] provides a schedule for general distributed

[2] In fact, the packet routing problem and this LLL-based method of it are typically covered in courses on randomized algorithms for introducing the Lovasz's Local Lemma, see for instance [Sin, Raja, Rajb, Kar].

protocols that is within an $O(\log n)$ factor of the trivial lower bound. Here the shared randomness is helpful because for each distributed protocol, there can be many (potentially distant) nodes that start it and delaying the protocol by a (controlled) random delay requires all nodes to do so in a consistent manner.

Theorem 10.1.2 (Informal[3], and Extension of [LMR94]). *Given shared randomness, one can distributedly run all the distributed protocols in* $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ *round, with high probability.*

Proof Sketch. We note that the description provided here is meant to just provide an informal sketch of the approach. See the proof of Theorem 10.4.1 for a formal proof of existence of the schedule.

We break time into phases, each consisting of $\Theta(\log n)$ consecutive rounds. Then, we treat each phase as one round, meaning that we make each distributed protocol perform its communications at the speed one protocol-round per phase. We delay the start of each protocol by a uniform random delay in $[O(\mathsf{congestion}/\log n)]$ phases. Then, for each edge e and each phase, each particular message that goes through e in one of the protocols has probability at most $O(\log n/\mathsf{congestion})$ to be scheduled to traverse e in this particular phase. Thus, the expected number of messages that are scheduled to traverse each edge in a given phase is $O(\log n)$. Then, a Chernoff bound[4] shows that with high probability, for each edge and each phase, $O(\log n)$ messages are scheduled to traverse this edge in this phase. Since the phase has $\Theta(\log n)$ rounds, there is enough time to deliver all the scheduled messages. Hence, with high probability, all protocols run concurrently and each will is done after $O(\mathsf{congestion}/\log n) + \mathsf{dilation}$ phases, that is, $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds. □

The above theorem leaves us with two main questions:

Questions:

1. Can one remove the $\log n$ factor and get an $O(\mathsf{congestion} + \mathsf{dilation})$ round schedule for general distributed protocols, perhaps using ideas similar to [LMR94, Rot13, Sch98, LMR99, BG99, Wie11, RT96, OR97]?
2. What can we do with only private randomness, that is, without assuming that nodes have access to shared randomness?

10.1.3 Our Technical Contributions

We provide two results, which provide answers for the above two questions, respectively.

[3] See Theorem 10.4.1 for a formalized existential version.

[4] Although we do not have independence between these random events, we can apply the Chernoff bound as we have negative independence. See the proof of Theorem 10.4.1.

First Result Regarding the first question, we show that interestingly, and unlike in packet routing, when scheduling general distributed protocols, the lower bound can be improved to essentially match the simple upper bound. Concretely, using the probabilistic method [AS04], we prove the existence of a hard instance of the scheduling problem which shows that:

Theorem 10.1.3 (Informal). *There are instances of the distributed protocol scheduling problem for which any schedule needs* $\Omega(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log\log n)$ *rounds.*

Second Result As for the second question, that is shared randomness, we face two rather orthogonal issues: One is about the amount of randomness to be shared and the other is how (and where) to share randomness. We note that the second question is somewhat orthogonal to the first question and it is relevant even if we want to share just one bit. While the first issue is simple, the second is deeper and requires considerably more work. For the first issue, the saving grace is that for the proof of Theorem 10.1.2, $\Theta(\log n)$-wise independence between the values of random delays is enough[5] and thus, thanks to the standard bounded-independence randomness constructions (e.g., via Reed-Solomon codes), sharing simply $O(\log^2 n)$ random bits is sufficient. As for the second issue, clearly one can elect a leader to pick the required "shared" randomness and broadcast it to all nodes. However, this, and moreover any such global sharing procedure, will need at least $\Omega(D)$ rounds, for D being the network diameter, which is not desirable. We explain how to solve the problem with private randomness in a time close to the case with shared randomness.

Theorem 10.1.4 (Informal). *There is a distributed algorithm that uses only private randomness and for any instance of the distributed protocol scheduling problem, computes a schedule of length* $O(\mathsf{congestion}+\mathsf{dilation}\cdot\log n)$*, in* $O(\mathsf{congestion}+\mathsf{dilation}\log^2 n)$ *rounds, with high probability.*

Roughly speaking, the approach is to break each protocol into sparsely overlapping sub-protocols that each span only small areas of the network. In particular, each of these sub-protocols will span an area of (weak) diameter $O(\mathsf{dilation}\log n)$ hops. Then, we share random bits only inside these smaller areas and show how to run these sub-protocols in a way that they simulate the main protocols.

We also explain that our approach is in fact more general and it can be used to remove the assumption of having shared randomness in a broad family of randomized distributed protocols, at the cost of an $O(\log^2 n)$ factor increase in their running time. Roughly speaking, the family that this result applies to is those protocols in which each node outputs one (canonical) output in the majority of the executions of the protocol (with the given input), that is, protocols where the randomness does not affect the output (significantly) and is used

[5] See [SSS95, Theorem 5] for a Chernoff bound for k-wise independent random variables.

only to speed up the computation. We note that recently this class was termed *Bellagio algorithms* [Gol12, GG11] as a subclass of randomized algorithms with some pseudo-deterministic behavior. This generalization is presented in Section 10.5.

10.2 Preliminaries

In this section, we present the notions and the definitions that we use throughout this chapter. In particular, in Section 10.2.1, we define the concept of *communication pattern* and describe various related notions, especially the *simulations* of *communication patterns*. These are key elements in formalizing our study of the scheduling problem. Then, in Section 10.2.2, we present the formal definition of the scheduling problem, in the context of the definitions provided in Section 10.2.1.

10.2.1 Communication Patterns and their Simulations

Time-Expanded Graph Given a network graph $G = (V, E)$, we define its T-round *time-expanded* graph $G \times [T]$ as follows: We have $T+1$ copies of V, denoted by $V_0, V_1, V_2, \ldots, V_T$, and T copies of E, denoted $E_1, E_2, \ldots, E_T$, where E_i defines the set of edges between V_{i-1} and V_i. We refer to the copy of node $v \in V$ that is in V_i as v_i. For each $i \in [1, T]$, $v_i \in V_i$ is connected with a direct edge to $u_{i+1} \in V_{i+1}$, that is $(v_i, u_{i+1}) \in E_{i+1}$, if and only if $(v, u) \in E$. We use the notations $V(G \times [T])$ and $E(G \times [T])$ to refer to, $V_0 \cup V_1 \cup V_2 \cdots \cup V_T$ and $E_1 \cup E_2 \cup \cdots \cup E_T$, respectively.

Communication Pattern To talk about running many distributed protocols simultaneously, we focus on the *communication patterns* [Lyn96, Page 143] of the protocols. In an informal sense, the communication pattern of (one execution of) a protocol records in which rounds and over which edges messages get sent in that (execution of the) protocol. Formally, the definition is as follows.

Fix an execution α of a protocol $\mathcal{P}$ and suppose that it has T rounds. The communications of this execution on network G correspond naturally to a subset of edges $P \subseteq E(G \times [T])$, in the following manner: For $v \in V_i$ and $u \in V_{i+1}$, we include the edge (v_i, u_{i+1}) in the subset P if and only if in the execution α, node v sends a message to node u in round $i+1$. We call this subset P the *communication pattern* of α. Note that a communication pattern does not represent the content of the messages. We refer to T as the *length* of this communication pattern, and use the notation $length(P) = T$ to refer to it. We also sometimes say that P is a T-round communication pattern. Figure 10-1 shows an example.

As explained above, the communication pattern is defined for one execution α of a protocol $\mathcal{P}$. However, each protocol might have many executions and thus also many communication patterns. These executions and communication patterns are determined as a function of the inputs to the nodes and also their randomness. Our goal is to address running a

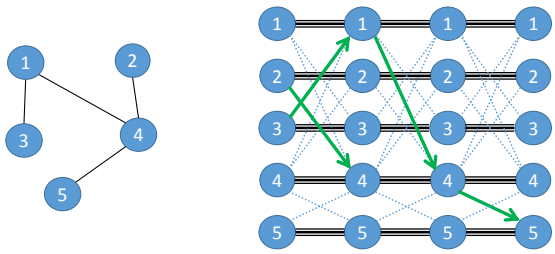


Figure 10-1: A simple graph, its 3 round time-expanded version, and a communication pattern on it. In round 1, nodes 2 and 3 send messages respectively to nodes 4 and 1; in round 2, node 1 sends a message to node 4; in round 3, node 4 sends a message to node 5.

collection of protocols $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ concurrently. To formalize this, we would like to have one communication pattern for each protocol $\mathcal{P}_i$. For that, we fix one execution for each protocol $\mathcal{P}_i$ in this collection, by fixing two things at the beginning of the execution: (1) the input to the nodes, (2) the random bits used by the nodes throughout the execution of $\mathcal{P}_i$. Once these two are fixed, the execution of the protocol $\mathcal{P}_i$ is uniquely determined. This execution determines also a unique communication pattern P_i for protocol $\mathcal{P}_i$. Thus, we now have a collection of fixed communication patterns $P_1, P_2, \ldots, P_k$, corresponding respectively to protocols $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$. In the following, our focus will be on this collection of communication patterns $P_1, P_2, \ldots, P_k$.

Congestion and Dilation for a Collection of Communication Patterns Given a collection of k communication patterns P_1 to P_k, the dilation of this collection is the maximum length among P_1 to P_k. That is, $\mathsf{dilation} = \max_{i \in [1,k]} length(P_i)$. Moreover, the congestion of this collection is the maximum over network edge e of the total number of messages that are sent through edge e over all P_1 to P_k. More precisely, for each edge e, let $c_i(e)$ be the number of messages sent over e in communication pattern P_i. Then, we define $\mathsf{congestion} = \max_{e \in E} \mathsf{congestion}(e)$ where $\mathsf{congestion}(e) = \sum_{i=1}^{k} c_i(e)$.

Causal Precedence in a Communication Pattern We first define *causal precedence* between nodes of $G \times [T]$ with respect to a given communication pattern $P \subseteq E(G \times [T])$. Consider the following two basic *precedence* relations: (1) for each $i \in [0, T-1]$, each node $v_i \in V_i$ causally precedes $v_{i+1} \in V_{i+1}$, and (2) node $v_i \in V_i$ causally precedes node $u_{i+1} \in V_{i+1}$ if $(v_i, u_{i+1}) \in P$. We define *causal precedence* by taking the reflexive transitive closure of these basic precedence relations. In other words, a node $v_i \in V_i$ *causally precedes* $u_j \in V_j$ if $v_i = u_j$ or if there is a sequence of basic precedence relations which start at v_i and end in u_j. Moreover, we say edge (v_i, u_{i+1}) *causally precedes* edge (v'_j, u'_{j+1}) if u_{i+1} causally precedes v'_j.

Local Equivalence of Two Communication Patterns We say two communication patterns $P_1, P_2 \subseteq E(G \times [T])$ are *locally equivalent for a node* $u_i \in V_i$, where $i \in [0, T]$, if the two sets of the edges that causally precede u_i in P_1 and P_2 are equal. Moreover, if P_1 and

P_2 are locally equivalent for u_T, we also say that P_1 and P_2 are *locally equivalent* for node $u \in G$.

Simulation of One Communication Pattern If $P \subseteq E(G \times [T])$ is a T-round communication pattern and $P' \subseteq E(G \times [T'])$ is a T'-round communication pattern for $T' \geq T$, then we define a *simulation* of P by P' to be a bijective mapping f from P to P' that preserves pairwise causal precedences. More precisely, f maps each edge (v_i, u_{i+1}) of communication pattern P to an edge $f((v_i, u_{i+1}))$ in the communication pattern P' in a way that preserves the causal precedence: that is, if edge (v_i, u_{i+1}) causally precedes edge (v'_j, u'_{j+1}) in P, then edge $f((v_i, u_{i+1}))$ causally precedes edge $f((v'_j, u'_{j+1}))$ in P'. We use the notation $P \underset{\mathsf{sim}}{\leq} P'$ to denote that there exists a simulation of P by P', we also sometimes say that P' *simulates* P if such a simulation exists.

Local Simulation of One Communication Pattern for a Node For a communication pattern P, we say that a communication pattern P' *locally simulates* P *for a node* $v \in G$ if there exists a communication pattern P'' such that $P \underset{\mathsf{sim}}{\leq} P''$ and P' and P'' are locally equivalent for node v. If that case, we also write $P \underset{\mathsf{sim@}v}{\leq} P'$.

Simulation of Many Communication Patterns For a collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns where each $P_i \subseteq E(G \times [T])$, we define a *simultaneous simulation* of P—or simply just a *simulation* of it—by a communication pattern $P' \subseteq E(G \times [T'])$ to be a set $\mathsf{F} = \{f_1, f_2, \ldots, f_k\}$ of mappings, where each f_i is a simulation from P_i to some communication pattern $P'_i \subseteq P'$ such that $P'_1, P'_2, \ldots, P'_k$ are mutually disjoint and $P' = \cup_i P'_i$.

Local Simulation of Many Communication Patterns for a Node For a collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns where each $P_i \subseteq E(G \times [T])$, we say that communication pattern $P' \subseteq E(G \times [T'])$ *locally simulates* P for node v if there is a partition of P' into disjoint communication patterns P'_1, P'_2, ..., P'_k such that for each i, we have $P_i \underset{\mathsf{sim@}v}{\leq} P'_i$.

10.2.2 The Distributed Protocol Scheduling Problem

Recall that our objective is to investigate running a collection $\mathcal{P}_1$ to $\mathcal{P}_k$ of protocols concurrently. To formalize this, we consider communication pattens P_1 to P_k, one for each of the protocols. Each communication pattern P_i is determined by fixing the input and the randomness in all nodes in the protocol $\mathcal{P}_i$, which defines one unique execution α_i for that protocol $\mathcal{P}_i$, and then letting P_i be the communication pattern of α_i. Our main goal is for each node v to learn its output in execution α_i for each protocol $\mathcal{P}_i$. An ideal formal specification of this would be to construct an execution such that its communication pattern is a simultaneous simulation of P_1 to P_k. However, this goal turns out to be unnecessarily

restrictive and it is unclear to us if it can be performed efficiently. Since our ultimate objective is that each node learns its output in execution α_i, for each protocol $\mathcal{P}_i$, it suffices for us to seek *local simulations* for each of the nodes. In this regard, we define the scheduling problem using *local simulations*, as follows.

Scheduling for Many Communication Patterns For a collection of communication patterns $P_1, P_2, \ldots, P_k \subseteq E(G \times [T])$, we call a communication pattern $P' \subseteq E(G \times [T'])$ a *scheduling* of $P_1, P_2, \ldots, P_k$ if and only if the following condition is satisfied: for each node $v \in G$, communication pattern P' contains a simultaneous local simulation of $P_1, P_2, \ldots, P_k$. We define the length of this *schedule* to be T', and we sometimes say that this is a T'-*round scheduling of* $P_1, P_2, \ldots, P_k$. We emphasize that the definition of the scheduling does not restrict how the local simulations of each communication pattern P_i in different nodes $v, v' \in G$ relate with each other. We note that this will not cause any problem, because the local simulations of P_i in different nodes enable them to compute outputs that are the same as the outputs that would be generated with the complete communication pattern P_i.

The Distributed Protocol Scheduling (DPS) Problem Given a collection $\mathcal{P}_1$ to $\mathcal{P}_k$ of protocols, and their fixed communication patterns P_1 to P_k, one for each, we define the problem of *scheduling distributed protocols* to be constructing an execution such that the corresponding communication pattern is a scheduling of communication patterns P_1 to P_k.

We next discuss a number of remarks about our results on the DPS problem.

Remarks About Our Lower Bounds Our lower bound, which is explained in Section 10.3, is existential and it applies even if the scheduling problem is solved using centralized computation. This means that everything—particularly the whole network topology and also the communication patterns of all the protocols—is known to the centralized scheduler, and the lower bound simply says that *there is no "short" schedule*. Notice that this is much stronger than saying that *a "short" schedule cannot be computed (distributedly)*.

Remarks About Our Upper Bounds For the purpose of our algorithm that schedules distributed protocols (formally their corresponding fixed communication patterns), and which is presented in Section 10.4, we assume that the distributed protocols $\mathcal{P}_1$ to $\mathcal{P}_k$ are given in the following format: For each protocol $\mathcal{P}_i$, when this protocol is run alone, in each round each node knows what to send in the next round. This clearly depends on the node's input and also what messages the node has received up to, and including, that round.

Moreover, we remark that protocols $\mathcal{P}_1$ to $\mathcal{P}_k$ might be randomized. To make the task of the simulation precise, as mentioned before, we consider the randomness used by each protocol $\mathcal{P}_i$ as a part of the input to the node. That is, at the start of the execution, each node samples all of its bits of randomness, and thus fixes them. For the remainder of the execution of the protocol $\mathcal{P}_i$, the node uses only these fixed bits, similarly to the way it uses

its fixed input. Hence, aside from using this input string of random bits, the protocol $\mathcal{P}_i$ behaves as a deterministic algorithm and has a unique execution α_i, as mentioned before, and thus also a unique communication pattern P_i. Even if we repeat the protocol many times, which we do in our algorithms, the protocol will use the same fixed string of random bits and the same inputs.

We emphasize that we cannot assume that the communication pattern P_i is known to the nodes before the start of the execution. This is because, often, the communication pattern conveys information about the things that the protocol is supposed to compute. Therefore, nodes cannot know this communication pattern before running the protocol. Even throughout the execution, a node v might not know which of its neighbors will send a message to v in the next round. A simple example to stress the significance of this issue is the case of Breadth First Search computation: before running the BFS protocol, node v does not know at which round and from which neighbors it will receive a message[6] during the execution of the protocol. In fact, the first message that it receives and the round in which it receives that first message determine the node's parent in the BFS tree and its distance to the root, respectively, and these are exactly what the protocol is designed to compute. Because of these, we do not assume that the nodes know the correct communication pattern a priori. In fact, if for some reason the schedule is mixed up and the node does not receive one of the messages that it is supposed to receive in the execution of protocol $\mathcal{P}_i$, the node might not realize this and it can proceed with executing the protocol, although generating a wrong execution. Our algorithms ensure that this does not happen.

Finally, we note that for our algorithmic result, we assume that nodes know upper bounds on congestion and dilation which are within a constant factor of them.

10.3 Lower Bound

In this section, we prove Theorem 10.1.3 (restated below). The main take-home message of this result is that the $\log n$ factor in Theorem 10.1.2—formalized in Theorem 10.1.4—is essentially unavoidable. That is, in a sharp contrast to the packet routing result of Leighton, Maggs, and Rao [LMR94], general distributed protocols do not always admit $O(\text{congestion} + \text{dilation})$ round schedules. Concretely, the theorem statement is as follows.

Theorem 10.1.3 *There is an instance of the distributed protocol scheduling problem for which any schedule needs* $\Omega(\text{congestion} + \text{dilation} \cdot \log n / \log\log n)$ *rounds.*

To prove this theorem about scheduling, we prove Theorem 10.3.1 which shows the existence of a collection of communication patterns that cannot be simulated by any short communication pattern. At the end of the section, in the proof of Theorem 10.1.3, we explain how Theorem 10.3.1 proves Theorem 10.1.3.

[6] Clearly one can add dummy messages to BFS to fix its communication pattern. However, this will increase the load on edges (i.e., congestion), which is undesirable.

Theorem 10.3.1. *There exists a collection* $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ *of communication patterns such that any communication pattern* P' *that simultaneously simulates* P *must have* $\mathit{length}(P') = \Omega(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log\log n)$.

Proof Outline From A Bird's-Eye View: We use the *probabilistic method* [AS04] to show the existence of such a "*hard*" problem instance. In particular, we present a probability distribution for a collection of communication patterns and show that, with a nonzero probability (and in fact with probability almost 1), for a random collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ taken from this distribution, no communication pattern P' of length $o(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log\log n)$ exists that simultaneously simulates P. For that, we study each fixed communication pattern P' of length $o(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log\log n)$ against a random collection P, and we show that this one communication pattern[7] P' has an extremely small probability to simulate the sampled collection P. This probability is so small that, even after we take a union bound over all possible communication patterns P', the probability that one of them simulates the sampled collection of communication patterns is strictly less than 1, and in fact close to 0. Hence, we conclude that there exists a collection P of communication patterns which does not admit any $o(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log\log n)$ round communication pattern P' that simulates them.

The Probability Distribution of the Communication Patterns: We start by describing the probability distribution of the communication patterns P_1 to P_k. We set $k = n^{0.2}$. The network $G = (V, E)$ is as follows: $V = \{v^0, v^1, \ldots, v^L\} \cup U^1 \cup U^2 \cup \cdots \cup U^L$, where $L = n^{0.1}$. Each set U^i contains $\eta = n^{0.9}$ nodes, and each node $u \in U^i$ is connected to v^{i-1} and to v^i. Figure 10-2 illustrates the structure of this network. We refer to the connections from v^{i-1} to U^i and those from U^i to v_i as the *connections of layer* i, or simply *layer* i.

The general format of each communication pattern P_i is as follows: in round 1, node v^0 sends a message to a subset $S^1 \subset U^1$. The choice of S^1 and more generally S^i will be described shortly. In round 2, each of the nodes in S^1 sends a message to v^1. In round 3, node v^1 sends a message to a subset $S^2 \subset U^2$. In round 4, each of the nodes in S^2 sends a message to v^2. The communication pattern proceeds similarly over the next layers, continuing with a speed of one hop progress per round.

In a random collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns, for each communication pattern P_i, each set S^j is determined randomly where each node $u \in U^j$ is added to S^j with probability $n^{-0.1}$, and the choices are independent between different nodes of the same layer, different layers, and different communication patterns. Figure 10-2 shows an example.

[7] Actually, this sentence and the two after it are not accurate. Instead of studying each fixed simulating communication pattern P', as we will see shortly, we will study a closely related structure, which we call *crossing pattern*. For now, before defining the concept of *crossing patterns*—which is somewhat detailed—it is convenient to imagine that the argument investigates each simulating communication pattern P' directly.

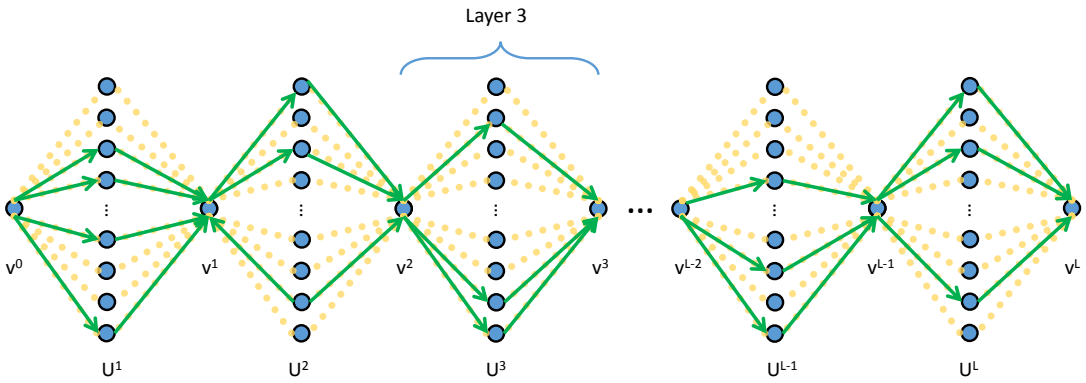


Figure 10-2: The figure depicts the network in the hard distribution and it also shows the communications of a sample communication pattern. We note that this is the base graph G and not its time-expansion, and thus, the communications are indicated on the base graph rather than on the time-expanded graph, as we did for illustrating communication patterns in Section 10.2.1. The orange dotted lines present all the edges of the base network. The green arrows indicate all the edges used by one sampled communication pattern. In particular, a green arrow from a node v^i to a node $u \in U^{i+1}$ indicates that a message is sent from node v^i to node u in round $2i+1$, and similarly, a green arrow from a node $u \in U^i$ to a node v^{i+1} indicates a message sent from node u to node v^{i+1} in round $2i$. In this illustration, the communications move through the network synchronously from left to right, that is from v^0 towards v^L, at a speed of one hop per round.

Having this probability distribution, we are now ready to start proving Theorem 10.3.1. The proof is composed of two lemmas, Lemma 10.3.2 and Lemma 10.3.3, which establish two properties for a random collection P of communication patterns in the distribution mentioned above:

(A) Lemma 10.3.2 shows that a random collection P has $\mathsf{congestion} = O(n^{0.1})$ and $\mathsf{dilation} = 2n^{0.1}$, with high probability.

(B) Lemma 10.3.3 shows that for a random collection P, with high probability, there is no short communication pattern P' that simultaneously simulates collection P. We call a communication pattern P' *short* if $\mathit{length}(P') \leq \frac{n^{0.1} \log n}{1000 \log \log n}$.

Proving property (B) will be the main technical part of the proof. At the end, we put the two lemmas, Lemma 10.3.2 and Lemma 10.3.3, together to prove Theorem 10.3.1.

Lemma 10.3.2. *For a randomly sampled collection* $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ *of communication patterns in the distribution mentioned above, with high probability, we have* $\mathsf{dilation} = 2n^{0.1}$ *and* $\mathsf{congestion} = O(n^{0.1})$.

Proof. Notice that in any instance of the distributed scheduling problem in the above setting, we have $\mathsf{dilation} = 2n^{0.1}$. We now argue that we also have $\mathsf{congestion} = O(n^{0.1})$, with high probability. Since we have $k = n^{0.2}$ communication patterns, $\mathbb{E}[\mathsf{congestion}] = kn^{-0.1} =$

$n^{0.2}n^{-0.1} = n^{0.1}$. Because of the independence between the communication patterns, we know that $\Pr[\mathsf{congestion} \geq 2n^{-0.1}] \leq e^{-\Theta(n^{0.1})}$. □

Lemma 10.3.3. *For a random collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns in the distribution mentioned above, with high probability, there is no short communication pattern P' that simultaneously simulates collection P.*

As mentioned before, proving Lemma 10.3.3 is the most technical part of the result. We next present our plan for proving Lemma 10.3.3. In particular, we specify the steps required for the proof. Then, we present the lemmas formalizing each of the steps of this plan, and finally we conclude with putting these lemmas together and presenting the proof of Lemma 10.3.3.

Proof Plan for Lemma 10.3.3. Our plan for proving the lemma is as follows. We want to prove that for a randomly sampled collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns, there is no short simulating communication pattern P', with high probability. For that, we first define a concept of *crossing pattern*. This definition is such that the existence of a short communication pattern P' that simulates collection P implies the existence of a crossing pattern with certain conditions (as formalized in Claim 10.3.6). We first upper bound the number of possible crossing patterns with these condition, in Lemma 10.3.4. We then study each possible crossing pattern with those conditions individually and in Lemma 10.3.7, we prove that each such individual crossing pattern has an extremely low probability to be *good*—clarified shortly—for a random collection P. This probability is so small that it allows us to union bound over all possible crossing patterns and conclude that with high probability, no crossing pattern is good for a random collection P. Then, we infer that for a random collection P, with high probability, there is no short communication pattern P' that simulates the collection P. □

We first present the abstract definition of the concept of *crossing patterns*, and a few related notions and helper lemmas. Then, we explain the interpretation of this abstract definition and how it connects to simulations of a collection P of communication patterns.

Defining Crossing Patterns: We break the time into $0.1n^{0.1}$ phases, each made of $\frac{\log n}{100 \log\log n}$ consecutive rounds. Then, a *crossing pattern* $\Phi : [k] \times [n^{0.1}] \to [0.1n^{0.1}] \cup \{\perp\}$ is simply a mapping from communication pattern index $i \in \{1, 2, \ldots, k\}$ and layer number $j \in \{1, 2, \ldots, n^{0.1}\}$ to a phase number $t \in \{1, 2, \ldots, 0.1n^{0.1}\}$ or to $\perp$ (which indicates undefined).

Notations Related to Crossing Patterns Consider an arbitrary crossing pattern Φ. For each layer $j \in [n^{0.1}]$ and each phase $t \in [0.1n^{0.1}]$, let $I(j,t)$ be the set of communication pattern indices $i \in [k]$ such that $\Phi(i,j) = t$, that is, $I(j,t) = \{i \mid \Phi(i,j) = t\}$. Define the load $L(j,t)$ in layer j at phase t by $L(j,t) = |I(j,t)|$. That is, $L(j,t)$ denotes the total number

of communication pattern indices i that cross layer j during phase t. We call a pair of layer number j and phase number t *heavy* if $L(j,t) > 0.9n^{0.1}$.

Semi-Full Crossing Patterns: We say a crossing pattern Φ is *semi-full* if for each $i \in [1,k]$, we have $|\{j \mid \Phi(i,j) = \perp\}| \leq \frac{n^{0.1}}{10}$. That is, if for each communication pattern index i, strictly greater than 9/10 of the layers have a defined crossing phase number.

We next argue that the number of possible semi-full crossing patterns is moderately small.

Lemma 10.3.4. *The number of semi-full crossing patterns is at most* $e^{\Theta(n^{0.3})}$.

Proof. We argue that for each communication pattern P_i, there are at most $e^{O(n^{0.1})}$ different ways to specify the crossing pattern part for P_i in a semi-full crossing pattern. To specify that, we need to specify at most $0.1n^{0.1}$ layers that are not crossed during a phase, and also specify the crossing phase number for the rest of the layers. For the layers that are not crossed during a phase, there are $\binom{n^{0.1}}{0.1n^{0.1}} = e^{O(n^{0.1})}$ options. To assign increasing phase numbers in range $[n^{0.1}]$ to the remaining $0.9n^{0.1}$ layers, there are $\binom{1.9n^{0.1}-1}{0.9n^{0.1}} = e^{O(n^{0.1})}$ options[8]. Hence, the total number of options for the crossing pattern of a single communication pattern P_i is at most $e^{\Theta(n^{0.1})}$. Therefore, over all the $k = n^{0.2}$ communication patterns P_1 to P_k, the number of all possible crossing patterns is at most $e^{\Theta(n^{0.3})}$. □

Claim 10.3.5. *Each semi-full crossing pattern has at least one heavy layer-phase pair* (j,t) *for* $j \in [n^{0.1}]$ *and* $t \in [0.1n^{0.1}]$.

Proof. We argue that there is at least one pair of layer number j and phase number t such that $L(j,t) > 0.9n^{0.1}$. The argument is as follows. There are $k = n^{0.2}$ communication patterns indices $i \in [k]$. For each $i \in [k]$, at least $0.9n^{0.1}$ of the layers is crossed during a phase, because the crossing pattern is semi-full. Thus, we have

$$\sum_{j,t} L(j,t) > n^{0.2} \cdot 0.9n^{0.1} = 0.9n^{0.3}.$$

On the other hand, there are $0.1n^{0.2}$ choices for pair (j,t). Hence, the average load over these pairs is at least $0.9n^{0.1}$. Therefore, there exists at least one pair (j,t) such that $L(j,t) \geq 0.9n^{0.1}$. □

We next connect crossing patterns to simulations of communication patterns.

[8]Recall that this is essentially a case of the so-called *stars and bars* problem: phases are the stars and layers are the bars, and how many stars there are before a bar indicates the phase in which the related layer is crossed.

The Crossing Pattern Corresponding to a Simulation F of a collection P by a Communication Pattern P': Fix a simulation F of a collection P of communication patterns by a commutation pattern P'. We say that F has crossing pattern Φ if the following condition holds: For any $i \in [k]$ and $j \in [n^{0.1}]$, we have $\Phi(i,j) = t$ if and only if in the simulation $f_i \in F$ of P_i in P', two conditions are satisfied: (1) all the messages from v^{j-1} to S^j are sent in phase t, and (2) all the messages of nodes of S^j to node v^j are sent in phase t. If no t satisfies these conditions, we have $\Phi(i,j) = \perp$.

We emphasize that Φ does not depend on the collection P, and it only maps indices $i \in [k]$ and the respective layer numbers to phase numbers. Also, although each simulation F defines one crossing pattern Φ, different simulations F_1 and F_2 might have the same crossing pattern.

Good Crossing Pattern For a Collection P of Communication Patterns: Consider a crossing pattern $\Phi : [k] \times [n^{0.1}] \to [0.1n^{0.1}] \cup \{\perp\}$. Recall the notation $I(j,t)$ which denotes the set of communication pattern indices $i \in [k]$ such that $\Phi(i,j) = t$, that is, $I(j,t) = \{i | \Phi(i,j) = t\}$. Fix an edge $e \in G$ that is in layer j. Fix a collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns and let $I_e(t)$ be the subset of indices $i \in I(j,t)$ such that communication pattern P_i contains a time-copy of edge e, that is, P_i sends a message along edge e at some point in time. We say edge e is *overloaded* in phase t if $|I_e(t)| > \frac{\log n}{100 \log\log n}$. We say that Φ is *good* for P if there is no overloaded edge in any phase.

We next connect the existence of a short simulating communication pattern P' to the existence of a good semi-full crossing pattern Φ. In particular, we prove the following claim:

Claim 10.3.6. *If there is a short communication pattern P' that simulates a collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns in our distribution, then there is a semi-full crossing pattern Φ that is good for P.*

Proof. Fix an arbitrary collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns in our distribution. Suppose that there is a short communication pattern P' that simulates collection P. We first generate a crossing pattern Φ by examining the simulation F of P' for the collection P. Then, we argue that Φ must be (1) semi-full and (2) good for P.

The rule for generating Φ is the same as the definition presented earlier when discussing the crossing pattern corresponding to a simulation. For simplicity, we repeat it here. For a communication pattern index i and layer j, set $\Phi(i,j) = t$ if in the simulation $f_i \in F$ of P_i in P', two conditions are satisfied: (1) all the messages from v^{j-1} to S^j are sent in phase t, and (2) all the messages of nodes of S^j to node v^j are sent in phase t. If no phase number t satisfies these two conditions, set $\Phi(i,j) = \perp$.

We now argue that Φ must be semi-full. Fix one $P_i \in \mathsf{P}$. If strictly more than $\frac{n^{0.1}}{10}$ layers of P_i do not have defined crossing phases, that is if $|\{j | \Phi(i,j) = \perp\}| > \frac{n^{0.1}}{10}$, then the length of the simulation of P_i would be more than $\frac{n^{0.1}}{10} \cdot \frac{\log n}{100 \log\log n} = \frac{n^{0.1} \log n}{1000 \log\log n}$ rounds. This would be in contradiction with P' being short. Thus, in P', at most $\frac{n^{0.1}}{10}$ layers of P_i have an

undefined crossing phase and are mapped to $\perp$. As this holds for each $i \in [k]$, we get that the crossing pattern Φ is semi-full.

We next argue that Φ must be good for the collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$. Fix an arbitrary layer number $j \in [n^{0.1}]$ and a phase number $t \in [0.1n^{0.1}]$. Recall from above that $I(j,t) = \{i | \Phi(i,j) = t\}$ and for each edge e of layer j, we use $I_e(t)$ to denote the subset of indices $i \in [k]$ such that communication pattern P_i uses a time-copy of edge e, that is, P_i sends a message along edge e at some point in time. Since P' is a simulation of $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$, it delivers the messages of all communication patterns with indices in $I_e(t)$ in phase t, in distinct rounds of the phase. On the other hand, each phase has only $\frac{\log n}{100 \log\log n}$ rounds. Therefore, it must be the case that $|I_e(t)| \leq \frac{\log n}{100 \log\log n}$. That is, edge e is not overloaded in phase t. As this holds for any phase t and any edge e, crossing pattern Φ is good for the collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$. □

We next investigate each semi-full crossing pattern and we argue that it has an extremely small probability to be good for a random collection P.

Lemma 10.3.7. *For each fixed semi-full crossing pattern, the probability that it is good for a random collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns is at most $e^{-n^{0.7}}$.*

Proof. Fix an arbitrary semi-full crossing pattern Φ, and a heavy layer-phase (j,t) of it, which is established to exist by Claim 10.3.5. Recall from above that $I(j,t) = \{i | \Phi(i,j) = t\}$. To upper bound the probability of Φ being good for P, we upper bound the probability that in phase t there is no overloaded edge e of layer j. We will examine only edges e of layer j that connect from v^{j-1} to a node in U^j. Recall that for each edge e of layer j, we use $I_e(t)$ to denote the subset of indices $i \in [k]$ such that communication pattern P_i uses a time-copy of edge e, that is, P_i sends a message along edge e at some point in time. We upper bound the probability that no such edge e has $|I_e(t)| > \frac{\log n}{100 \log\log n}$. Recall that if there exists such an overloaded edge e, then Φ is not good for P.

Consider each edge e of layer j. It is easy to see that $\mathbb{E}[|I_e(t)|] \geq 0.9$. That is, the expected number of communication pattern indices $i \in I(j,t)$ such that P_i includes a time-copy of e is at least $0.9n^{0.1} \cdot n^{-0.1} = 0.9$. The reason is as follows: We have $L(j,t) = |I(j,t)| \geq 0.9n^{0.1}$, and each of the communication patterns P_i for $i \in I(j,t)$ includes edge e with probability $n^{-0.1}$. Notice that this bound on the expectation is rather small expectation, and in particular, far lower than our threshold $\frac{\log n}{100 \log\log n}$ for the edge being overloaded. We next argue that there is a non-negligible probability that the load $|I_e(t)|$ of at least one such edge e deviates significantly from this lower bound on its expectation and e is overloaded in phase t. In particular, for each edge e of layer j, we have

$$Pr[|I_e(t)| \geq \frac{\log n}{100 \log\log n}] \geq \sum_{\ell = \frac{\log n}{100 \log\log n}}^{0.9n^{0.1}} \binom{0.9n^{0.1}}{\ell} (n^{-0.1})^{\ell} (1 - n^{-0.1})^{0.9n^{0.1} - \ell} \geq n^{-0.2}.$$

On the other hand, for each communication pattern P_i, the choices of different edges of

layer j (which each connect from v^{j-1} to a node in U^j) being in P_i are independent. Thus, the events of different edges e being overloaded are independent. Therefore, the probability that no edge e of layer j is overloaded in phase t is at most $(1-n^{-0.2})^{n^{0.9}} \leq e^{-n^{0.7}}$. That is, the probability that the fixed crossing pattern Φ is good for a random collection P of communication patterns is at most $e^{-n^{0.7}}$. □

Proof of Lemma 10.3.3. Now, we wrap up the proof using a union bound over all semi-full crossing patterns. Lemma 10.3.4 shows that the number of all possible semi-full crossing patterns is at most $e^{\Theta(n^{0.3})}$. Lemma 10.3.7 shows that the probability of each semi-full crossing pattern Φ being good for a random collection P of communication patterns is at most $e^{-n^{0.7}}$. Thus, a union bound over all the possible crossing patterns tells us that the probability that there is a semi-full crossing pattern that is good for the randomly sampled collection P of communication patterns is at most $e^{-n^{0.7}} \cdot e^{\Theta(n^{0.3})} \ll 1$. Thus, for a random collection P, with high probability, there is no crossing pattern that is good for it. Claim 10.3.6 shows that if there is a short communication pattern P' that simulates a collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ of communication patterns in our distribution, then there is a semi-full crossing pattern Φ that is good for that collection P. Hence, we can conclude that for a random collection P, there is no short communication pattern P' that simulates P. □

Finally, we put Lemma 10.3.2 and Lemma 10.3.3 together and prove Theorem 10.1.3.

Proof of Theorem 10.3.1. Lemma 10.3.2 shows that collection P has $\mathsf{congestion} = O(n^{0.1})$ and $\mathsf{dilation} = 2n^{0.1}$, with high probability. Lemma 10.3.3 shows that a random collection P has no short simulating communication pattern P', with high probability. A union bound over these two shows that the random collection P satisfies the two properties together, with high probability: (1) $\mathsf{congestion} = O(n^{0.1})$ and $\mathsf{dilation} = 2n^{0.1}$, and (2) there is no short communication pattern P' that simultaneously simulates this collection P. □

We now finish this section by explaining how the above results, and in particular Theorem 10.3.1, prove the main theorem of this section, Theorem 10.1.3. For simplicity, let us recall the statement.

Theorem 10.1.3 *There is an instance of the distributed protocol scheduling problem for which any schedule needs* $\Omega(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log\log n)$ *rounds.*

Proof of Theorem 10.1.3. The hard instance claimed in Theorem 10.1.3 is simply a set of protocols where each protocol $\mathcal{P}_i$ has a communication pattern P_i in the collection $\mathsf{P} = \{P_1, P_2, \ldots, P_k\}$ proven to exist by Theorem 10.3.1. Notice that for these communication patterns, any local simulation for node v^L is indeed a simulation for the whole communication pattern, simply because all the communications in each P_i causally influence v^L. Therefore, any schedule for protocols $\mathcal{P}_1$ to $\mathcal{P}_k$ would imply the existence of a simulation for communication patterns P_1 to P_k with the same length. Theorem 10.3.1 establishes that any such simulation must have length $\Omega(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log\log n)$ rounds. Thus, any schedule for $\mathcal{P}_1$ to $\mathcal{P}_k$ also needs $\Omega(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log\log n)$ rounds. □

10.4 Upper Bound

In this section, we present our algorithmic result on distributed protocol scheduling, which allows us to run protocols $\mathcal{P}_1$ to $\mathcal{P}_k$ concurrently, in a short span of time.

To formally discuss running a collection $\mathcal{P}_1$ to $\mathcal{P}_k$ of protocols concurrently, we fix their inputs and randomness at the beginning of the execution. More concretely, for each $i \in [k]$, we assume that (1) the input of protocol $\mathcal{P}_i$ for each node v gets fixed at the beginning of the execution (provided to node v as its input in $\mathcal{P}_i$), and moreover, (2) each node v fixes the randomness that it will use when executing protocol $\mathcal{P}_i$. Once these two are fixed in all nodes, we have a unique execution α_i for each protocol $\mathcal{P}_i$ (when $\mathcal{P}_i$ is run alone). In particular, this unique execution leads to a unique output $out_i(v)$ for each node v. Our general goal in running the collection $\mathcal{P}_1$ to $\mathcal{P}_k$ concurrently is to make each node node learn $out_i(v)$ for all $i \in [1, k]$. The core issue in doing that is to arrange the communications of different protocols in the shortest possible span of time. To discuss these communications, for each protocol $\mathcal{P}_i$, we use P_i to denote the communication pattern of the fixed execution α_i of $\mathcal{P}_i$. In the remainder of this section, our focus will be mainly on scheduling these fixed communication patterns P_1 to P_k. We present a distributed algorithm that computes a schedule of optimal length for these communication patterns P_1 to P_k. Moreover, this algorithm will allow the nodes to also learn their outputs, that is, each node v learns $out_i(v)$ for all $i \in [1, k]$.

Roadmap We start in Section 10.4.1 by presenting a formalized proof of the existential part of Theorem 10.1.2, which serves as a warm-up for the discussions in the rest of this section. Then, in Section 10.4.2, we give an intuitive explanation of the challenge in scheduling distributed protocols and a high level description of the ideas we use to overcome this challenge. Finally, in Section 10.4.3, we describe the algorithm and its analysis.

10.4.1 Warm-Up: Existence Proof for a Short Schedule

As a warm-up for our algorithm for distributed protocol scheduling, we first explain a formalization of the existential[9] aspect of the result claimed in Theorem 10.1.2. This result shows the existence of schedules with a length nearly matching the lower bound of Theorem 10.1.3.

Theorem 10.4.1 (Extension of [LMR94]). *Given a collection of k communication patterns P_1 to P_k, there exists a communication pattern P' that is a simultaneous simulation of P_1 to P_k and has length $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds.*

Proof. We construct P' randomly and prove that with high probability, it is a simultaneous simulation of P_1 to P_k.

[9] We note that this randomized construction can be turned into a randomized distributed algorithm for scheduling, if nodes have access to shared randomness. For simplicity, we forgo describing the formal details of the setting with shared randomness.

The randomized construction of P' is as follows: Divide time into *phases*, each consisting of $R = c \log n$ consecutive rounds, for a sufficiently large constant $c \geq 100$. More formally, this means grouping each $\Theta(\log n)$ consecutive layers of the time-expanded graph into one group, to which we refer as a *phase*. We later discuss how many phases are used in the construction of P'. We construct P' phase by phase, as follows: pick a random delay δ_i for each communication pattern P_i, where δ_i is chosen according to a uniform distribution on integers $[0, \mathsf{congestion}/\log n]$. The simulation of communication pattern δ_i starts in phase i and it proceeds in simulating P_i but at a slower speed, by simulating each round of P_i in one phase of P'.

We now argue that if we follow this rule, with high probability, we have the following property: for each edge e in the network, and each given phase, there are at most $O(\log n)$ messages that need to be sent during this phase over edge e. Notice that the expected number of messages that will need to traverse edge e in this phase is at most $\mathsf{congestion}(e)/(\mathsf{congestion}/\log n) = \log n \leq R/100$. We have independence between the messages of different communication patterns as their random delays are independent. On the other hand, for each communication pattern and for the messages that it sends through an edge e, the events of these different messages falling on the fixed phase under consideration are negatively correlated. This is because, if one message is scheduled to go through e in this particular phase, none of the other messages of that communication pattern that need to go through e can be in that phase. Note that a Chernoff bound is applicable for negatively correlated random variables (see e.g. [DP09, DR96]). Therefore, using a Chernoff bound, we get that with high probability, for each edge and each phase, at most $R/10$ messages are scheduled to traverse this edge in this phase. Since the phase has R rounds, there is enough time to deliver all these messages. Hence, with high probability, all the communication patterns are simulated in P' and the simulations are edge-disjoint, hence providing a simultaneous simulation of collection $\mathsf{P} = \{P_1, \ldots, P_k\}$.

We now argue that $length(P') = O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$. The reason is as follows: for each communication pattern P_i, the maximum random delay is at most $\mathsf{congestion}/\log n$ phases, and thus each communication pattern's simulation starts after at most $\mathsf{congestion}/\log n$ phases. Furthermore, each communication pattern's simulation takes at most $\mathsf{dilation}$ phases, once started. Thus, P' has $O(\mathsf{congestion}/\log n) + \mathsf{dilation}$ phases. Since each phase has $R = O(\log n)$ rounds, this means that $length(P') = O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$. □

10.4.2 The Main Challenge and the Outline of Our Approach

The discussions in this subsection are informal, and intend to merely provide some intuition about the challenge and how we tackle it. The formal approach is presented in Section 10.4.3.

A Challenge The key insight in the random delays strategy of Leighton, Maggs, and Rao [LMR94], explained above and used in the proof of Theorem 10.4.1, is that randomly delaying messages spreads them over time in such a way that the number of packets scheduled

to go through an edge per unit time is small. When working with distributed protocols, implementing this random delay strategy is not straightforward: we cannot have the same randomly chosen *delay* value for a given protocol over the whole network. This is because having the same random value over the whole network would require spending at least D rounds, where D denotes the network diameter. Note that adding D rounds to the complexity would usually be considered an undesirable overhead, especially in the case of local distributed algorithms.

One piece of good news is that we actually do not need to have the same delay value over the whole network. Let us focus on one node v. In each of the protocols $\mathcal{P}_i$, node v is influenced by only the actions of nodes within its dilation-neighborhood. This is because protocol $\mathcal{P}_i$ has round complexity at most dilation. Events outside the dilation-neighborhood of v cannot affect v in less than dilation rounds. Hence, for node v, it would be sufficient if we apply (an appropriate version of) the random delay strategy only among the nodes within the dilation-neighborhood of v. That is, we can imagine that we have carved out a ball of radius dilation hops around v and we apply the random delay technique only inside this ball. That would suffice for node v to learn its output $out_i(v)$ in each protocol $\mathcal{P}_i$. However, actually leveraging this insight is non-trivial, because we need to perform something of this kind for each of the nodes, all at the same time. In particular, each edge e can be in the dilation-neighborhood balls of a large number of nodes. Hence, separately executing protocols in each of these balls would require an undesirably large running time.

Our Approach in a Nutshell To overcome the above challenge, our algorithm makes use of a generic method of *graph partitioning* [LN05,CKR05,Bar04,FRT04], colloquially referred to as *ball carving*. In particular, the closest to what we do is the approach that Bartal [Bar98] used for probabilistically approximating arbitrary metric spaces with tree-metrics. The end result of the graph partitioning part will be a number of clusters—that is subset of the vertices of the graph— that are grouped into $L = \Theta(\log n)$ *layers*, with the following three properties: (1) the clusters in each layer are vertex-disjoint. This implies that each edge is in at most $L = O(\log n)$ clusters, at most one per layer. (2) Each cluster has diameter $O(\text{dilation} \cdot \log n)$. (3) For each node v, there are at least $L/100 = \Theta(\log n)$ layers in each of which the whole dilation-neighborhood of node v is fully contained in one cluster.

In Section 10.4.3, we explain an efficient distributed method for computing this graph partitioning in the CONGEST model. This method mostly follows the approach of Bartal [Bar98]. But it also involves a number of smaller new ideas for implementing the approach in the CONGEST model. We believe these new ideas might be useful also in other distributed graph partitioning algorithms in the CONGEST model.

Once we have the aforementioned partitioning of the graph, we share random bits in each of the clusters. More concretely, we make one node of the cluster choose a sequence of random bits locally, and then broadcast this sequence to all the nodes of the cluster. These random sequences are fed into a $\Theta(\log n)$-wise-independent pseudo-random generator, which receives these bits and generates a much larger sequence of random values that are

$\Theta(\log n)$-wise independent. We then use these values as the locally-shared random delay values in scheduling distributed protocols, via locally simulating each distributed protocol $\mathcal{P}_i$ (formally, its fixed communication pattern P_i) in each cluster. That is, each cluster runs a copy of each protocol. The simulations of different clusters do not depend on each other. We show how using the aforementioned locally-shared random delay values, we can run all the distributed protocols in all the clusters in $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds. This allows us to produce a schedule of length $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds for the corresponding communication patterns.

There is one more issue to discuss: recall that the final goal from running all the protocols concurrently was to let each node v learn its outputs $out_i(v)$ for all protocols $\mathcal{P}_i$. However, now in each clustering layer, each node v runs one copy of each protocol $\mathcal{P}_i$. Hence, node v is involved in many copies of the protocol, each spanning a local area of the network. Node v needs to pick its output according to one of these. We explain how each node v can find a correct *local simulation* of each communication pattern P_i, thus allowing v to learn its output $out_i(v)$. For that, node v needs to find a clustering layer in which the whole $\mathsf{dilation}$-neighborhood of the node v is contained in the cluster of v. In the simulation of each such layer, we are guaranteed that the output of node v is the same as its output $out_i(v)$ in the fixed execution α_i of $\mathcal{P}_i$. This is because the simulation in any such layer is indistinguishable for v from a complete global simulation. Recall that the graph partitioning algorithm guarantees that there are at least $L/100$ such clustering layers for each node v.

10.4.3 Scheduling Distributed Protocols via Locally Sharing Randomness

Here, we describe our algorithm for scheduling distributed protocols. This algorithm follows the outline discussed above and achieves the following result:

Theorem 10.1.4 *There is a distributed algorithm that uses only private randomness and for any instance of the distributed protocol scheduling problem, computes a schedule of length $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$, in $O(\mathsf{congestion} + \mathsf{dilation} \log^2 n)$ rounds, with high probability. In particular, given a collection $\mathcal{P}_1$ to $\mathcal{P}_k$ of protocols, and the corresponding fixed executions α_1 to α_k as discussed above, this algorithm allows each node v to learn its output $out_i(v)$ in execution α_i for each $i \in [1, k]$.*

The algorithm has three parts, as follows:

- (**Part I**) We first describe the aforementioned graph partitioning algorithm.
- (**Part II**) We then explain how to share randomness in each of the clusters of this graph partition.
- (**Part III**) Finally, we conclude with how to concurrently run the distributed protocols—formally scheduling the corresponding communication patterns—using these locally shared random sequences.

We next proceed to explaining each of these parts.

Part I — Graph Partitioning

Recall from above that a key ingredient is the graph partitioning method. We use this to break each communication pattern into smaller communication patterns, each of which has communications that span only a small local area of the network. We first present the guarantees provided by the algorithm, formalized in Lemma 10.4.2. We then present the algorithm and afterwards its analysis, in the proof of Lemma 10.4.2. The graph partitioning algorithm is abstracted by the following guarantee:

Lemma 10.4.2. *[**The Graph Partitioning Lemma**] There is a distributed algorithm, which we call the Graph Partitioning Algorithm (GPA), that runs in $O(\mathsf{dilation}\log^2 n)$ rounds and creates $L = \Theta(\log n)$ layers of clustering of the graph such that: (1) in each layer, the clusters are node-disjoint, (2) each cluster has weak diameter $O(\mathsf{dilation}\cdot\log n)$, (3) w.h.p, for each node v, there are at least $L/100 = \Theta(\log n)$ layers such that the $\mathsf{dilation}$-neighborhood of v is fully contained in one of the clusters of this layer, and*

We next present the outline of the algorithm, which is partially based on Bartal's algorithm [Bar98, Section 3]. Afterwards, we explain the details of the distributed implementation of this algorithm in the CONGEST model.

The Overall Plan of the Graph Partitioning Algorithm (GPA) We have $L = \Theta(\log n)$ layers of clustering, each obtained from an independent repetition of the following scheme: each node u picks a random radius $r(u)$ from a truncated exponential probability where $\Pr[r(u) = z] = (\frac{n}{n-1})\frac{1}{R}e^{-z/R}$ for $R = \Theta(\mathsf{dilation})$. Node u also picks a random label $\ell(u) \in \{0,1\}^{4\log n}$. Note that w.h.p., for each two nodes $u \neq u$, $\ell(u) \neq \ell(u')$. The radius $r(u)$ defines a ball of radius $r(u)$ centered at u, which we denote $B(u)$. Each node v joins the cluster centered at node w^* where w^* is defined as the node that has the smallest label $\ell(w^*)$ among the labels of nodes w such that $v \in B(w)$.

Distributed Implementation of the Graph Partitioning Algorithm We now discuss the distributed implementation details of GPA. We run the layers of clustering separately, each in $O(\mathsf{dilation}\log n)$ rounds. Thus, overall this graph partitioning algorithm takes $O(\mathsf{dilation}\log^2 n)$ rounds.

For each layer of clustering, we do as follows: each node u draws a random radius $r(u)$ from the truncated exponential distribution explained above, as well as a random label $\ell(u) \in \{0,1\}^{4\log n}$ where each bit is uniformly distributed in $\{0,1\}$. Then the output for each node v is the node ID of the node $w^* = \mathit{argmin}_{w \text{ s.t. } dist(v,w)\leq r(w)}\ell(w)$. In this case, w^* is the center of the cluster of v and this cluster is defined by the set of nodes that have the same cluster center.

To compute these outputs, the distributed algorithm is as follows: Each node u initiates a message m_u containing its ID and label $\ell(u)$, and an initial "hop-count" which is set to $H - r(u)$, where $H = \Theta(\mathsf{dilation} \log n)$. This hop-count mimics the scenario that the message m_u has already traveled $r(u)$ hops by the time it reached u. We increment the hop-count with every hop that the message traverses, and we allow the message's hop-count to reach only H; beyond that point, the message is not forwarded anymore. Hence, given the initial hop-count, the message can traverse at most $r(u)$ hops. That is, the message m_u can reach only nodes that are within $r(u)$ hops of u.

The message forwarding procedure is as follows: in each round $i \in [1, H]$, each node v forwards the message with hop-count i that has the smallest label among the messages of hop-count i or less that have reached v. Each received message gets its hop-count incremented by 1. At the end, for each node v, the smallest label $\ell(w^*)$ that has reached v indicates that cluster center of node v is node w^*.

Proof of Lemma 10.4.2. Properties (1) and (2) follow immediately from the definition, and the property (3) follows from the analysis of Bartal [Bar98, Section 3] which shows that in each layer, each $\mathsf{dilation}$-neighborhood is fully contained in one of the clusters with constant probability.

We next argue that the distributed implementation presented above indeed implements the plan. In particular, we first argue that each message m_u can reach only nodes that are in $B(u)$. Then, we argue that each node v will indeed receive the message m_u of the node u that is supposed to be the cluster center of node v according to the overall plan. Finally, we argue that node v will not receive a message from a node u' with a smaller label $\ell(u')$ than the label of this supposed the cluster center u of node v. We next explain these steps.

First, we argue that each message m_u can reach only nodes that are in $B(u)$, that is, it cannot reach nodes outside $B(u)$. This is because of the initial hop-count of m_u, which is set to $H - r(u)$, which implies that m_u can travel $r(u)$ hops from the origin until it reaches the threshold hop-count of H and thus does not get forwarded afterward.

Second, we argue that for any node v, if $\ell(u)$ is the smallest label among the labels of balls that contain v, then m_u will indeed reach v. This is because of the following: if m_u does not reach v, it must be that in a round i, there is a node w on the shortest path from u to v that has received m_u but does not forward m_u because it instead forwards (or has forwarded) $m_{v'}$ for a node v' such that $\ell(v') < \ell(v)$ and at w, the hop count of $m_{v'}$ was less than or equal to that of m_v. But this implies that u is in fact in $B(v')$, which is in contradiction with v having the smallest label among the balls that contain u.

Third, $\ell(u)$ will clearly be the smallest label that v hears, and thus v will join the cluster centered at u. □

Covered Radii For our algorithms, we need the nodes to know one more thing: In each layer ℓ, each node v should know what is the maximum *radius* around v that is fully contained in the same cluster of layer ℓ that contains node v. We call this the *covered radius* of node

v in layer ℓ. We next explain a simple distributed algorithm that allows us to inform nodes about their covered radii, in $O(\mathsf{dilation} \log^2 n)$ rounds.

The Algorithm for Learning Covered Radii (LCR) We perform the following procedure for each layer ℓ in $O(\mathsf{dilation} \log n)$ rounds. Performing it one by one for all the layers takes $O(\mathsf{dilation} \log^2 n)$ rounds. First, each node v sends its cluster label—i.e. the label of its cluster center—$\ell(w^*)$ to each of its neighbors. Then, if node v receives a cluster label from its neighbors that is different than its own cluster label, then v marks itself as a *cluster boundary* node. In the next $O(\mathsf{dilation} \log n)$ rounds, we perform a boundary message propagation as follows: each boundary node sends a special 'BOUNDARY' message, and each other node that has received such a message forwards it to its neighbors in the next rounds. The output for each node v is a number $i - 1$ where i is such that v sent the 'BOUNDARY' message for the first time in round i.

Lemma 10.4.3. *After executing the LCR algorithm, each node v learns its covered radii, that is, node v learns what radius around v is fully contained in the cluster of v in each layer.*

Proof. Let us focus on one layer. It is easy to see that each node that is k hops away from boundaries will send BOUNDARY for the first time in round $k + 1$, and thus it will set its output to be k. Hence, if h' neighborhood of a node v is the maximum neighborhood that is fully contained in the cluster of node v, then node v will set its covered radius to be h'. Thus, node v learns what radius around it is fully contained in its own cluster. □

To summarize, at this point, we have a partitioning of the graph in layers, which is known in the following format: each node knows its cluster—via knowing the ID of the cluster center—in each layer of the graph partitioning. Moreover, each node knows its covered radii in each layer of the clustering.

Part II — Randomness Sharing in the Clusters of the Graph Partition

Given the graph partitioning provided by Lemma 10.4.2, we now explain how to share (a sufficient amount of) randomness in the clusters of this partition.

We note that if we had to share $\Theta(\log n)$ random bits in each cluster, then each cluster center node would simply pick this randomness and append it to its initial message. However, we actually need to share $\Theta(\log^2 n)$ bits per cluster. The reason for this amount becomes clear later in the proof of Lemma 10.4.5, where we turn these bits of randomness into much longer sequences of random values. Sharing $\Theta(\log^2 n)$ bits requires sending $O(\log n)$ messages by the cluster center, because each message can carry only $O(\log n)$ rounds. Thus, a naive extension of the message forwarding approach used in proving Lemma 10.4.2 would become $O(\mathsf{dilation} \log^2 n)$ rounds per layer, that is, $O(\mathsf{dilation} \log^3 n)$ in total, which is more than our promised round complexity of $O(\mathsf{dilation} \log^2 n)$. We next explain how to achieve this randomness sharing in $O(\mathsf{dilation} \log^2 n)$ rounds.

Randomness Sharing Algorithm (RSA) in the Clusters of One Layer We broadcast randomness in clusters of one layer in $O(\mathsf{dilation} \log n)$ rounds. Repeating this scheme $\Theta(\log n)$ times, once for each clustering layer, solves the problem in $O(\mathsf{dilation} \log^2 n)$ rounds[10]. For each clustering layer, we do as follows: each cluster center v creates $\Theta(\log n)$ messages, each containing $\Theta(\log n)$ random bits. Each of these messages m carries three header information, aside from its payload of random bits: (1) an initial hop-count, (2) the label $\ell(v) = ID(v)$ of node v, and (3) an additional counter $\ell'(m)$ from $\{1, 2, \Theta(\log n)\}$ to distinguish between these $\Theta(\log n)$ messages of node v. The initial hop-count is set as in Lemma 10.4.2 for all these messages, that is, it is set equal to $H - r(v)$ where $r(v)$ is the radius of the cluster of v (chosen by v randomly in the graph partitioning algorithm). Then, for $H + \Theta(\log n) = O(\mathsf{dilation} \log n)$ rounds, in each round, each node forwards the message with lexicographically smallest (hop-count, $\ell(v), \ell'(m)$) that it has not sent before.

Lemma 10.4.4. *In $O(\mathsf{dilation} \log^2 n)$ rounds, the RSA algorithm shares $\Theta(\log^2 n)$ random bits in each of the clusters created by the GPA algorithm, as abstracted in Lemma 10.4.2.*

Proof Sketch. The pipelining analysis of Lenzen [LP13] shows that after $H + \Theta(\log n) = O(\mathsf{dilation} \log n)$ rounds of the message forwarding process explained above, each node v receives the smallest $\Theta(\log n)$ messages starting in the H neighborhood of v. Here, smallest is with respect to the lexicographical ordering of $(\ell(v), \ell'(v))$. Because of the initial hop-count settings, this H-neighborhood translates to all centers that could potentially reach v, and particularly means node v will receive all the $\Theta(\log n)$ messages coming from the center u of the cluster that contains v. □

The $\Theta(\log^2 n)$ bits of randomness shared in each cluster are not (directly) sufficient for determining all the random delay values. We next explain how to expand this sequence of randomness, while maintaining a sufficient degree of independence. We note for this part, we simply apply a standard method in generating pseudo-random numbers, see e.g. [AS04, Theorem 15.2.1] or [DvM, Section 3].

Lemma 10.4.5. *Using deterministic local computations, each node v can transform the $\Theta(\log^2 n)$ fully-independent random bits shared in each cluster that contains v into $\mathrm{poly}(n)$ many $\Theta(\log n)$-bit random values that are $\Theta(\log n)$-wise independent.*

Proof. Each node v feeds the $\Theta(\log^2 n)$ random bits of each cluster that contains v into the classical k-wise independent pseudo-randomness construction via Reed-Solomon codes, for $k = \Theta(\log n)$. See for instance [DvM, Section 3] and also [AS04, Theorem 15.2.1][11] for the

[10] The author believes that one might be able to get to a bound of $O(\mathsf{dilation} \log n + \log^3 n)$ rounds, but this is an issue to be worked out for the journal version of the paper.

[11] We note that [AS04, Theorem 15.2.1] describes only the construction on the field GF(2) which yields binary random values that are k-wise independent. But, as also described in [DvM, Section 3], the construction readily extends to any $GF(p)$, for any prime number $p \in \mathrm{poly}(n)$. Furthermore, when desiring random delays in range $[\Theta(R)]$ for a given R, pick them from a range $\{1, \ldots, p\}$ for a prime $p \in \Theta(R)$. Note that by Bertrand's postulate, there are many such primes, as there is at least one between a and $2a - 2$, for any integer $a \geq 4$.

explanations of this pseudo-randomness construction. This transforms the randomness to $\text{poly}(n)$ many $\Theta(\log n)$-bit random values that are $\Theta(\log n)$-wise independent. □

The random values generated by the transformation of the above lemma will be used to determine the random delays of different communication patterns in each of the clusters. In particular, we assume that each communication pattern P_i for $i \in k$ has a unique *protocol identifier* $\mathsf{PID}(i)$ in a range of size $K = \text{poly}(n)$. We divide the generated random values into K buckets, each containing $\text{poly}(n)$ random values. The communication pattern P_i will choose its random delays based on the random values in bucket $\mathsf{PID}(i)$.

To summarize, at this point, what nodes have learned throughout the algorithms of the past two parts is as follows: (1) We have a partitioning of the graph in layers, which is known in the following format: each node knows its cluster—via knowing the ID of the cluster center—in each layer of the graph partitioning. Moreover, each node knows its covered radii in each layer of the clustering. (2) We have shared randomness and we have expanded it via Lemma 10.4.5. At the end, all nodes of each cluster have shared randomness in the following format: for each protocol index $i \in k$, there are $\text{poly}(n)$ random values, which are known to all nodes of the cluster, and are stored in each node in an array identified by $\mathsf{PID}(i)$. The random sequences of different clusters are independent.

Part III — Locally Simulating the Communication Patterns using the Locally Shared Randomness

Now, we explain how to use the randomness shared in the clusters to derive a schedule of length $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds for the communication patterns. In particular, this will also allow each node v to learn its output $out_i(v)$ for each protocol $\mathcal{P}_i$. Using this, at the end of this section, we prove Theorem 10.1.4.

We first explain a simpler algorithm that leads to a schedule of length $O(\mathsf{congestion} \cdot \log n + \mathsf{dilation} \cdot \log n)$. Then, we explain how to improve this to the near-optimal schedule length of $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds, using a careful choice of a non-uniform distribution of random delays.

Basic Scheduling Algorithm (BSA) We run an instance of each protocol in each cluster. Before explaining how we run these protocols simultaneously, we need to explain how we run each instance only within each cluster. Since we have partitioned the graph into clusters and we want to run each protocol instance only inside these clusters, we need to be careful about something: consider a node v such that the $\mathsf{dilation}$-neighborhood is fully contained in its own cluster. We should run the protocol instance in the cluster of v such that it is indistinguishable for v from running the full protocol over the whole network. Consider a communication pattern P_i and a clustering layer ℓ. Recall that the covered radius of each node v in layer ℓ is the maximum neighborhood radius h' such that the h'-neighborhood of v is contained in the cluster of v in layer ℓ. If in layer ℓ, the covered radius of a node v is h', then v will execute only the first h' rounds of P_i. It will discard the messages that it would

be sending in the later rounds. This change does not affect the execution from the viewpoint of the nodes that their dilation-neighborhood is fully contained in a cluster. This is because, if dilation-neighborhood of node w is fully contained in its cluster, all the messages that can influence w will still be sent.

We divide time into *big-rounds*, each of which consists of $\Theta(\log n)$ consecutive rounds. For each cluster, for each communication pattern P_i, the start of P_i is delayed by a random delay of δ big-rounds, where δ is a random value uniformly distributed in $[1, \mathsf{congestion}]$. This random value is chosen using the randomness shared in the cluster, from the bucket $\mathsf{PID}(i)$ of random values explained above. Then, each communication pattern proceeds in a synchronous manner at a rate of one communication pattern round per big-round, but only inside each cluster. The execution finishes after $\mathsf{dilation} + \mathsf{congestion}$ big-rounds. Later in the analysis, presented as the proof of Claim 10.4.6, we will argue that, w.h.p., the number of the messages that need to go through each edge in each big-round is at most $O(\log n)$. This means that the big-round has enough time to deliver all of these messages.

At the end, we determine the outputs. Each node v picks its output $out_i(v)$ for each protocol $\mathcal{P}_i$ based from the instance of protocol $\mathcal{P}_i$ in a layer ℓ such that the covered radius of v in layer ℓ is at least dilation, that is, dilation-neighborhood of v is contained in the cluster of v. Recall from property (3) of the graph partitioning lemma (Lemma 10.4.2) that there are many such layers ℓ. Also, notice that as node v knows its covered radius in each layer (as shown by Lemma 10.4.3), node v knows which of the layers satisfy this condition.

Claim 10.4.6. *Given the local randomness sharing as done in Lemma 10.4.4, the BSA algorithm schedules the communication patterns P_1 to P_k in $O(\mathsf{congestion} \cdot \log n + \mathsf{dilation} \cdot \log n)$ rounds. Furthermore, each node v learns its output $out_i(v)$ for each protocol $\mathcal{P}_i$.*

Proof. We can see that the expected number of messages that will need to go through each edge is at most $O(\log n)$ per big-round. The reason is as follows: for each edge e, there are at most $O(\mathsf{congestion} \cdot \log n)$ many messages that need to be sent through edge e. This is because for each communication pattern P_i that contains (a time-copy of) e, there can be up to $\Theta(\log n)$ copies of each of the messages that P_i sends over e, at most one per layer. Hence, the number of messages that need to go through e can increase from congestion to at most $O(\mathsf{congestion} \cdot \log n)$. Now, we have a random delay in the range $[\mathsf{congestion}]$ for each of these messages. Hence, the probability that each message needs to go through e in a given big-round is $1/\mathsf{congestion}$. We conclude that the expected number of messages that need to go through e in a given big-round is at most $O(\mathsf{congestion} \cdot \log n)/\mathsf{congestion} = O(\log n)$. Furthermore, we have $\Theta(\log n)$-wise independence between the random delays of different protocols. This allows us to apply a Chernoff bound (see e.g. [SSS95, Theorem 2]) and infer that, with high probability, at most $O(\log n)$ messages need to go through each edge per big-round. This means that the big-round has enough time to deliver all the messages that need to go through edge e in that big-round.

Because of the above, we are able to run all the copies of the protocols in the clusters, in $O(\mathsf{congestion} + \mathsf{dilation})$ big-rounds. This is $O(\mathsf{congestion} \cdot \log n + \mathsf{dilation} \cdot \log n)$ rounds.

We next argue that this is indeed a scheduling of the related communication patterns P_1 to P_k and at the end, each node v knows its output $out_i(v)$ in each protocol $\mathcal{P}_i$.

In particular, we argue that for each node v, if the dilation-neighborhood of v is contained in a cluster completely, then the run of each protocol $\mathcal{P}_i$ in this cluster will be indistinguishable from running the protocol over the whole network. Notice that by property (3) of the graph partitioning lemma (Lemma 10.4.2), there are many such layers. Also note that since node v knows its covered radius in each layer (as shown by Lemma 10.4.3), node v knows which of the layers satisfy this condition. For each node w in the dilation-neighborhood of v, in each communication pattern P_i, node v causally depends only on the actions of w in the first $h' = \mathsf{dilation} - dist(v, w)$ rounds. Since the dilation-neighborhood of v is fully contained in the cluster, also the h'-neighborhood of w is fully contained in that cluster. Hence, w will not discard its messages in those rounds that can influence v. Moreover, it is easy to see that, for any message in the communication pattern that it is not discarded, none of the previous messages in the same communication pattern that can causally affect it is discarded. Hence, when we run the protocol $\mathcal{P}_i$ only in this cluster, we generate a communication pattern that locally simulates communication pattern P_i for node v. Indeed, this run of the protocol $\mathcal{P}_i$ is indistinguishable for v from running the protocol over the whole network and thus, v learns its output $out_i(v)$. □

Improved Scheduling Algorithm We now explain how to improve the schedule length to the near-optimal bound of $O(\mathsf{congestion}+\mathsf{dilation}\cdot\log n)$ rounds. The key point we leverage is the fact that we need only one copy of each message, instead of all the potentially up to $O(\log n)$ copies in the basic solution, one per layer. The main ingredient in our approach for utilizing this point is a well-crafted nonuniform distribution of the random delays. We next describe this distribution, and then explain how it affects the schedule.

The probability mass function of the random delay is as follows: We have a parameter $B = \Theta(\frac{\mathsf{congestion}}{\log n})$. The distribution is over positive integers in the range $[1, \Theta(\frac{\mathsf{congestion}}{\log n})]$. The support of the distribution is divided to $\beta = \Theta(\log n)$ *blocks*, each being a set of consecutive positive integers. The total probability mass given to each block is $1/\beta$. The i^{th} block contains $B\alpha^{i-1}$ integers, and the distribution of the probability mass inside the block is uniform, that is, each number in the i^{th} block has probability $\frac{1/\beta}{B\alpha^{i-1}}$. Here, α is a positive constant slightly less than 1, which we fix later. Note that the total support of the distribution has $\sum_{i=1}^{\Theta(\log n)} B\alpha^{i-1} \leq \frac{B}{1-\alpha} = \Theta(\frac{\mathsf{congestion}}{\log n})$ numbers.

Above, we described the probability distribution for the initial delay in each communication pattern. Once a communication pattern starts, it proceeds synchronously at the speed of one communication pattern round per each big-round, similar to the basic algorithm. However, there is one more change: this time, if there is a message scheduled to be sent over e and a copy of it has been sent before, this message gets dropped. On the other hand, when a node is about to create a message when simulating a round j of a communication pattern, it takes into account all the messages that it has received in the past about rounds up to $j-1$ of the simulations of the same communication pattern, and uses them to create this

message for simulating round j.

Lemma 10.4.7. *Given the local randomness sharing as done in Lemma 10.4.4, the ISA algorithm schedules the communication patterns P_1 to P_k in $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds. Furthermore, at the end of this algorithm, each node v learns its output $out_i(v)$ for each protocol $\mathcal{P}_i$.*

Proof of Lemma 10.4.7. The argument showing that the above procedure produces a scheduling of the communication patterns P_1 to P_k, and that is allows each node v to learn its output $out_i(v)$ in each protocol $\mathcal{P}_i$, is similar to the analogous argument in the claim above about the basic algorithm.

What is left to show is that with the new distribution of delays, w.h.p., for each big-round, there are at most $\Theta(\log n)$ messages that need to go through an edge e. Note that a message must go through e in a given big-round if no copy of it has done so in the past, that is, if among the $\Theta(\log n)$ copies of the simulations of the corresponding communication pattern, this is the first execution scheduled. For a copy of a communication pattern to be the first scheduled, all the $\Theta(\log n)$ copies must be scheduled afterward. We show that for each big-round and each edge, the probability that the first copy of a message is scheduled to go through this edge in that big-round is at most $\Theta(\frac{\log n}{\mathsf{congestion}})$.

The above claim is trivially true for delays in the first block of distribution of the delays as the probability for each of those delays is $\frac{1}{B\beta}$. For the second block, the probability that none of the $\Theta(\log n)$ copies of the communication pattern have a delay in the first block is $(1-\frac{1}{\beta})^{\Theta(\log n)}$ which is a constant, say $\gamma < 1$. It is sufficient to pick the constant α equal to this constant γ. Then, the probability that a delay that is in the second block is chosen and is the first among the delays is at most $\frac{\gamma/\beta}{B\alpha} \leq \frac{1}{B\beta}$. Generally, for a delay value in the i^{th} block for $i \geq 2$, the probability that this value is the first is at most $(1-\frac{1}{\beta})^{(i-1)\Theta(\log n)} \leq \gamma^{i-1}$, and hence, the probability that it is chosen and is the first among the related delays is at most $\frac{\gamma^{i-1}/\beta}{B\alpha^{i-1}} \leq \frac{1}{B\beta} = \Theta(\frac{\log n}{\mathsf{congestion}})$. Hence, we conclude that the total number of messages to be sent over each edge per big-round is at most $\Theta(\log n)$, with high probability. Since each big-round is made of $\Theta(\log n)$ rounds, there is enough time to send all these messages across the edge in that big-round. Note that the range of the initial delays is $\Theta(\frac{\mathsf{congestion}}{\log n})$ big-rounds, and that each communication pattern runs for only $\mathsf{dilation}$ big-rounds. Hence, the whole schedule has a length $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds, thus completing the proof of the lemma. □

Finally, we point out that Theorem 10.1.4 follows from putting the three algorithmic parts explained above together:

Theorem 10.1.4 *There is a distributed algorithm that uses only private randomness and for any instance of the distributed protocol scheduling problem, computes a schedule of length $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$, in $O(\mathsf{congestion} + \mathsf{dilation} \log^2 n)$ rounds, with high probability. In particular, given a collection $\mathcal{P}_1$ to $\mathcal{P}_k$ of protocols, and the corresponding fixed executions*

α_1 to α_k as discussed above, this algorithm allows each node v to learn its output $out_i(v)$ in execution α_i for each $i \in [1, k]$.

Proof. First, we use the GPA algorithm to partition the graph. We then run the LCR algorithm to make each node learn its covered radii. As Lemma 10.4.2 and Lemma 10.4.3 show, these two steps take $O(\mathsf{dilation} \log^2 n)$ rounds, each.

Then, we run the RSA algorithm to broadcast randomness in each of the clusters of the graph partitioning. As Lemma 10.4.4 shows, this takes $O(\mathsf{dilation} \log^2 n)$ additional rounds. Then, we use Lemma 10.4.5 to transform these shared bits of randomness into poly(n)-length sequences of randomness. Notice that this step is a local computation and does not incur any round complexity.

Finally, we apply the ISA algorithm to use these shared randomness sequences to run all protocols $\mathcal{P}_1$ to $\mathcal{P}_k$ concurrently. As Lemma 10.4.7 shows, this algorithm schedules the communication patterns P_1 to P_k in $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds. Furthermore, at the end, each node v learns its output $out_i(v)$ for each protocol $\mathcal{P}_i$.

Taking all these steps into account, the overall round complexity of this algorithm is $O(\mathsf{congestion} + \mathsf{dilation} \log^2 n)$ rounds. □

10.5 A Recipe for Removing Shared Randomness in Bellagio Distributed Algorithms

In this section, we sketch out how to use the general approach of the previous section to remove the assumption of having shared randomness in a broad range of distributed protocols, at the cost of a small slow-down factor.

Generally, having *shared randomness* among all nodes of the network is a far stronger assumption than assuming that each node has its own *private randomness*. Thus, it is more ideal to have distributed algorithms that do not to rely on such the assumption of having shared randomness. Here, we present an informal sketch that explains to remove the assumption of having shared randomness in a large class of randomized distributed algorithms, by using the approach that we presented in Section 10.4. We note that to maintain the generality of the approach, we present only an informal sketch. This should be regarded more as a suggestive general recipe, rather than a precise and detailed algorithm for each case. Applying this general recipe to each concrete algorithm might require adjustments in the details.

The class of the algorithm that our approach applies to is what Goldwasser recently named Bellagio algorithms [Gol12]. We note that although the name is new, a wide range of classical and old randomized algorithms fall into this category. The definition, translated to distributed algorithms, is as follows: for each fixed set of inputs to the nodes, each node must have a *canonical output value* and it should output this value in at least 2/3

of the executions, that is, with probability at least 2/3. Note that in many problems, the correct output for each node is unique. In such cases, any randomized algorithm will indeed be a Bellagio algorithm. In a few other problems, although a priori there is no unique correct output, the problem can be changed slightly, by adding certain restrictions, to make the output unique without affecting the problem's complexity significantly. Again, in such cases, the randomized algorithms would be necessarily Bellagio and thus our scheme would be applicable.

On the other hand, we note that there are classical distributed graph problems for which it is not clear how to obtain an efficient Bellagio algorithm. For instance, obtaining a poly $\log n$-round Bellagio algorithm for computing a Maximal Independent Set remains open[12].

The reason that our approach applies only to Bellagio algorithms is as follows: we try to simulate the algorithm with shared randomness by only locally sharing randomness, as we did in Section 10.4. Thus, we will have many partial executions of the algorithm, each cut to a small local area. The Bellagio property allows us to *paste* these together and make sure that the outputs of these locally-cut executions are consistent.

Meta-Theorem 10.5.1. *For any problem that has a T-round Bellagio randomized distributed algorithm which uses R bits of shared randomness and where each node outputs a canonical solution with probability at least 2/3, there is a randomized algorithm with round complexity $O(T \log^2 n + R)$ that uses only private randomness and w.h.p. each node outputs its canonical output. Furthermore, if the input given to each node can be described using $\text{poly}(n)$ bits, a different technique can be used to reduce R to $O(\log n)$, thus giving an $O(T \log^2 n)$ round algorithm.*

Since this is cast as a meta-theorem rather than a concrete and precise theorem, in the following, we explain a general recipe for it, instead of providing a formal proof. This explanation uses a few examples to make some points concrete.

We first use an example to explain the first part of the meta-theorem for removing shared randomness and running the algorithm in $O(T \log^2 n + R)$ rounds. Then, we describe the general approach for the second part which reduces the amount of the shared randomness in the original shared-randomness Bellagio algorithm to $O(\log n)$, in most cases of interest, while keeping it Bellagio. Simulating this new algorithm via our techniques in Lemmas 10.4.2, 10.4.4, and 10.4.7, leads to the claimed round complexity of $O(T \log^2 n)$ round.

An example for removing shared randomness One of the key multi-party computation operations in which shared randomness gets used frequently is hashing. Many variants of hashing are used in different algorithms, but typically the hash function is constructed via a randomness seed that comes from shared randomness. We now explain how this shared randomness assumption can be removed. To have a concrete example, we discuss a very

[12] It is not hard to see that a Bellagio randomized distributed algorithm with round complexity poly $\log n$ for computing an MIS can be transformed to a deterministic distributed algorithm for MIS with round complexity poly $\log n$. Obtaining the latter remains among the most well-known open problems of the area.

simple case of *dimensionality reduction* via hashing. We present only a rough sketch which illustrates the main idea without getting caught up in the details of this particular example.

Suppose each node $v \in V$ receives as input a string $s_v \in \{0,1\}^L$ for $L = \text{poly}(n)$, and the objective is for each node v to know the number of distinct strings within d-hops of v, to within a $(1+\varepsilon)$ factor, for a small $\varepsilon > 0$. Note that even two neighbors exchanging their strings would require $L = \text{poly}(n)$ rounds, so intuitively a first natural step is to reduce the dimension $L$of the problem, and this we do via a simple hashing.

Using shared randomness, nodes can pick a random hash function $h : \{0,1\}^L \to \{0,1\}^{\Theta(\log n)}$ at random, which would give the property that among the at most n strings in the graph, w.h.p., there is no hash-collision. This dimension reduction to $O(\log n)$ already captures the usage of shared randomness that we wanted to illustrate and it opens the road for computing the number of distinct elements via standard algorithm. Later, we will talk more about the details of how to actually solve this approximate distinct elements problem, but let us here focus more on the heart of the story, that is, how to mimic the shared randomness via only local sharing

To keep the hash function collision free w.h.p., pairwise independence is enough. This implies that sharing $O(\log n)$ bits of randomness is sufficient in this example. Now, this solution relies on that for each node v, all nodes within its h-neighborhood have picked the same hash function, that is, the same $\Theta(\log n)$ bits of hashing seed. Obviously we cannot have the same seed of randomness in all nodes in $o(D)$ rounds. We use the method based on graph partitioning here.

Using the graph partitioning algorithm explained above and abstracted by Lemma 10.4.2, we can carve $\Theta(\log n)$ layers of clustering, where each cluster has radius $\Theta(d \log n)$, and such that each node's h-neighborhood is fully contained in $\Theta(\log n)$ of such balls. This takes $\Theta(d \log^2 n)$ rounds. Then, we share $\Theta(\log n)$ bits of the randomness in each cluster picked by the center, similar to Lemma 10.4.4, in $\Theta(d \log n)$ rounds. In fact here, in this simple example where we want $\Theta(\log n)$ bits, this sharing can be done at the same time as we are carving the clusters. But in general when $R = \omega(\log n)$, we would this randomness sharing after the graph paritioning algorithm, similar to what we did above in Lemma 10.4.4. This would take $O(d \log n + R)$ rounds. Finally, we can simulate the hash functions locally for each of the clusters. Hence, the hash function construction part takes $O(h \log^2 n + R)$ rounds.

Note that each node will be in $\Theta(\log n)$ clusters and thus will have $\Theta(\log n)$ hash functions, one for each cluster layer. Hence, given an algorithm $\mathcal{P}$ that uses the hash-functions to solve the problem, we still need to simulate $\mathcal{P}$ for each of these cluster layers, as we did in Lemma 10.4.7. But if $\mathcal{P}$ takes T rounds, we can similarly simulate $\mathcal{P}$ also in local clustering areas, each of diameter at most $O(T \log n)$, in total in $O(T \log^2 n)$ time.

Before going to the issue of reducing shared randomness, since we have started the discussion about the problem of distinct elements, let us first finish this discussion. We explain how to approximate it to within $1+\varepsilon$, in $O(d \log n/\varepsilon^3)$ rounds, and just present a rough sketch: Consider a threshold $k = (1+\varepsilon)^j$. We can compare the number of distinct elements in each node's h-hop neighborhood with this threshold k as follows: Use a hash function

$h'_1 = \{0,1\}^{\Theta(\log n)} \to \{0,1\}$ where for each $s \in \{0,1\}^{\Theta(\log n)}$, $\Pr[\forall s \in h'_1(s) = 1] = 1 - 2^{-1/k} \approx 1/k$, and these events are independent among different k. Then, in d rounds, each node v can know whether there is a node u in its d-hop neighborhood for which $h'_1(h(s_u)) = 1$. One can see that if the number of distinct elements in the d-hop neighborhood of v is at least $(1+\varepsilon/2)k$, then the probability that there is at least one node node u in its d-hop neighborhood for which $h'_1(h(s_u)) = 1$ is at least $0.5 + \Theta(\varepsilon)$. Furthermore, if the number is at most $k/(1+\varepsilon/2)$, then the probability is at most $0.5 - \Theta(\varepsilon)$. This $\Theta(\varepsilon)$ gap is the key distinguishing element. If we repeat this process for $\Theta(\log n/\varepsilon^2)$ iterations (different binary hash functions h'_i), Hoeffding's bound tells us that at the end, w.h.p, each node v knows whether the number of distinct elements in its d-neighborhood is above $(1+\varepsilon/2)k$ or below $k/(1+\varepsilon/2)$, as indicated by the majority of the experiments. Repeating this for all the $\Theta(\log n/\varepsilon)$ different thresholds—that is, different values of $j \in \{1, \ldots, \Theta(\log n/\varepsilon)\}$)—node v can know the number of distinct elements in its neighborhood to within $1+\varepsilon$ factor. In fact, each $\Theta(\log n)$ iterations can be bundled together because the CONGEST model admits $O(\log n)$ bit messages, with the adjustment that now a bit-wise OR of the received message is forwarded. This gets us to the bound $O(d \log n/\varepsilon^3)$.

Reducing the Shared Randomness Now we explain a distributed generalization of a classical observation of Newman [New91] for two-party protocols, which shows that $O(\log n)$ bits of shared randomness is sufficient if in the problem, there are at most $2^{\text{poly}(n)}$ possibilities for the input given to each node, that is, if each node's input can be described in $\text{poly}(n)$ bits. Note that almost all problems of interest in theoretical distributed computing fall within this category. For instance, the edges of a node, which are typically a key part of the input, can be described in at most $n \log n$ bits.

An algorithm with R bits of shared randomness is simply a collection $\mathcal{F}$ of 2^R deterministic algorithms. We know that for each fixed set of inputs, in 2/3 or more of the algorithms, node v is outputting the same canonical output. We claim that we can find a smaller collection $\mathcal{F}'$ of these deterministic algorithms, with size $\text{poly}(n)$, such that for each set of fixed input, each node outputs the same canonical output with probability say at least 3/5. The argument is by an application of the probabilistic method: simply pick $\text{poly}(n)$ many of the deterministic algorithms in $\mathcal{F}$ at random and define the resulting collection to be a candidate for being collection $\mathcal{F}'$. Regarding one fixed set of inputs, using the Chernoff bound, we know that the probability that node v outputs the correct canonical value in at least 3/5 of the algorithms in the candidate collection is at least $1 - 2^{-\Theta(\text{poly}(n))}$. Now, there are $2^{\text{poly}(n)}$ sets of possible inputs to all nodes, n nodes, and furthermore at most $2^{O(n^2)}$ possible graphs between the nodes. We can union bound over all of these possibilities and say that the probability that the candidate $\mathcal{F}'$ collection is good for all these choices is at least $1 - 2^{-\Theta(\text{poly}(n))} \times 2^{\text{poly}(n)} \geq 1 - 2^{-\Theta(\text{poly}(n))}$. That is, with very high probability, the candidate collection is good. Hence, in fact there exists such a good smaller collection $\mathcal{F}'$ with $|\mathcal{F}'| = \text{poly}(n)$. Now, note that to pick one of the algorithms in collection $\mathcal{F}'$ only takes

$O(\log n)$ bits.

We note that this argument is simply existential in the sense that it proves that there exists a good collection $\mathcal{F}'$, and thus an algorithm which uses $O(\log n)$ bits of shared randomness. The argument does not provide a fast (centralized) method for finding this algorithm. However, if we ignore the local computations, which is standard in the area of distributed computing [Pel00], nodes can deterministically search through the space of all collections, using a simple deterministic brute force, each running it on its own, and consistently find the first good collection $\mathcal{F}'$. Here, "first" is with respect to the deterministic search order.

10.6 Concluding Remarks and Future Work

This chapter is centered around the issue of running many independent distributed protocol together, and we presented an existential schedule length lower bound of $\Omega(\mathsf{congestion} + \mathsf{dilation} \log n / \log\log)$ as well as an algorithm that produces a schedule with a length almost matching this lower bound, that is, $O(\mathsf{congestion}+\mathsf{dilation} \log n)$ rounds, after $O(\mathsf{dilation} \log^2 n)$ rounds of pre-computation.

We believe that this is merely a starting point, as there are many seemingly fundamental aspects that are not addressed here. Next, we first discuss some detailed technical questions about the bound, and then, we discuss some more speculative and conceptually deeper issues.

Detailed Questions

One particular question is rooted in the fact that the lower bound we presented is existential and shows the existence of some hard instances for the scheduling problem in which one cannot obtain $O(\mathsf{congestion} + \mathsf{dilation})$ round schedules. Is it possible to provide a necessary and sufficient classification of special cases of distributed protocol scheduling problems for which $O(\mathsf{congestion} + \mathsf{dilation})$ round schedules are admissible? Alternatively, can we get at least a sufficient condition that covers a broad range of problems of interest? If yes, for those, can we also find these schedules distributedly?

Another question is regarding Theorem 10.1.4. Recall that this theorem provides a schedule of length $O(\mathsf{congestion}+\mathsf{dilation} \log n)$, which is nearly optimal as Theorem 10.1.3 implies, but it uses $O(\mathsf{dilation} \log^2 n)$ rounds of pre-computation to find this schedule. Can we improve the latter bound, ideally to $O(\mathsf{dilation} \log n)$ rounds? Such an improvement would mean that the whole pre-computation and scheduling is done in $O(\mathsf{congestion} + \mathsf{dilation} \log n)$ rounds?

Broader and More Speculative Questions

Throughout the chapter, we worked under the assumption that the protocols $\mathcal{P}_1$ to $\mathcal{P}_k$ are fixed, and we just want to run them. However, one can imagine a different viewpoint, when considering a higher-level picture: The end goal that we have is to solve the problems that $\mathcal{P}_1$ to $\mathcal{P}_k$ were designed to solve, which means we are allowed to change these protocols to

make them fit with each other better, so long as they solve the same problems. In other words, roughly speaking, this coincides with the question of, given k problems, what is the best collection of protocols $\mathcal{P}_1$ to $\mathcal{P}_k$ that produce the minimum congestion and dilation. To make the objective single-measure, one might consider congestion + dilation $\cdot \log n$ as one possibly good objective function to be minimized. In fact, once we design a set of protocols optimizing this measure, then we can use the algorithms presented in this chapter to run these protocols essentially optimally, which would be a near-optimal method overall. Hence, one can say that our algorithmic results provide the following corollary: they essentially (and approximately) reduce the general question of how to solve k problems together to that of designing a set of k protocols that minimize the measure congestion + dilation $\cdot \log n$.

This design question seems rather broad and it is not completely clear how to tackle such a question for general choice of k problems. A special but also cleaner case, which might provide us with valuable insights, is the setting where the k problems are independent instances of the same problem. Here, the problem statement becomes cleaner in the sense that, we have k independent shots of a single problem, and we want to solve all of them. We note that this question loosely resembles the well-studied *parallel repetitions* problem (see e.g. [Raz98]) in the complexity theory which, roughly speaking, asks can one solve k independent shots of a given problem using "resources" less than k times that of the single-shot version. However, the author believes that this is rather a superficial similarity and the connection is not strong, mainly because our focus is on the distributed round complexity and we clearly know that pipelining often allows us to solve k shots faster than k times the round complexity of solving a single shot. Although, there might be other less-direct connections between the two areas; one possible direction is via using *direct product* or *direct sum theorems* in communication complexity to derive distributed round complexity lower bounds for k shots of a given problem.

Going back to the distributed k-shot question, let us use a classical (single-shot) problem as an example. Consider the k-shot version of the MST problem: For the network graph $G = (V, E)$, we are given k different weight functions w_1 to w_k, each $w_i : E \to \mathbb{R}$, and we want to find the k Minimum Spanning Trees corresponding to these weight functions. One can see that for this setting, naively running the best single-shot protocol k times, even when allowing the best possible pipelining of these k protocols, is not the optimal. Generally, one can easily see that, the best protocol that is to be run k times for the k-shot problem (run in parallel) is not the same as the best single-shot protocol. The core of the matter is that, the single-shot protocol is designed to minimize its running time—that is, dilation in our terminology—but it typically does not lead to the best congestion.

To make the discussion concrete, let us consider the MST problem again. For MST, the almost a century-old protocol of Boruvka [NMN01], which has the same outline as the one used by Gallager, Humblet, and Spira [GHS83], has dilation $\tilde{O}(n)$ rounds and running it once gives a very low congestion of $O(\log n)$. On the other hand, an alternative near-linear time protocol can be achieved by filtering edges—discarding heaviest edge in each cycle—while they are being upcast on a tree towards the root, which leads to dilation and congestion

both being $\tilde{O}(n)$. Furthermore, the newer protocol of Kutten and Peleg [KP95], has an almost optimal running time of $\mathsf{dilation} = \tilde{O}(D + \sqrt{n})$ rounds, but at the cost of having $\mathsf{congestion} = \Theta(\sqrt{n})$. This is a simple example showing that the protocols designed to have optimal dilation do not necessarily have good congestion. For the case of the MST, we can actually understand the tradeoff between these two parameters quite well. Using constructs similar to those of Das Sarma et al. [DSHK+11], one can see that there is indeed an inevitable tradeoff and an MST protocol that has congestion at most L will require a dilation of at least $\tilde{O}(n/L)$ rounds. Interestingly, one can algorithmically match this

$$\mathsf{congestion} = L, \mathsf{dilation} = \tilde{O}(D + n/L)$$

tradeoff by a simple parameter tuning in the protocol of Kutten and Peleg [KP95]. For the k-shot version, the aforementioned lower bound can be combined with a *strong direct sum theorem* for communication complexity of k independent shots of set disjointness [Kla10], and some parameter tuning, to show that solving k-shots of MST requires at least $\Omega(D + \sqrt{kn})$ rounds. Again, this can be matched: we use the parameter-optimized protocol of Kutten and Peleg, with the optimal choice $L = \sqrt{n/k}$, as our single-shot protocol, and we run k copies of it in parallel using our $O(\mathsf{congestion} + \mathsf{dilation} \log n)$ round schedule. As a result, we get a protocol that solves k independent instances of MST in $\tilde{O}(D + \sqrt{nk})$ rounds, thus essentially matching the lower bound.

Perhaps it is simply a coincidence that in the case of MST, we already know how to solve k-shots of it optimally, simply by doing some parameter tuning in the single-shot protocol and running k copies of it in parallel. Can we achieve such a result for a broader family of protocols (hopefully, ones which have a different "nature" compared to that of MST [13])?

To summarize this point, we believe that when designing distributed protocols, it is valuable to keep track of both dilation and congestion, as opposed to merely dilation which is the standard objective in the recent years of theoretical distributed computing. This is because, when the protocol is not run alone, congestion is one parameter that gives us a sense of what will happen when we run this protocol along with others. It would be ideal if one can also have a protocol with a controllable tradeoff between the two parameters, as we saw above for the MST problem. Furthermore, this tradeoff deserves a study from a lower bound viewpoint, and even congestion deserves a study of its own; given a problem, can we solve it with a congestion below the given threshold? We note that, a measure that received more attention in the old days of theoretical distributed computing was *message complexity*. In fact message complexity has some correlation with congestion; however, it is quite easy to see that there is no tight relation and small message complexity does not necessarily imply a small congestion. For instance, a protocol with message complexity $O(m)$ can have congestion anywhere between $O(1)$ to $O(m)$.

[13] For instance, a k-shot version of minimum cut approximation can also be computed in $\Omega(D + \sqrt{kn})$ rounds, by extending [GK13]. This bound is also is optimal due to essentially the same lower bound constructions. However, this is not really a novel addition but merely an extension of the same tradeoff.

Bibliography

[ABCP92] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Fast network decomposition. In *Proc. 11th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 169–177, 1992.

[ABCP96] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Fast distributed network decompositions and covers. *Journal of Parallel and Distributed Computing*, 39(2):105–114, 1996.

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986.

[AC04] A. Agarwal and M. Charikar. On the advantage of network coding for improving network throughput. In *Information Theory Workshop, 2004. IEEE*, pages 247–249. IEEE, 2004.

[ACF+03] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. Optimal oblivious routing in polynomial time. In *Proc. 35th ACM Symp. on Theory of Computing (STOC)*, pages 383–388, 2003.

[ACLY00] R. Ahlswede, N. Cai, S. Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[AdHV99] Friedhelm Meyer Auf der Heide and Berthold Vöcking. Shortest-path routing in arbitrary networks. *Journal of Algorithms*, 31(1):105–131, 1999.

[AGHK13] Noga Alon, Mohsen Ghaffari, Bernhard Haeupler, and Majid Khabbazian. Broadcast throughput in radio networks: routing vs. network coding. *Manuscript*, 2013.

[AGHK14] Noga Alon, Mohsen Ghaffari, Bernhard Haeupler, and Majid Khabbazian. Broadcast throughput in radio networks: routing vs. network coding. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1831–1843, 2014.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design & Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[ALGP89] Baruch Awerbuch, M. Luby, A.V. Goldberg, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. 30th IEEE Symp. Foundations of Computer Science (FOCS)*, pages 364–369, 1989.

[AMS06] Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, April 2006.

[ARVX12] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1132–1139, 2012.

[AS04] Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

[AWF02a] Khaled M Alzoubi, Peng-Jun Wan, and Ophir Frieder. Message-optimal connected dominating sets in mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 157–164. ACM, 2002.

[AWF02b] Khaled M Alzoubi, Peng-Jun Wan, and Ophir Frieder. New distributed algorithm for connected dominating set in wireless ad hoc networks. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3849–3855. IEEE, 2002.

[Bar95] Francisco Barahona. Packing spanning trees. *Mathematics of Operations Research*, 20(1):104–115, 1995.

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 161–168, 1998.

[Bar04] Yair Bartal. Graph decomposition lemmas and their role in metric embedding methods. In *Algorithms–ESA 2004*, pages 89–97. Springer, 2004.

[BDTC05] Jeremy Blum, Min Ding, Andrew Thaeler, and Xiuzhen Cheng. Connected dominating set in sensor networks and manets. In *Handbook of Combinatorial Optimization*, pages 329–369. Springer, 2005.

[BE10] Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.

[BE13] Leonid Barenboim and Michael Elkin. Distributed graph coloring: Fundamentals and recent developments. *Synthesis Lectures on Distributed Computing Theory*, 4(1):1–171, 2013.

[Bec91] József Beck. An algorithmic approach to the lovász local lemma. I. *Random Structures & Algorithms*, 2(4):343–365, 1991.

[BEPSv3] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *Foundations of Computer Science (FOCS) 2012*, pages 321–330. IEEE, 2012, also coRR abs/1202.1983v3.

[BG99] Dimitris Bertsimas and David Gamarnik. Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[BK96] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proc. 28th ACM Symp. on Theory of Computing (STOC)*, pages 47–55, 1996.

[BKP14] Tushar Bisht, Kishore Kothapalli, and Sriram Pemmaraju. Brief announcement: Super-fast t-ruling sets. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 379–381. ACM, 2014.

[BKR03] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proc. 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 24–33, 2003.

[BM08] J Bondy and U Murty. Graph theory (graduate texts in mathematics, vol. 244), 2008.

[Bol98] Béla Bollobás. *Random graphs*. Springer, 1998.

[BR06] Béla Bollobás and Oliver Riordan. *Percolation*. Cambridge University Press, 2006.

[BRS94] Bonnie Berger, John Rompel, and Peter W. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 454–477, 1994.

[BYCHGS16] Reuven Bar-Yehuda, Keren Censor-Hillel, Mohsen Ghaffari, and Gregory Schwartzman. Fast distributed approximation of maximum independent set and maximum matching. manuscript, 2016.

[BYCHS16] Reuven Bar-Yehuda, Keren Censor-Hillel, and Gregory Schwartzman. A distributed $(2+\epsilon)$-approximation for vertex cover in $O(\log \Delta/\epsilon \log\log \Delta)$ rounds. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, 2016.

[CGG+15] Keren Censor-Hillel, Mohsen Ghaffari, George Giakkoupis, Bernhard Haeupler, and Fabian Kuhn. Tight bounds on vertex connectivity under vertex sampling. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2006–2018, 2015.

[CHGK13] K. Censor-Hillel, M. Ghaffari, and F. Kuhn. A New Perspective on Vertex Connectivity. *arXiv, http://arxiv.org/abs/1304.4553*, 2013.

[CHGK14a] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 156–165, 2014.

[CHGK14b] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. A new perspective on vertex connectivity. In *Proc. 25th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2014.

[CHL+03] Xiuzhen Cheng, Xiao Huang, Deying Li, Weili Wu, and Ding-Zhu Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks*, 42(4):202–208, 2003.

[CHPS16] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *arXiv preprint arXiv:1608.01689*, 2016.

[CKP16] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the local model. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, page to appear, 2016.

[CKR05] Gruia Calinescu, Howard Karloff, and Yuval Rabani. Approximation algorithms for the 0-extension problem. *SIAM Journal on Computing*, 34(2):358–372, 2005.

[CL01] Fan Chung and Linyuan Lu. The diameter of random sparse graphs. *Advances in Applied Math*, 26(4):257–279, 2001.

[CL02] Yuanzhu Peter Chen and Arthur L Liestman. Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 165–172. ACM, 2002.

[CM88] Joseph Cheriyan and SN Maheshwari. Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs. *Journal of Algorithms*, 9(4):507–537, 1988.

[Coo83] Stephen A Cook. An overview of computational complexity. *Communications of the ACM*, 26(6):400–408, 1983.

[CP11] A. Chattapodhyay and T. Pitassi. The story of set disjointness. *SIGACT News Complexity Theory Column*, 67:59–85, 2011.

[CPS14] Kai-Min Chung, Seth Pettie, and Hsin-Hao 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. ACM, 2014.

[CV00] Robert Carr and Santosh Vempala. Randomized metarounding (extended abstract). In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 58–62, New York, NY, USA, 2000. ACM.

[CWD08] Xiuzhen Cheng, Feng Wang, and Ding-Zhu Du. Connected dominating set. In *Encyclopedia of Algorithms*, pages 1–99. Springer, 2008.

[DB97] Bevan Das and Vaduvur Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *Communications, 1997. ICC 97 Montreal,'Towards the Knowledge Millennium'. 1997 IEEE International Conference on*, volume 1, pages 376–380. IEEE, 1997.

[DGG+13] Sebastian Daum, Mohsen Ghaffari, Seth Gilbert, Fabian Kuhn, and Calvin Newport. Maximal independent sets in multichannel radio networks. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 335–344, 2013.

[DHK+12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. on Comp.*, 41(5):1235–1265, 2012.

[DMP+03] Devdatt Dubhashi, Alessandro Mei, Alessandro Panconesi, Jaikumar Radhakrishnan, and Arvind Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 717–724, 2003.

[DP09] D.P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[DR96] Devdatt P Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996.

[DSHK+11] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 363–372, 2011.

[DvM] Holger Dell and Dieter van Melkebeek. Lecture notes on k-wise uniform (randomness) generators. http://pages.cs.wisc.edu/~dieter/Courses/2013s-CS880/Scribes/PDF/lecture04.pdf. Accessed: 2015-02.

[DW04] Fei Dai and Jie Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *Parallel and Distributed Systems, IEEE Transactions on*, 15(10):908–920, 2004.

[EFS56] P. Elias, A. Feinstein, and C. E. Shannon. Note on maximum flow through a network. *IRE Transactions on Information Theory IT-2*, pages 117–199, 1956.

[EKV13] Alina Ene, Nitish Korula, and Ali Vakilian. Connected domatic packings in node-capacitated graphs. *http://arxiv.org/abs/1305.4308/v2*, 2013.

[Elk04a] M. Elkin. Distributed approximation: a survey. *SIGACT News*, 35(4):40–57, 2004.

[Elk04b] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 359–368. Society for Industrial and Applied Mathematics, 2004.

[Elk04c] Michael Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 331–340, 2004.

[Eve75] Shimon Even. An algorithm for determining whether the connectivity of a graph is at least k. *SIAM Journal on Computing*, 4(3):393–396, 1975.

[Fei96] Uriel Feige. A threshold of $\ln n$ for approximating set cover (preliminary version). In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 314–318, 1996.

[FF56] L. R. Ford and D. R. Fulkersonn. Maximal flow through a network. *Canad. J. Math.*, 8:399–404, 1956.

[FF62] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 2010 (first published in 1962).

[FHK00] Uriel Feige, Magnús M. Halldórsson, and Guy Kortsarz. Approximating the domatic number. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 134–143, 2000.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[FS07] Christina Fragouli and Emina Soljanin. Network coding fundamentals. *Foundations and Trends in Networking*, 2(1), 2007.

[FW90] Michael L. Fredman and D.E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. In *Proc. 31st IEEE Symp. Foundations of Computer Science (FOCS)*, 1990.

[Gab91] Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, STOC '91, pages 112–122, New York, NY, USA, 1991. ACM.

[Gab00] H.N. Gabow. Using expander graphs to find vertex connectivity. In *Proc. 41st IEEE Symp. Foundations of Computer Science (FOCS)*, pages 410–420, 2000.

[Gal80] Zvi Galil. Finding the vertex connectivity of graphs. *SIAM Journal on Computing*, 9(1):197–199, 1980.

[GG11] Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 18, page 136, 2011.

[GGLS14] Rati Gelashvili, Mohsen Ghaffari, Jerry Li, and Nir Shavit. On the importance of registers for computability. In *International Conference on Principles of Distributed Systems*, pages 171–185. Springer, 2014.

[GGNT12] Mohsen Ghaffari, Seth Gilbert, Calvin Newport, and Henry Tan. Optimal broadcast in shared spectrum radio networks. In *International Conference On Principles Of Distributed Systems*, pages 181–195. Springer, 2012.

[GH13a] Mohsen Ghaffari and Bernhard Haeupler. Fast structuring of radio networks for multi-message communications. In *Proc. of the International Symposium on Distributed Computing (DISC)*, 2013.

[GH13b] Mohsen Ghaffari and Bernhard Haeupler. Near optimal leader election in multi-hop radio networks. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 748–766, 2013.

[GH14] Mohsen Ghaffari and Bernhard Haeupler. Optimal error rates for interactive coding II: Efficiency and list decoding. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 394–403. IEEE, 2014.

[GH16a] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks i: Planar embedding. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, 2016.

[GH16b] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks ii: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2016.

[Gha14] Mohsen Ghaffari. Near-optimal distributed approximation of minimum-weight connected dominating set. In *the Pro. of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2014.

[Gha15a] Mohsen Ghaffari. Distributed broadcast revisited: Towards universal optimality. In *the Pro. of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 638–649, 2015.

[Gha15b] Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 3–12, 2015.

[Gha16a] Mohsen Ghaffari. Faster interactive welfare maximization, and distributed MIS in congested clique. manuscript, 2016.

[Gha16b] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 270–277, 2016.

[GHK13] Mohsen Ghaffari, Bernhard Haeupler, and Majid Khabbazian. Randomized broadcast in radio networks with collision detection. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 325–334, 2013.

[GHLN12] Mohsen Ghaffari, Bernhard Haeupler, Nancy Lynch, and Calvin Newport. Bounds on contention management in radio networks. In *Proc. of the International Symposium on Distributed Computing (DISC)*, pages 223–237, 2012.

[GHS83] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and systems (TOPLAS)*, 5(1):66–77, 1983.

[GHS14] Mohsen Ghaffari, Bernhard Haeupler, and Madhu Sudan. Optimal error rates for interactive coding I: Adaptivity and other settings. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 794–803. ACM, 2014.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[GK98] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, April 1998.

[GK99] Sudipto Guha and Samir Khuller. Improved methods for approximating node weighted steiner trees and connected dominating sets. *Information and computation*, 150(1):57–74, 1999.

[GK08] Ashish Goel and Sanjeev Khanna. On the network coding advantage for wireless multicast in euclidean space. In *Proc. 7th Conf. on Inf. Processing in Sensor Netw. (IPSN)*, pages 64–69, 2008.

[GK13] M. Ghaffari and F. Kuhn. Distributed minimum cut approximation. In *Proc. of the International Symposium on Distributed Computing (DISC)*, pages 1–15, 2013.

[GKK10] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Graph sparsification via refinement sampling. *arXiv preprint arXiv:1004.4915*, 2010.

[GKK+15] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 81–90, 2015.

[GKLN14] Mohsen Ghaffari, Erez Kantor, Nancy Lynch, and Calvin Newport. Multi-message broadcast with abstract mac layers and unreliable links. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 56–65, 2014.

[GKP93] J.A. Garay, S. Kutten, and D. Peleg. A sub-linear time distributed algorithm for minimum-weight spanning trees. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 1993.

[GKPar] Mohsen Ghaffari, David Karger, and Debmalya Panigrahi. Random contractions and sampling for hypergraph and hedge connectivity. Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), 2017 (to appear).

[GKS16] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Generalizing the congested clique. manuscript, 2016.

[GL14] Mohsen Ghaffari and Chistoph Lenzen. Near-optimal distributed tree embedding. In *Proc. of the International Symposium on Distributed Computing (DISC)*, 2014.

[GLN13] Mohsen Ghaffari, Nancy Lynch, and Calvin Newport. The cost of radio network broadcast for different models of unreliable links. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 345–354, 2013.

[GLS11] Mohsen Ghaffari, Nancy Lynch, and Srikanth Sastry. Leader election using loneliness detection. In *Proc. of the International Symposium on Distributed Computing (DISC)*, pages 427–450, 2011.

[GMRL15] Mohsen Ghaffari, Cameron Musco, Tsvetomira Radeva, and Nancy Lynch. Distributed house-hunting in ant colonies. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 57–66, 2015.

[GN16a] Mohsen Ghaffari and Calvin Newport. How to discreetly spread a rumor in a crowd. In *Proc. of the International Symposium on Distributed Computing (DISC)*, 2016.

[GN16b] Mohsen Ghaffari and Calvin Newport. Leader election in unreliable radio networks. In *the Pro. of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2016.

[Gol12] Shafi Goldwasser. Pseudo-deterministic algorithms. In *STACS'12 (29th Symposium on Theoretical Aspects of Computer Science)*, volume 14, pages 29–29. LIPIcs, 2012.

[GP16a] Mohsen Ghaffari and Merav Parter. MST in log-star rounds of congested clique. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, 2016.

[GP16b] Mohsen Ghaffari and Merav Parter. Near-optimal distributed algorithms for fault-tolerant tree structures. In *the Proceedings of the Symposium on Parallel Algorithms and Architectures*, 2016.

[GP16c] Mohsen Ghaffari and Merav Parter. A polylogarithmic gossip algorithm for plurality consensus. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, 2016.

[GSar] Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), 2017 (to appear).

[GW88] Harold Gabow and Herbert Westermann. Forests, frames, and games: algorithms for matroid sums and applications. In *Proc. 20th ACM Symp. on Theory of Computing (STOC)*, pages 407–421, 1988.

[Hae11] Bernhard Haeupler. Analyzing network coding gossip made easy. In *Proc. 43rd ACM Symp. on Theory of Computing (STOC)*, pages 293–302, 2011.

[Hen97] Monika Rauch Henzinger. A static 2-approximation algorithm for vertex connectivity and incremental approximation algorithms for edge and vertex connectivity. *J. Algorithms*, 24(1):194–220, 1997.

[HF01] Bert L. Hartnell and Douglas F.Rall. Connected domatic number in planar graphs. *Czech. Math. Journal*, 51:173–179, 2001.

[HHR03] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proc. 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 34–43, 2003.

[HKRL07] Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, Harald Räcke, and Tom Leighton. Oblivious routing on node-capacitated and directed graphs. *ACM Trans. Algorithms*, 3(4), November 2007.

[HL83] S. T. Hedetniemi and R.C. Laskar. Connected domination in graphs. Technical Report 414, Clemson University, SC, USA, 1983.

[HL16] Stephan Holzer and Nancy Lynch. Brief announcement - beeping a maximal independent set fast. In *Proc. of the International Symposium on Distributed Computing (DISC)*, page to appear, 2016.

[HRG96] Monika Rauch Henzinger, Satish Rao, and Harold N Gabow. Computing vertex connectivity: new bounds from old techniques. In *Proc. 37th IEEE Symp. Foundations of Computer Science (FOCS)*, pages 462–471. IEEE, 1996.

[HW12] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 355–364, 2012.

[IR88] Alon Itai and Michael Rodeh. The multi-tree approach to reliability in distributed networks. *Information and Computation*, 79(1):43–59, 1988.

[JRS01] Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 32–42, 2001.

[Kan90] Arkady Kanevsky. On the number of minimum size separating vertex sets in a graph and how to find all of them. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithm (SODA)*, pages 411–421, 1990.

[Kar] Anna Karlin. Lecture notes on Randomized Algorithms and Probabilistic Analysis. http://courses.cs.washington.edu/courses/cse525/13sp/scribe/lec10.pdf. Accessed: 2015-02.

[Kar93] David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-out algorithm. In *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 21–30, 1993.

[Kar94a] David R. Karger. Random sampling in cut, flow, and network design problems. In *Proc. 26th ACM Symp. on Theory of Computing (STOC)*, pages 648–657, 1994.

[Kar94b] David R. Karger. Using randomized sparsification to approximate minimum cuts. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 424–432, 1994.

[Kar95] David R. Karger. A randomized fully polynomial time approximation scheme for the all terminal network reliability problem. In *Proc. 27th ACM Symp. on Theory of Computing (STOC)*, pages 11–17, 1995.

[Kar96] David R. Karger. Minimum cuts in near-linear time. In *Proc. 28th ACM Symp. on Theory of Computing (STOC)*, STOC'96, pages 56–63, 1996.

[Kar00] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, January 2000.

[KKT95] David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995.

[KL98] David R. Karger and Matthew S. Levine. Finding maximum flows in undirected graphs seems easier than bipartite matching. In *Proc. 30th ACM Symp. on Theory of Computing (STOC)*, pages 69–78, 1998.

[Kla10] Hartmut Klauck. A strong direct product theorem for disjointness. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 77–86. ACM, 2010.

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Rogert Wattenhofer. What cannot be computed locally! In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 300–309, 2004.

[KMW10] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *arXiv preprint arXiv:1011.5470*, 2010.

[KMW16] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *J. ACM*, 63(2):17:1–17:44, March 2016.

[Knu96] Donald E. Knuth. *Stable Marriage and Its Relation to Other Combinatorial Problems: An Introduction to the Mathematical Analysis of Algorithms*. AMS, 1996.

[KP95] Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and applications. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 238–251, 1995.

[KP12] Kishore Kothapalli and Sriram Pemmaraju. Super-fast 3-ruling sets. In *32nd International Conference on Foundations of Software Technology and Theoretical Computer Science*, page 136, 2012.

[KPST94] Philip Klein, Serge Plotkin, Clifford Stein, and Eva Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23(3):466–487, 1994.

[KR95] Philip Klein and R Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms*, 19(1):104–115, 1995.

[KRH+06] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 243–254, 2006.

[KS92a] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math*, 5(4):545–557, 1992.

[KS92b] Samir Khuller and Baruch Schieber. On independent spanning trees. *Information Processing Letters*, 42(6):321–323, 1992.

[KS93] David R. Karger and Clifford Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 757–765, New York, NY, USA, 1993. ACM.

[Kuh15] Fabian Kuhn. Personal Communication, 06-15-2015.

[Kun74] S. Kundu. Bounds on the number of disjoint spanning trees. *J. of Comb. Theory, Series B*, 17(2):199 – 203, 1974.

[KW84] Richard M Karp and Avi Wigderson. A fast parallel algorithm for the maximal independent set problem. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 266–272. ACM, 1984.

[KW03] Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 25–32, 2003.

[Len13] Christoph 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.

[Lin92] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[LLL09] Zongpeng Li, Baochun Li, and Lap Chi Lau. A constant bound on throughput improvement of multicast network coding in undirected networks. *IEEE Trans. Inf. Theory*, 55(3):1016–1026, 2009.

[LM09] Michael Langberg and Muriel Médard. On the multiple unicast network coding conjecture. In *Proc. 47th Allerton Conf. on Comm., Control, and Computing*, pages 222–227, 2009.

[LMR94] Frank Thomson Leighton, Bruce M Maggs, and Satish B Rao. Packet routing and job-shop scheduling in O(congestion+ dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[LMR99] Tom Leighton, Bruce Maggs, and Andrea W Richa. Fast algorithms for finding O(congestion+ dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999.

[LN05] James R Lee and Assaf Naor. Extending lipschitz functions via random metric partitions. *Inventiones mathematicae*, 160(1):59–95, 2005.

[LP72] Micael V. Lomonosov and V. P. Polesskii. Lower bound of network reliability. *Problems of Information Transmission*, 8:118–123, 1972.

[LP13] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 375–382, 2013.

[LPPSP03] Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in $O(\log \log n)$ communication rounds. In *the Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 94–100. ACM, 2003.

[LPS13] Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages: Extended abstract. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 381–390, 2013.

[LPS14] Christoph Lenzen and Boaz Patt-Shamir. Improved distributed steiner forest construction. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, 2014.

[LPSP15] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. *Journal of the ACM (JACM)*, 62(5), 2015.

[LPSR09] Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. Distributed approximate matching. *SIAM Journal on Computing*, 39(2):445–460, 2009.

[LRY15] Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *preprint arXiv:1502.04022*, 2015.

[Lub85] Michael Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 1–10. ACM, 1985.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM journal on computing*, 15(4):1036–1053, 1986.

[LW11] Christoph Lenzen and Roger Wattenhofer. MIS on Trees. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 41–48. ACM, 2011.

[Lyn96] Nancy A Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

[Mat93] David W. Matula. A linear time $2 + \varepsilon$ approximation algorithm for edge connectivity. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 500–504, 1993.

[MDJ+06] Manki Min, Hongwei Du, Xiaohua Jia, Christina Xiao Huang, Scott C-H Huang, and Weili Wu. Improving construction for connected dominating set with steiner tree in wireless sensor networks. *Journal of Global Optimization*, 35(1):111–119, 2006.

[MR10] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the Symposium on Theory of Computing (STOC)*, 2014.

[New91] Ilan Newman. Private vs. common random bits in communication complexity. *Information processing letters*, 39(2):67–71, 1991.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discret. Math.*, 5(1):54–66, February 1992.

[NMN01] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1):3–36, 2001.

[NS14] Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *Proc. 27th Symp. on Distributed Computing (DISC)*, pages 439–453, 2014.

[NW61] C.St.J.A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *J. of the London Math. Society*, 36:445–450, 1961.

[OR97] Rafail Ostrovsky and Yuval Rabani. Universal O(congestion+ dilation+ $\log^{1+\epsilon} n$) local control packet switching algorithms. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 644–653. ACM, 1997.

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[Pem01] Sriram V. Pemmaraju. Equitable colorings extend chernoff-hoeffding bounds. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 924–925, 2001.

[PQ82] J. C. Picard and M. Queyranne. Selected applications of minimum cuts in networks. *INFOR.*, 20:19–39, 1982.

[PR99] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed MST construction. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 1999.

[PR01] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed computing*, 14(2):97–100, 2001.

[PR07] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1):183–196, 2007.

[PS92] Alessandro Panconesi and Aravind Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 581–592. ACM, 1992.

[PST91] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. In *Proc. 32nd IEEE Symp. Foundations of Computer Science (FOCS)*, pages 495–504, 1991.

[PW11] Britta Peis and Andreas Wiese. Universal packet routing with arbitrary bandwidths and transit times. In *Integer Programming and Combinatoral Optimization*, pages 362–375. Springer, 2011.

[Rac02] Harald Racke. Minimizing congestion in general networks. In *Proc. 43rd IEEE Symp. Foundations of Computer Science (FOCS)*, pages 43–52, 2002.

[Rac08] Harald Racke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 35th ACM Symp. on Theory of Computing (STOC)*, pages 255–264, 2008.

[Raja] Rajmohan Rajaraman. Lecture notes on Algorithmic Power Tools. http://www.ccs.neu.edu/home/rraj/Courses/7880/F09/Lectures/GeneralLLL_Apps.pdf. Accessed: 2015-02.

[Rajb] Rajmohan Rajaraman. Lecture notes on Algorithmic Power Tools. http://www.ccs.neu.edu/home/rraj/Courses/7880/F09/Lectures/PacketRouting.pdf. Accessed: 2015-02.

[Raz92] A. A. Razborov. On the distributional complexity of disjointness. *Theor. Comp. Sci.*, 106:385–390, 1992.

[Raz98] Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.

[Rot13] Thomas Rothvoß. A simpler proof for O(congestion + dilation) packet routing. In *Integer Programming and Combinatorial Optimization*, pages 336–348. Springer, 2013.

[RS97] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 475–484, 1997.

[RT96] Yuval Rabani and Éva Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 366–375. ACM, 1996.

[RTVX11] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Symposium on Innovations in Computer Science*, pages 223–238, 2011.

[Sch98] Christian Scheideler. *Universal routing strategies for interconnection networks*, volume 1390. Springer, 1998.

[SET03] P. Sanders, S. Egner, and L. Tolhuizen. Polynomial time algorithms for network information flow. In *Proc. 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 286–294, 2003.

[SEW13] Johannes Schneider, Michael Elkin, and Roger Wattenhofer. Symmetry breaking depending on the chromatic number or the neighborhood growth. *Theoretical Computer Science*, 509:40–50, 2013.

[Sin] Alistair Sinclair. Lecture notes on Randomness and Computation. http://www.cs.berkeley.edu/~sinclair/cs271/n22.pdf. Accessed: 2015-02.

[SM90] Farhad Shahrokhi and David W Matula. The maximum concurrent flow problem. *Journal of the ACM (JACM)*, 37(2):318–334, 1990.

[SSS95] Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

[ST04] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proc. 36th ACM Symposium on Theory of Computing (STOC)*, pages 81–90, 2004.

[Tar74] R. Endre Tarjan. Testing graph connectivity. In *Proc. 6th ACM Symp. on Theory of Computing (STOC)*, STOC '74, pages 185–193, 1974.

[Tho01] Mikkel Thorup. Fully-dynamic min-cut. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 224–230. ACM, 2001.

[Thu95] Ramakrishna Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. In *the Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 28–37, 1995.

[Thu97] Ramakrishna Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. *Journal of Algorithms*, 23(1):160 – 179, 1997.

[Top85] Donald M. Topkis. Concurrent broadcast for information dissemination. *Software Engineering, IEEE Transactions on*, pages 1107–1112, 1985.

[Tut61] W. T. Tutte. On the problem of decomposing a graph into n connected factors. *J. of the London Math. Society*, 36:221–230, 1961.

[Val82] Leslie G Valiant. Parallel computation. In *Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science*, 1982.

[WAF02] Peng-Jun Wan, Khaled M Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *the Proc. of IEEE INFOCOM*, volume 3, pages 1597–1604, 2002.

[WG75] Henry William Watson and Francis Galton. On the probability of the extinction of families. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 4:138–144, 1875.

[WGS01] Jie Wu, Ming Gao, and Ivan Stojmenovic. On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. In *Parallel Processing, International Conference on, 2001.*, pages 346–354. IEEE, 2001.

[Wie11] Andreas Wiese. *Packet Routing and Scheduling*. Cuvillier, 2011.

[Win04] Aaron Windsor. A simple proof that finding a maximal independent set in a graph is in nc. *Information processing letters*, 92(4):185–187, 2004.

[Zel86] B. Zelinka. Connected domatic number of a graph. *Math. Slovaca*, 36:387–392, 1986.

[ZI89] Avram Zehavi and Alon Itai. Three tree-paths. *Journal of Graph Theory*, 13(2):175–188, 1989.