Towards Universal Optimality in Distributed Optimization

Goran Žužić

CMU-CS-20-121
August 2020

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Bernhard Haeupler, Chair
Anupam Gupta
Gary Miller
Keren Censor-Hillel, Technion

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Supported in part by NSF grants CCF-1527110, CCF-1618280, CCF-1814603, CCF-1910588, NSF CAREER award CCF-1750808, a Sloan Research Fellowship and the 2018 DFINITY Scholarship. Some of this work was completed while visiting the Nagoya Institute of Technology.

Keywords: distributed computing, CONGEST, distributed optimization, network coding, coding gaps, low-congestion shortcuts, tree-restricted shortcuts, universal optimality, minor-free graphs, treewidth-bounded graphs, genus-bounded graphs, moving cut

To my family that shaped my youth, and to my friends that shaped my adulthood.

Abstract

The modern computation and information processing systems shaping our world have become massively distributed and a fundamental understanding of distributed algorithmics has never been more important. This shift towards distributed systems has resulted in increased interest and fast acceleration in our theoretical understanding of distributed optimization problems. At the same time, extremely general lower bounds uncovered that any distributed optimization requires $\tilde{\Omega}(\sqrt{n})$ rounds on some worst-case network topology, even if the diameter of the network is small. Many fundamental optimization problems, including MST, shortest paths, and cut/flow problems, now have "optimal" algorithms matching this worst-case performance bound.

Real-world networks, however, are never worst-case and no network of interest shares the limiting bottleneck characteristics of the lower bound topology. In fact, there is no known barrier for ultra-fast polylogarithmic-round distributed algorithms on any network of interest.

In this thesis, we develop a theoretical understanding of when it is possible to go below the $\tilde{\Theta}(\sqrt{n})$ bound. Our results include:

1. We show that planar networks, genus-bounded networks, and treewidth-bounded networks admit $\tilde{O}(D)$ round algorithms, where D is the network diameter. Similarly, we show that minor-free networks, a wide family subsuming all of the above, admit $\tilde{O}(D^2)$ round algorithms. Moreover, we give a single (uniform) and simple algorithm that works on all of these network classes by introducing a new framework called tree-restricted shortcuts. Prior to our work, only the planar network result was known and was significantly more complicated compared to ours.
2. We resolve the following 25-year-old open problem asked by Garay, Kutten, and Peleg: "What network topology parameters determine the complexity of distributed optimization?" We show that a previously studied parameter, the general shortcuts of best congestion+dilation, characterizes the complexity of distributed optimization for every network topology. In particular, this includes showing the first known non-trivial unconditional lower bound that is universal (i.e., applies to all graphs) and constructively matching that bound in the distributed known-topology setting.

Acknowledgments

First of all, I would like to thank my advisor Bernhard Haeupler: his guidance, expertise, support, and kindness have been nothing short of amazing and will be a source of inspiration for a lifetime.

I would also like to thank the rest of my thesis committee: Anupam Gupta, Gary Miller, and Keren Censor-Hillel. I am grateful to Gary for both his academic and non-academic guidance throughout my graduate studies. He generously allocated his time for me to expand my research horizons with new directions: in particular, he sparked my interest in spectral graph theory and continuous methods which eventually lead me to explore important questions at the tail end of my Ph.D. I am grateful to Anupam for enabling and mentoring a research collaboration with Domagoj Bradač and Sahil Singla that introduced me to stochastic optimization. I am grateful for his patience in answering my various class- and research-related questions I have posed to him over the years. Finally, I am also grateful to Keren for her helpful feedback and introducing me to her students Michal Dory and Yuval Efron with whom I had many fruitful discussions.

I have had the privilege of co-authoring papers with many people during my time in graduate school: Domagoj Bradač, D. Ellis Hershkowitz, Taisuke Izumi, Jason Li, Arayank Mehta, Sahil Singla, D. Sivakumar, David Wajc and Di Wang. I would like to thank them for the numerous useful discussions and collaborations we had throughout my time as a graduate student.

I would like to thank VMware Research and my mentors, Udi Wieder and Ittai Abraham, for giving me an excellent summer internship opportunity, their guidance greatly widened my scope and appreciation of theory as a field in the fledgling phase of my graduate studies. I would also like to thank Google Research and my mentors D. Sivakumar and Aranyak Mehta for providing me with an amazing summer internship that allowed me to immerse myself in new areas of computer science.

I would deeply like to thank my many friends who greatly shaped my experience while at CMU—I genuinely appreciate their time, patience, friendship, and support they have given me. The last five years passed in a flash in large part due to them. The list of friends I am grateful to is too large to be included here, but I will do my best to show my gratitude to each and every one of them.

I am indebted to my teachers and professors that had a transformative impact on me by sparking, shaping, nurturing, and developing my interests in mathematics, computer science, and programming before I started my graduate studies. I would specifically like to thank Predrag Brođanac, Luka Kalinovčić, Spomenka Mihočinec, Mile Šikić, and Tibor Kulcsar.

Finally, I would like to thank my family: my parents Ljiljana and Drago and my sister Lorena, who set me on my path and encouraged my passions. Without their active and ongoing support and love, I would have never been able to even start my Ph.D. journey, let alone complete this dissertation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Decentralization and distributed computing are becoming the computing paradigms of the future. In fact, modern computation and information processing systems are already massively distributed for various reasons: Moore's law is approaching the limits of physics, data to be processed is vastly exceeding what can be stored on a single machine, and modern services are increasingly decentralized. A deep and fundamental understanding of distributed algorithmics is a prerequisite to building efficient and reliable distributed systems of tomorrow and has never been more important than today.

The *CONGEST model* [121] has been the standard mathematical model to study communication in distributed algorithmics. In this model, the network topology is abstracted as an undirected graph with n nodes and diameter D. Communication occurs in synchronous rounds in which each node can send a bounded amount of information to each neighbor, typically $O(\log n)$ bits. The complexity measure is the number of rounds required to solve an optimization problem. While this classic model is decades old, it remains highly relevant and influential because it allows for a clean mathematical study of communication bottlenecks in distributed optimization algorithms. Indeed, synchronous message-passing algorithms that are closely inspired by the CONGEST model are used by systems that account for much of modern large-scale graph processing and network analysis, such as Google's Pregel [111], Facebook's Giraph [26, 72], or Apache's Spark GraphX [58]. These applications signify that communication remains a critical bottleneck in any practical distributed system.

This relevance has motivated a recent, broad, concentrated, and highly successful effort to advance our understanding of distributed algorithmics with a strong focus on algorithms for fundamental network optimization problems, such as MST [37, 89], shortest paths [36, 44, 78, 80, 82, 84, 101, 102, 113], flows [55], cuts [51, 114], etc. As a result of these efforts, for the majority of such fundamental optimization problems, we now know methods that achieve running times (close to) $\widetilde{O}(D+\sqrt{n})$[1] CONGEST rounds (where D is the diameter of the topology graph and n is the number of nodes). Moreover, such results cannot be improved in general due to a pervasive $\widetilde{\Omega}(D+\sqrt{n})$ lower bound that applies to all of the above problems [27, 122]. Therefore, in some

[1]Throughout this thesis, we mostly ignore polylogarithmic factors in n. We use $\tilde{\Omega}$ and $\tilde{O}$ notation to hide such factors. For example, $\tilde{O}(f(n)) = O(f(n)\log^{O(1)} n)$.

sense, the results are optimal.

1.1 Overview of the Thesis

In this section, we give a brief overview of the motivation, goals, and contributions of this thesis.

1.1.1 Beyond worst-case networks

While matching upper and lower bounds for distributed algorithmic problems are remarkable achievements, this thesis emphatically argues that one cannot settle for such results, particularly when the provable performance guarantees are too weak to be practically relevant. The justification of optimality simply does not match any real barriers that are observed in practice. Understanding this claim and its importance, especially in the case of distributed optimization algorithms, requires a more detailed look at different notions of optimality.

A key abstraction in the theoretical study of algorithms is the focus on worst-case running times expressed as asymptotic functions of the instance size, e.g., the number of nodes n for graph problems. This perspective successfully abstracts away differences between concrete computational models/implementations, enables definitions of complexity classes like P and NP, and provides a way to compare the efficiency and scalability of different algorithms. The theory of distributed computing has historically followed this paradigm. The running time of a distributed algorithm is measured against all possible inputs and all possible network topologies. For example, one of the first distributed MST algorithms [45] runs in $O(n \log n)$ rounds. This was later improved to an "optimal" $O(n)$ round algorithm by Awerbuch [12].

The precise way in which the algorithm of [12] is optimal is the following: There *exists a (pathological worst-case) topology* on n nodes on which no algorithm can do better, namely the n-node cycle. Indeed, it is easy to see that for most network optimization problems nodes require some knowledge about inputs that are far away from them to determine a solution. This leads to a trivial $\Omega(D)$ lower bound on topologies with diameter D, which is $\Omega(n)$ in the pathological cycle network. An algorithm for which the asymptotic worst-case running time, over all inputs and networks of size n, that matches the worst-case running time of the best possible algorithm, is called optimal, or more precisely, "*existentially optimal*" with respect to n.

Existential optimality, however, says nothing about how the performance of an algorithm compares to what is achievable on non-worst-case topologies. Topologies of practical interest in particular might allow for drastically faster running times compared to a pathological worst-case instance. Indeed, essentially all real-world topologies have diameters which are (poly-)logarithmic in the size of the network and the trivial $\Omega(D)$ lower bound for global optimization algorithms on such networks is merely $\Omega(\log^c n)$. This is exponentially faster than the algorithm of Awerbuch, which requires $\Theta(n)$ rounds on any topology.

Researchers soon realized that existentially optimal algorithms in terms of n are far from satisfactory and that a finer-grained way to analyze and specify distributed running times was required.

Kutten and Peleg [95] made progress in this direction by giving their celebrated $\tilde{O}(D + \sqrt{n})$ round MST algorithm, achieving a quadratic performance improvement on any topology of interest. Later, a series of works that started with Peleg and Rubinovich [122] and culminated in the work of Das Sarma et al. [27] gave strong unconditional lower bounds using communication complexity. They constructed certain n-node networks with logarithmic diameter in which solving any non-trivial global optimization problem requires $\tilde{\Omega}(\sqrt{n})$ rounds. This result makes the Kutten-Peleg MST algorithm existentially optimal with respect to n and D because for any n and D there exists a pathological worst-case network on which no algorithm can run faster. It is this notion of existential optimality with respect to n and D which applies to most state-of-the-art distributed optimization algorithms. I.e., on any input and low-diameter topology these algorithms achieve a $\tilde{\Theta}(\sqrt{n})$ running time, which is the best possible—at least on the pathological worst-case network of [27].

Unfortunately, this form of optimality and the $\tilde{\Theta}(D+\sqrt{n})$ performance guarantees of these algorithms suffer from the same drawbacks as before: Real-world networks are never worst-case and no network of interest comes close to exhibiting any of the limiting bottleneck characteristics of the pathological worst-case topology which is used to demonstrate (existential) optimality. In fact, there is no plausible barrier known for ultra-fast polylogarithmic round CONGEST algorithms on any network of interest for most fundamental network optimization problems. On the other hand, essentially all optimal state-of-the-art $\tilde{\Theta}(\sqrt{n} + D)$ algorithms are specifically designed to achieve a $\Theta(\sqrt{n})$ running time and always require $\Theta(\sqrt{n})$ many rounds, even when no communication bottlenecks are present and faster algorithms are possible. This exponential gap between optimal worst-case algorithms that exhibit $\tilde{\Theta}(\sqrt{n})$ round complexities on every topology and the $O(\log^c n)$ performance which is likely possible in most, if not all, real-world settings forms the starting point for the studies of this thesis and is summarized in the following objective.

Objective 1.1.1. *Develop general tools for designing distributed optimization algorithms that provably achieve ultra-fast polylogarithmic running times on practical networks.*

We tackle this objective by introducing a novel *tree-restricted shortcut* framework that can be used to design simple and efficient solutions for various distributed optimization tasks. In particular, the framework provides optimal $\tilde{O}(D)$ distributed algorithms on planar, bounded genus graphs, bounded treewidth graphs, and excluded-minor graphs. Notably, we provide a simple distributed algorithm that is *uniform* over different graph classes—the same algorithm works without any knowledge of which graph class is it being run on. (Chapters 2 to 4.)

Prior to the work of this thesis, only the $\tilde{O}(D)$ round algorithm for planar graphs was known, and it was significantly more complicated: requiring a distributed computation of a planar embedding in $\tilde{O}(D)$ rounds. Our methods significantly simplified this result and extended it to other graph classes. This simplicity and uniformity give the framework qualities that might make the framework usable even in practice.

1.1.2 Universal optimality

A broader goal that applies to each network (and not just real-world networks) is to uncover what graph properties enable/preclude ultra-fast algorithms. On a high-level, this can be summarized via the following objective.

Objective 1.1.2. *Develop a theory that explains how the network topology influences optimal distributed optimization algorithms.*

As mentioned before, the network diameter D is an example of such a parameter: any distributed algorithm that takes less than D cannot exchange information between nodes that are D hops away from each other, hence cannot solve (global) distributed optimization problems like the MST. Two things are important to note. First, the diameter parameter is in general not achievable, as showcased by the pathological instance of [27] that has diameter $D = O(\log n)$ and requires $\tilde{\Omega}(\sqrt{n})$ rounds for distributed optimization. Second, the diameter gives a lower bound that applies to all networks, i.e., it applies *universally* (as opposed to existentially). Universal lower bounds give raise to a much stronger notion of impossibility than the existential bounds as they provide evidence on why a specific network of interest cannot admit a fast distributed algorithm. This generality might be one of the reasons why no universal lower bound other than the (trivial) diameter one was known (prior to this thesis).

We tackle a more ambitious goal: we aim to not only find a universal lower bound but to also match it with an algorithm. Such an algorithm would be "as fast as possible on each topology" and is called *universally optimal*. The fundamental question regarding their existence was formulated in an influential FOCS'93 paper by Garay, Kutten and Peleg [46]. The below is a verbatim quote:

"*The interesting question that arises is therefore whether it is possible to identify the inherent graph parameters associated with the distributed complexity of various fundamental network problems, and develop* universally-optimal *algorithms[, that are as fast as possible on* every *topology.]*" [46]

We note that even the MST algorithm proposed by Garay, Kutten and Peleg achieves only existential optimality—it terminates in $\tilde{O}(n^{0.61\ldots} + D)$ rounds. However, at the time of their writing, they were arguing against settling for an "optimal" $O(n)$ algorithm; they proposed that a parametrization with respect to both n and D is more meaningful. Of course, a similar reasoning could lead researchers to consider an increasingly complex set of parameters and find (existentially) optimal algorithm with respect to those parameters. On the other hand, universal optimality cuts through the question of finding the right parametrization as it does not depend on any parameters: it is parametrized by the entire graph, without losing any topological information.

The question has remained open even for significantly simpler problems than the MST. For instance, Ghaffari [48] proposed an algorithm for distributed k-message broadcast that achieves universal optimality under certain assumptions about the optimal solution in the known-topology setting[2]. In fact, Ghaffari writes:

[2]Interestingly, the methods presented in this thesis can be used to show the algorithm of [48] is unconditionally

"*Although the motivation for universal optimality is clear and strong, achieving it is not straightforward and thus, to the best of our knowledge, there is no known distributed algorithm that (non-trivially) achieves universal optimality.*" [48].

Our contribution with respect to universal optimality focuses on the *known-topology* setting. In this setting nodes know the graph upfront, but not the input to the problem (e.g., for the MST problem they know G, but not the edge costs). This setting can be naturally motivated by the practical need to solve multiple distributed optimization instances on the same network, hence potentially amortizing the preprocessing overhead required to learn the graph topology before the input is given.

Our contribution can be summarized in three parts.

1. We contribute a formal definition of universal optimality that is aligned with the notion used in prior work (a precise definition was never put forward, most likely due to no little progress being made towards this goal). (Chapter 6.)
2. Second, we prove that a known graph parameter called *shortcut quality* is a universal lower bound for many distributed optimization problems (e.g., MST, single-source shortest path, min-cut). We also provide several equivalent quantities (up to polylogarithmic factors) that are based on various communication and combinatorial graph properties. Moreover, we show that the lower bound is universal even in the known-topology setting (making the lower bound stronger). The bound is derived via a combination of classic reductions and novel information-theoretic arguments that characterize the limits of information transfer in a graph while taking both the distance and congestion into account; the bound is unconditional. (Chapters 5 and 6.)
3. Third, we design a distributed algorithm in the known-topology setting that works in $\tilde{O}(\text{shortcut quality})$ rounds, matching the lower bound up to polylogarithmic factors. Together, these give raise to universally optimal algorithms in the known-topology setting. The result has several important ramifications: many important distributed optimization tasks belong to a newly identified complexity class of problems with equivalent distributed running times. This complexity class even provides a simple and clean prototypical problem (called part-wise aggregation) that a network designer can focus on if they wish to explore the limits of communication in a new network. Furthermore, the result stipulates that the shortcut quality is a "sufficient statistic" that characterizes the optimal running time of a large class of distributed tasks. It might be interesting to explore the shortcut quality of real-world networks and see whether these theoretical predictions can be matched in practice.

   The largest obstacle we had to overcome to achieve these universally optimal algorithms is to show there exists an extension of the classic oblivious routing strategies [126] that take both the dilation (i.e., distance) and congestion into account (current strategies only take the congestion into account). In fact, the construction of such $\text{congestion} + \text{dilation}$ oblivious routings was an open problem that was studied on special classes of graphs like mesh graphs and geometric networks [20, 21]. Moreover, $\text{congestion} + \text{dilation}$ oblivious

universally optimal in the known-topology setting

routings for general graphs with polylogarithmic approximation ratios were believed to be impossible due to an impossibility result of Räcke [125, pg. 59]. We side-step this barrier by using a slightly more general definition of oblivious routing that is still usable in the distributed setting. We believe our results will be of independent interest to the approximation algorithms community. (Chapter 7.)

1.2 Technical Preliminaries

In this section, we formally define the CONGEST model and the low-congestion shortcut framework.

1.2.1 The CONGEST model

We work in the classical CONGEST model [121]. In this setting, a network is given as a connected undirected graph $G = (V_G, E_G)$ with $n = |V|$ nodes and diameter D. We sometimes write V or E if the graph is clear from the context. Initially, nodes only know their immediate neighbors and they collaborate to compute some global function of the graph like the MST. Communication occurs in synchronous rounds; during a round, each node can send $O(\log n)$ bits to each of its neighbors. The nodes know (a polynomial bound) on the value of n, always correctly follow the protocol, and never fail. Many of these assumptions can be removed via standard methods with a slight round overhead but these are not in the scope of this thesis. The goal is to design protocols that minimize the resource of time—the number of rounds before the nodes compute the solution.

We now precisely formalize the notion of solving a problem in this model, e.g., how the input and output are given. While the formalization is specifically given for the MST, any other problem is completely analogous. All nodes synchronously wake up in the first round and start executing some given protocol. Every node initially only knows its immediate neighbors and the weight of each of its incident edges. After a specific number of rounds, all nodes must simultaneously output (i) the weight of the computed MST τ, and (ii) for each edge e incident to it, a $0/1$ bit indicating if $e \in \tau$.

1.2.2 Low-congestion shortcut framework

Next, we briefly provide some background and details on the low-congestion shortcuts framework. While the framework itself was introduced in prior work [49], this thesis both makes extensive use and extends the framework.

Broadly speaking, the shortcut framework shows how many distributed problems can be reduced to solving a simple and natural communication problem. It also defines low-congestion shortcuts as natural routing structures that can be used to algorithmically solve this communication problem. Lastly, it shows how the topological structure of a network influences the quality of short-

cuts in the network and how this quality is directly coupled to the efficiency of shortcut-based distributed algorithms. More formally, consider the following *part-wise aggregation problem*.

Definition 1.2.1 (Part-wise aggregation problem)**.** *Let $G = (V, E_G)$ be a graph. Given disjoint and internally-connected* ***parts*** *$P_1, P_2, \ldots, P_N \subseteq V$, we want to distributedly compute some simple part-wise aggregate (e.g., sum or max) of nodes' private values. Specifically, each node is initially given its part ID (or $\perp$ if none) and a private value x_v; at the end of the computation each node v belonging to some part P_i should know the aggregate value of $\{x_v \mid v \in P_i\}$.*

This problem arises naturally in many divide-and-conquer algorithms in which a network is subdivided and simple distributed computations need to be solved in each part. A prominent example is Boruvka's MST algorithm [115] in which the MST is constructed across $O(\log n)$ iterations in which each connected component of edges selected so far computes and selects the smallest weight edge leaving it. Up to a factor of $O(\log n)$ in the running time, the classic MST problem, therefore, reduces to solving the part-wise aggregation problem.

Fact 1.2.2 ([49])**.** *There is a distributed CONGEST algorithm for the MST problem that makes $O(\log n)$ oracle calls to the part-wise aggregation problem and succeeds with high probability.*

Moreover, all known MST distributed algorithms can be seen as simply finding efficient ways to solve the part-wise aggregation problem. Maybe more surprisingly, recent related work has shown that many other, seemingly unrelated, distributed network optimization problems like finding approximate min-cuts [49] or shortest paths [64] similarly reduce to solving the part-wise aggregation problem.

Ideally one would like to solve this part-wise aggregation problem in D time, where D is the diameter of the network. This would lead to essentially optimal $\tilde{O}(D)$ round optimization algorithms. Unfortunately, the lower bound of [27] shows that this is not possible, at least not on the pathological worst-case network they construct. The reason for this, however, is somewhat subtle. In particular, since parts are disjoint and connected one can easily solve the aggregation problem by having nodes repeatedly forward the minimum value seen so far to neighbors in their part. It is easy to see that the number of rounds needed for this trivial flooding strategy to converge is equal to the strong diameter of any part (i.e., the diameter of the induces subgraph $G[P_i]$ in isolation of the rest of the graph). Unfortunately, the strong diameter of a connected subgraph can be much larger than the diameter D of the underlying network graph.

Low-congestion shortcuts were introduced as a natural way to overcome the above issue. We allow each part P_i to use a set of extra edges $H_i \subseteq E_G$ to more efficiently communicate with other nodes in the same part. More precisely, part P_i is permitted to use the edges $E_G[P_i] \cup H_i$ for communication, where $E_G[P_i]$ are edges with both endpoints in P_i. However, if too many parts try to communicate using the same network edge they cause congestion which slows down communication. Overall, solving part-wise aggregation efficiently requires communicating over *short* paths whose edges have *low congestion*. Therefore, a useful shortcut needs to balance the

congestion c and the **dilation** d. In particular, a d-dilation c-congestion shortcut for a set of parts is naturally defined as a set of shortcut edges for every part which, if added, makes the diameter of a part at most d while each edge is used by at most c parts. The following definition formalizes these notions.

Definition 1.2.3. *Let $G = (V, E_G)$ be an undirected graph with vertices subdivided into **disjoint and connected** subsets $\mathcal{P} = (P_1, P_2, ..., P_N), P_i \subseteq V$. In other words, $E_G[P_i]$ is connected and $P_i \cap P_j = \emptyset$ for $i \neq j$. The subsets P_i are called **parts**. We define a **shortcut** $\mathcal{H}$ as $(H_1, H_2, ..., H_N)$, $H_i \subseteq E_G$. A shortcut is characterized by the following parameters:*

1. *$\mathcal{H}$ has **congestion** c if each edge $e \in E_G$ is used in at most c different sets $E_G[P_i] \cup H_i$, i.e., $\forall e \in E_G : |\{i : e \in E_G[P_i] \cup H_i\}| \leq c$. Note that the sets $\{E_G[P_i]\}_{i=1}^N$ are disjoint.*
2. *$\mathcal{H}$ has **dilation** d if for each $i \in [N]$ the diameter of $E_G[P_i] \cup H_i$ is at most d.*

We note that the congestion and dilation are classic parameters used in routing that were co-opted for low-congestion shortcuts. The celebrated result of Leighton, Maggs and Rao [99] shows that given a set of paths in a graph with congestion c (each edge is used by at most c paths) and dilation d (each path has at most d hops), one can simultaneously send packets from the start to the end of each path in $O(c + d)$ rounds while only sending one packet per edge per round. However, this result is constructive and is not clearly applicable to distributed models where one is unable to schedule packets in a centralized way (e.g., nodes in CONGEST have knowledge only about their immediate neighborhood and need to distributedly learn information about other parts of the graph).

However, one can recover a slightly weaker result: having access to a congestion c and dilation d shortcut with respect to a part-wise aggregation enables the aggregation to be completed in $\tilde{O}(c + d)$ CONGEST rounds. This motivates the following definition.

Definition 1.2.4. *The **quality** q of a congestion c and dilation d shortcut is $q = c + d$.*

The definition of quality enables the user of the shortcut framework to focus on minimizing a single quantity rather than a $(\text{congestion}, \text{dilation})$ pair. We summarize the usefulness of the above definitions via the following result.

Fact 1.2.5 ([49])**.** *Let $\mathcal{P} = (P_1, \ldots, P_N)$ be a set of parts in a graph G. Given distributed knowledge of quality-q shortcut (i.e., each node incident to an edge e knows the IDs of the parts allowed to use e), a randomized distributed CONGEST algorithm can complete part-wise aggregation on $\mathcal{P}$ in $\tilde{O}(q)$ rounds with high probability.*

The above result is relatively simple and utilizes the well-known technique of random delays. Furthermore, we note that many practical networks of interest admit shortcuts of quality $\tilde{O}(D)$ (more on this in later chapters). However, the result assumes the shortcuts can be (efficiently) constructed. Shortcut construction is often the bottleneck in designing distributed algorithms and the efficient construction of good-quality shortcuts in distributed settings is the central remaining

open problem regarding low-congestion shortcuts (at least in the setting when nodes do not know the topology in advance).

Open Problem 1.2.6. *Let $\mathcal{P} = (P_1, \ldots, P_N)$ be a set of parts in a graph G. Distributedly construct shortcuts of quality $\widetilde{O}(q)$ in $\widetilde{O}(q)$ rounds, where q is the optimal shortcut quality (with respect to $\mathcal{P}$).*

One can side-step this issue of construction by considering a more-structured **tree-restricted shortcut** (defined in Chapter 2) which comes with an efficient, uniform, and distributed construction technique. Alternatively, in Chapter 7 we provide efficient construction techniques in the distributed known-topology setting.

1.3 Structure of the Thesis

The rest of the thesis is structured into (mostly self-contained) chapters that, together, culminate in both (1) a mature tree-restricted shortcut framework, and (2) universally optimal algorithms in the distributed known-topology setting. We note that (almost all of) the chapters were originally published as standalone conference or journal papers, and have been slightly edited for the purposes of this thesis.

Chapter 2: The Tree-Restricted Shortcut Framework. We present the tree-restricted shortcut framework that leads to simple $\widetilde{O}(D)$ distributed optimization algorithms for many graph classes (e.g., planar, genus-bounded, treewidth-bounded, minor-free graphs), where D is the diameter of the graph. Previous constructions were exceedingly complicated: even relying on having access to a distributed planar embedding of a planar graph [49]. Our framework side-steps this problem by defining a slightly restricted and more structured form of shortcuts and giving a novel construction algorithm that efficiently finds a shortcut which is, up to a logarithmic factor, as good as the best shortcut that exists for a given network. This new construction algorithm directly leads to an $\widetilde{O}(D)$-round algorithm for solving optimization problems like MST for any topology for which good tree-restricted shortcuts exist.

Chapter 3: Shortcuts for Treewidth-Bounded and Genus-Bounded Graphs. We broaden the utility of the tree-restricted shortcut framework by showing that networks with pathwidth or treewidth k allow for good tree-restricted shortcuts. This leads to fast $\widetilde{O}(kD)$ distributed optimization algorithms. We also improve the dependence on genus g from $\widetilde{O}(gD)$ to $\widetilde{O}(g\sqrt{D})$. Lastly, we prove lower bounds which show that the dependence on k and g in our shortcuts is optimal. Overall, this significantly refines and extends the understanding of how the complexity of distributed optimization problems depends on the network topology.

Chapter 4: Shortcuts for Minor-Free Graphs. We prove that excluded minor graphs admit high-quality shortcuts, leading to an $\widetilde{O}(D^2)$ round algorithm for the distributed optimization

problem. To work with excluded minor graph families, we utilize the Graph Structure Theorem of Robertson and Seymour. To the best of our knowledge, this is the first time the Graph Structure Theorem has been used for an algorithmic result in the distributed setting. Even though the proof is involved, merely showing the existence of good shortcuts is sufficient to obtain simple, efficient distributed algorithms. In particular, the shortcut framework can efficiently construct near-optimal shortcuts and then use them to solve the optimization problems. This, combined with the very general family of excluded minor graphs, which includes most other important graph classes, makes this result of significant interest.

Chapter 5: Network Coding Gaps for Completion-time of Multiple Unicasts. We study network coding gaps for the problem of makespan minimization of multiple unicasts. While the chapter tackles a somewhat different setting than the rest of the thesis, the tooling developed in this chapter is crucial to proving the main contributions of Chapter 6.

In the problem of makespan minimization of multiple unicasts, distinct packets at different nodes in a network need to be delivered to a destination specific to each packet, as fast as possible. The *network coding gap* specifies how much coding packets together in a network can help compared to the more natural approach of routing. While makespan minimization using routing has been intensely studied for the multiple unicasts problem, no bounds on network coding gaps for this problem are known. We develop new techniques that allow us to upper bound the network coding gap for the makespan of k unicasts, proving this gap is at most polylogarithmic in k. Complementing this result, we show there exist instances of k unicasts for which this coding gap is polylogarithmic in k. Our results also hold for average completion time, and more generally any ℓ_p norm of completion times.

Chapter 6: Shortcuts are a Universal Lower Bound for Distributed Optimization. We formally define the notion of *universal optimality* and prove that *shortcut quality* is a universal lower bound for many distributed optimization tasks on any graph. We also provide several different quantities that are non-trivially equivalent to the shortcut quality, up to polylogarithmic factors. Some of them might be more appropriate in different circumstances, even though all of them (together with the material from the next chapter) characterize the inherent complexity of distributed optimization.

Chapter 7: Near-Optimal Distributed Known-Topology Shortcut Construction. We show how to construct shortcuts of quality q in $\tilde{O}(q)$ rounds of known-topology CONGEST. This result (together with the lower bound of Chapter 6) resolves the 25-year-old open problem that asks "What network topology parameters determine the complexity of distributed optimization?" To which the answer is shortcut quality (or some other equivalent quantity). Furthermore, this result shows that low-congestion shortcuts are a sufficient tool to achieve universally optimal algorithms and uncovers a novel distributed complexity class of problems all of which have running times equivalent up to polylogarithmic factors. The main technical contribution is an oblivious routing construction that minimizes $\mathrm{congestion} + \mathrm{dilation}$ up to polylogarithmic factors, a result which we believe is of independent interest for the approximation algorithms community.

Chapter 8: Conclusion and Open Questions. We summarize the contribution of this thesis and address the implications, impacts on other fields and open problems.

Chapter 2

The Tree-Restricted Shortcut Framework

The results of this chapter were published in [65] with Bernhard Haeupler and Taisuke Izumi as co-authors. The work was supported in part by KAKENHI No. 15H00852 and 16H02878 as well as NSF grants CCF-1527110 "Distributed Algorithms for Near Planar Networks" and NSF-BSF grant "Coding for Distributed Computing".

2.1 Introduction

2.1.1 Background and motivation

Problems such as the MST problem can be solved in the distributed CONGEST setting using $O(\sqrt{n}\log^* n + D)$ rounds of communication [95]. Moreover, and perhaps more surprisingly, this bound was shown to be the best possible (up to polylogarithmic factors). Specifically, there are graphs in which one cannot do any better than $\widetilde{\Omega}(\sqrt{n} + D)$. While clearly, no algorithm can solve any global network optimization problem faster than $\Omega(D)$, the $\widetilde{\Omega}(\sqrt{n})$ factor is harder to discern. To make matters worse, the $\widetilde{\Omega}(\sqrt{n} + D)$ lower bound was shown to be far-reaching. It applies to a multitude of important network optimization problems including MST, minimum-cut, weighted shortest-path, connectivity verification, and so on [27].

While this bound precludes the existence of more efficient algorithms in the general case, it was not clear does it hold for special families of graphs. This question is especially important because any real-world application on huge networks should exploit the special structure that the network provides. The mere existence of "hard" networks for which one cannot design any fast algorithm might not be a limiting factor.

In the first result that utilizes network topology to circumvent the lower bound, Haeupler and Ghaffari designed an $\widetilde{O}(D)$-round distributed MST algorithm for planar graphs [49]. Note that this algorithm offers a huge advantage over older results for planar graphs with small diameters.

They achieve this by introducing an elegant abstraction for designing distributed algorithms named **low-congestion shortcuts**. Their methods could in principle be used to achieve a similar

result for genus-bounded graphs, but their presented algorithms have a major technical obstacle: they require a surface embedding of the planar/genus bounded graph to construct the low-congestion shortcuts. While computing a distributed embedding for planar graphs has a complex $\widetilde{O}(D)$-round solution [49], this remains an open problem for genus-bounded graphs [49].

This chapter side-steps the issue by vastly simplifying the construction of low-congestion shortcuts. We define a more structured version of low-congestion shortcuts called **tree-restricted shortcuts** and propose a simple and general distributed algorithm for finding them. The algorithm is completely oblivious to any intricacies of the underlying topology and finds universally near-optimal tree-restricted shortcuts. As a simple consequence of our construction technique, we get a $\widetilde{O}(gD)$-round algorithm for genus g graphs, which is a novel result. We believe that this simplicity makes the algorithm usable even in practice.

2.1.2 Our contribution

Roughly speaking, there are two challenges in the design of shortcut-based algorithms. Let $\mathcal{G}$ be the target class of graphs we want to design distributed algorithms. The first challenge is to identify the optimal (smallest) value q such that $\mathcal{G}$ has shortcuts of quality q. This is purely a graph-theoretic problem. The second challenge is to convert the existential result proved by the first challenge to the constructive result, i.e., we must design a distributed algorithm constructing efficient shortcuts for that class. This is a distributed computing problem that might be distinctively harder than the former one. Indeed, while one can prove that bounded genus graphs have good-quality shortcuts, the proof is not constructive because it requires access to an embedding [49]; this is the primary reason why fast algorithms for bounded genus graphs were not known. Even in the planar case, distributedly constructing such an embedding is known, but complicated.

A natural idea to simplify algorithm design would be to come up with a generic procedure that finds a shortcut of quality q for the best (or approximately best) q. Such a result would automatically lift a purely existential result to a constructive one. However, such a result is possible only in the known-topology setting (Chapter 7) and only via complicated techniques. The problem is still open in the standard CONGEST model (we restate the formal statement for convenience).

Open Problem 1.2.6. *Let $\mathcal{P} = (P_1, \ldots, P_N)$ be a set of parts in a graph G. Distributedly construct shortcuts of quality $\widetilde{O}(q)$ in $\widetilde{O}(q)$ rounds, where q is the optimal shortcut quality (with respect to $\mathcal{P}$).*

In this chapter, we resolve Open Problem 1.2.6 for some important classes of graphs. We introduce a more structured definition of shortcuts called **tree-restricted shortcuts** and give a constructive algorithm that finds the nearly optimal tree-restricted shortcuts in any graph that contains them. While the new shortcut definition is a strict subset of the old definition, we leverage them to design optimal $\widetilde{O}(D)$ round distributed algorithms for many graphs of interest (e.g., all planar graphs and all bounded genus graphs).

The details of our contribution are summarized as follows:

- In Section 2.2, we introduce tree-restricted shortcuts, which can only use edges of some fixed spanning tree $T \subseteq G$. Such shortcuts are characterized by congestion c and **block parameter** b (which substitutes the classic dilation parameter). The block parameter is more appropriate for tree-restricted shortcuts due to their highly-structured nature: in particular, the new parameter is stronger in the sense that it implies an upper bound of $O(bD)$ on the dilation. The block parameter (upper-)bounds the number of components of P_i, where two nodes are in different components if they cannot reach each other via H_i. In Section 2.2.3 we propose deterministic algorithms for broadcast, convergecast, and leader election (for all parts in parallel) utilizing tree-restricted shortcuts. These yield a $O(b(D + c))$ round solution to the part-wise aggregation problem (assuming constructed tree-restricted shortcuts), a solution simpler and often faster as compared to the general-case randomized algorithms from [49].
- In Section 2.3, we present a generic algorithm for constructing tree-restricted shortcuts. Given a spanning tree T, we can find near-optimal T-restricted shortcuts, as formalized in the following statement.

  Theorem 2.1.1. *Let $\mathcal{P} = (P_1, \ldots, P_N)$ be parts in the graph G with a spanning tree $T \subseteq G$ such that there exists a T-restricted shortcut with congestion c and block parameter b. There exists a randomized distributed CONGEST algorithm that finds a T-restricted shortcut with congestion $O(c \log N)$ and block parameter $3b$. The shortcut can be found in $\widetilde{O}(b(D + c))$ rounds.*

  Notably, when a tree-restricted shortcut with parameters $b = \widetilde{O}(1)$ and $c = \widetilde{O}(D)$ exists, our construction yields $\widetilde{O}(D)$-quality shortcuts (since dilation is at most $O(bD)$) and, by extension, (optimal) $\widetilde{O}(D)$-round algorithms for MST and approximate Min-Cut.

  Note: The algorithm does not know the values of b and c upfront if one is willing to suffer a $\tilde{O}(1)$ performance hit. In particular, it is possible to find a feasible pair (b, c) that yields a near-optimal value of $b(D + c)$. Given an arbitrary $Q > 0$, one can check if there exists a valid pair of parameters (b, c) that yield a running time of at most $\tilde{O}(b(D + c)) \leq Q$. This is done by trying all $O(\log n)$ possible powers-of-two b that guarantee $\tilde{O}(bD) \leq Q$ and $\tilde{O}(bc) \leq Q$ and truncating the execution after Q rounds. Given this procedure, one can search for the smallest power-of-two Q for which the above procedure succeeds (by checking all $O(\log n)$ possibilities).
- The final question we tackle is what graph families admit good-quality tree-restricted shortcuts. Fortunately, one can reinterpret prior work in the novel terminology of tree-restricted shortcuts to conclude that (any $O(D)$-depth spanning tree of) genus-g graphs contain tree-restricted shortcuts with congestion $O(gD \log D)$ and block parameter $O(\log D)$. In Section 2.2.4, we can obtain a distributed algorithm that constructs a tree-restricted shortcut with congestion $O(gD \log D \log N)$ and block parameter $O(\log D)$ for graphs with genus at most g. For bounded genus graphs (i.e. $g = O(1)$), the algorithms based on our shortcut construction achieve near-optimal time complexity (up to a polylogarithmic factor).

2.1.3 Subsequent work: a short survey

Significant progress has been made since the initial conference version of this chapter was published [65]. Subsequent work has expanded on the utility of the framework by extending it to new graph classes, new problems, and provided better construction guarantees. We intend this section to serve as a short and convenient survey of the tree-restricted shortcut framework.

Tree-restricted shortcut quality and construction. For a spanning tree of depth $O(D)$, we define the **T-quality** (denoted q_T) of a T-restricted shortcut as $q_T := bD + c$ (where b is the block parameter and c is the congestion). This definition is simply the $\text{congestion} + \text{dilation}$, i.e. quality, when one upper-bounds the dilation as $O(bD)$ (see Section 2.2 for a proof of this fact).

T-quality combines the congestion and the block parameter into a single value that sufficiently describes the shortcut construction and routing performance without the need to keep track of multiple parameters.

Definition 2.1.2. *A graph $G = (V, E_G)$ of diameter D **admits** tree-restricted shortcuts of T-quality q_T if for each spanning tree T of depth $O(D)$ and each set of disjoint and connected parts $(P_i \subseteq V)_{i=1}^N$ there exists a T-restricted shortcut of congestion c and block parameter b satisfying $b \cdot D + c \leq q_T$.*

It is not hard to see that if one can efficiently construct shortcuts of T-quality q_T, then a randomized algorithm can solve the part-wise aggregation problem in $\tilde{O}(q_T)$ rounds using standard random delay ideas [49]. However, the key benefit of using the tree-restricted shortcut framework (as opposed to the general shortcut framework) is that near-optimal tree-restricted shortcuts can be efficiently and distributedly constructed.

Theorem 2.1.3 (Theorem 1.2 of [67])**.** *Suppose that a graph $G = (V, E_G)$ admits tree-restricted shortcuts of T-quality q_T. There exists a distributed CONGEST algorithm that finds a T-restricted shortcut with T-quality $\tilde{O}(q_T)$ in $\tilde{O}(q_T)$ rounds and sends at most $\tilde{O}(|E_G|)$ messages during its execution with high probability (with probability at least $1 - n^{-O(1)}$, where any constant can be chosen in the exponent). Moreover, the algorithm does not need to know the value of q_T upfront.*

Note: We slightly reworded the main Theorem of [67]. An appealing property of the tree-restricted shortcut framework (shared between this and subsequent work) is that one does not need to know the optimal tree-restricted shortcut T-quality q_T^* upfront. This can often yield much better shortcuts than guaranteed by the theoretical bound, a property often desired in practical applications. While the paper typically assumes the algorithm knows the congestion c and block parameter b, one can circumvent this issue with a simple exponential parameter search like the one described in Section 2.1.2.

Comparing Theorem 2.2.8 and [67]. Notably, the construction of [67] (unlike Theorem 2.2.8) controls the number of messages throughout the algorithm. Furthermore, it completes in $\tilde{O}(q_T) =$

Graph Family		**Tree-Restricted Shortcut Parameters**			**Lower Bound**
		Block	Congestion	T-quality	$\Omega(d+c)$
General	[49]	1 [2]	$O(\sqrt{n})$	$O(D+\sqrt{n})$	$\widetilde{\Omega}(D+\sqrt{n})$
Pathwidth k	Chapter 3	$O(k)$	$O(k)$	$O(kD)$	$\Omega(kD)$
Treewidth k	Chapter 3	$O(k)$	$O(k\log n)$	$O(kD+k\log n)$	$\Omega(kD)$
Genus g	Chapter 3	$O(\sqrt{g})$	$O(\sqrt{g}D\log D)$	$O(\sqrt{g}D\log D)$	$\Omega(\frac{\sqrt{g}D}{\log g})$
Planar	[49]	$O(\log D)$	$O(D\log D)$	$O(D\log D)$	$\Omega(D\frac{\log D}{\log\log D})$
Minor-excluded	[50]	$O(1)$	$O(D\log n)$	$O(D\log n)$	trivial $\Omega(D)$
No δ-dense minors	[50]	$O(\delta)$	$O(\delta D\log n)$	$O(\delta D\log n)$	$\Omega(\delta D)$

Table 2.1: Upper and lower bounds for tree-restricted shortcuts.

$\tilde{O}(bD+c)$ rounds, while the construction of Theorem 2.2.8 takes $\tilde{O}(b(D+c))$ rounds. The latter result is significantly slower when $b = \log^{\omega(1)} n$, in e.g., genus- or treewidth-bounded graphs with super-polylogarithmic genus or treewidth (see Table 3.1 below). Furthermore, the results of [67] can be made deterministic (with slightly worse guarantees, see below).

Deterministic construction. Many of the aforementioned randomized results can be recovered in the deterministic setting while suffering only a small performance penalty. Notably, one can still construct near-optimal tree-restricted shortcuts and solve the part-wise aggregation problem in $\tilde{O}(b(D+c))$ rounds instead of $\tilde{O}(q_T) = \tilde{O}(bD+c)$ rounds (as guaranteed by the randomized procedure), even while controlling the message complexity.

Theorem 2.1.4 (Deterministic construction of [67])**.** *Suppose that a spanning tree T of a graph $G = (V, E_G)$ admits tree-restricted shortcuts of congestion c and block parameter b. There exists a deterministic distributed CONGEST algorithm that finds a T-restricted shortcut of congestion $\tilde{O}(c)$ and block parameter $\tilde{O}(b)$ in $\tilde{O}(b(D+c))$ rounds and $\tilde{O}(|E_G|)$ messages. Furthermore, one can solve the part-wise aggregation problem with the same guarantees.*

Graph families. Various graph families admit good-quality tree-restricted shortcuts. Table 3.1 lists the known results. The last row of the table references graphs that exclude δ-dense minors, meaning that all minors of G have a density (i.e., the ratio between the number of edges and vertices) at most δ. We note that the result of [50] implies all other known upper bounds in the table (up to logarithmic factors): for instance, minor-excluded families have $\delta = O(1)$.[1]

[1]The excluded-dense-minor result of [50] improves the best-known quality of tree-restricted shortcuts in minor-excluded graph families from $\widetilde{O}(D^2)$ (proved in [68]) to $\widetilde{O}(D)$.

[2]For general graphs, each part of size $|P_i| \geq \sqrt{n}$ is assigned the entire tree; giving them a block param. of 1 and congestion of at most $\sqrt{n}$. Smaller parts can be handled separately in $\tilde{O}(\sqrt{n})$ rounds by using intra-part edges.

Applications. Numerous distributed optimization tasks can be simplified and optimized by utilizing the part-wise aggregation primitive as a black-box subroutine. Applications include the MST, approximate Min-Cut, and approximate single-source shortest path (SSSP) [49, 64, 67].

Corollary 2.1.5. *Suppose that a graph G admits tree-restricted shortcuts of T-quality q_T. One can compute an (exact) MST in $\tilde{O}(q_T)$ rounds and $\tilde{O}(m)$ messages with high probability.*

As a reminder, in the Min-Cut problem, one is given a graph $G = (V, E_G)$ with integer weights $w : E_G \to [1, \text{poly}(n)]$ and needs to compute a set of edges $F \subseteq E_G$ that disconnect G into at least 2 components while minimizing the sum $\sum_{e \in F} w_e$. An α-approximation to Min-Cut finds a set of edges that disconnects the graph whose aggregate weight is at most a multiplicative α factor larger than the optimal value.

Corollary 2.1.6. *Suppose that a graph G admits tree-restricted shortcuts of T-quality q_T. One can compute an $(1+\varepsilon)$-approximate (weighted) Min-Cut in $\tilde{O}(q_T) \cdot \text{poly}(1/\varepsilon)$ rounds and $\tilde{O}(m) \cdot \text{poly}(1/\varepsilon)$ messages with high probability.*

In the Single-Source Shortest Path (SSSP), one is given a graph $G = (V, E_G)$ with integer weights $w : E_G \to [1, \text{poly}(n)]$, a source $s \in V$, and needs to compute a spanning tree $T \subseteq E_G$ such that for each node u we have that $d_T(s, u) = d(s, t)$ where $d(u, v)$ is the distance between $u, v \in V$ in G with respect to the weight w, and $d_T(u, v)$ is their distance in the tree (with respect to w). An α-approximation to SSSP requires the tree to satisfy $d_T(u, v) \leq \alpha \cdot d(u, v)$ (note that the inequality $d_T(u, v) \geq d(u, v)$ is always satisfied).

Corollary 2.1.7. *Suppose that a graph $G = (V, E_G)$ admits tree-restricted shortcuts of T-quality q_T. Each edge $e \in E_G$ has a weight w_e, and let L be the weight-diameter of G. For any $\beta = (\log n)^{-\Omega(1)}$ one can compute an $L^{O(\log\log n)/\log(1/\beta)}$-approximate SSSP in $\tilde{O}(q_T/\beta)$ rounds and $\tilde{O}(m/\beta)$ messages with high probability.*

For instance, in the above corollary, setting $\beta = n^{-\varepsilon}$, $\beta = 2^{-\Theta(\sqrt{n})}$, and $\beta = \log^{-\Theta(1/\varepsilon)} n$ for a constant $\varepsilon > 0$ one obtains a $\log^{O(1)} n$, $2^{\sqrt{\log n}}$, and L^{ε} approximations to SSSP, respectively. [64]

General shortcuts vs. tree-restricted shortcuts. One can easily construct pathological graph examples that admit good-quality *general* shortcuts, but do not admit good-quality *tree-restricted* shortcuts. For example, one can take the lower bound graph of [27] which requires $\widetilde{\Omega}(\sqrt{n})$ rounds to solve MST and replace each edge with $\sqrt{n}$ parallel multi-edge copies. This immediately yields a $\widetilde{O}(D) = \widetilde{O}(1)$ MST solution via general shortcuts, whereas tree-restricted shortcuts are constrained by the original $\widetilde{\Omega}(\sqrt{n})$ lower bound. Moreover, general shortcuts allow faster algorithms for several important graph families. For example, expander graphs and Erdős-Rényi random graphs admit general shortcuts of $\text{dilation} + \text{congestion} = \widetilde{O}(1)$ for any set of parts; no such result is possible in the tree-restricted setting. However, it seems that a distributed construction of general shortcuts is a burdensome task even in highly structured graphs. The best-known result for shortcut construction and part-wise aggregation in expander graphs has

round complexity $2^{O(\sqrt{\log n})} = n^{o(1)}$, significantly worse than the best existential result [52].

2.2 Tree-Restricted Shortcuts

In this section we define tree-restricted shortcuts: a restricted version of low-congestion (i.e., general) shortcuts that are (i) simpler to work with, (ii) often equally powerful as the general shortcuts, (iii) offer deterministic routing schemes and, most importantly, (iv) can be efficiently constructed on any graph that contains them. Following the definitions, we rephrase the relevant prior work in our new term, showcase an efficient deterministic routing scheme, and finally state our main result and applications.

2.2.1 Definition

Tree-restricted shortcuts are low-congestion shortcuts with the additional property that H_i is restricted to (the edges of) some spanning tree T. The user of the shortcut can typically choose any tree T, so a cogent choice would be the BFS tree because of its optimal depth (or any tree of depth $O(D)$).

Definition 2.2.1. *Let $\mathcal{H} = (H_1, H_2, ..., H_N)$ be a (general) shortcut on the graph $G = (V, E_G)$ with respect to the parts $\mathcal{P} = (P_i)_{i=1}^N$. Given a rooted spanning tree $T = (V, E_T) \subseteq G$ we say that a shortcut $\mathcal{H}$ is* tree-restricted *or T-**restricted** if for each $i \in [N], H_i \subseteq E_T$ i.e., every edge of H_i is a tree edge of T.*

Congestion and dilation are still well-defined for tree-restricted shortcuts. However, it is more convenient to use an alternative **block parameter** in place of dilation. The block parameter upper-bounds the number of connected components that H_i induces P_i (note that H_i might not be connected, even though $E_G[P_i] \cup H_i$ is).

Definition 2.2.2. *Let $\mathcal{H} = (H_1, H_2, ..., H_N)$ be a T-restricted shortcut on the graph $G = (V, E_G)$ with respect to the parts $\mathcal{P} = (P_i)_{i=1}^N$. Fix a part P_i and consider the connected components of the spanning subgraph (V, H_i). If such a connected component* intersects *P_i we call it a **block component**. Furthermore, we say $\mathcal{H}$ has **block parameter** b if for each part the number of block components associated with that part is at most b.*

The intersection property ensures that we do not count trivial components unrelated to P_i. Lemma 2.2.3 argues that a block parameter of b implies the dilation of $b(2 \cdot \text{depth}(T) + 1)$. From now on, we will assume that T is chosen to have depth $O(D)$, which is asymptotically minimal and achievable via a BFS tree. We note that distributively computing a BFS tree is a classic problem with a simple $O(D)$ rounds CONGEST algorithm [121].

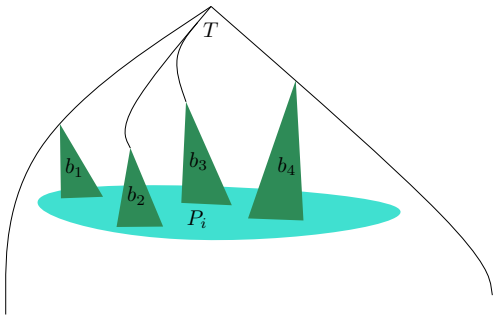


Figure 2.1: Illustration of a T-restricted shortcut subgraph for a part P_i, composed of block components b_1, b_2, b_3 and b_4.

Lemma 2.2.3. *Let T be a spanning tree with depth D and let $\mathcal{H} = (H_i : i \in [N])$ be a T-restricted shortcut with congestion c and block parameter b with respect to parts $\mathcal{P} = (P_i : i \in [N])$. Then the dilation of $\mathcal{H}$ is at most $b(2D + 1)$.*

Proof. Fix $i \in [N]$. Contract every block component of H_i into a supernode and remove all other nodes. This supergraph will contain $b' \leq b$ supernodes and will be connected (because $E_G[P_i]$ is connected). Hence its diameter is $b' - 1 \leq b - 1$. Every supernode corresponds to a block component of diameter $2D$, implying the diameter of $E_G[P_i] \cup H_i$ is at most $2bD + b - 1 < b(2D + 1)$. □

2.2.2 Shortcuts on genus-bounded and planar graphs

Tree-restricted shortcuts are particularly useful on genus-bounded (e.g., planar) graphs. In particular, we can reinterpret the low-congestion result of Haeupler and Ghaffari [49] using our notation.

Theorem 2.2.4 (Haeupler and Ghaffari [49])**.** *Let G be a graph with genus g and diameter D, and let T be any tree with depth $O(D)$ (e.g., BFS tree). There exists a T-restricted shortcut with congestion $O(gD \log D)$ and block parameter $O(\log D)$.*

The paper originally also provided a $O(D \log D)$ upper bound on the dilation of the shortcut. However, this bound can be implicitly recovered from Lemma 2.2.3 and block parameter $O(\log D)$. Note that the Theorem proves only the existence of such shortcuts. While the original paper does describe an algorithm that can in principle be used to compute them, it requires an embedding of G on a surface of genus g. It is an open problem to compute such an embedding efficiently in the CONGEST model [49].

2.2.3 Deterministic routing on tree-restricted shortcuts

In this section, we show how the structure of tree-restricted shortcuts can be useful in facilitating communication within parts. On a high-level, the tree-like structure allows for fast, deterministic, and simultaneous broadcasting/convergecasting on block components; this can be easily extended to true part-wise aggregation. For clarity, broadcast is defined as an operation on a rooted (sub)tree that floods some value from the root down to all other nodes; convergecast is defined as an aggregation of nodes' private values starting from the leaves and towards the root (ending in the root knowing the final aggregate). Lemma 2.2.5 gives a way how to simultaneously perform these primitives on subtrees.

Lemma 2.2.5 (Routing on subtrees)**.** *Let T be a rooted tree of depth $O(D)$ and let $T_1, T_2, \ldots, T_k \subseteq T$ be a family of subtrees where each edge of T is contained in at most c subtrees, i.e., $|\{i \mid e \in T_i, i \in [k]\}| \leq c$. There is a simple deterministic algorithm that can perform a convergecast/broadcast on all of the subtrees in $O(D + c)$ CONGEST rounds.*

Proof. We describe the convergecast algorithm. Each message sent during the algorithm will have a subtree-ID i associated with it. Suppose that a node v is in a subtree T_i (a node can be contained in multiple subtrees). We say (v, i) is active when v receives a message associated with i from all of its T_i-children (if v is a leaf in T_i, then (v, i) is immediately active). When (v, i) becomes active, it will schedule an ID-i message to be sent along its T-parent edge; note that two messages scheduled along the same edge cannot have the same ID. Each round, if multiple messages are scheduled over the same T-edge, the algorithm sends the message associated with the ID i that minimizes $\mathrm{depth}_T(\mathrm{root}(T_i))$. Here, $\mathrm{depth}_T(v)$ is the length of the unique path between $\mathrm{root}(T_i)$ and v path in T. Ties are broken by the ID i itself. The convergecast and broadcast operations are symmetric, so we will only prove the Lemma for convergecasts.

We now analyze the algorithm. Fix a node v. It is sufficient to prove that no message gets transmitted along v's parent edge after $\mathrm{height}_T(v) + c = O(D + c)$ rounds where $\mathrm{height}_T(v)$ is the maximum distance between v and any leaf in T that is a descendant of v (the unique path between the T-root and the leaf goes through v).

Note that any message that gets transmitted along v's parent edge must belong to a subtree T_i that contains that edge. Let $I = (i_1, i_2, ..., i_k)$ be the IDs of subtrees that contain v's parent edge, ordered by their priority (as described). In particular, we say that T_{i_p} has priority p. The congestion condition stipulates that $k \leq c$.

We will prove by induction that for $p \in [k]$ the message associated with i_p will be transmitted no later than round $\mathrm{height}_T(v) + p$. The claim clearly holds for the leaves of T. Note that (i) the relative priority-ordering between I is unchanged with respect to any node of T (other than v), (ii) any subtree T_i that is contained in the set of descendants of v, but does not contain the parent edge of v will have lower priority than any subtree in I.

Fix i_p. By the induction hypothesis, messages corresponding to $\{i_1, \ldots, i_{p-1}\}$ will be sent strictly before round $\mathrm{height}_T(v) + p$. It is sufficient to argue that v has received messages corresponding to i_p from all of its T_{i_p}-children before round $\mathrm{height}_T(v) + p$. However, this can be directly

argued from the induction: for any child $w \in T_{i_p}$ we have $\text{height}_T(w) \leq \text{height}_T(v) - 1$, hence the priority of i_p is at most p with respect to w. Hence v will send the message corresponding to i_p no later than round $\text{height}_T(v) + p$ and we are done.

□

Convergecast and broadcast are used to facilitate routing in tree-restricted shortcuts. We can intuitively envision the shortcut edges H_i as a family of subtrees (in our notation: block components). Aggregation of values within each block component can be exactly achieved by simultaneously convergecasting and broadcasting in all block components. We extend this result to true part-wise aggregation.

Theorem 2.2.6 (Routing on tree-restricted shortcuts)**.** *Given a T-restricted shortcut with congestion c and block parameter b, there are deterministic distributed algorithms that terminate in $O(b(D + c))$ rounds for the following problems.*

1. *Electing a leader for each of the parts in parallel.*
2. *Convergecasting $O(\log n)$-bit messages to the leader of each part in parallel.*
3. *Broadcasting a $O(\log n)$-bit message from the leader of each part in parallel.*

Proof. All of these algorithms have a common flavor: for each part we perceive its shortcut edges H_i as a supergraph of at most b supernodes where each supernode corresponds to a block component. We proceed to describe each of the algorithms on the supergraph and implicitly assume that intra-block communication happens after each step of the algorithm.

Communication within block components can be done in parallel using Lemma 2.2.5: all the nodes of a block component convergecast the relevant information to the block-root and subsequently the block-root broadcasts the result back.

Electing a leader for each part is performed by electing a leader for each supernode (block component) and broadcasting the leader to all neighborhood supernodes for b steps. Every supernode keeps the smallest leader ID ever seen as its current leader. After b rounds all the supernodes have the same leader. The algorithm requires $O(b(D + c))$ rounds as each of the b broadcasting steps is followed by an $O(D + c)$ intra-block communication step.

Broadcasting/convergecasting from/to the leader for each part can be done by building a BFS tree from the leader-supernode. We can utilize the standard distributed BFS algorithm on the supergraph requiring $O(b)$ steps. The algorithm similarly requires $O(b(D+c))$ rounds as each of the $O(b)$ BFS steps is followed by an $O(D + c)$-round intra-block communication step. □

We also state a simple technical lemma we use for the construction of tree-restricted shortcuts.

Lemma 2.2.7. *Given a T-restricted shortcut with congestion c, a deterministic distributed algorithm can identify all parts with at most b' block components. Specifically, after the algorithm terminates each node within a part i knows if P_i is composed of more than b' block components. The algorithm executes in $O(b'(D + c))$ rounds.*

Proof. Similarly to the proof of Theorem 2.2.6, for each part P_i we consider the (connected) supergraph where each supernode corresponds to a block component of H_i. We need to find all parts whose supergraphs have at most b' supernodes.

Each supernode broadcasts its leader for exactly b' rounds and every supernode keeps the minimum ID as their current leader. Subsequently, each leader r (there may be multiple ones as we have not bounded the block parameter) tries to build a BFS tree comprised of all the nodes that believe r is the leader. We can detect the existence of multiple leaders as in that case each BFS tree will contain two neighboring supernodes in different BFS trees and report failure. If this is not the case (all the supernodes of a part belong to the same BFS tree), we can convergecast the number of supernodes back to the root and subsequently broadcasts their count back. □

Comparison with routing on general shortcuts: Ghaffari and Haeupler [49] give a method for routing on general shortcuts in $O(\text{dilation} \cdot \log n + \text{congestion})$ rounds that is randomized and assumes a leader is already elected for each part. They describe a process of leader election via a complicated randomized bootstrapping process that takes $O(\text{dilation} \cdot \log^2 n + \text{congestion} \cdot \log n)$ rounds. We contrast those results with our current tree-restricted shortcut routing where leader election is simple, deterministic, and essentially no more difficult than a single convergecast+broadcast. The downside is that non-tree-restricted shortcuts sometimes offer better quality guarantees and therefore better performance.

2.2.4 Main result and applications

The main contribution of the chapter is to introduce a general framework for finding good-quality shortcuts in graphs where the only assurance is that they exist.

Theorem 2.2.8. *Let G be a graph with a spanning tree $T \subseteq G$ such that there exists a T-restricted shortcut with congestion c and block parameter b. There exists a distributed algorithm that finds a T-restricted shortcut with congestion $O(c \log N)$ and block parameter $3b$ with high probability (with probability at least $1 - n^{-O(1)}$, where any constant can be chosen in the exponent). The shortcut can be found in $O(D \log n \log N + bD \log N + bc \log N)$ rounds.*

We note that the Theorems 2.2.4 and 2.2.8 immediately give a novel result: an algorithm for constructing shortcuts on bounded genus graphs.

Corollary 2.2.9. *Given a genus-g graph with diameter D and N parts there is a distributed algorithm that computes a tree-restricted shortcut with congestion $O(gD \log D \log N)$ and block parameter $O(\log D)$ in $O(gD \log^2 D \log N)$ rounds.*

Next, we explain how to use tree-restricted shortcuts to distributedly compute the Minimum Spanning Tree (MST) on genus-g graphs. Similarly to [49], we incorporate the shortcuts into the classic 1926 algorithm of Boruvka [115].

Lemma 2.2.10. *Given a genus-g graph with n nodes and diameter D, there is a distributed algorithm that computes the Minimum Spanning Tree in $O(gD \log^2 D \log^2 n)$ rounds.*

For completeness we give a brief proof outline:

Proof. Boruvka's algorithm runs in $O(\log n)$ phases. Each phase starts with a partition of the graph into connected parts; each part has previously computed the MST on the subgraph induced by the part. Initially, the algorithm starts with the trivial partition in which each node is in its own part. During each phase, each part P_i suggests a merge along the minimum-weighted edge going out of P_i. It is well-known that all such edges belong to some MST. By computing a tree-restricted shortcut for each part in $O(gD \log^2 D \log n)$ rounds and using our convergecast algorithm on it in $O(gD \log^2 D)$ rounds we can compute the min-weight outgoing edge from each part. A slight difficulty remains: many parts could chain together to form a new part, making the assignment of part IDs in the newly merged part difficulty. This can be avoided by restricting the merge shapes to be star graphs: each part can independently mark itself as a **head** or **tail** with probability $\frac{1}{2}$; we are only allowed to merge tails to heads. The number of phases remains $O(\log n)$ as every minimum-weighted outgoing edge will be used for merging with probability at least $\frac{1}{4}$, thus reducing the expected number of parts by a constant. □

2.3 Constructing Tree Restricted Shortcuts

In this section, we describe an algorithmic framework that solves the problem of finding near-optimal tree-restricted shortcuts.

2.3.1 Overview of the algorithmic framework

Our algorithm `FindShortcut` uses two separate subroutines:

- **Core:** This subroutine finds a good-quality shortcut with respect to at least a constant fraction of the parts. As a prerequisite, we assume we constructed a tree T with depth $O(D)$ such there exists a T-restricted shortcut with congestion c and block parameter b. Note that we only assume the shortcut's existence.

  Lemma 2.3.1. *Let T be a spanning tree with depth $O(D)$ and assume there exists a T-restricted shortcut with congestion c and block parameter b. The subroutine `CoreFast` finds a T-restricted shortcut $\mathcal{H}' = (H'_i)_{i=1}^N$ with the following properties:*

  1. *The congestion of $\mathcal{H}'$ is at most $8c$ with high probability.*
  2. *There exists a subset of parts $\mathcal{P}' \subseteq \mathcal{P}$ with size at least $|\mathcal{P}'| \geq \frac{|\mathcal{P}|}{2}$ such that each part in $\mathcal{P}'$ has at most $3b$ block components.*

  The subroutine takes $O(D \log n + c)$ CONGEST rounds to execute. Upon completion, each node knows for each of its incident edges which parts are they assigned to in $\mathcal{H}'$.

We divide out the exposition of the core subroutine in two versions: a deterministic, and simper `CoreSlow` requiring $O(D \cdot c)$ rounds; and a randomized `CoreFast` requiring $O(D \log n + c)$ rounds. We note that the `CoreFast` subroutine is the only randomized building block of our framework. Therefore, we can replace it with a deterministic (albeit slower) version at a cost of an additional $\frac{c}{\log n}$ factor.

- **Verification:** This subroutine is used to identify the parts whose shortcut edges H_i have a sufficiently small number of block components.

> **Lemma 2.3.2.** *Given a tree T with depth D and a tentative T-restricted shortcut $\mathcal{H}'$ with congestion c, the deterministic subroutine `Verification` finds all parts $\mathcal{P}' \subseteq \mathcal{P}$ whose designated shortcuts have at most b' block components. The subroutine takes $O(b'(D+c))$ CONGEST rounds to execute. Upon completion, each node knows whether its part is in the set $\mathcal{P}'$ or not.*

We use the subroutines in `FindShortcut` that implements the construction of Theorem 2.2.8.

Algorithm FindShortcut: We run the `CoreFast` subroutine that computes a shortcut $\mathcal{H}' = \{H'_1, \ldots, H'_N\}$ with congestion $8c$, but possibly an unacceptably large block parameter. The next step is to run the `Verification` subroutine that finds all parts whose computed shortcut edges H'_i have at most $3b$ block components. We call those parts **good** and fix their computed shortcut edges and discard the rest. The subroutine is iteratively repeated for $O(\log N)$ rounds at which point the parts have been marked as good.

Proof of Theorem 2.2.8. By Lemma 2.3.1, in each iteration we find a shortcut with congestion $8c$ and block parameter $3b$ for at least a half of the parts that have not yet been marked as good, w.h.p. This implies that after $O(\log N)$ iterations all the parts are marked as good. This further implies that the congestion of $\mathcal{H}'$ is $O(c \log N)$ as the congestion of the union of partial shortcuts is at most the sum of congestion of individual partial shortcuts.

Finally, the number of rounds is at most $O(\log N)$ times the combined number of rounds of the `CoreFast` and `Verification` subroutines, namely $O(\log N \cdot (D \log n + c + bD + bc)) = O(D \log N \log n + bD \log N + bc \log N)$. □

2.3.2 Warm-up: an $O(D \cdot c)$-round version of the core subroutine

In this section, we explain a simple and deterministic, but slower version of the core subroutine named `CoreSlow` that terminates in $O(D \cdot c)$ CONGEST rounds. We improve its round complexity to $O(D \log n + c)$ in the following section.

On a high level, the subroutine takes each part P_i and tries to assign the T-ancestors of nodes in P_i to its shortcut edges H'_i. However, this might lead to a large congestion on some edges. We address this issue by declaring an edge **unusable** if more than $2c$ different parts try to use it. This ensures the congestion is at most $2c$. We show the process provably leads to a constant fraction of parts having small congestion and a small block parameter.

Preliminaries: As standard, assume we fix a spanning tree $T = (V, E_T)$ of depth $O(D)$ such that G has a T-restricted shortcut with congestion c and block parameter b. During the execution of the algorithm, some of the edges will be marked as **unusable**. Furthermore, we say that a tree edge $e \in E_T$ **can see** a node $v \in V$ if v is in the subtree of e and no edge on the unique path between the lower endpoint of e and v is unusable. Analogously, an edge can see a part P_i if it can see any node in P_i.

Outline of the `CoreSlow` subroutine: Initially, no edge is unusable. We process the (tree) edges of T in order of decreasing depth (bottom to top). An edge e is assigned to all parts P_i that e can see. If an edge is assigned to more than $2c$ different parts, we mark this edge e as **unusable** disallow e from being used at all by any part.

Detailed description of the `CoreSlow` subroutine: Each node v maintains a list L_v of part IDs that its T-parent edge can see. The lists L_v are initially empty. The subroutine runs in $\text{depth}(T)$ phases where in phase k all the nodes at depth $\text{depth}(T) - k$ update L_v simultaneously and send the entire list L_v to its T-parent. Consider a node v that receives $L_{v'}$ for all its T-children v'. We assign the union of all received lists and the singleton part ID of v (if any) to L_v. If $|L_v| \leq 2c$, we assign the parent edge of v to all the parts in L_v and transmit L_v to its parent (potentially requiring $2c$ rounds). Otherwise, if $|L_v| > 2c$, we declare the parent edge as unusable.

A direct implementation of this would lead to a subroutine that takes $O(D \cdot c)$ rounds in the CONGEST model. Each of the $O(D)$ levels of T must propagate at most $2c$ part IDs to their parent nodes. However, this bottleneck can be improved by random sampling, as we show in the next section with the subroutine `CoreFast`.

Algorithm 1 `CoreSlow`

1. At time k each node v at depth $depth(T) - k$ does the following in parallel:
   (a) if v is an element of P_i, set $L_v \leftarrow \{i\}$, otherwise $L_v \leftarrow \emptyset$
   (b) receive all the part IDs from v's children and assign their union to L'
   (c) $L_v \leftarrow L_v \cup L'$
   (d) if $|L_v| > 2c$, mark v's parent edge as unusable
   (e) otherwise (serially) send all the part IDs of L_v up to v's parent node
2. For each node v:
   (a) if the parent edge e of v is marked as unusable, e will not be assigned to any part
   (b) otherwise, e will be assigned to all $H_i, \forall i \in L_v$

Lemma 2.3.3. *Let T be a spanning tree of depth $O(D)$ and assume there exists a T-restricted shortcut with congestion c and block parameter b. The subroutine `CoreSlow` finds a T-restricted shortcut $\mathcal{H}' = (H'_1, H'_2, ..., H'_N)$ with the following properties:*

1. *The congestion of $\mathcal{H}'$ is at most $2c$.*
2. *There exists a subset of parts $\mathcal{P}' \subseteq \mathcal{P}$ with size at least $|\mathcal{P}'| \geq \frac{|\mathcal{P}|}{2}$ such that each part*

in $\mathcal{P}'$ has at most $3b$ block components.

The subroutine takes $O(D \cdot c)$ CONGEST rounds to execute. Upon completion, each node knows for each of its incident edges which parts are they assigned to in $\mathcal{H}'$.

Proof. Let $\mathcal{H} = (H_i)$ be any T-restricted shortcut with congestion c and block parameter b and let $\mathcal{H}' = (H'_i)$ be the shortcut computed by `CoreSlow`. We call $\mathcal{H}$ the **canonical** shortcut and $\mathcal{H}'$ the **computed** shortcut.

By construction, the congestion of $\mathcal{H}'$ is $2c$ as any edge that would be assigned to more than $2c$ parts is marked as unusable. Hence we proved property 1.

Let $U \subseteq E_T$ be the set of unusable edges marked by the subroutine. In this paragraph, we find an upper bound for $|U|$. Consider **blaming** a part P_i for congesting an unusable edge $e \in U$ when $e \notin E_G[P_i] \cup H_i$ and e can see P_i, i.e., edge e was not in the canonical shortcut H_i, but e was congested by part P_i (and ultimately declared unusable). Each part can be blamed at most b times because each block component can only be blamed for the first unusable edge in his T-tree path towards the T-root. Furthermore, if e is unusable, it takes at least $2c - c$ different block components (from different parts) to be blamed for congesting e. Therefore $|U| \leq N\frac{b}{c}$.

We say that a part P_i **missed** an edge e when $e \in E_G[P_i] \cup H_i$ and $e \in U$ (consequently, $e \notin H'_i$). Furthermore, call a part **bad** if it missed at least $2b$ edges and **good** otherwise. Note that if a part P_i is good, the block parameter of H'_i is at most $2b + \text{blockParameter}(\mathcal{H}) = 3b$. This is because each missed edge induces a new block component in $\mathcal{H}'$ (more precisely, we can identify each block component of $\mathcal{H}'$ by either a unique block component of $\mathcal{H}$ or a unique missed edge $e \in U$). Consequently, it is sufficient to prove that the subroutine finds at least $\frac{1}{2}N$ good parts.

As any unusable edge is assigned to at most c parts in the canonical shortcut, and for a part to be bad we need at least $2b$ edges to be missed, we have that the number of bad parts is at most $|U|\frac{c}{2b} \leq \frac{1}{2}N$. Hence, the subroutine finds at least $\frac{1}{2}N$ good shortcuts, proving property 2.

The subroutine terminates in $O(D \cdot c)$ rounds: on each of the $O(D)$ levels of the tree T, all the nodes in parallel must send the part IDs trying to use its parent edge up the tree. A node can send up to $2c$ IDs, each requiring one round for its transmission. □

2.3.3 A faster $O(D \log n + c)$-round version of the core subroutine

In this section, we describe a faster version of the core subroutine named `CoreFast`. On a high level, we lower the running time of `CoreSlow` by estimating the number of parts trying to use an edge by random sampling. In particular, each part becomes **active** with probability p and we declare an edge unusable when $\Omega(c \cdot p)$ active parts try to use that edge.

Preliminaries: In addition to the preliminaries of `CoreSlow` we need shared randomness between all the nodes within a part. In other words, all the nodes of the same part must have access to the same seeds for a pseudorandom generator. This can be done by sharing $O(\log^2 n)$ random bits among all the nodes of G in $O(D + \log n)$ rounds, as described in [49].

Outline of the `CoreFast` subroutine: Each part becomes **active** with probability $p = \frac{\gamma \log n}{2c}$

where $\gamma > 0$ is sufficiently large constant. We (basically) follow the `CoreSlow` subroutine, but instead of propagating all $O(c)$ part IDs of L_v, we propagate only the active ones. An edge is declared **unusable** if at least $4c \cdot p = \Omega(\log n)$ (active) part IDs want to use it. Hence, by a standard Chernoff bound argument we can claim with high probability that (i) we never propagate more than $O(\log n)$ part IDs through an edge, (ii) each unusable edge has at least $2c$ part IDs trying to use that edge, and (iii) each usable (non-congested) edge has at most $8c$ part IDs. After determining which edges are unusable in $O(D \log n)$ rounds, `CoreFast` must nevertheless find the complete set of part IDs that can use each edge. This is a tree routing problem where each message (part ID) has to be routed up the tree T until the first unusable edge. No message needs to travel more than D edges and no edge needs to transmit more than $8c$ different part IDs w.h.p. Hence this routing can be done in $O(D + c)$ using Lemma 2.2.5.

Detailed description of the `CoreSlow` subroutine: Due to shared randomness, each part independently becomes **active** with probability $p = \frac{\gamma \log n}{2c}$ (all the nodes within the part agree on this label). Similarly, as in `CoreSlow`, each node v maintains a list $\tilde{L}_v$ of active part IDs that its (T) parent edge can see. The lists $\tilde{L}_v$ are initially empty. The subroutine runs in $\text{depth}(T)$ phases where in phase k all the nodes at depth $\text{depth}(T) - k$ try to update $\tilde{L}_v$ in parallel and send $\tilde{L}_v$ to its T-parent. Consider a node v that receives $L_{v'}$ for all its T-children v'. We assign the union of all received lists and the singleton part ID of v (if any) to L_v. If $|L_v| \leq 4c \cdot p$, we assign the parent edge of v to all the parts in L_v and transmit L_v to its parent (potentially requiring $O(\log n)$ rounds). Otherwise, if $|L_v| > 4 \cdot p$, we declare the parent edge as unusable. This finalizes the first part of the subroutine where we determine all unusable edges. It remains to forward the complete set of part IDs (and not just the sampled ones) that can use some edge e to the endpoints of e. This is a classic tree routing problem where no route has its length larger than D and no edge intersects more than $8c$ paths w.h.p. Lemma 2.2.5 provides a method to route all part IDs in at most $O(D + c)$ rounds. Note that any two part-IDs whose routes share an edge have the same endpoint (lowest unusable ancestor edge), so any routing priority between the messages gives the aforementioned $O(D + c)$ bound w.h.p.

Lemma (Restated Lemma 2.3.1)**.** *Let T be a spanning tree with depth $O(D)$ and assume there exists a T-restricted shortcut with congestion c and block parameter b. The subroutine `CoreFast` finds a T-restricted shortcut $\mathcal{H}' = (H'_i)_{i=1}^N$ with the following properties:*

1. *The congestion of $\mathcal{H}'$ is at most $8c$ with high probability.*
2. *There exists a subset of parts $\mathcal{P}' \subseteq \mathcal{P}$ with size at least $|\mathcal{P}'| \geq \frac{|\mathcal{P}|}{2}$ such that each part in $\mathcal{P}'$ has at most $3b$ block components.*

The subroutine takes $O(D \log n + c)$ CONGEST rounds to execute. Upon completion, each node knows for each of its incident edges which parts are they assigned to in $\mathcal{H}'$.

Proof. This proof extensively utilizes methods used in the proof of Lemma 2.3.3. For completeness, we redefine all of the used terminologies and reprove all of the intermediate results.

Let $\mathcal{H} = (H_i)$ be any T-restricted shortcut with congestion c and block parameter b and let $\mathcal{H}' = (H'_i)$ be the shortcut computed by `CoreFast`. We call $\mathcal{H}$ the **canonical** shortcut and $\mathcal{H}'$ the **computed** shortcut.

Algorithm 2 `CoreFast`

1. Each part becomes active with probability $p = \frac{\gamma \log n}{2c}$
2. At time k each node v at depth $depth(T) - k$ does the following in parallel:
   (a) if v is an element of P_i and P_i is active, set $\tilde{L}_v \leftarrow \{i\}$, otherwise $\tilde{L}_v \leftarrow \emptyset$
   (b) receive all the active part IDs from v's children and assign their union to L'
   (c) $\tilde{L}_v \leftarrow \tilde{L}_v \cup L'$
   (d) if $|\tilde{L}_v| \geq 4c \cdot p$, mark v's parent edge as unusable
   (e) otherwise send all the part IDs $\tilde{L}_v$ up to v's parent node
3. Each node v initializes Q_v with its part ID (or $\emptyset$ if not in any part)
4. Each node v does the following in parallel:
   (a) add all received IDs to the Q_v
   (b) if the parent edge of v is not unusable and $\exists i \in Q_v$ that was never forwarded
      i. forward minimum such i along the parent edge
5. Each part ID in Q_v can use the parent edge of v unless it is unusable

As $4c \cdot p = \Omega(\log n)$, a standard Chernoff bound argument demonstrates that any edge that is not marked as unusable can see at most $8c$ different part IDs w.h.p. Hence, the congestion of $\mathcal{H}'$ is $8c$ w.h.p.

Let $U \subseteq E_T$ be the set of unusable edges marked by the subroutine. In this paragraph, we find an upper bound for $|U|$. Consider **blaming** a part P_i for congesting an unusable edge $e \in U$ when $e \notin E_G[P_i] \cup H_i$ and e can see P_i, i.e., edge e was not in the canonical shortcut H_i, but e was congested by part P_i (and ultimately declared unusable). We argue via a Chernoff bound that each unusable edge $e \in U$ can see at least $2c$ parts, hence we blame at least $2c - \text{congestion}(\mathcal{H}) = c$ parts for congesting e. Each part can be blamed at most b times because each block component can only be blamed for the first unusable edge in his T-tree path towards the T-root. Furthermore, if e is unusable, it takes at least $2c - c$ different block components (from different parts) to be blamed for congesting e. Therefore $|U| \leq N\frac{b}{c}$.

We say that a part P_i **missed** an edge e when $e \in E_G[P_i] \cup H_i$ and $e \in U$ (consequently $e \notin H'_i$). Furthermore, call a part **bad** if it missed at least $2b$ edges and **good** otherwise. Note that if a part P_i is good, the block parameter of H'_i is at most $2b + \text{blockParameter}(\mathcal{H}) = 3b$. This is because each missed edge induces a new block component in $\mathcal{H}'$ (more precisely, we can identify each block component of $\mathcal{H}'$ by either a unique block component of $\mathcal{H}$ or a unique missed edge $e \in U$). Consequently, it is sufficient to prove that the subroutine finds at least $\frac{1}{2}N$ good parts.

As any unusable edge is assigned to at most c parts in the canonical shortcut and for a part to be bad we need at least $2b$ edges to be missed, we have that the number of bad parts is at most $|U|\frac{c}{2b} \leq \frac{1}{2}N$. Hence, the subroutine finds at least $\frac{1}{2}N$ good shortcuts.

The subroutine takes $O(D \log n + c)$ rounds: on each of the $O(D)$ levels of the tree T, all the nodes in parallel must send the active part IDs that its parent edge can see. If an edge e is not

unusable, a Chernoff bound proves that at most $O(c \cdot p) = O(\log n)$ active part IDs can be seen from e, hence the number of rounds for determining unusable edges is $O(D \log n)$ w.h.p. Finally, propagating the part IDs upwards along T described in Lemma 2.2.5 takes $O(D + c)$ rounds, bringing the total number of rounds to $O(D \log n + c)$. □

2.3.4 Verification subroutine

In this section, we describe the `Verification` subroutine. Given a tree-restricted shortcut with congestion c and possibly unbounded block parameter, it inspects each part in parallel and marks the ones that have at most $b' = 3b$ block components.

The subroutine runs precisely the algorithm described in Lemma 2.2.7 which we restate here.

Lemma (Restated Lemma 2.2.7)**.** *Given a T-restricted shortcut with congestion c, a deterministic distributed algorithm can identify all parts with at most b' block components. Specifically, after the algorithm terminates each node within a part i knows if P_i is composed of more than b' block components. The algorithm executes in $O(b'(D + c))$ rounds.*

The Lemma provides a direct method to implement the formal requirements of the `Verification` subroutine which we restate here for clarity.

Lemma (Restated Lemma 2.3.2)**.** *Given a tree T with depth D and a tentative T-restricted shortcut $\mathcal{H}'$ with congestion c, the deterministic subroutine `Verification` finds all parts $\mathcal{P}' \subseteq \mathcal{P}$ whose designated shortcuts have at most b' block components. The subroutine takes $O(b'(D + c))$ CONGEST rounds to execute. Upon completion, each node knows whether its part is in the set $\mathcal{P}'$ or not.*

Chapter 3

Shortcuts for Treewidth-Bounded and Genus-Bounded Graphs

The results of this chapter were published in [66] with Bernhard Haeupler and Taisuke Izumi as co-authors. The work was supported in part by KAKENHI No. 15H00852 and 16H02878 as well as NSF grants CCF-1527110 "Distributed Algorithms for Near Planar Networks" and NSF-BSF grant "Coding for Distributed Computing".

3.1 Introduction

We show that many distributed network optimization problems can be solved much more efficiently in structured and topologically simple networks. We show this by utilizing the tree-restricted shortcut framework. We show that good-quality tree-restricted shortcuts exist in pathwidth-bounded graphs, treewidth-bounded graphs, well-separated graphs and genus-bounded graphs. This existence result, together with the uniform tree-restricted shortcut construction (Theorem 2.1.3) yields a constructive results. We also exhibit lower bounds that show one cannot do better using any shortcut-related method (and using the methods presented in Chapter 6 one can show that no distributed algorithm can do better).

It was known that there exist low-congestion shortcuts on genus g graphs of quality $\tilde{O}(gD)$. However, it was not clear how to construct them, making the result algorithmically unusable [49]. Furthermore, it wasn't known what other families allow for faster algorithms and how tight is the bound. This chapter makes progress in exactly this direction. We show that on g-genus, treewidth-k, pathwidth-k and k-well-separated graphs one can do much better and circumvent the $\tilde{\Omega}(\sqrt{n})$ lower bound. In particular, we show that bounded genus graphs admit $\tilde{O}(\sqrt{g}D)$ round algorithms, while the others admit $\tilde{O}(kD)$ round algorithms for MST, min-cut and other problems. All of this bounds are tight up to logarithmic factors and they represent the first tight results for special families of graphs.

3.2 Technical Results

The contribution of this chapter is to show the existence of good-quality tree-restricted shortcuts for multiple classes of graphs: bounded pathwidth, bounded treewidth, bounded genus and well-separated graphs. These results, using Theorem 2.1.3, imply the first distributed MST algorithm that circumvents the $\tilde{\Omega}(\sqrt{n})$ lower bound for those graphs.

Furthermore, we show that by using the low-congestion shortcut framework, one cannot hope to do much better. Specifically, we prove a lower bound on the quality (i.e., on $d + c$) for any low-congestion shortcut with dilation d and congestion c on pathwidth bounded and genus bounded graphs. These lower bounds almost match (within logarithmic factors) the proved upper bounds. Those two lower bounds show that one typically does not lose any power by restricting oneself from low-congestion shortcuts to tree-restricted shortcuts.As will be noted later, the k-pathwidth bounded graphs are also k-treewidth bounded graphs and k-well-separated graphs, so the lower bound for pathwidth bounded graphs applies to all of those classes. The results and lower bound are summarized in Table 3.1. Note that $O(bD + c)$ is the analogue of $O(d + c)$ for the tree-restricted case.

Graph Family	**Tree-restricted Shortcut T-Quality**			**Lower Bound**
	Block	Congestion	$O(bD + c)$	$\Omega(d + c)$
Pathwidth k	$O(k)$	$O(k)$	$O(kD)$	$\Omega(kD)$
Treewidth k	$O(k)$	$O(k \log n)$	$O(kD + k \log n)$	$\Omega(kD)$
k-Well-Separated Graphs	$O(k \log n)$	$O(k \log n)$	$O(kD \log n)$	$\Omega(kD)$
Genus g Graphs	$O(\sqrt{g})$	$O(\sqrt{g}D \log D)$	$O(\sqrt{g}D \log D)$	$\Omega(\frac{\sqrt{g}D}{\log g})$
Planar Graphs [49]	$O(\log D)$	$O(D \log D)$	$O(D \log D)$	$\Omega(D\frac{\log D}{\log \log D})$

Table 3.1: Upper and lower bounds for tree-restricted shortcuts in this chapter

We note here one important technical difficulty that applies to distributed algorithms on genus bounded graphs. While we prove that optimal $\tilde{O}(\sqrt{g}D)$ congestion and $\tilde{O}(\sqrt{g})$ block parameter shortcuts do exist, their construction via Theorem 2.2.8 proven in Chapter 2 takes $\tilde{O}(gD)$ rounds. To mitigate this, we invoke the improved construction via Theorem 2.2.8 to produce the shortcuts in $\tilde{O}(bD + c)$ rounds, giving a $\tilde{O}(\sqrt{g}D)$ construction for optimal genus bounded shortcuts.

The rest of the chapter is organized as follows: we first show that good-quality tree-restricted shortcuts exist for pathwidth bounded graphs in Section 3.3, followed by the existence of good-quality tree-restricted shortcuts for treewidth bounded graphs in Section 3.4. Tree-restricted shortcuts for well-separated graphs are deferred to Section 3.7. We then prove the lower bound for these three mentioned classes by exhibiting a lower bound for pathwidth bounded graphs in Section 3.5. After that, we turn our attention to tree-restricted shortcuts for bounded genus graphs in Section 3.6 and, finally, exhibit a tight lower bound for them in Section 3.6.3.

3.3 Pathwidth Bounded Graphs

In this section we show that k-pathwidth graphs admit tree-restricted shortcuts with congestion $O(k)$ and block parameter $O(k)$. As noted before, this enables us to leverage the Construction Theorem 2.2.8 to design efficient algorithms for pathwidth bounded graphs.

Given a graph $G = (V, E_G)$, a **path decomposition** of G is a sequence of subsets $\mathcal{PD} = (X_1, X_2, ..., X_r), X_i \subseteq V$ with the following properties: *i)* $\bigcup_{i=1}^{r} = V$; *ii)* For all $\{v, w\} \in E_G$ there exists i such that $a \in X_i, b \in X_i$; *iii)* For all $v \in V$ there exists $1 \leq s_v \leq t_v \leq r$ such that $v \in X_i \iff i \in [s_v, t_v]$. We call the subsets X_i **bags**. The width of the path decomposition $\mathcal{PD}$ is $k := \max_i |X_i| - 1$. The minimal possible width of a path decomposition of G is called the **pathwidth** of G.

For $v \in V$ let $I(v)$ be the set of indices of bags that contain v. Note that property *iii)* implies that $I(v)$ is an interval of integers. Similarly, for a set $P \subseteq V$ we define $I(P)$ as $\bigcup_{v \in P} I(v)$. Note that for a connected vertex set P (such as any part), $I(P)$ is also an interval of integers.

Lemma 3.3.1. *Let $\mathcal{PD}$ be a k-width path decomposition of a graph $G = (V, E_G)$. For any rooted spanning tree $T = (V, E_T) \subseteq G$, there exists a T-restricted shortcut with congestion $O(k)$ and block parameter $O(k)$.*

Proof. Deferred to Section 3.8 □

3.4 Treewidth Bounded Graphs

In this section we show that k-treewidth graphs with n nodes admit tree-restricted shortcuts with congestion $O(k \log n)$ and block parameter $O(k)$.

Given a graph $G = (V, E_G)$, a **tree decomposition** of G is a tree $\mathcal{TD} = (\mathcal{X}, E_{\mathcal{T}})$. The nodes of $\mathcal{TD}$, $\mathcal{X} = (X_1, ..., X_{|\mathcal{X}|})$ are called **bags**. Each bag corresponds to a subset of V, the nodes of the original graph G. For the sake of presentation, we will identify the bag X_i and this corresponding subset of nodes. The tree decomposition has to satisfy these properties: *i)* the union of all sets X_i equals V, i.e., $\mathcal{X}$ is a partition of V; *ii)* for each $v \in V$ the bags containing vertex v form a connected subtree of $\mathcal{TD}$; *iii)* for every edge $\{a, b\} \in E_G$ there is a bag X_i that contains both a and b. The **width** of the tree decomposition $\mathcal{TD}$ is $k := \max_i |X_i| - 1$. The minimal possible width of a tree decomposition of G is called the **treewidth** of G.

Lemma 3.4.1. *Let $\mathcal{TD}$ be a k-width tree decomposition of a graph $G = (V, E_G)$ rooted in an arbitrary bag such that its depth is $D_{\mathcal{TD}}$. For any rooted spanning tree $T = (V, E_T) \subseteq G$ there exists a T-restricted shortcut with congestion $O(kD_{\mathcal{TD}})$ and block parameter $O(k)$.*

Proof. Deferred to Section 3.8 □

Corollary 3.4.2. *Given a n-node graph G with treewidth k and a spanning tree $T \subseteq G$, there exist a T-restricted shortcut with congestion $O(k \log n)$ and block parameter $O(k)$.*

Proof. By Bodlaender and Hagerup [17], for a graph with treewidth k there exists a $O(k)$-width tree decomposition with depth $O(\log n)$. Applying Lemma 3.4.1 finishes the argument. □

3.5 Lower Bound for Pathwidth Bounded Graphs

In this section we prove a lower bound for general low-congestion shortcuts for pathwidth bounded graphs. In particular, we prove that there exists a family of pathwidth bounded graphs $\mathcal{G}_P(\Gamma, w, \delta)$ for which any low-congestion shortcut either must have large congestion or large dilation. More precisely, we exhibit a k-pathwidth graph family such that for any shortcut with dilation d and congestion c it must hold that $d + c = \Omega(kD)$. Note that this result also implies a lower bound for treewidth bounded and well-separated graphs as k-pathwidth graphs are both k-treewidth and k-well-separated.

We now describe the graph family $\mathcal{G}_P(\Gamma, w, \delta)$ in detail, depicted in Figure 3.1. All parameters of the graph Γ, w, δ are positive integers. Furthermore, $\Gamma \geq 2, w \geq 2, \delta \geq 2$ and w is a power of 2.

The construction consists of two main parts: Γ different **lanes** and a **tree** $\mathcal{T}$. The lanes are denoted by $\{L^1, L^2, ..., L^\Gamma\}$. Each lane L^l is constructed in two steps: first we take a path consisting of w **named nodes** $v_0^l, v_1^l, ..., v_{w-1}^l$ connected by single edges, and then we subdivide each edge by adding $2\delta - 1$ **unnamed nodes** in its interior.

The tree $\mathcal{T}$ is a perfect binary tree with w leaves (note again that w is a power of 2). $\mathcal{T}$ has $p = 1 + \log_2 w$ different levels (depths) where the root is on level $p - 1$ and the leaves are on level 0. The tree nodes on level l are denoted by $u_0^l, u_1^l, ..., u_{2^l-1}^l$. Finally, the tree and the lanes are connected: each named node on the lane v_i^l is connected by a **cross edge** to the leaf u_i^0 in the tree.

Observation 3.5.1. *The graph $\mathcal{G}_P(\Gamma, w, \delta)$ has $\Theta(\Gamma w \delta)$ nodes. Its diameter is $\Theta(\log w + \delta)$ and pathwidth is $O(w)$.*

Proof. The only non-trivial part is the pathwidth. We construct a $O(w)$-width path decomposition of $\mathcal{G}_P$. First, construct a $O(1)$-width path decomposition of each lane L^l in isolation. Next, to each bag in a decomposition of the lane L^l we add all w named nodes of that lane. Next, concatenate the path decompositions together and obtain a $O(w)$ path decomposition of the union of all lanes. Next, add all the nodes of the tree $\mathcal{T}$ to each bag in every lane (there are $O(w)$ nodes that are added). Finally, we have a valid $O(w)$-width path decomposition of $\mathcal{G}_P$. □

Lemma 3.5.2. *There exists a node partition on $\mathcal{G}_P(\Gamma, w, \delta)$ such that the (general) shortcut for this partition either has dilation $\Omega(w\delta)$ or congestion $\Omega(\frac{\Gamma}{\log w})$.*

Proof. Deferred to Section 3.8 □

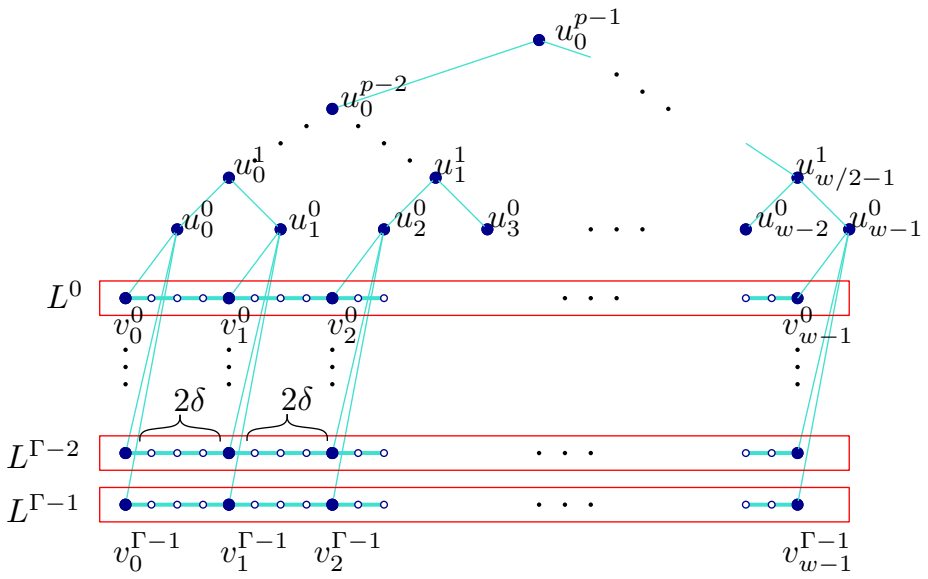


Figure 3.1: Graph $\mathcal{G}_P(\Gamma, w, \delta)$

Corollary 3.5.3. *Given $k \geq 2, D = \Omega(\log k)$ and a sufficiently large n, there exists a graph with $O(n)$ nodes that has* i) *pathwidth $O(k)$,* ii) *diameter $\Theta(D)$,* iii) *there exists a node partition $\mathcal{P}$ such that any (general) shortcut for that partition must have either dilation $\Omega(kD)$ or congestion $\Omega(\frac{n}{Dk \log k})$.*

Proof. This corollary follows directly from Lemma 3.5.2 by taking the graph $\mathcal{G}_P(\Gamma, w, \delta)$ with $\Gamma = \frac{n}{kD}, \delta = D$ and $w = \Theta(k)$ (note that we can always find a power of 2 within a constant factor of any k). □

3.6 Genus-Bounded Graph

The main idea of our construction is a reduction from the planar-graph case: We first construct another planar graph J related to the original genus-g graph G, and compute a good shortcut for J. Then, we map each shortcut subgraph in J to a subgraph in G as a shortcut in G. We first introduce the general framework of this "mapping" strategy.

3.6.1 Graph Extension

Definition 3.6.1. *A graph J is an* extension *of a graph G if G is obtained from J by deleting edges or contracting vertex pairs (contracting pair may not be adjacent, and multiedges caused by a contraction is merged into a single edge).*

Throughout this section we use notation $V(G)$ and $E(G)$ to indicate the sets of vertices and edges in G respectively. Node contraction maps several nodes in J to a node in G. The mapping

is denoted by $f : V(J) \to V(G)$. Let $\overline{E}$ be the set of deleted edges. By the definition of contraction, for any two nodes $v, u \in V(J)$ such that $f(v) \neq f(u)$, if $(v, u) \in E(J) \setminus \overline{E}$ holds, $(f(v), f(u)) \in E(G)$ also holds. That is, there exists a mapping from $E(J) \setminus \overline{E}$ to $E(G)$. We commonly use function f to indicate this edge mapping. We define, we define $f^{-1}(v) = \{v' \in V(J) | f(v') = v\}$ for any $v \in V(G)$, and define $f^{-1}(e)$ for edge $e \in E(G)$ similarly. The cardinality of $f^{-1}(e)$ for edge $e \in E(G)$ is called the *multiplicity* of e. The maximum multiplicity of all edges in G are denoted by μ. The mappings f and f^{-1} are also extended for the set of vertices or edges. For example, for $U \in V(G)$, we define $f^{-1}(U) = \cup_{u \in U} f^{-1}(u)$. All other cases are defined similarly. Let $\lambda = |V(J)| - |V(G)|$ for short.

A hurdle of converting a shortcut in J to G is that given a connected component U in G, $G[f^{-1}(U)]$ is not necessarily connected (i.e., each part in G is fragmented into several subparts in J). The following lemma states the bound on the number of fragments.

Lemma 3.6.2. *Let G be a graph and J be its extension. Given a node subset $U \subset V(G)$ such that the subgraph of G induced by U is connected, the subgraph of J induced by $f^{-1}(U)$ consists of at most $|f^{-1}(U)| - |U| + 1$ connected components.*

Proof. Deferred to Section 3.8 □

Lemma 3.6.3. *Let G be a graph, J be its extension, T be a spanning tree of G, T^{-1} be a spanning tree of J, and $\nu = |E(T^{-1}) \setminus f^{-1}(T)|$. If J has a T^{-1}-restricted shortcut with congestion c and block parameter b, then G has a T-restricted shortcut with congestion $\mu c + \alpha$ and block parameter $(\lambda b + \nu c)/\alpha + 1$ for any $\alpha \geq 1$.*

Proof. Deferred to Section 3.8 □

3.6.2 Optimal Shortcut for Genus-g Graphs

The core of the proof for genus-g graphs is the following lemma.

Lemma 3.6.4. *Let G be any graph of genus g and diameter D. Then there exists an extension J of G satisfying the following conditions:* i) *J is planar,* ii) *There exists a spanning tree T^{-1} with depth at most $2D + 1$, and* iii) *$\mu = 2$, $\lambda \leq 12gD$, and $\nu \leq 12g$ for T^{-1}.*

To prove this lemma, we prepare several notions related to graph embeddings on surfaces: Let G be a graph of genus g. In the following argument we assume that G is 2-cell embedded in an orientable surface of genus g, which is denoted by Σ_g[1]. A loop on a surface Σ is a continuous function $f : [0, 1] \to \Sigma$ satisfying $f(0) = f(1)$. For any spanning tree T of G and an edge e not contained in T, graph $G + e$ contains exactly one simple cycle. We denote it by $loop(T, e)$. We also use the notation $loop(T, e)$ as the loop on surface Σ if G is embedded in Σ. A key tool of our proof is the following theorem.

[1] 2-cell embedding is the embedding where every face on Σ is topologically isomorphic to an open disk.

Theorem 3.6.5 (Eppstein [39]). *Let G be a graph of genus g and consider its arbitrary 2-cell embedding on Σ_g. Given any node $v_x \in V$, let T be the shortest path tree of G rooted by v_x. Then there exists a set B of $2g$ edges $= \{e'_1, e'_2, \ldots, e'_{2g}\}$ such that a set of loops $loop(T, e'_1)$, $loop(T, e'_2), \ldots, loop(T, e'_{2g})$ generates the fundamental group of the surface Σ_g whose base point is v_x.*

Let G' be the subgraph of G induced by the set of edges $\cup_{i=1}^{2g} E(loop(T, e'_i))$, and T' be a subtree of T obtained from G' by removing all edges in B. Then the following lemma holds:

Lemma 3.6.6. *Given any 2-cell embedding of G into Σ_g, we remove all edges and vertices in $G - G'$. After the removal, we obtain a embedding of G' into Σ_g. This embedding is still a 2-cell embedding and the number of faces is one.*

Proof. Deferred to Section 3.8 □

This lemma implies that by "cutting" Σ_g along the (embedded) edges in $E(G')$, it becomes topologically equivalent to a disk. In other words, if we embed some graph on Σ_g without crossing $\cup_{i=1}^{2g} loop(T, e'_i)$, it becomes a planar embedding. The proof of lemma 3.6.4 is to identify a graph J which is an extension of G and planarly embeddable on Σ_g in the sense above.

Proof. Deferred to Section 3.8 □

It is known that any planar graph has a T-restricted shortcut for congestion $O(D_T \log D_T)$ and blocking parameter $O(\log D_T)$ [49, 65], where D_T is the depth of T. Combining that fact with Lemmas 3.6.3 and 3.6.4 with $\alpha = \sqrt{g} D \log D$, we obtain the main theorem.

Theorem 3.6.7. *Any graph with genus g has a T-restricted shortcut with congestion $O(\sqrt{g}$ $D \log D)$ and blocking parameter $O(\sqrt{g})$ for any spanning tree T with diameter D.*

3.6.3 Lower Bounds for Genus Bounded Graphs

In this section we prove a lower bound for general low-congestion shortcuts for genus bounded graphs. More precisely, we exhibit a g-genus graph family $\mathcal{G}_P(\Gamma, w, \delta)$ such that for any shortcut with dilation d and congestion c it holds that $d + c = \tilde{\Omega}(\sqrt{g} D)$.

We now describe the graph family $\mathcal{G}_G(w, \delta)$ in detail. The family is depicted in Figure 3.2. The construction and reasoning are very similar to the pathwidth lower bound graphs $\mathcal{G}_P$.

The parameters of the graph $w \geq 2, \delta \geq 2$ are positive integers and w is a power of 2. The construction consists of two main parts: $\delta \cdot (w - 1) + 1$ different **lanes** and a **tree** $\mathcal{T}$. The lanes are denoted by $\{L^0, L^1, L^2, ..., L^{\delta(w-1)}\}$. Each lane L^l is constructed in two steps: first we take a path consisting of w **named nodes** $v^l_0, v^l_1, ..., v^l_{w-1}$ connected by single edges, and then we subdivide each edge by adding $\delta - 1$ **unnamed nodes** in its interior. Next, every we have add w **vertical paths**. The i'th vertical path connects $t^i_0, t^i_1, .., t^i_{\delta(w-1)}$, where t^i_i is the $\delta \cdot i$'th (named or unnamed) vertex on the i'th lane.

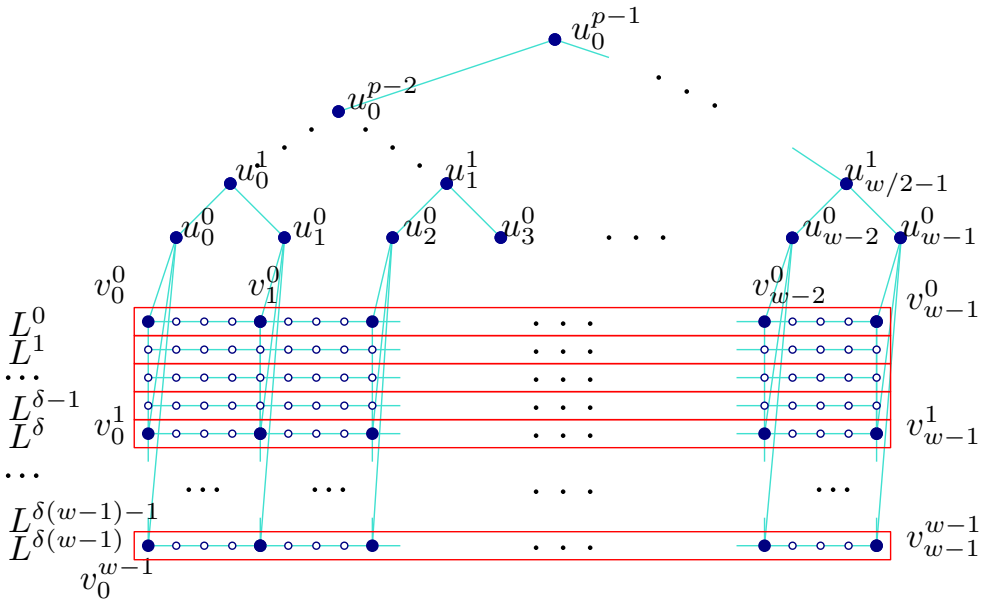


Figure 3.2: Graph $\mathcal{G}_G(w, \delta)$

The tree $\mathcal{T}$ is a perfect binary tree with w leaves (note again that w is a power of 2). $\mathcal{T}$ has $p = 1 + \log_2 w$ different levels (depths) where the root is on level $p - 1$ and the leaves are on level 0. The tree nodes on level l are denoted by $u_0^l, u_1^l, ..., u_{2^l-1}^l$. Finally, the tree and the lanes are connected: each named node on the lane v_i^l is connected by a **cross edge** to the leaf u_i^0 in the tree.

Observation 3.6.8. *The graph $\mathcal{G}_G(w, \delta)$ has $\Theta(w^2\delta^2)$ nodes. Its diameter is $\Theta(\log w + \delta)$ and genus is $O(w^2)$.*

Proof. The only non-trivial part is the genus. Note that if we remove the $\Theta(w^2)$ cross edges we have a planar graph (union of a grid and a tree). Also, adding an edge increases the genus by at most 1, hence the genus is $O(w^2)$. □

Lemma 3.6.9. *There exists a node partition on $\mathcal{G}_G(w, \delta)$ such that the (general) shortcut for this partition either has dilation $\Omega(w\delta)$ or congestion $\Omega(\frac{w\delta}{\log w})$.*

Proof Sketch. Deferred to Section 3.8 □

Corollary 3.6.10. *Given $g \geq 2, D = \Omega(\log g)$ and a sufficiently large n, there exists a graph with $O(n)$ nodes that has* i) *genus $O(g)$,* ii) *diameter $\Theta(D)$,* iii) *there exists a node partition $\mathcal{P}$ such that any (general) shortcut for that partition must have either dilation $\Omega(\sqrt{g}D)$ or congestion $\Omega(\frac{\sqrt{g}D}{\log g})$.*

Proof. This corollary follows directly from Lemma 3.6.9 by taking the graph $\mathcal{G}_G(w, \delta)$ with $w = \sqrt{g}$ and $\delta = D$ (note that we can always find a power of 2 within a constant factor of any $\sqrt{g}$). □

3.7 Chapter Appendix: Graphs with Small Separators

In this section we show that any well-separated graph admits good tree-restricted shortcuts. We first define the necessary preliminaries and then proceed to prove the result.

Preliminaries: Let $E_G[S]$ denote the set of edges in E_G with both endpoints in S. Let $G[S]$ denote the induced subgraph of G, namely $(S, E_G[S])$. Furthermore, for some $S \subseteq V$, the notation $G - S$ denotes the subgraph $(V_G - S, \{\{a, b\} | \{a, b\} \in E_G, a \notin S, b \notin S\})$.

Let $G = (V_G, E_G)$ be a (undirected) graph and $S \subseteq V_G$ be a subset of its vertices. A **well-balanced k-separator** of G is a subset $S \subseteq V_G$ such that $|S| \leq k$ and all components of $G - S$ contain at most $\frac{2}{3}|V_G|$ vertices. A graph G is **k-well-separated** if all of its subgraphs $F \subseteq G$ contain a well-balanced k-separator.

Main result of this section: Let $G = (V_G, E_G), n := |V_G|$ be a k-well-separated graph and let $T = (V_G, E_T)$ be any rooted spanning tree of G. We show that G has a T-restricted shortcut with congestion $O(k \log n)$ and block $O(k \log n)$.

We set up a recursive algorithm that takes a subgraph $F = (V_F, E_F)$ as a parameter and finds shortcut subgraphs for all parts completely contained in V_F. We now present the algorithm in detail.

Recursive algorithm: Given a subgraph $F = (V_F, E_F)$, we find a well-balanced separator $S \subseteq V_F, |S| \leq k$. The graph $F - S$ collapses into several connected components. Let their vertex sets be $\{C_1, C_2, .., C_t\}, C_i \subseteq V_F$ where the well-separation property guarantees that $|C_i| \leq \frac{2}{3}|V_F|$. For each component $C_i, i \in [t]$ we recursively apply the algorithm on the subgraph $(C_i \cup S, E_F[C_i \cup S] - E_F[S])$. Finally, each part that intersects S and is completely contained in F is given $E_F \cap E_T$ as its shortcut subgraph, i.e., such parts can use all the tree edges of T contained in F. The recursion terminates when $|V_F| \leq 10k$, at which point we give all of $E_F \cap E_T$ to all parts completely contained in F.

Lemma 3.7.1. *Let $G = (V_G, E_G), n := |V_G|$ be a k-well-separated graph and let $T = (V_G, E_T)$ be any rooted spanning tree of G. Then G admits a T-restricted shortcut with congestion $O(k \log n)$ and block $O(k \log n)$.*

Proof. Before analyzing the recursive algorithm described above we have to describe some preliminaries. For a subgraph $(V_F, E_F) = F \subseteq G = (V_G, E_G)$ we define the **boundary** ∂F as the set of nodes $v \in V_F$ such that there exists an edge $(v, a) \in E_G$ where $a \in V_G - V_F$.

First note that the depth of the recursion d is at most $O(\log n)$ as we always find a well-balanced separator, thereby exponentially decreasing $|V_F|$ in each subsequent child recursive call. Also, note that in any recursive call with F as its argument it holds that $|\partial F| \leq k \cdot d = O(k \log n)$. To prove this, denote the separating sets of the parent recursive call by $S_1, S_2, ..., S_{d-1}$ and let $\mathscr{S} = \bigcup_{i=1}^{d-1} S_i$. Then $\partial F \subseteq \mathscr{S}$ since any $v \in V_F$ not contained in $\mathscr{S}$ has the same set of direct neighbors in F as in G, hence cannot be on the boundary of F.

We first bound the congestion. For each non-leaf recursive call we find a separator S and assign the T-tree edges of F to at most $|S| \leq k$ parts, as each part that is assigned a shortcut sub-

graph must intersect S. Similarly, in a leaf recursive call, tree edges are assigned to $O(k)$ parts. Also, note that if $e = \{a, b\} \in E_F$, then e will appear either in *i)* no child recursive calls (if $a \in S, b \in S$) or *ii)* exactly one child recursive call (otherwise). Therefore, if we consider an edge $e \in E_T$, the number of recursive calls (with argument F) in which $e \in E_F$ is bounded by the depth of the recursion, namely $O(\log n)$. In each of those recursive calls e can be assigned to at most $O(k)$ different shortcut subgraphs, hence the congestion is $O(k \log n)$.

We now bound the number of block components for a fixed part i. Similarly to the proof of Lemma 3.4.1 we will count the maximum number of block components by counting the maximum number of possible distinct roots of those block components. All block component roots can be constructed in the following manner: start with a node in the part $v \in P_i$ and travel along its T-parent edges while the edge exists and is in the shortcut subgraph H_i. The process clearly ends in the root of the block component r_{bc}. It is sufficient to prove that either *i)* r_{bc} is the root of T or *ii)* $r_{bc} \in \partial F$, hence there can be $O(k \log n)$ different possibilities for r_{bc} and, consequently, $O(k \log n)$ block components.

By construction, i will be assigned a shortcut subgraph in exactly one recursive call. Let F be the argument subgraph given to that specific call. Take any $v \in P_i$ and travel upwards along T, as described above. Denote the block component root we end up in by r_{bc}. Assume, for the sake of contradiction, that r_{bc} is not the root of T and is not in ∂F. Then all of its original direct neighbors in G are still neighbors in F. Specifically, its T-parent is still incident to it, hence the process will not stop at such a node. This proves the claim.

□

Corollary 3.7.2. *A graph with n nodes, diameter D and treewidth k admits a tree-restricted shortcut with congestion $O(k \log n)$ and block $O(k \log n)$.*

Proof. We set T to be the BFS tree of G and apply Lemma 3.7.1. □

Note that the result for treewidth bounded (Corollary 3.4.2) and well-separated graphs are equivalent up to logarithmic factors (Lemma 3.7.1). A graph with treewidth k is also k-well-separated and a graph that is k-well-separated has treewidth $O(k \log n)$ [60].

3.8 Chapter Appendix: Deferred Proofs

Lemma (Restated Lemma 2.2.3)**.**

Lemma (Restated Lemma 3.3.1)**.** *Let $\mathcal{PD}$ be a k-width path decomposition of a graph $G = (V, E_G)$. For any rooted spanning tree $T = (V, E_T) \subseteq G$, there exists a T-restricted shortcut with congestion $O(k)$ and block parameter $O(k)$.*

Proof. Denote the parts as $\mathcal{P} = (P_1, P_2, ..., P_N)$ and fix a part P_i. Call a node $v \in V$ **admissible** if $I(v) \subseteq I(P_i)$, i.e., if the interval of the node if a subset of the partwise interval. Let A_i be the set of all admissible nodes.

The shortcut subgraphs H_i can be constructed in the following way: H_i contains all tree edges $\{a, b\} \in E_T$ where a is closer to the root if $b \in A_i$.

We first prove that congestion of this tree-restricted shortcut is $O(k)$. Fix an edge $e = \{a, b\} \in E_T$ as before and denote by L_b the lowest-numbered bag containing b. If a shortcut subgraph $H_i \ni e$ then by construction there exist a node $v \in P_i$ that is contained in L_b. Hence the number of shortcut subgraphs that contain e is at most $|L_b| = O(k)$.

To bound the block parameter, fix a part P_i. Call a node $v \in V$ **absorbing** if it is contained in either the lowest-numbered or highest-numbered bag of $I(P_i)$. Denote all absorbing nodes by B_i and note that $|B_i| \leq 2k + 2 = O(k)$. To upper bound the number of block components of part i, we will count the number of nodes that can be the root of a block component (each block component can be bijectively represented by its root). Since every block component of part i must intersect P_i, we can generate the set of roots in the following manner: start with a node $v \in P_i$ and travel along its T-parent edges while the edge exists and is in H_i. The process clearly ends in the root of the block component r_{bc}. It is sufficient to prove that either r_{bc} is the root of T or $r_{bc} \in B_i$, hence there can be $O(k)$ different possibilities for r_{bc} and, consequently, $O(k)$ block components.

We start the process in some $v \in P_i$. By construction, $v \in A_i \implies v \in A_i \cup B_i$. In each step it holds that either: *i)* v is the root of T, in which case we are done; *ii)* $v \in B_i$, in which case $r_{bc} = v$ and we are done. Note that its parent is not in H_i; *iii)* $v \in A_i$, in which case we move to its T-parent v'. Note that, by construction, $v' \in A_i \cup B_i$. Hence by induction we can prove that we always end in the root of T or B_i, which proves the claim. □

Lemma (Restated Lemma 3.4.1)**.** *Let $\mathcal{TD}$ be a k-width tree decomposition of a graph $G = (V, E_G)$ rooted in an arbitrary bag such that its depth is $D_{\mathcal{TD}}$. For any rooted spanning tree $T = (V, E_T) \subseteq G$ there exists a T-restricted shortcut with congestion $O(kD_{\mathcal{TD}})$ and block parameter $O(k)$.*

Proof. Denote the parts as $\mathcal{P} = (P_1, P_2, ..., P_N)$. Fix P_i and let $\mathcal{B}_i$ be the set of all $\mathcal{TD}$ bags that intersects P_i or whose ancestors (in the $\mathcal{TD}$ tree) intersects P_i and let B_i be the union of nodes contained in any bag of $\mathcal{B}_i$. The set of bags $\mathcal{B}_i$ corresponds to a (connected) subtree on the (rooted) tree decomposition, hence its **lower common ancestor bag** L_i is well-defined.

We now define A_i, the set of **admissible nodes** for part P_i. A_i contains all nodes that are in bags $\mathcal{B}_i$, but are not contained in the bag L_i.

We are ready to define the set of edges contained in the shortcut subgraph H_i. Specifically, H_i contains all T-tree edges $\{a, b\} \in E_T$ where a is closer to the root if $b \in A_i$.

We first prove that the congestion of this construction is $O(kD_{\mathcal{TD}})$. Let $\{a, b\} \in E_T$ be an edge contained in H_i where a is closer to the root of T. Then, by construction, $b \in A_i$, which implies $b \in B_i$ and $b \notin L_i$.

But there exists only $O(D_{\mathcal{TD}})$ possible bags for L_i: L_i has to not contain b and one of its ancestors has to contain b - but only $O(D_{\mathcal{TD}})$ bags satisfy this condition. Namely, if we distinguish the bags that contain b, those bags correspond to a connected subtree in $\mathcal{TD}$. L_i must be a ancestor of the lowest common ancestor of this distinguished subtree. Furthermore, only $k + 1$ different parts

can have some fixed bag X as its lowest common ancestor bag L_i. This follows because each such part P_i must intersect X, and there are at most $k+1$ nodes in X. Hence $\{a, b\}$ is contained in at most $O(kD_{\mathcal{TD}})$ parts, as required.

We now prove that for each fixed part P_i, the number of block components is $O(k)$. Each block component can be represented by its root. Since every block component of part i must intersect P_i, we can generate the set of roots in the following manner: start with a node $v \in P_i$ and travel along its T-parent edges while the edge exists and is in H_i. The process clearly ends in the root of the block component r_{bc}. It is sufficient to prove that either r_{bc} is the root of T or $r_{bc} \in L_i$, hence there can be $O(k)$ different possibilities for r_{bc} and, consequently, $O(k)$ block components.

We start the process in $v \in P_i$. By construction, $v \in B_i$. In each step it holds that either:

i) v is the root of T, in which case we are done.

ii) $v \in L_i$, in which case $r_{bc} = v$ and we are done. Note that its parent is not in H_i.

iii) $v \in A_i$, in which case we move to its T-parent v'. Note that, by construction, $v' \in B_i$.

Hence by induction we can prove that we always end in the root of T or L_i, which proves the claim. □

Lemma (Restated Lemma 3.5.2)**.** *There exists a node partition on $\mathcal{G}_P(\Gamma, w, \delta)$ such that the (general) shortcut for this partition either has dilation $\Omega(w\delta)$ or congestion $\Omega(\frac{\Gamma}{\log w})$.*

Proof. We let each lane L^l be its own part $P_l, l \in [\Gamma]$, as depicted by a red box in Figure 3.1. There are Γ parts in total.

In order to prove that a shortcut subgraph for a part either has large dilation or has to congested edges of $\mathcal{T}$, we define **potential** on all of the edges in $\mathcal{G}_P(\Gamma, w, \delta)$ in the following way:

- every cross edge is assigned a potential of 0
- every edge between two nodes on a lane is assigned a potential of 1
- every tree edge between u_i^l and u_j^{l+1} is assigned a potential of $\delta 2^l$

Define the **potential of a path** as the sum of potentials of the edges on that path. Observe that the potential of any path between any leftmost node of a lane v_0^l and rightmost node of the same lane v_{w-1}^l is at least $(w-1)2\delta = \Omega(w\delta)$. Also, note that the sum of potentials of all edges in $\mathcal{T}$ is $O(\delta w \log w)$.

For the sake of contradiction assume that there exists a shortcut $\mathcal{H}$ with dilation $d = o(w\delta)$ and congestion c. Then for each part (i.e., lane) P_l there exists a path in $\mathcal{G}_P[P_l] + H_l$ of length $O(d)$ between the leftmost and rightmost node in its lane. The potential of that path is at least $\Omega(w\delta)$, but at at most $O(d)$ of this potential can come from edges from a lane. Hence, at least $\Omega(w\delta - d) = \Omega(w\delta)$ of the potential has to come from edges on the tree $\mathcal{T}$. In other words, if we define T_l as the subset of tree edges of $\mathcal{T}$ that shortcut subgraph H_l uses and define $\phi(T_l)$ as the sum of their potentials, then $\phi(T_l) = \Omega(w\delta)$. Consequently, $\sum_{i=0}^{\Gamma-1} \phi(T_l) = \Omega(\Gamma w\delta)$.

But on the other hand, each (tree) edge in $\mathcal{T}$ can only be contained in c different shortcut subgraphs, so $\sum_{i=0}^{\Gamma-1} \phi(T_l) = O(c\delta w \log w)$ since the sum of potentials of all tree edges in $\mathcal{T}$ is $O(\delta w \log w)$. If follows that $c = \Omega(\frac{\Gamma}{\log w})$, as required. □

Lemma (Restated Lemma 3.6.2)**.** *Let G be a graph and J be its extension. Given a node subset $U \subset V(G)$ such that the subgraph of G induced by U is connected, the subgraph of J induced by $f^{-1}(U)$ consists of at most $|f^{-1}(U)| - |U| + 1$ connected components.*

Proof. Let $J_U = J[f^{-1}(U)]$ for short. Let T_U be any spanning tree of $G[U]$. By the definition of f^{-1}, $f^{-1}(e_1)$ and $f^{-1}(e_2)$ are disjoint for any $e_1, e_2 \in V(G[U])$ if $e_1 \neq e_2$. Thus $f^{-1}(E(T_U))$ is a set of edges in J_U with a size at least $|E(T_U)| = |U| - 1$. In addition, the edge set $f^{-1}(E(T_U))$ is cycle-free (because if it contains a cycle C, T_U must contain a cycle $f(C)$, which is a contradiction). Thus J_U contains a forest $(V(J_U), f^{-1}(E(T_U))$ with $|U| - 1$ edges. Since $|V(J_U)| = \sum_{u \in U} |f^{-1}(u)|$ holds, the number of connected components in that forest is $\sum_{u \in U} |f^{-1}(u)| - |U| + 1 = |f^{-1}(U)| - |U| + 1$. This gives an upper bound on the number of connected components in J_U. □

Lemma (Restated Lemma 3.6.3)**.** *Let G be a graph, J be its extension, T be a spanning tree of G, T^{-1} be a spanning tree of J, and $\nu = |E(T^{-1}) \setminus f^{-1}(T)|$. If J has a T^{-1}-restricted shortcut with congestion c and block parameter b, then G has a T-restricted shortcut with congestion $\mu c + \alpha$ and block parameter $(\lambda b + \nu c)/\alpha + 1$ for any $\alpha \geq 1$.*

Proof. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be the node partition of $V(G)$ for which we want to compute a shortcut. Now we define the node partition of $V(J)$ from $\mathcal{P}$ as follows: Letting $Q_i = f^{-1}(P_i)$ for any $i \in [1, N]$, we denote by $\mathcal{Q}_i = \{Q_{i,1}, Q_{i,2}, \ldots Q_{i,\gamma(i)}\}$ the family of the vertex sets of the connected components in $J[Q_i]$, where $\gamma(i)$ is the number of connected components in $J[Q_i]$. First, we construct a T^{-1}-restricted shortcut for J and partition $\cup_{0 \leq i \leq N} \mathcal{Q}_i$. Let $I'_{i,k}$ be the shortcut subgraph augmented with $Q_{i,k}$, and $I_{i,k} = I'_{i,k} - E(T^{-1}) \setminus f^{-1}(T)$. The number of edges removed from $I'_{i,k}$ is denoted by $\nu_{i,k}$. Since $I'_{i,k}$ consists of at most b subtrees of T^{-1}, after removing $\nu_{i,k}$ edges, $I_{i,k}$ consists of $b + \nu_{i,k}$ subtrees of T^{-1} because one edge removal separates one tree into two (sub)trees. Let $b_i = \gamma(i)b + \sum_{k \in [1,\gamma(i)]} \nu_{i,k}$. The shortcut edges H_i for part P_i in G is constructed as follows: *i)* Set $H_i = G[f(I_i)]$ if $b_i \leq (\lambda b + \nu c)/\alpha + 1$ *ii)* Otherwise, $H_i = T$. Since node contraction never increases the number of blocks, b_i is the upper bound on the number of block components for part P_i. Thus the construction above trivially achieves the block parameter $(\lambda b + \nu c)/\alpha + 1$. The remaining issue is that the maximum congestion is bounded by $\mu c + \alpha$. Since at most μ edges in T^{-1} are mapped into the same edge in T, the maximum congestion incurred by the first-case construction is bounded by μc. Then it suffices to show that at most α parts apply the second case. Suppose for contradiction that more than α parts have block parameters larger than $(\lambda b + \nu c)/\alpha + 1$. Without loss of generality, we assume $P_1, P_2, \ldots P_{\alpha+1}$ are those parts. Then we have $\sum_{i \in [1,\alpha+1]} b_i \geq (\alpha + 1)((\lambda b + \nu c)/\alpha + 1) > (\lambda b + \nu c) + \alpha + 1$. In addition, for any P_i such that $i > \alpha + 1$, we also have $b_i \geq 1$. Consequently. $\sum_{i \in [1,N]} b_i > \lambda b + \nu c + N$. On the other hand, since an edge in $E(T^{-1}) \setminus f^{-1}(T)$ is used at most c times as a shortcut edge, $\sum_{i \in [1,N]} \nu_i$ is at most νc. Furthermore, by Lemma 3.6.2, $\sum_{i \in [1,N]} \gamma(i) \leq \lambda + N$ also holds, and thus we obtain $\sum_{i \in [1,N]} b_i \leq \lambda b + \nu c + N$. It is a contradiction, and the lemma holds. □

Lemma (Restated Lemma 3.6.6)**.** *Given any 2-cell embedding of G into Σ_g, we remove all edges and vertices in $G - G'$. After the removal, we obtain a embedding of G' into Σ_g. This embedding is still a 2-cell embedding and the number of faces is one.*

Proof. It is obvious that the embedding of G' stated by the lemma is 2-cell embedding: If it has a face not topologically isomorphic to a disk, by cutting and capping all the boundaries in that face by disks, we can obtain an embedding of G' on a surface with genus $g - 1$ or less, which contradicts the fact that G' is the union of the generators of Σ_g. The number of faces is obtained by applying Euler's formula. Since $E(G')$ consists of a subtree T' of T spanning G' and $2g$ edges not in T but whose endpoints are both in T'. Thus the total number of the edges is $|V(G')| - 1 + 2g$. Since G' is 2-cell embedded in Σ_g, by applying Euler's formula, we can conclude that the number of faces for that embedding is one. □

Lemma (Restated Lemma 3.6.4)**.** *Let G be any graph of genus g and diameter D. Then there exists an extension J of G satisfying the following conditions:* i) *J is planar,* ii) *There exists a spanning tree T^{-1} with depth at most $2D + 1$, and* iii) *$\mu = 2$, $\lambda \leq 12gD$, and $\nu \leq 12g$ for T^{-1}.*

Proof. We prove the lemma in a constructive way. Let $\overline{G} = G - G'$ for short. Now we have a 2-cell embedding of G (and thus an embedding of G') into a genus-g orientable surface. It uniquely determines a clockwise cyclic ordering of the set of edges incident to each vertex. Let $e_i^1, e_i^2, \ldots, v_i^{\delta(i)}$ be that ordering around v_i (where $\delta(i)$ is the degree of v_i in G), and without loss of generality we assume that e_i^1 be the upward edge in T. In addition, we also denote by $e_i^{t_1}, e_i^{t_2}, e_i^{t_3}, \ldots, e_i^{t_{\Delta(i)}}$ be the ordering of the edges in G' around v_i, where $\Delta(i)$ is the degree of v_i in G'. Let $e_i[j] = \{e_i^h | t_j < h < t_{j+1}\}$ if $j < \Delta(i)$ and $e_i[j] = \{e_i^h | t_j < h \leq \delta(i)\}$ otherwise.

The construction of J and its embedding on Σ_g follow the steps below:

1. For each $v_i \in V(G')$, we define S_i as a mutually disjoint region topologically isomorphic to a disk on Σ_g containing v_i (i.e, each S_i contains only one vertex v_i). We also define $S_{i,j}$ as a connected region containing S_i, S_j and edge (v_i, v_j) properly (i.e., any single point does not separate the region), but excluding all other vertices and edges not intersecting S_i or S_j (see Figure 3.3(1)). Note that those regions are always definable by taking a small regions whose boundary arbitrarily close to the edge (v_i, v_j). Let $S = \cup_{(v_i,v_j)\in V(G')} S_{i,j}$ for short.

2. Since exactly one node lies in S_i, the set of edges in G' (more precisely, their drawing as a curve on Σ_g) separates S_i into $\Delta(i)$ subregions. Let S_i^h be the subregion separated by the edges $e_i^{t_h}$ and $e_i^{t_{h+1}}$. We add new $\Delta(i)$ nodes to G, each of which is placed in each subregion. The node placed in S_i^h is denoted by v_i^h. For each $h \in [1, \Delta(i)]$, we replace the endpoint v_i of all edges in $e_i[h]$ by v_i^h .This replacement can be done without crossing any edges (as depicted in Figure 3.3(2)).

3. For any edge $(v_i, v_j) \in E(G')$, we add two corresponding edges (v_i^{r-}, v_j^s) and (v_i^r, v_j^{s-}), where r and s are the values such that t_r and t_s corresponds to the orders of edge (v_i, v_j) around v_i and v_j and $r-$ and $s-$ mean the predecessors of r and s in the cyclic ordering (see Figure 3.3(3)). We can draw these edges within $S_{i,j}$ so that they do not cross any edge, and the number of edges augmented to each node is exactly two.

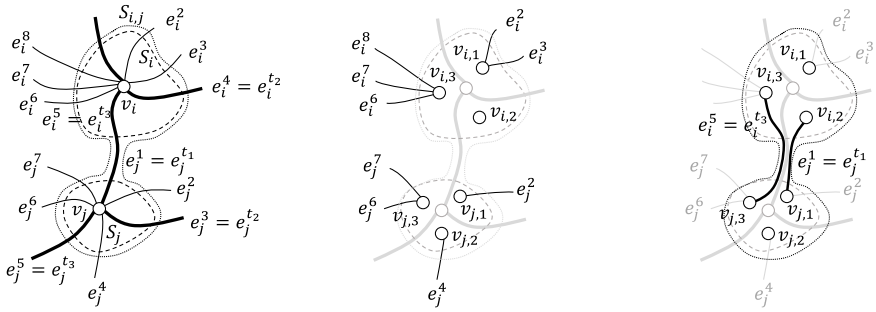


Figure 3.3: Steps 2 and 3 in the construction of J

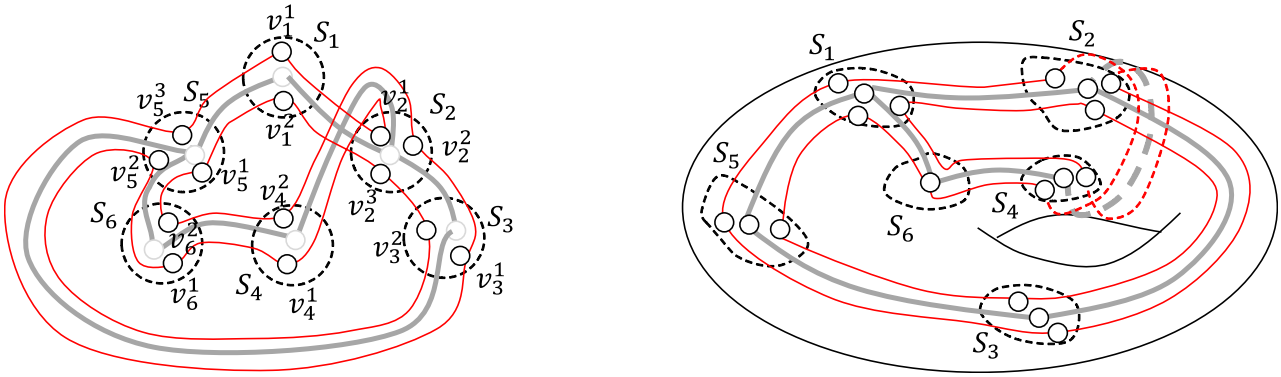


Figure 3.4: Cycle C (The right figure is a drawing on σ_1 of the left side).

4. We remove all the vertices and edges in G'. From Lemma 3.6.6, this operation makes the graph becomes planar, which is drawn on the single disk defined by the embedding of G'. In addition, since any edge in G' becomes the boundary of the disk, it can be drawn on the plane so that all the nodes and edges strictly contained in S belong to the outer boundary (see Figure 3.4). It also implies that all the nodes in S (i.e., newly-added nodes) forms a cycle (denoted by C) because in the subgraph induced by those vertices any node has degree two. The final step of the construction is to shorten the diameter by augmenting edges. Letting $u_1, u_2, \cdots u_l$ be the ordering of vertices in C (u_1 is chosen arbitrarily), we augment edge (u_1, u_{iD}) for all $i \in [1, \lfloor l/D \rfloor]$. The set of edges augmented in this step is denoted by E''. It is obvious that we can preserve the planarity for this augmentation (see Figure 3.5). The resultant graph is J we construct.

It is obvious from the construction that (1) J is planar and (2) J is an extension of G: The construction above provides a planar embedding of J, and by removing all edges in E'' and contracting $v_i^1, v_i^2, \ldots v_i^{\Delta(i)}$ into one node v_i for all $v_i \in V(T')$ we can recover the original graph G. Then an edge (v_i, v_j) in G' with order r and s around v_i and v_j has the correspondence from two edges (v_i^{r-}, v_j^s) and (v_i^r, v_j^{s-}), and all other edges have one-to-one correspondence. Thus we

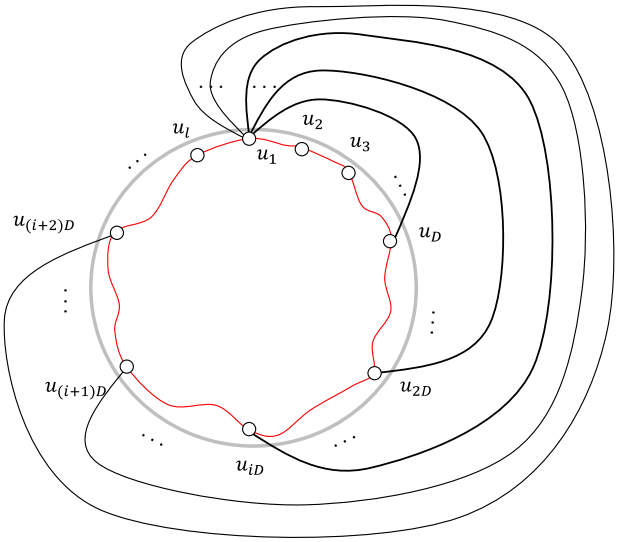


Figure 3.5: Edge augmentation in Step 4

obtain $\mu = 2$. The construction above separates one node in G' into $\Delta(j)$ nodes. That is, the total number of nodes in J is $|V(G)| + \sum_{v_i \in V(T')} \Delta(j) - 1$. It follows that $\lambda \leq 2(|E(G')|)$ holds. Since T' has a depth at most D and has at most $4g$ leaves (because any leaf node must be an endpoint of edges in B), the total number of edges in G' is at most $4gD + 2g$. That is, $\lambda \leq 12gD$.The spanning tree T^{-1} is constructed as follows: we first take the shortest path tree of $C + E''$, and add all the edges $f^{-1}(T)$. Since $f^{-1}(T)$ becomes a forest spanning all the nodes in $V(J) \backslash V(C)$ where each subtree is rooted by a node in C and has a depth at most D, the subgraph T^{-1} above actually spans all the nodes in J. In addition, the shortest path tree of $C + E''$ has a depth at most $D + 1$, the depth of T^{-1} is bounded by $2D + 1$. The spanning tree T^{-1} consists of the edges in $f^{-1}(T) \cup f^{-1}(B) \cup E''$, the number of edges in $E(T^{-1}) \setminus f^{-1}(T)$ is bounded by $|E''| + |f^{-1}(B)| \leq |C|/D + |f^{-1}(B)| \leq 2|E(G')|/D + |f^{-1}(B)| \leq 12g$. The lemma is proved. □

Lemma (Restated Lemma 3.6.9)**.** *There exists a node partition on $\mathcal{G}_G(w, \delta)$ such that the (general) shortcut for this partition either has dilation $\Omega(w\delta)$ or congestion $\Omega(\frac{w\delta}{\log w})$.*

Proof Sketch. The proof is very similar to the pathwidth bounded lower bound. We similarly define the potentials of edges such that the lowest potential of any path between the first and the last node of any lane is $\Omega(w\delta)$. A dilation d and congestion c low-congestion shortcut with respect to every lane either has $\Omega(w\delta)$ dilation or every lane uses up $\Omega(w\delta)$ potential from the tree $\mathcal{T}$, for a total of $\Omega(w^2\delta)$ potential. But the total potential of the tree is $O(w \log w)$, hence in this case the congestion must be $\Omega(\frac{w\delta}{\log w})$. □

Chapter 4

Shortcuts for Minor-Free Graphs

The results of this chapter were published in [68] with Bernhard Haeupler and Jason Li as co-authors. The work was supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.

We note that most of the results of this chapter have been very recently improved by (a preprint published by) Ghaffari and Haeupler [50].

4.1 Introduction

This chapter provides a fast distributed algorithm for such problems in excluded minor graphs in the CONGEST model.

The next major question in algorithmic design is to determine whether one can bypass this barrier by restricting the class of network graphs. One immediate question that arises is, what family of graphs should one consider? Ideally, such a class of graphs should be inclusive enough to admit most "realistic" networks, yet be restrictive enough to disallow the pathological lower bound instances.

In our search for a restricted graph family to study, we focus on three criteria. First, we desire a family with a *rich and rigorous mathematical theory*, so that our result is technically meaningful. Second, the family should capture many networks in practice. And finally, we want *robustness*: a graph with a few added or perturbed edges and vertices should still remain in the family. Robustness is an important goal, since we want our graph family to be resistant to noise. For example, planar graphs satisfy the first two criteria, but fail to be robust since often adding a single random edge will make the graph non-planar. Indeed, most algorithms on planar graphs fail completely when run on a planar graph with a few perturbed edges and vertices. Next, one might try genus-bounded graphs, but they also suffer from similar problems since adding a single randomly connected vertex can arbitrarily increase the genus.

A candidate graph family that fulfills all three conditions is the family of *excluded minor* graphs, namely the graphs which do not have a fixed graph H as a minor. This family encompasses

several classes of naturally occuring networks. For example, trees which exclude K_3, planar graphs that capture the structure of two-dimensional maps exclude K_5 and $K_{3,3}$, and, series-parallel graphs that capture many network backbones exclude K_4 [1, 42]. Excluded minor graphs also have a history of deep results, including the series of Graph Minor papers by Robertson and Seymour.

In this chapter, we provide efficient distributed algorithms for the class of excluded minor graphs which break the $\tilde{\Omega}(\sqrt{n}+D)$ lower bound for general graphs, giving evidence that most practical networks admit efficient distributed algorithms. We show an $\widetilde{O}(D^2)$ algorithm for MST and $(1+\varepsilon)$ approximate min-cut, among other results. For networks having low diameter, such as $D = \text{poly}(\log n)$ or $D = n^{o(1)}$, our algorithms are optimal up to $\text{poly}(\log n)$ or $n^{o(1)}$ factors, respectively. This is a significant improvement over previous MST and min-cut algorithms, which run in $\Omega(\sqrt{n})$ time even on an excluded minor graph with $D = \text{poly}(\log n)$, such as a planar graph with an added vertex attached to every other node.

Our results use the framework of **low-congestion shortcuts**, a powerful combinatorial abstraction to designing distributed algorithms. As a reminder, it introduces a simple, combinatorial problem involving **shortcuts** on a graph, and guarantees that a good-quality solution to this combinatorial problem automatically translates to a simple, efficient distributed algorithm for MST and $(1+\varepsilon)$ approximate min-cut, among other problems; the concepts of shortcuts and quality will be defined later. Actually, the algorithm is the same regardless of the network or the graph family; the purpose of the combinatorial shortcuts problem is to prove that the algorithm runs *efficiently* on the graph or family.

To solve the shortcuts problem on excluded minor graphs, we appeal to the Graph Structure Theorem of Robertson and Seymour [129, 130]. At a high level, the Graph Structure Theorem decomposes every excluded minor graph into a set of almost-planar graphs connected in a tree-like fashion. Our solution to the shortcuts problem is in fact a series of results, one for each step in the structure decomposition. We remark that our result is, to the best of our knowledge, the first in distributed computing to make use of the Graph Structure Theorem to claim a distributed algorithm is fast. The absence of such a preceding result in distributed computing is unsurprising, since algorithms working with the Graph Structure Theorem generally require computing the required decomposition beforehand, and no efficient distributed algorithm to do so is known. Even the best classical algorithm still takes $O(n^3)$ time [88], so even a sublinear distributed algorithm is still out of reach. However, our result is unique in that we merely show the *existence* of a solution to the shortcuts problem in excluded minor graphs, and as a consequence, the simple algorithm of Chapter 2—which does not look at any structure in the network graph, let alone compute a decomposition—is proven to run efficiently on excluded minor graphs.

The fact that this algorithm does not actually compute the Graph Structure Theorem should be stressed further. Since the framework of Chapter 2 computes a shortcut competitive to the optimal one, a consequence is that the running time of this algorithm rarely depends on the (large) constants appearing in the Graph Structure Theorem. In other words, while we can only *prove* that the constants in the running time are bounded by (some functions of) the constants in the Graph Structure Theorem, the actual running time of the algorithm is likely to be much smaller. In fact, for most excluded minor networks, we expect the running time to be $\widetilde{O}(D^2)$ with

a small constant, or even $\widetilde{O}(D)$. In contrast, algorithms that explicitly compute a Graph Structure Theorem decomposition have an inherent bottleneck in the form of the potentially huge constants of the Graph Structure Theorem.

This chapter is structured as follows. After the introduction, we begin with introducing the two main tools necessary for our main result, namely the Graph Structure Theorem and the low-congestion shortcuts framework. Then, we prove the existence of good shortcuts one step at a time, following the step-by-step construction in the Graph Structure Theorem.

4.1.1 Outline of the Proof

The goal of this section is to outline the proof of our main result, without delving into the technical details. Our main technical result is showing the existence of good tree-restricted shortcuts in excluded minor graphs.

Theorem 4.1.1. *[Main Theorem] Every graph in a graph family excluding a fixed minor H admits tree-restricted shortcuts of T-quality $q_T(d) = \tilde{O}(d^2)$. The constants in the big-O depend only on the minor H.*

Using Theorem 4.1.1, we get out main result.

Corollary 4.1.2. *There exists an $\tilde{O}(D^2)$-round distributed algorithm for MST and $(1+\varepsilon)$-approximate min-cut, for any $\varepsilon > 0$, on graph networks excluding a fixed minor.*

For excluded minor graphs of diameter $n^{o(1)}$, as is the case for many practical networks, our algorithms also run in $n^{o(1)}$ time, which is optimal up to lower order terms. This is a significant improvement over the previously known $\tilde{\Omega}(\sqrt{n})$ time algorithms and it avoids the $\tilde{\Omega}(\sqrt{n})$ lower bound for general graphs, even when they have $n^{o(1)}$ diameter.

Corollary 4.1.3. *There exists an $n^{o(1)}$-round distributed algorithm for MST and $(1+\varepsilon)$-approximate min-cut, for any $\varepsilon > 0$, on graph networks with diameter $n^{o(1)}$ and excluding a fixed minor.*

In order to work with excluded minor families, we appeal to the Robertson-Seymour Graph Structure Theorem. At a high level, this theorem states that every graph in an excluded minor family can be decomposed into a set of graph **almost embeddable** in a bounded genus surface that are glued together in a tree-like fashion. Naturally, our approach is to first construct good-quality shortcuts for the entire family of almost embeddable graphs, and then modify them in a robust manner as they are patched together in the composition. While this approach works in general, the patching required is very involved because of various interactions between the many ingredients involved in the decomposition. For example, one step in the construction of an almost embeddable graph is the addition of an "apex" vertex that connects arbitrarily to all previous vertices. While the addition of only one vertex appears harmless at first glance, observe that the diameter can shrink arbitrarily, e.g., to 2 if the apex is connected to all other vertices. If

the graph without the apex has large diameter D, and its shortcuts solution leads to an $\widetilde{O}(D^2)$-round algorithm, this same algorithm will not suffice on the graph with the apex, which can have diameter 2. A lot of technical effort goes into reconstructing shortcuts upon the addition of an apex, in order to handle the arbitrary decrease in graph diameter. Hence, as a consequence of all these difficulties, we settle for $\tilde{O}(d^2)$-quality shortcuts, and leave the improvement to $\tilde{O}(d)$-quality shortcuts as an open problem.

4.1.2 Literature note

The Graph Structure Theorem originates from a series of deep results on graph minor theory by Robertson and Seymour [129, 130]. It provides a structural decomposition to all excluded minor graphs, transforming a negative property—not containing a minor—to a positive and constructive property that is more useful for algorithm design. The original statement of the theorem only states that such a structure exists, but in a later breakthrough, Demaine et al. developed a polynomial-time algorithm to compute the decomposition guaranteed by the theorem[30]. Since then, the graph structure decomposition has found numerous algorithmic applications on excluded minor graphs, such as polynomial-time approximation schemes [30, 59], subexponential algorithms [29], graph coloring [31], and computing separators [1, 59]. Since then, simpler proofs of the Graph Structure Theorem have been discovered [88], as well as more efficient algorithms, with the fastest known one running in time $f(H) \cdot n^3$ for a function f depending only on the excluded minor H [88]. However, we believe this is the first time the theorem is used for an algorithmic result in the distributed setting.

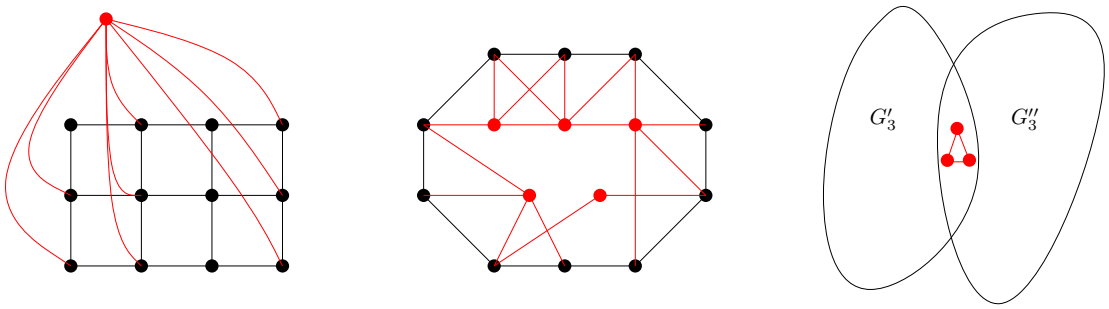
4.1.3 Preliminaries

For a graph G, let $V(G)$ and $E(G)$ denote the vertices and edges, respectively. Given $P \subseteq V(G)$, $G[P]$ denotes the induced subgraph, namely, the one obtained by removing $V(G) \setminus P$ from G. Finally, when G is the underlying network graph, we always assume that G is connected and contains no self-loops (which can be ignored in the distributed setting anyway).

Graph Structure Theorem

In this section we introduce Robertson and Seymour's Graph Structure Theorem, following the survey of Lovász [109]. This theorem is instrumental in our shortcut construction, since it provides structure for all graphs that are H-free, for any minor H. At a high level, the theorem says that every H-free graph can be glued together in a tree-like fashion from graphs that can be "almost" embedded in a fixed surface. To elaborate on this statement, we need a few definitions. The first definition, k-clique-sum, captures the tree-like structure of the graph.

Definition 4.1.4 (k-clique-sum)**.** *Let G_1 and G_2 be two graphs, and let $S_i \subseteq G_i$ be a k-clique for $i = 1, 2$. Let G be obtained by identifying S_1 with S_2 and deleting some (possibly none,*



(a) A planar graph with an added apex.

(b) A cycle with an added vortex of depth 2.

(c) 3-clique-sum between G'_3 and G''_3.

Figure 4.1: Ingredients of the Graph Structure Theorem

*possibly all) edges between the nodes in $S_1 = S_2$. We say that G is a k-**clique-sum** of G_1 and G_2. More generally, G is a k-clique-sum of $G_1, G_2, \ldots, G_\ell$ if G is formed by starting with G_1 and iteratively taking the k-clique-sum of the current graph with G_i, for $i = 2, \ldots, \ell$ in that order.*

The next few definitions classify the graphs which are almost embeddable on a surface. We start with the three main ingredients in constructing such a graph, and then define what it means to be almost-embeddable.

Definition 4.1.5 (Apex)**.** *Define **adding an apex to graph** G as follows: create a new vertex called the **apex**, and connect it to an arbitrary subset of the vertices in G.*

Definition 4.1.6 (Surface of genus g)**.** *A graph G has genus g if there is a **2-cell embedding** in a surface of genus g. In other words, this means: (i) there exists an oriented or unoriented surface (i.e., 2-manifold) Σ of genus g, (ii) vertices of G are mapped to distinct points of Σ, (iii) edges are mapped to simple paths whose interiors do not contain any vertices of G nor do path interiors mutually intersect, and (iv) each face defined by such embedding is homeomorphic to a unit disk, i.e., contains no holes or handles in it.*

Definition 4.1.7 (Vortex [109])**.** *Let G be a graph with a 2-cell embedding. Let C be a cycle in G that corresponds to a face on the surface. Call a continuous interval on the cycle an **arc**. Select a family of arcs on C so that each node is contained in at most k of these arcs. For each arc A, create a new node v_A and connect v_A to a subset of the vertices in C that lie on arc A. Such nodes v_A are called **internal vortex nodes**. Finally, for any two arcs A and B sharing a common vertex in C, we may add the edge $\{v_A, v_B\}$. We call this operation **adding a vortex of depth** k **to cycle** C.*

Definition 4.1.8. *A graph G is (q, g, k, ℓ)-**almost-embeddable** if it can be constructed according to the three steps below.*

(i) Start with a graph G' embedded on a surface of genus at most g.

(ii) We select at most ℓ faces of G' and add a vortex of depth at most k to each of them. Call the result G''.

(iii) We add q apices to G'', connected arbitrarily to vertices in G'' and to each other, and obtain the desired graph G.

*For simpler notation, we say a graph is h-**almost-embeddable** if it is (h, h, h, h)-almost embeddable.*

By this definition, the planar graphs are precisely the $(0, 0, 0, 0)$-almost-embeddable graphs, and the genus-g graphs are precisely the $(0, g, 0, 0)$-almost-embeddable graphs. Later on, we will study the planar graphs with added vortices, in particular the $(0, 0, k, \ell)$-almost-embeddable graphs for constants k and ℓ.

As a final ingredient to the Graph Structure Theorem, we construct a graph family $\mathcal{L}_k$ as follows.

Definition 4.1.9. *Let $\mathcal{L}_k$ denote all graphs that can be represented as a k-clique-sum of k-almost-embeddable graphs. That is, a graph G is in $\mathcal{L}_k$ if there exist k-almost-embeddable graphs $G_1, G_2, \ldots, G_\ell$ such that G is a k-clique-sum of $G_1, G_2, \ldots, G_\ell$.*

In other words, take any set of k-almost-embeddable graphs $G_1, G_2, \ldots, G_\ell$ for $\ell \geq 1$, and let G be their k-clique-sum. Construct a graph G by repeatedly taking a k-clique-sum operation between multiple G_i's constructed using step (i)–(iii) and connect them in a tree-like fashion. Define $\mathcal{L}_k$ as precisely all graphs G that can be constructed in this way.

Finally, we present the Graph Structure Theorem, which states that for any H, there is a k such that $\mathcal{L}_m$ includes (but does not exactly characterize) all graphs that are H-free [109].

Theorem 4.1.10 (Graph Structure Theorem)**.** *For every graph H there is a fixed integer $k = k(H)$ such that any H-free graph G is contained in $\mathcal{L}_k$.*

Below, we include additional terminology on clique-sums and vortices used in the shortcut construction.

Definition 4.1.11 (Vortex terminology)**.** *Let C be a cycle of G, and add a vortex of depth k to C, following Definition 4.1.7. Let $v_1, v_2, \ldots$ be the vertices created when adding a vortex of depth k to C. The vertices in C form the **vortex boundary**, and the added vertices $v_1, v_2, \ldots$ are called **inside the vortex** and **internal vortex** nodes. Moreover, suppose an internal vertex v_i corresponds to **arc** A_i of C in the vortex construction. Define the **vortex decomposition** to be the map $\mathcal{P}$ satisfying $\mathcal{P}(v_i) = A_i$. Finally, if G is embedded on a closed surface such that C forms a face in the embedding, then that face is called the **vortex face**.*

Definition 4.1.12 (k-Clique-sum decomposition tree)**.** *Let the graph G be constructed as the k-clique-sum of subgraphs $B_1, B_2, \ldots, B_\ell$. The subgraphs $B_i \subseteq G$ are denoted as **bags**.*

*A k-**clique-sum decomposition tree** of G is a tree $\mathbb{DT}$ whose vertices $V(\mathbb{DT})$ are identified with bags B_i. The edges of the decomposition $f \in E(\mathbb{DT})$ correspond to a clique in two of the bags, with possibly some removed edges. Therefore, we refer to them as **partial k-cliques** C_f. The decomposition satisfies the following properties:*

1. $\bigcup_{i \in \mathbb{DT}} V(B_i) = V(G)$.
2. *For all $i \in V(\mathbb{DT})$, $B_i \subseteq G$.*
3. *For all $f = \{i, j\} \in E(\mathbb{DT})$, $B_i \cap B_j = C_f$.*
4. *For all $v \in V(G)$, the set $\{i \in V(\mathbb{DT}) \mid v \in V(B_i)\}$ is connected in $\mathbb{DT}$.*
5. *For all $e \in E(G)$, there exists $i \in V(\mathbb{DT})$ with $e \in E(B_i)$.*

We conclude with a statement that the above clique-sum decomposition tree captures all possible ways to take clique-sums of graphs.

Fact 4.1.13. *Let a graph G be the k-clique-sum of graphs from a family $\mathcal{F}$. Then, G has a k-clique-sum decomposition tree whose bags are graphs in $\mathcal{F}$.*

4.2 Shortcuts in Excluded Minor Graphs

Our main result extends tree-restricted shortcut results to excluded minor graphs, showing that any family of graphs excluding a fixed minor has good tree-restricted shortcuts. We repeat Theorem 4.1.1 with a bit of extra detail.

Theorem 4.2.1. *[Main Theorem, Extended Version] The family of graphs excluding a fixed minor H admits tree-restricted shortcuts of T-quality $q_T(d) = \widetilde{O}(d^2)$. More generally, the family admits block parameter $b(d) = O(d)$ and congestion $c(d) = O(d \log n + \log^2 n)$. The constants in the big-O depend only on the minor H.*

Using the shortcuts framework, the above theorem translates to the algorithmic result of Corollary 4.1.2.

4.2.1 Two Parts of the Proof

Recall that the Graph Structure Theorem says that any excluded minor graph can be represented as a k-clique-sum of k-almost-embeddable graphs, for some constant k depending on the excluded minor. As with most results utilizing the Graph Structure Theorem, our proof is split into two parts, one handling the k-clique-sums and one for the k-almost-embeddable graphs.

Our proof has two main components, namely, Theorem 4.2.2 and Theorem 4.2.3 that we state below. It should be clear that they are sufficient to prove the main technical result, Theorem 4.2.1.

Clique Sums Part: In the k-clique-sums part, we show that if a family of graphs admits shortcuts with good T-quality, then so does any k-clique-sum of graphs from this family, for any constant k. In other words, having good tree-restricted shortcuts is a property robust under taking k-clique-sums for a fixed integer k. The theorem below is proved in Section 4.3.

Theorem 4.2.2. *[Shortcuts in Clique Sums] Let $\mathcal{F}$ be a family of graphs that admits tree-restricted shortcuts with block parameter $b_{\mathcal{F}}$ and congestion $c_{\mathcal{F}}$. Let G be a k-clique-sum of graphs in $\mathcal{F}$. Then G admits tree-restricted shortcuts with block parameter $b_G(d) \leq 2k + O(b_{\mathcal{F}}(d_T))$ and congestion $c_G(d) \leq O(k \log^2 n) + c_{\mathcal{F}}(d_T)$.*

On a high level, the proof relies on carefully charging the congestion to bags in the k-clique-sum decomposition. This leads to a bound that relies on the depth of the decomposition, which can be controlled by folding up long bag-paths in the decomposition.

To prove the full result, we use Theorem 4.2.2 with $\mathcal{F}$ as the family of k-almost-embeddable graphs, which we show admits tree-restricted shortcuts with block parameter and congestion $\widetilde{O}(d)$. Plugging in these parameters, we obtain $b_G(d) = 2k + \tilde{O}(d)$ and $c_G(d) = O(k \log^2 n) + \tilde{O}(d)$ for the final result, which are both $\tilde{O}(d)$ since k is a constant. Note that Theorem 4.2.2 does not assume that $\mathcal{F}$ is any particular family, so it may be of independent interest.

Almost Embeddable Part: The second part of the proof establishes good T-quality shortcuts for k-almost-embeddable graphs, namely the theorem below, proved in Section 4.4.

Theorem 4.2.3. *[Shortcuts in Almost Embeddable Graphs] An (q_T, g, k, ℓ)-almost-embeddable graph G admits tree-restricted shortcuts with block parameter $b(d) = O(q_T + (g+1)k\ell^2 d)$ and congestion $c(d) = O(q_T + k\ell^2 d(g + \log n))$.*

The proof is fairly technical and uses several novel ideas. The most prominent one is the construction of a structure we call the "combinatorial gate". Intuitively, when given a partition of a genus-bounded graph into balls of low diameter, there exists a small number of special vertices such that any subgraph that intersects many balls has to contain many of these special vertices. We use these combinatorial gates to set up a careful charging scheme that allows high-diameter parts to be given more shortcut edges without a catastrophic increase in the congestion.

Putting Them Together: The main theorem, restated below, follows immediately from Theorem 4.2.2 and Theorem 4.2.3.

Theorem 4.2.1. *[Main Theorem, Extended Version] The family of graphs excluding a fixed minor H admits tree-restricted shortcuts of T-quality $q_T(d) = \widetilde{O}(d^2)$. More generally, the family admits block parameter $b(d) = O(d)$ and congestion $c(d) = O(d \log n + \log^2 n)$. The constants in the big-O depend only on the minor H.*

Proof. By Theorem 4.1.10, there is a constant k such that the family of H-free graphs is contained in $\mathcal{L}_k$, so it suffices to prove the claim for $\mathcal{L}_k$. Let $\mathcal{F}$ be the family of k-almost-embeddable graphs. By Theorem 4.2.3, $\mathcal{F}$ admits tree-restricted shortcuts with block parameter $b_{\mathcal{F}}(d) = O(d)$ and congestion $c_{\mathcal{F}}(d) = O(d \log n)$. Plugging in $\mathcal{F}$, $b_{\mathcal{F}}$, and $c_{\mathcal{F}}$ into Theorem 4.2.2,

we conclude that $\mathcal{L}_k$ admits tree-restricted shortcuts with block parameter $O(d)$ and congestion $O(d \log n + \log^2 n)$, as desired. □

4.3 Shortcuts in Clique Sum Graphs

In this section, we prove Theorem 4.2.2, restated below.

Theorem 4.2.2. *[Shortcuts in Clique Sums] Let $\mathcal{F}$ be a family of graphs that admits tree-restricted shortcuts with block parameter $b_{\mathcal{F}}$ and congestion $c_{\mathcal{F}}$. Let G be a k-clique-sum of graphs in $\mathcal{F}$. Then G admits tree-restricted shortcuts with block parameter $b_G(d) \leq 2k + O(b_{\mathcal{F}}(d_T))$ and congestion $c_G(d) \leq O(k \log^2 n) + c_{\mathcal{F}}(d_T)$.*

Local and Global Shortcuts: The intuition behind our construction is as follows. Let G be a k-clique-sum of graphs in $\mathcal{F}$, and consider a k-clique-sum decomposition tree $\mathbb{DT}$ of G. Its existence is guaranteed by Fact 4.1.13. Consider a part $P \subseteq V(G)$, which could either span much of a single bag in $\mathbb{DT}$, or traverse through multiple bags, or both. As a result, we construct two types of shortcuts—**local** shortcuts and **global** shortcuts—to handle each case separately. At a high level, local shortcuts, which are constrained within a single bag, are meant to deal with parts that behave wildly within a bag, while global shortcuts, which can span multiple bags, treat parts that stretch across many different bags. In particular, for each part P, we specify one bag on which we construct local shortcuts for P, and let global shortcuts handle the rest. The shortcut for P is simply the union of the local and global shortcuts.

Root $\mathbb{DT}$ at an arbitrary bag, and define $d_{\mathbb{DT}}$ to be the depth of the rooted tree $\mathbb{DT}$. We first prove a weaker result whose global shortcut depends on the value of $d_{\mathbb{DT}}$ in its congestion, then later show how to "compress" $\mathbb{DT}$ to a low depth independent of $d_{\mathbb{DT}}$, thereby removing the dependence of $d_{\mathbb{DT}}$.

Lemma 4.3.1. *Let $\mathcal{F}$ be a family of graphs that admits tree-restricted shortcuts with block parameter $b_{\mathcal{F}}$ and congestion $c_{\mathcal{F}}$. Let G be a k-clique-sum of graphs in $\mathcal{F}$ with decomposition tree $\mathbb{DT}$. Then G admits tree-restricted shortcuts with block parameter $b_G(d_T) \leq k + b_{\mathcal{F}}(d_T)$ and congestion $c_G(d_T) \leq k d_{\mathbb{DT}} + c_{\mathcal{F}}(d_T)$. (Note the dependence on $d_{\mathbb{DT}}$, the depth of the decomposition tree $\mathbb{DT}$, which is unrelated to d_T, the diameter of the spanning tree T.)*

Proof. Let T be an arbitrarily rooted spanning tree of G of diameter d_T. Take a k-clique-sum decomposition tree $\mathbb{DT}$, root it at an arbitrary bag, and suppose that the rooted tree has depth $d_{\mathbb{DT}}$. In the rooted setting, define the set $\text{desc}(i) \subseteq V(\mathbb{DT})$ for $i \in V(\mathbb{DT})$ to be i along with all of its descendants in $\mathbb{DT}$.

Consider a part $P \subseteq V(G)$. Since P is connected, we know, by properties (4) and (5) of Definition 4.1.12, that the set of bags $S_P := \{j \in V(\mathbb{DT}) \mid V(B_j) \cap P \neq \emptyset\}$ is connected in $\mathbb{DT}$. Therefore, the lowest common ancestor, denoted by h_P, of S_P is also inside S_P. Similarly, for an edge $e \in E(G)$ we can define the set of bags that contain that edge $S_e := \{j \in V(\mathbb{DT}) \mid e \in E(B_j)\}$ and its lowest common ancestor $h_e \in S_e$.

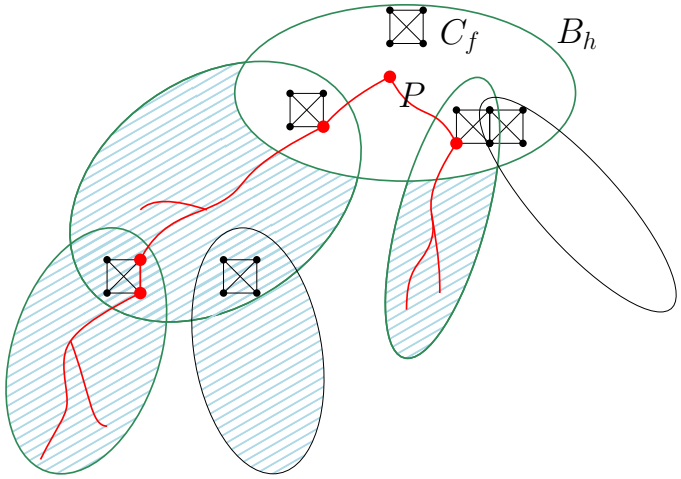


Figure 4.2: Global shortcut construction. The part P is shown in red. The global T-restricted shortcut is the intersection of T (not shown) with the shaded region. C_f denotes the partial clique leading to the parent of h, which is not used in the global shortcut.

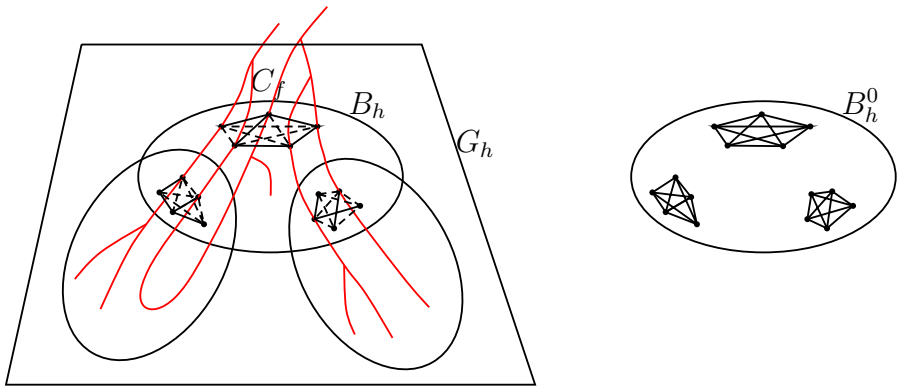


Figure 4.3: Local shortcut construction. On the left, T is solid red. Dotted black edges are edges absent from the partial k-cliques. On the right is B_h^0 for the B_h on the left.

Global Shortcuts: See Figure 4.2. The construction of the global shortcut is simple. For each edge f' to a child i of h_P such that $P \cap V(C_{f'}) \neq \emptyset$, allow part P to use all edges in $\left(\bigcup_{j \in \text{desc}(i)} E(B_j) \cap T \right) \setminus E(B_{h_P})$. Informally, the global shortcut "takes care" of all vertices in P except for those in B_{h_P}, which leaves constructing the local shortcut for P in B_{h_P}. More precisely, remember that T is rooted and consider the roots of the block components of P when using only the global shortcut: they are restricted to B_{h_P}.

We now argue about the congestion. Consider an edge $e \in E(G)$, and let $\mathcal{B}$ be the set of bags on the $\mathbb{DT}$-root-path to h_e, including h_e. Clearly, $|\mathcal{B}| \leq d_{\mathbb{DT}}$. Edge e can only be assigned to parts that contain a vertex in the partial-clique on a parent edge of a bag in $\mathcal{B}$. Hence its congestion is at most $k|\mathcal{B}| \leq kd_{\mathbb{DT}}$.

Local Shortcuts: See Figure 4.3. Let h be an arbitrary bag, we apply the following argument to all of them. We now focus on the local shortcut within B_h. Let $T_h^1 := T \cap B_h$ be the forest when we look at B_h in isolation (note that the tree T can become disconnected). We will repair T_h^1 in the next paragraph.

Let $B_h^0 \in \mathcal{F}$ be the original bag of B_h, which is B_h with all partial k-cliques involved in the clique-sum completed to full k-cliques (see Figure 4.3). In particular, $V(B_h) = V(B_h^0)$. We emphasize that $B_h^0 \in \mathcal{F}$ by the definition of partial-cliques.

In order to find a tree-restricted shortcut on B_h, we have to define the tree. The forest $T_h^1 := T \cap B_h^0$ might be disconnected, so we have to repair it. First, we define a **path contraction** operation between two vertices $s, t \in V(B_h^0)$. Consider the unique path between s and t in T, represented as a sequence of vertices $s = u_0, u_1, \ldots, u_* = t$. Delete any vertex $u_i \notin V(B_h)$ and one is left with (a sequence of vertices representing a) valid path in B_h^0 between s and t. Note that the contracted path is a graph minor of T.

We form the repaired tree T_h^2 in the following way: for every two $s, t \in V(B_h^0)$, take the path contraction between them and union it into T_h^2. It is clear that (1) T_h^2 is a subgraph of B_h^0, in fact, it is a spanning tree of B_h^0, (2) T_h^1 is a subgraph of T_h^2, and (3) T_h^2 is a contraction of T. The last property implies that T_h^2 is connected and that its diameter is at most d_T. Also, note that the same argument shows that for any part P, its restriction $B_h^0[P]$ is also connected since we can contract any path inside P and the resulting path is still in B_h^0 and contains only vertices in P—the only unimportant difference being that this path might be on T.

Next, construct a T_h^2-restricted shortcut, discard all edges in $T_h^2 \setminus T = T_h^2 \setminus T_h^1$, and discard all edges contained in C_f, where f is the parent $\mathbb{DT}$-edge of h. The resulting assignment is the local shortcut of B_h.

The congestion of the local shortcut is $c_{\mathcal{F}}(d_T)$. Fix an edge $e \in E(G)$, and note that it is only locally assigned in the bag h_e (due to discarding edge of C_f). But the local congestion of h_e is $c_{\mathcal{F}}(d_T)$, as claimed. The total congestion is at most the sum of the local and global one, hence it is at most $kd_{\mathbb{DT}} + c_{\mathcal{F}}(d_T)$.

Bounding the Block Parameter: With all shortcut edges established, we now upper bound the block parameter for each part $P \subseteq V(G)$. Remember that T is, arbitrarily, rooted. We will bound the number of nodes $v \in V(G)$ that are roots of block components. Note that $v \in B_{h_P}$ since otherwise the global shortcut assigns the T-parent edge of v to P. But in the lowest common ancestor B_{h_P}, v can be a block root only if either (a) it is a vertex in C_f, where f is the parent $\mathbb{DT}$-edge of h_P, or (b) it is a block root of a local shortcut inside B_{h_P}. Summing up the contributions of these two cases, the total number of block roots, and therefore block components, can be at most $k + b_{\mathcal{F}}(d_T)$.

□

To improve the $d_{\mathbb{DT}}$ factor in the congestion and prove the main result of this section, we compress the decomposition tree $\mathbb{DT}$ to reduce its depth to $O(\log^2 n)$, in a similar way to the compression scheme in [16] for treewidth decompositions.

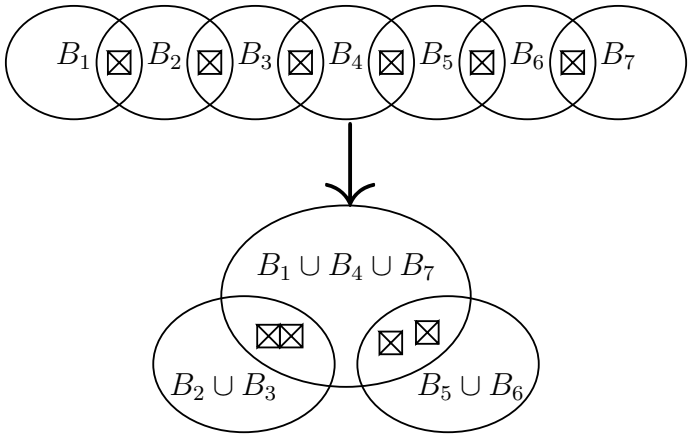


Figure 4.4: Compressing a k-clique-sum decomposition tree with high depth.

Theorem 4.2.2. *[Shortcuts in Clique Sums] Let $\mathcal{F}$ be a family of graphs that admits tree-restricted shortcuts with block parameter $b_{\mathcal{F}}$ and congestion $c_{\mathcal{F}}$. Let G be a k-clique-sum of graphs in $\mathcal{F}$. Then G admits tree-restricted shortcuts with block parameter $b_G(d) \leq 2k + O(b_{\mathcal{F}}(d_T))$ and congestion $c_G(d) \leq O(k \log^2 n) + c_{\mathcal{F}}(d_T)$.*

Proof. Let $\mathbb{DT}$ be a k-clique-sum decomposition tree of G. To motivate the main proof, we first consider the case when $\mathbb{DT}$ is a single path from root to leaf. This case will directly help in the general case, in which we apply *heavy-light decomposition* to the tree, breaking it up into chains, and then treat each chain as a single path; we will present this general case next.

Case When $\mathbb{DT}$ Is a Path: Assume that $\mathbb{DT}$ is a rooted path with bags $B_1, \ldots, B_{d_{\mathbb{DT}}}$, in that order. We recursively construct a balanced binary decomposition tree $\mathbb{DT}'$ as follows.

1. Group the bags $B_1, B_{\lceil d_{\mathbb{DT}}/2 \rceil}, B_{d_{\mathbb{DT}}}$ into a single bag B_r.
2. Recursively solve the paths $B_2, \ldots, B_{\lceil d_{\mathbb{DT}}/2 \rceil - 1}$ and $B_{\lceil d_{\mathbb{DT}}/2 \rceil + 1}, \ldots, B_{d_{\mathbb{DT}}}$.
3. Attach the two resulting trees as subtrees of B_r (see Figure 4.4).

We call this operation **folding** a path.

Call the new decomposition tree $\mathbb{DT}'$; it is almost a k-clique-sum decomposition tree, with one exception: an edge may no longer a partial k-clique, but a union of two partial k-cliques. We call such edges **double edges**. Note that, while we can add edges within each of the two partial k-cliques and keep the graph in the family $\mathcal{F}$, we cannot add edges between a vertex in one partial k-clique and a vertex in the other. Hence, we cannot simply treat the union of two partial k-cliques as a single partial $2k$-clique. However, a bag B_i can have at most two children connected by double edges.

Using the terminology of the above proof, let B_h^0 still be the bag B_h with all partial cliques filled in with edges (the union of two cliques in a double edge will not have edges between them). The only difference this incurs in the proof is the following: in the global shortcut, partial cliques on

the edge of $\mathbb{DT}$ can now contain $2k$ vertices instead of k, doubling the congestion; and, in the local shortcut, a part restricted to to a bag $B_h^0[P]$ might not be connected anymore. However, we claim that it consists of at most $O(1)$ connected components: for each connected component we find a "representative vertex" in that component as follows. If (1) the component touches a partial clique in a double edge to a child, then the representative is the lowest numbered vertex in such a partial clique, and otherwise (2) we pick any vertex in the component. Now there will be at most $O(1)$ different representatives, thereby finishing the claim since no two different components can have the same representative. One can see this by arguing if (1) a part touches a partial clique in a double edge to a child, the it has at most 4 possibilities; otherwise (2) the part is already connected via the previous proof.

We construct local shortcuts considering connected components of the parts as separate (sub)parts and union the assignment in the end. This only decreases the congestion, and increases the block by a multiplicative $O(1)$ to a total of $2k + O(b_{\mathcal{F}}(d_T))$.

We now discuss the general case, when $\mathbb{DT}$ is an arbitrary tree. The main steps of the proof are as follows. First, we compute a *heavy-light decomposition* [73] of $\mathbb{DT}$. Then, we fold every chain in the heavy-light decomposition the same way we fold a single path, so that the resulting tree decomposition has depth $O(\log^2 n)$.

Heavy-Light Decomposition: The heavy-light decomposition is a decomposition of any rooted tree into vertex-disjoint paths, called **heavy chains**, such that any path from the root to a leaf changes at most $O(\log n)$ heavy chains, where n is the number of vertices in the tree. The decomposition is simple: for each non-leaf vertex of the tree, connect it to the child vertex with the largest number of vertices in its own subtree. On any path from root to leaf, if traveling from vertex u to vertex v changes heavy chains, then vertex u has at least twice as many vertices in its subtree than does v; such an event can only occur $\log_2 n$ times along the path.

Folding a Chain: Once we compute the heavy-light decomposition, we partition the vertices of $\mathbb{DT}$ into heavy chains, and then fold each chain independently. Then, we connect the resulting binary trees in the following natural way: if the root of chain C_1 is a child of some vertex v, then we connect the root of the binary tree of C_1 to v. Note that this is not a double edge. We get a rooted tree $\mathbb{DT}'$ of depth $O(\log^2 n)$ with the following key property: while every vertex in the new decomposition tree can have many children, it has at most two children connected via double edges. Therefore, the same argument for double edges in the single path case also applies here. With the depth of $\mathbb{DT}'$ reduced to $O(\log^2 n)$, the result follows. □

4.4 Shortcuts in Almost Embeddable Graphs

In this section, we prove Theorem 4.2.3. In particular, we prove that k-almost-embeddable graphs admit good shortcuts. Recall that these graphs have bounded genus with an additional constant number of apices and vortices of constant depth added.

4.4.1 Warm-up: Non-Apex Graphs

As a warm-up, we disregard apices and only consider graphs of bounded genus with vortices, i.e., the "**Genus+Vortex**" graphs. We establish tree-restricted shortcuts with block parameter $O((g+1)kD)$ and congestion $O((g+1)kD \log n)$ for graphs of genus g with a k-vortex included. We first show that such a graph must have treewidth at most $O((g+1)kD)$, and then use the treewidth-based shortcut construction of [66]. We note that this lemma is not novel, it is a simple consequence of the work by Dujmovic, Morin and Wood [32], but we chose to include it because it illustrates how to deal with vortices.

At this point, we introduce our notation for treewidth decompositions. A *treewidth decomposition* of a graph G is a tree $\mathbb{DT}$ whose vertices, called *bags*, are subsets of $V(G)$. The tree $\mathbb{DT}$ satisfies three properties: (i) the union of vertices over all bags equals $V(G)$; (ii) for each $v \in V$, the set of bags containing v is connected in $\mathbb{DT}$; (iii) for each edge $(u, v) \in E(G)$, there is a bag containing both u and v. The *treewidth* of a graph G is the minimum k such that there exists a tree decomposition $\mathbb{DT}$ of G whose bag sizes are all at most $k+1$.

Lemma 4.4.1. *A graph G of diameter D and genus g with a single vortex of depth k has treewidth $O((g+1)kD)$.*

Proof. First, we transform G into a graph G' of genus g and diameter at most $D+1$ as follows: remove all the vertices inside the vortex, and add a single vertex r in the vortex face with an edge to all vertices on the vortex boundary. Since pairwise distances between vertices on the boundary do not increase by more than 1, the diameter of G' is at most $D+1$.

Eppstein [38] proves that graphs of genus g have treewidth $O((g+1)D)$. Therefore, there exists a tree decomposition $\mathbb{DT}'$ of G' with bag size $O((g+1)D)$. Remove r from $\mathbb{DT}'$. To add the vortex back in, first take a vortex decomposition $\mathcal{P}$. Then, for each vertex v inside the vortex that was removed, add v to every bag in $\mathbb{DT}'$ that intersects $\mathcal{P}(v)$, i.e., contains a boundary vertex on the corresponding arc of v. It remains to prove that the resulting tree decomposition $\mathbb{DT}$ is valid and has bag size $O((g+1)kD)$.

To show the former, fix a vertex v inside the vortex. Since the neighboring boundary vertices in $\mathcal{P}(v)$ are connected by edges, there exists a common bag between every two neighboring boundary vertices. Therefore, the entire set of bags containing v is connected. In addition, v shares a common bag with any boundary vertex in $\mathcal{P}(v)$, as well as any other vertex v' in the vortex with $\mathcal{P}(v) \cap \mathcal{P}(v') \neq \emptyset$. It follows that for each edge incident to v, there exists a bag containing both of its endpoints.

Finally, since the vortex decomposition $\mathcal{P}$ has depth at most k, each vertex on the boundary is responsible for at most k new vertices in its bags. Since each bag has at most $O((g+1)D)$ boundary vertices, the new bag size is $O((g+1)kD)$. □

This proof easily generalizes to the case when G has ℓ vortices, each of depth k.

Lemma 4.4.2. *A graph G of diameter D and genus g with ℓ vortices of depth k has treewidth $O((g+1)k\ell D)$.*

Finally, applying the treewidth-based shortcut construction gives the desired result.

Theorem 4.4.3. *A genus g and diameter D graph with ℓ vortices of depth k has tree-restricted shortcuts with congestion $O((g+1)k\ell D \log n)$ and block parameter $O((g+1)k\ell D)$.*

In particular, since planar graphs have genus 0, we get the following corollary:

Corollary 4.4.4. *A diameter D planar graph with ℓ vortices of depth k has tree-restricted shortcuts with congestion $O(k\ell D \log n)$ and block parameter $O(k\ell D)$.*

4.4.2 Apex Graphs

In this section, we add apices to $(0, g, k, l)$-almost-embeddable ("Genus+Vortex") graphs. At first glance, the addition of an apex to a graph might seem trivial, since the graph only changes by one vertex, and using that vertex can only make the shortcuts better. However, notice that the diameter of the graph can shrink arbitrarily with the addition of an apex, and our shortcuts on the apex graph must be competitive with the new diameter. Hence, we need ideas beyond our shortcut constructions for the graph without the apex. For a simple example, in a cycle graph, shortcuts with T-quality $\Theta(n)$ are considered good. However, by adding a single central vertex, we can transform the graph into the wheel graph where "good" shortcuts should have T-quality $\Theta(1)$. While good shortcuts actually do exist in the wheel graph, there are examples of graphs with good shortcuts where adding a apex makes good shortcuts impossible.

To streamline our arguments for (q_T, g, k, l)-almost-embeddable graphs ("Apex+Genus+Vortex") graphs, we will define a couple of intermediate properties which do not depend on the graph topology. More precisely, we will define the notions of β**-cell-assignment** and s**-combinatorial gates**. On a very high level, We will show that:

1. A Genus+Vortex graph has an s-combinatorial gate, for an appropriately chosen s. (Section 4.4.4 and the Appendix)
2. Graphs with s-combinatorial gate are β-cell-assignable, for appropriately chosen β and some technical stipulations. (Section 4.4.3)
3. Graphs that are β-cell-assignable and each cell locally admits good tree-restricted shortcuts also globally admit good tree-restricted shortcuts, barring various technicalities. (Section 4.4.5)

In each part, we separately prove the statements with Genus+Vortex graphs replaced by planar graphs. It is recommended that the reader, in their first reading, focus only on the lemmas regarding planar graphs with a single apex, namely Lemmas 4.4.9, 4.4.10, 4.4.12, and 4.4.14.

4.4.3 Cell Partitions, β-Cell-Assignment and s-Combinatorial Gate

In this section, we first introduce the notions of "cell partitions", "β-cell-assignment" and "s-combinatorial gates". Second, we prove that the second property implies the first.

Definition 4.4.5. *A **cell partition** of G is simply a partition of V_G into disjoint, connected components with a small diameter, called the **cells**.*

Note that the diameter condition is the only thing differentiating it from the definition of parts. It is helpful to think of cells as low-diameter components, whereas parts may be long and skinny. A canonical example for a cell partition is the following. Given an apex graph of diameter D, remove the apex and start a concurrent BFS from each node adjacent to the removed apex. Each node in the graph (except the apex) gets assigned to exactly one BFS component. We call such BFS components cells. For most of this section, we will ignore any extra property that a cell partition might have and assuming nothing besides them being disjoint, connected and having a controlled diameter.

A graph is cell-assignable if we can relate its cells and parts in a way that no cell is assigned to too many parts and parts are assigned to **almost all** intersecting cells.

Definition 4.4.6. *A graph $G = (V_G, E_G)$ is β-cell-assignable if the following holds. For every valid family of parts $\mathcal{P}$ (as in the part-wise aggregation Definition 1.2.1) and every valid cell partition $\mathcal{C}$ of diameter d there exists a relation $\mathcal{R} \subseteq \mathcal{C} \times \mathcal{P}$ with the following properties:*

*(i) each part is in relation with **all** cells it intersects, **except** for at most 2 of them*

(ii) each cell is in relation with at most β parts

Note: β is a function of the cell diameter d.

We will not prove directly that Genus+Vortex graphs are β-cell-assignable. Instead, we focus on a combinatorial property that we show implies cell-assignment. This property is called a "combinatorial gate" and it intuitively asserts that every two touching cells have a "gate" that covers all the edges between them. Furthermore, the boundary of such a gate is called a "fence" and its size should be controlled. The reader is encouraged to review Figure 4.5 for a mental picture of combinatorial gates on a planar graph.

Definition 4.4.7. *For a subset of vertices $S \subseteq V$, define the ∂S to be the set of vertices in S on the boundary of S, i.e., the vertices in S whose neighborhoods intersect $V \setminus S$.*

Definition 4.4.8. *Let $G = (V, E)$ be a graph from a family $\mathcal{F}$, and let $\mathcal{C}$ be a partition of G into cells. We define a s-**combinatorial gate** to be a collection $\mathcal{S} = \{(F_i, S_i)\}_i$ where $F \subseteq V$ are called **fences**, $S \subseteq V$ are **gates**, and the following properties hold:*

1. *Fences are a subset of their corresponding gates. I.e., $F \subseteq S$ for all $(F, S) \in \mathcal{S}$.*

2. *The boundary of a gate are included in its fence. I.e., $\partial(S) \subseteq F$ for all $(F, S) \in \mathcal{S}$.*
3. *Each edge $\{a, b\} \in E$ whose endpoints are in different cells must be covered by some gate. I.e., $a \in S \wedge b \in S$ for some gate S.*
4. *Each gate S intersects at most two cells in $\mathcal{C}$.*
5. *The non-fence vertices of the gates are disjoint. I.e., for every $v \in V$ there is at most one $(F_i, S_i) \in \mathcal{S}$ s.t. $v \in S_i \setminus F_i$.*
6. *The average size of fences compared to the number of cells is at most s. I.e., $\sum_{(F,S)\in\mathcal{S}} |F| \leq s|\mathcal{C}|$.*

Since this condition is entirely combinatorial, the proofs that imply β-cell-assignment are also combinatorial. Therefore, these results are self-contained and disregard any possible structure in the graph, for example, planarity. Next, we prove that the s-combinatorial boundary implies β-cell-assignment via the following two lemmas.

Lemma 4.4.9. *Suppose a graph G with cell partition $\mathcal{C}$ has an s-combinatorial gate $\mathcal{S}$. Then, for any collection of parts $\mathcal{P}$, either there exists a part intersecting at most two cells, or there exists a cell intersecting at most $2s$ parts.*

Proof. Let $\mathcal{I} \subseteq \mathcal{P} \times \mathcal{C}$ be the set of pairs (P, C) s.t. $P \cap C \neq \emptyset$. Let the "degree of a part P" be the cardinality $deg(P) := |\{C \mid (P, C) \in \mathcal{I}\}|$, and similarly define the "degree of a cell C" $deg(C)$.

We are done if there is a part with degree at most 2, hence we can assume $deg(P) \geq 3, \forall P \in \mathcal{C}$. Fix a part P and define $B_P = \{i \mid P \cap F_i \neq \emptyset, F_i \text{ is a fence}\}$ be the indices of fences it intersects. Then $\sum_{P\in\mathcal{P}} |B_P| \leq \sum_i |F_i|$ since every fence vertex can be contained in at most 1 part.

Furthermore, fix a part P; we claim that $deg(P) \leq |B_P| + 1 \leq 2|B_P|$. This paragraph proves the first inequality, the second being trivial. Since P intersects $deg(P)$ many different cells, there must be $deg(P) - 1$ edges whose endpoints are in different cells and are both in P; even more, each of these edges connects a different (unordered) pair of cells. Property (3) of the combinatorial gate definition implies that all of these edges must be inside some gate S_i. However, it is impossible that $P \subseteq S_i$, otherwise property (4) would imply $deg(P) \leq 2$. Therefore, P must contain a vertex $\partial(S_i)$, which is also included in the fence $\partial(S_i) \subseteq F_i$ by property (2). We conclude that $i \in B_P$. Moreover, the i's corresponding to different edges are distinct since the unordered pair of cells they are connecting is different.

We are now ready to prove the Lemma via the following claim: $|\mathcal{I}| = \sum_{P\in\mathcal{P}} \leq 2\sum_{P\in\mathcal{P}} |B_P| \leq 2s|\mathcal{C}|$. Hence $\frac{|\mathcal{I}|}{|\mathcal{C}|} \leq 2s$, implying there exists a cell with degree at most $2s$ by the pigeonhole principle. □

Lemma 4.4.10. *Let $\mathcal{F}$ be a family of graphs that is closed under taking minors. Suppose that there is a function $s(d) : \mathbb{N} \to \mathbb{N}$ such that every graph $G \in \mathcal{F}$ satisfies the following property:*

- *If G has a cell partition of diameter d, then there exists an $s(d)$-combinatorial gate $\mathcal{S}$ of subsets of $V(G)$.*

Then, every graph $G \in \mathcal{F}$ with a cell partition of diameter d is $2s(d)$-cell-assignable.

Proof. Fix a graph $G \in \mathcal{F}$, cells $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ of diameter d, and parts $\mathcal{P} = \{P_1, \ldots, P_{|\mathcal{P}|}\}$. We construct an assignment $\mathcal{R}$ following Definition 4.4.6. Assume that G is connected; otherwise, we can repeat the argument below on each connected component of G. We proceed by induction on $|\mathcal{C}| + |\mathcal{P}|$, with the base case $|\mathcal{C}| = 1$ or $|\mathcal{P}| = 1$ being trivial.

Suppose that $|\mathcal{C}| > 1$ and $|\mathcal{P}| > 1$. By Lemma 4.4.9, either there is a part $P \in \mathcal{P}$ intersecting at most two cells, or there exists a cell intersecting at most $2s(d)$ parts. In the former case, we do not assign any cell to P in $\mathcal{R}$ and proceed by induction on the instance $(G, \mathcal{C}, \mathcal{P} \setminus P)$.

In the latter case, we find a cell $C \in \mathcal{C}$ intersecting at most $2s(d)$ parts and $\mathcal{R}$-assign C to all parts it intersects. Then, iteratively remove C from the graph by contractions. Repeatedly pick any remaining $v \in C$. If v belongs to some part $P \in \mathcal{P}$ and has a neighbor in P, then contract v along any incident edge that has both of its endpoints in P. Note that, by the connectedness of P, $v \in P$ must have a neighbor in P unless $P = \{v\}$. Otherwise, contract a vertex v it along any incident edge.

Let G' be the resulting graph, let $\mathcal{P}' = \{P \setminus C \mid P \in \mathcal{P}\}$ be the new partition, and $\mathcal{C}' = \mathcal{C} \setminus C$ be the remaining cells. Note that, by our edge contraction scheme, all parts in $\mathcal{P}'$ remain connected in G', all remaining cells remain connected, and incidences between the remaining cells and parts are unchanged. In addition, since the graph family $\mathcal{F}$ is closed under edge contraction, $G' \in \mathcal{F}$. We apply induction on the instance $(G', \mathcal{C}', \mathcal{P}')$ and union the resulting relation $\mathcal{R}'$ with the assignments made in the current iteration. □

While Lemma 4.4.10 works well for planar graphs that are closed under taking minors, Genus+Vortex graphs do not have that property due to the existence of a bounded number of vortices. In particular, if one contracts an edge inside the vortex, the resulting graph is not Genus+Vortex. Therefore, we will deal with cells touching vortices as "special cells" that are not allowed to be contracted.

Lemma 4.4.11. *Let $\mathcal{F}$ be a family of graphs, not necessarily closed under taking minors. Suppose that there is a function $s(d) : \mathbb{N} \to \mathbb{N}$ such that every graph $G \in \mathcal{F}$ satisfies the following property:*

- *If G has a cell partition of diameter d, then there exists an $s(d)$-combinatorial gate $\mathcal{S}$ of subsets of $V(G)$.*

Consider a graph $G \in \mathcal{F}$ with a cell partition into two types of cells—normal cells and ℓ special cells—both of diameter d. Let E^* denote the set of edges in special cells. Assume that any graph G' obtained by deleting vertices and contracting edges outside of special cells is still in $\mathcal{F}$. Then, G is $2\ell s(d)$-cell-assignable with respect to a cell partition of only the normal cells.

Proof. Fix a graph $G \in \mathcal{F}$, normal cells $\mathcal{C}^0$, special cells $\mathcal{C}^*$, and parts $\mathcal{P} = \{P_1, \ldots, P_{|\mathcal{P}|}\}$. Similarly to Lemma 4.4.10. we proceed by induction on $|\mathcal{C}^0| + |\mathcal{P}|$, with the same base case

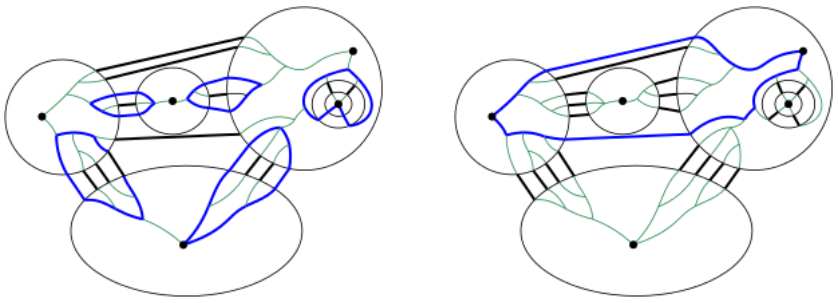


Figure 4.5: Graphical overview of our boundary construction. The black circles are the cells. Note that there is a cell completely contained inside another cell in the planar embedding. The green edges are the spanning trees T_i as defined in the proof. The blue edges form our boundary construction.

being trivial.

Suppose that $|\mathcal{C}^0| > 1$ and $|\mathcal{P}| > 1$. If there is a part $P \in \mathcal{P}$ intersecting at most two cells, then we proceed as in Lemma 4.4.10. Otherwise, as in the proof of Lemma 4.4.9, we show that the total number of pairs (P, C) where a part $P \in \mathcal{P}$ intersects a cell $C \in \mathcal{C} \cup \mathcal{C}^*$ is at most $2s(d)|\mathcal{C}^0 \cup \mathcal{C}^*|$. Since $|\mathcal{C}^0| \geq |\mathcal{C}^0 \cup \mathcal{C}^*| - \ell \geq |\mathcal{C}^0 \cup \mathcal{C}^*|/\ell$, the number of (P, C) is at most $2\ell s(d)|\mathcal{C}^0|$, so there exists a normal cell intersecting at most $2\ell s(d)$ parts. The rest of the proof is identical to that in Lemma 4.4.10, except we note that since we only remove vertices in a normal cell, the new graph G' still satisfies the conditions in the lemma. □

4.4.4 Graphs with s-Combinatorial Gate Property

In this section, we show that Genus+Vortex graphs satisfy the s-combinatorial property. We highlight our main ideas by proving the statement for planar graphs before moving on to genus-bounded graphs (Section 4.6).

Lemma 4.4.12. *Let G be a planar graph with a cell partition of diameter d. Then, there is an $36d$-combinatorial gate $\mathcal{S}$.*

Proof. Fix a planar embedding of G in the planar region $\mathbb{R}^2$. Define an auxiliary graph A formed by contracting each cell into a single vertex, then removing parallel edges. In other words, two vertices in A are adjacent iff their corresponding cells are connected by an edge; we call two such cells "adjacent". Our goal is to, for each pair of adjacent cells, define a closed loop that separates the planar embedding in a laminar way[1]. Figure 4.5 gives a graphical overview of our boundary construction, which we make more precise below. We note for later that the planarity of A implies $|E(A)| \leq 3|V(A)| - 6 = 3|\mathcal{C}| - 6$.

For each cell C_i, define T_i to be a spanning tree of C_i with diameter at most d. For any two cells $C_i, C_j \in \mathcal{C}$, define the set of "(C_i, C_j)-inter-cell edges" as those in F that connect the two cells.

[1]A family of sets is laminar when any two members are either disjoint or one is a subset of another.

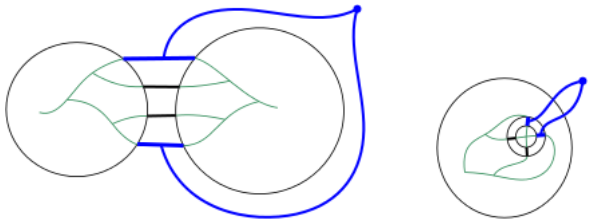


Figure 4.6: Definition of extremal edges between two different cells. T_i and T_j are shown in green. The extremal edges are the blue edges reachable from the outer face, as indicated by the paths in blue, while the other inter-cell edges are the black edges.

Given two (C_i, C_j)-inter-cell edges $e_u := (u_i, u_j)$ and $e_v := (v_i, v_j)$, define the "cycle along e_u and e_v" to be the union of edge e_u, edge e_v, the path along T_i from u_i to v_i, and the path along T_j from u_j to v_j. We denote the cycle by $cyc(e_u, e_v)$. Note that every such cycle has at most $4d + 2$ vertices. Our next step is to find two special (C_i, C_j)-inter-cell edges e_L and e_R such that if we consider the cycle along e_L and e_R, the boundary and interior of this loop, i.e., the set of points in the plane enclosed by the cycle, contain all (C_i, C_j)-inter-cell edges. We call e_L and e_R the "extremal edges" between C_i and C_j.

Intuitively, we choose the extremal edges to be the "left-most" and "right-most" (C_i, C_j)-inter-cell edges, but these can be formally defined as follows (see Figure 4.6). Define a planar graph T_{ij} to be the union of T_i, T_j, and all (C_i, C_j)-inter-cell edges. Note that $T_{ij} \subseteq G$ inherits the planar embedding from G. Draw a loop starting and ending in the outside face that encloses T_i but lies outside of T_j; this can always be done because the trees are disjoint. The first and last (C_i, C_j)-inter-cell edge intersected by the loop must lie on the outside face and form our extremal edges e_L, e_R[2].

Take all pairs of adjacent cells (C_i, C_j) and consider the cycle $K_{ij} \subseteq G$ along their extremal edges $cyc(e_L, e_R)$. Let $\mathcal{K}$ be the set of all such cycles.

For a cycle $K \in \mathcal{K}$ define the set of points in the embedding enclosed within the cycle as $reg(K)$. Note that $reg(K)$ is closed set. An important property of $\{reg(K) \mid K \in \mathcal{K}\}$ is that two $reg(K), reg(K')$ are either disjoint or one is a subset of another, i.e., they form a laminar family and the notions of minimal cycle and maximal cycle are well-defined inside the family.

We are ready to construct the combinatorial gate. Given a cycle $K_{ij} \in \mathcal{K}$, let $own(K) := reg(K) \setminus \left(\cup_{reg(K') \subseteq reg(K)} int(reg(K'))\right)$, where $int(\cdot)$ is the topological interior. In other words, $reg(K)$ is the set of points in the embedding that are enclosed in the cycle, but are outside of the strict interiors of any other cycles $K' \in \mathcal{K}$ enclosed in K. Let $S_{ij} \subseteq V(G)$ be the set of vertices $v \in C_i \cup C_j$ where the corresponding point in the embedding $p_v \in \mathbb{R}^2$ is in $own(K)$. Similarly, let $F_{ij} \subseteq V(G)$ be the set of vertices $v \in C_i \cup C_j$ which are not in the topological interior of $own(K_{ij})$ (again, in terms of the embedding). The combinatorial gate is then defined as $\mathcal{S} := \{(F_{ij}, S_{ij}) \mid K_{ij} \in \mathcal{K}\}$.

Property (1), i.e., $F_{ij} \subseteq S_{ij}$, and Property (4), i.e., $S_{ij} \subseteq C_i \cap C_j$, trivially follow from the definition. Property (5): the laminarity of $\{reg(K) \mid K \in \mathcal{K}\}$ implies that $\{own(K) \mid K \in \mathcal{K}\}$

[2]If there is only one (C_i, C_j)-inter-cell edge, then we set both e_L and e_R to that edge.

can only share a boundary, hence their interiors are disjoint; therefore, non-fence vertices have at most one $own(K)$ region they are contained in, implying the Property.

Property (3): fix any (C_i, C_j)-inter-cell edge e. By construction, $p_e \subseteq reg(K_{ij})$, where $p_e \subseteq \mathbb{R}^2$ is the set of points the edge corresponds to in the embedding. We also claim that $p_e \subseteq own(K_{ij})$, which would imply the Property. Assume this is not true, then $p_e \subseteq reg(K_{i'j'})$ for some $K_{i'j'} \neq K_{ij}$ such that $reg(K_{i'j'}) \subseteq reg(K_{ij})$. We can assume without loss of generality that $i \notin \{i', j'\}$. By planarity of the graph and the connectedness of C_i, the points in the embedding corresponding with C_i would have to lie inside of $reg(K_{i'j'})$ since no edge of C_i can cross $K_{i'j'} \subseteq C_{i'} \cup C_{j'}$. But that contradicts the assumption that $reg(K_{i'j'}) \subseteq reg(K_{ij})$, implying the Property.

Finally, we prove Property (6) with the parameter $s := 36|\mathcal{C}|d$. Every fence vertex $v \in F_{ij}$ must either lie on K_{ij} or on a *maximal* cycle nested within K_{ij}. If $v \in K_{ij}$, we will charge it to K_{ij}; otherwise, if v is on a maximal nested cycle $K' \in \mathcal{K}$ inside of K_{ij}, we charge it to K'. Note that for any $K \in \mathcal{K}$, only vertices on K and the the unique enclosing cycle (if one exists) can charge to K. Therefore, the total number of vertices charged to any $K \in K$ is at most $2 \cdot \max_{K' \in \mathcal{K}} |K'| \leq 2 \cdot (4d + 2) \leq 12d$, from before. The number of incident cells, $|\mathcal{K}|$, is equal to $|E(A)| \leq 3|\mathcal{C}| - 6 \leq 3|\mathcal{C}|$; leading to $\sum_{(F,S)\in\mathcal{S}} |F| \leq 3|\mathcal{C}| \cdot 12d \leq 36|\mathcal{C}|$.

□

The full result is rather technical and tangential; hence is proved in Section 4.6.

Lemma 4.4.13. *Let G be a genus-g graph with (a possibly unbounded number of) vortices of depth k, and consider a cell partition of diameter d such that no vortex is split between more than one cell. Then, there exists an $O((g+1)kd)$-combinatorial gate of G.*

4.4.5 Wrapping Up: From β-Cell-Assignment to Good Shortcuts

In this section, we finalize our proof for tree-restricted shortcuts in almost embeddable graphs. We do this by showing that if an $(0, g, k, l)$-almost-embeddable ("Genus+Vortex") graph is β-cell-assignable for small enough parameter β, then the same graph with q_T added apices admits good tree-restricted shortcuts. We first assume that the apex graph has exactly one apex, then establish a simple reduction from the multiple apices case. We begin with the same statement for $(1, 0, 0, 0)$-almost-embeddable ("Apex+Planar") graphs, continue with the full statement apart from the single apex, and finally finish with the most general statement.

Lemma 4.4.14. *Let G be a planar graph with a single apex and a diameter d_T spanning tree T of G. For a given set of parts, there exists a T-restricted shortcut with block parameter $O(\log d_T)$ and congestion $O(d_T \log d_T)$.*

Proof. Let x be the apex, and let $H := G - x$ be the planar region. First, if a part P contains x, we give P the entire spanning tree; there can be at most one such part, so the congestion does not change asymptotically. From now on, assume that no part contains the apex x.

Consider removing the apex x, which breaks the tree T into multiple connected subtrees in H. Note that each subtree has diameter at most d_T. For each subtree, let its vertices be a new cell C, and denote the subtree by $T[C]$. Since the family of planar graphs is closed under edge contraction, we can invoke Lemma 4.4.10, so H with cell partition $\mathcal{C}$ is β-cell-assignable for $\beta(d) := O(d)$, so there is a corresponding relation $\mathcal{R} \subseteq \mathcal{C} \times \mathcal{P}$. As in Lemma 4.3.1, we construct local and global shortcuts separately, using ideas from Section 4.4.1 for the local shortcuts and the relation $\mathcal{R}$ for the global shortcuts.

We first begin with global shortcuts. For each part $P \in \mathcal{P}$ and every cell $C \in \mathcal{C}$ assigned to P in the relation $\mathcal{R}$, assign all edges in $T[C]$ to part P, as well as the edge connecting the apex x to T_C, called the **uplink**. Since every edge, with the exception of uplinks, belongs to one cell, and since every cell is in relation with at most $\beta(d_{T_C})$ parts, the congestion on each edge is $\beta(d_{T_C}) \leq \beta(d_T)$ from global shortcuts.

Next, we define local shortcuts by repeating the following for each $C \in \mathcal{C}$ individually. (i) Take the graph H, and iteratively contract all edges in $E_H \setminus E_{H[C]}$, i.e. all edges outside the graph induced by C. (They are contracted in the same way as in Lemma 4.4.10, so that all parts $P \cap C$ remain connected in the resulting graph. Denote the graph by H_C. (ii) Next, construct a $b(d) := O(\log d)$ block parameter, $c(d) := O(d \log d)$ congestion T_C-restricted shortcut using Theorem 2.2.4 on H_C with the parts $\{P \cap C : P \in \mathcal{P}\}$. Note that the parts are still connected via the contractions. Add it as a local shortcut of the cell C. Each edge in T has congestion $c(d_{T_C}) \leq c(d_T)$ from local shortcuts.

Finally, we argue about block parameter. For each part P, there are at most 2 cells intersecting P but not in relation with P in $\mathcal{R}$. Each of these cells C generates $b(d_{T_C}) \leq b(d_T)$ additional blocks, and together with the single block from the global shortcuts, gives a blocking parameter of $1 + 2 \cdot b(d_T)$.

Plugging in $\beta(d_T) = O(d_T)$, $b(d_T) = O(\log d_T)$, and $c(d_T) = O(d_T \log d_T)$, we get block parameter $1 + 2 \cdot b(d_T) = O(\log d_T)$ and congestion $\beta(d_T) + c(d_T) = O(d_T \log d_T)$. □

Lemma 4.4.15. *Let G be a genus-g graph with ℓ vortices of depth k and a single apex and T a spanning tree of G. For a given set of parts, there exists a T-restricted shortcut with block parameter $O((g+1)k\ell^2 d_T)$ and congestion $O(k\ell^2 d_T(g + \log n))$.*

Proof. We proceed similarly as in Lemma 4.4.14, with different parameters β, b, and c. Let G be the entire graph, let x be the apex, and let $H := G - x$ be the graph without the apex. Let $\mathcal{C}$ be the cell partition as defined in Lemma 4.4.14. To obtain the actual partition $\mathcal{C}'$ that we can use as a precondition in Lemma 4.4.13, we first start with $\mathcal{C}$ and then, iteratively, for each vortex in H, merge all cells that intersect the vortex. Note that if a cell in $\mathcal{C}'$ intersects a vortex, then it completely contains the vortex, and a cell may contain multiple vortices. We let all cells that contain, or equivalently, intersect, a vortex to be special, so that there are at most ℓ special cells. The remaining cells are normal cells. At this point, all normal cells have diameter $O(d_T)$, but special cells can have unbounded diameter due to the individual vortices. To remedy this issue, for each vortex, we create a *star vertex* that connects to all boundary vertices of the vortex, and add it to the special cell containing this vortex. Doing so increases the depth of each vortex by at

most 1 and decreases the diameter of each special cell to $O(\ell d_T)$. Denote the normal cells by C^0 and the special cells by $\mathcal{C}^*$.

Define a graph family $\mathcal{F}$ to be all genus-g graphs with at most ℓ vortices of depth at most k, so that $H \in \mathcal{F}$. Note that for each graph $F \in \mathcal{F}$, contracting an edge outside of any vortex still leaves a graph in $\mathcal{F}$. Since every normal cell is in the genus-g region of the graph, any graph obtained by contracting edges in normal cells in H is still in $\mathcal{F}$. Therefore, we can apply Lemma 4.4.13 with cell diameter $O(\ell d_T)$ to get an $O((g+1)k\ell d_T)$-combinatorial gate, and then apply Lemma 4.4.11 to obtain a relation $\mathcal{R} \subseteq \mathcal{C}' \times \mathcal{P}$ with $\beta(d_T) := 2\ell((g+1)k\ell d_T)$. Note that our combinatorial gate applies to the graph with the extra star vertices added, not the original graph. However, only special cells get star vertices and the relation $\mathcal{R}$ does not touch special cells, so $\mathcal{R}$ is valid for the original graph.

The global shortcuts for normal cells are the same as those in Lemma 4.4.14, giving a congestion of at most $\beta(d_T)$. Note that there are no global shortcuts for special cells, since $\mathcal{R}$ does not associate special cells. For local shortcuts in a normal cell $C \in \mathcal{C}^0$, we define H_C as follows: we first contract all edges in $E_H \setminus \left(E_{H[C]} \cup \bigcup_{C' \in C^*} E_{H[C']}\right)$, i.e., all edges not inside C or any special cell, in the same way as in Lemma 4.4.10. We then contract the edges in $\bigcup_{C' \in C^*} E_{H[C']} \setminus E_{H[C]}$, i.e., the remaining edges not inside C, in the same way as in Lemma 4.4.10 to obtain H_C. The first set of contractions leaves C with the special cells. With the star vertices added, this graph is a genus g graph with ℓ vortices and diameter $O(\ell d_T)$, so by Lemma 4.4.2, it has treewidth $O((g+1)k\ell^2 d_T)$; disregarding the star vertices can only decrease the treewidth. The second set of contractions also cannot increase the treewidth, so H_C also has treewidth $O((g+1)k\ell^2 d_T)$. Applying the treewidth-based shortcut construction from [66] gives block parameter $b(d_T) := O((g+1)k\ell^2 d_T)$ and congestion $c(d_T) := O((g+1)k\ell^2 d_T \log n)$.

For local shortcuts in special cells, we construct them in all special cells simultaneously. Let $C^* := \bigcup_{C \in C^*} C$ be the union of all special cells. Define the tree $T^* := T[x \cup C^*]$ to be the union of C^* with all uplinks in C^*. Take the graph H and contract all edges in $E_H \setminus \bigcup_{C \in C^*} E_{H[C]}$, i.e., all edges not inside any special cell, in the same way as in Lemma 4.4.10, obtaining a genus g graph with ℓ vortices and diameter $O(\ell d_T)$; This graph has treewidth $O((g+1)k\ell^2 d_T)$. Add the apex x back to H and connect it to its neighbors in G that are vertices of H, which increases the treewidth by at most 1. The resulting graph is spanned by T^*, so it has shortcuts with block parameter $b(d_T)$ and congestion $c(d_T)$, as defined above.

Finally, we argue about block parameter. For each part P, there are at most 2 normal cells intersecting P but not in relation with P in $\mathcal{R}$, and at most ℓ special cells. Each of these cells generates $b(d_T)$ additional blocks, and together with the single block from the global shortcuts, gives a block parameter of $1 + (2+\ell) \cdot b(d_T)$.

Plugging in $\beta(d_T) = O((g+1)k\ell^2 d_T)$, $b(d_T) = O((g+1)k\ell^2 d_T)$, and $c(d_T) = O(k\ell^2 d_T \log n)$, we get block parameter $1 + 2 \cdot b(d_T) = O((g+1)k\ell^2 d_T)$ and congestion $\beta(d_T) + c(d_T) = O(k\ell^2 d_T(g + \log n))$. □

We finally prove the main theorem of this section, with multiple apices.

Theorem 4.2.3. *[Shortcuts in Almost Embeddable Graphs] An (q_T, g, k, ℓ)-almost-embeddable graph G admits tree-restricted shortcuts with block parameter $b(d) = O(q_T + (g+1)k\ell^2 d)$ and congestion $c(d) = O(q_T + k\ell^2 d(g + \log n))$.*

Proof. Let G be the apex graph and T a the spanning tree of G. If a part contains one of the q_T apices, we give the entire tree T to the part. This increases the congestion by at most q_T. For the remaining parts, we do the following. First, add an auxiliary new vertex x that connects to each of the q_T apices; the diameter can grow by at most 1. Contract these $q_T + 1$ vertices to a single apex to form graph G'; T might now contain cycles, so take a spanning subtree of depth d_T in the contracted T. Apply Lemma 4.4.15 to the single apex graph G'. If we extend the shortcuts for each part in the natural way to G, the congestion does not change any further. Furthermore, the block parameter increases by at most $q_T - 1$ because a block component containing x splits into at most q_T block components. □

4.5 Conclusion and Open Problems

We have proved all the ingredients we need to prove our main theorem, which we restate for convenience.

Theorem 4.2.1. *[Main Theorem, Extended Version] The family of graphs excluding a fixed minor H admits tree-restricted shortcuts of T-quality $q_T(d) = \widetilde{O}(d^2)$. More generally, the family admits block parameter $b(d) = O(d)$ and congestion $c(d) = O(d \log n + \log^2 n)$. The constants in the big-O depend only on the minor H.*

An obvious open question is whether the block parameter $O(d_T)$ can be improved to $\tilde{O}(1)$, which would result in a near-optimal $\tilde{O}(D)$-round algorithm for MST and $(1+\varepsilon)$-approximate mincut on excluded minor network graphs. The bottleneck in the current proof lies in the treewidth argument when arguing about Genus+Vortex graph, which produces the $O(d_T)$ block parameter. This treewidth argument cannot be improved due to lower bounds on treewidth-k graphs, as presented in [66]. Hence, an improvement on Genus+Vortex graphs requires a better understanding of vortices, beyond treating them as simply low-treewidth (or pathwidth) graphs.

4.6 Chapter Appendix: Combinatorial Gate in Genus+Vortex graphs

In this section we prove Lemma 4.4.13. Namely, that the Genus+Vortex graphs have an s-combinatorial gate. The exposition is split into three parts: (1) a general "Planarization" Lemma that converts a genus-g graph into a planar one, (2) proof that Genus-g graphs have an s-combinatorial boundary, and finally (3) the proof of Lemma 4.4.13.

4.6.1 Planarization of Genus-g Graphs

Genus-g graphs are relatively hard to analyze. This is in contrast to planar graphs for which many tools are available. In this section we state a lemma that will help us analyzing genus-g graphs by "cutting and developing them on a plane". The high-level idea is to "cut" the genus-g graph SG along multiple cycles, providing us with a "planarization" PG that is planar (see Figure 4.7). By "cutting" we informally mean taking scissors and cutting along edges in a cycle in a way that splits each edge into two sub-edges, one for each side of the cut. This also splits the nodes into multiple sub-nodes, possibly more than 2. For example, if a node lies on k edge-disjoint cutting paths, we split the node into $2k$ sub-nodes. Such a planarization is illustrated on Figure 4.7.

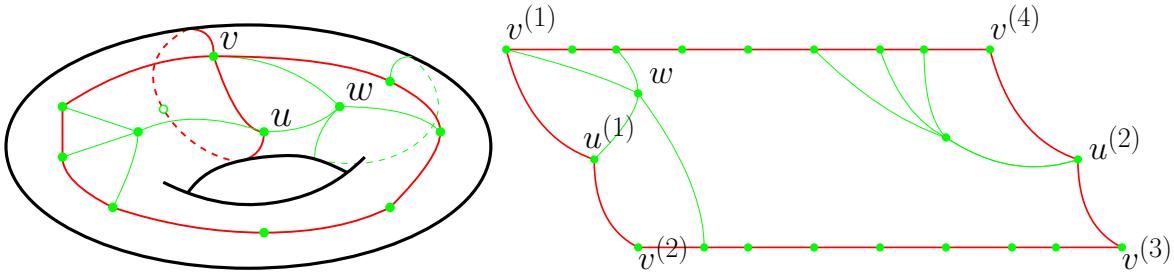


Figure 4.7: A graph embedded on a torus and its planarization after cutting the generators colored in red. Note how the vertex v gets duplicated into $v^{(1)}, \ldots, v^{(4)}$.

On a more technical note, we have some control over the cycles which are cut. We can choose any spanning tree $T \subseteq SG$. Then the cut can be represented by g non-tree edges and their induced cycles w.r.t. T. A cycle induced by a non-tree edge e is the unique cycle in $T \cup \{e\}$. We first formalize the cutting procedure and then we state the Planarization Lemma in all all detail.

Definition 4.6.1 (Cut Graph)**.** *Given a graph SG embedded on a surface and a subset of cut edges $R \subseteq E(SG)$, we define the **cut graph** PG as follows.*

- *For each $v \in V(SG)$ consider the local planar embedding on the surface (effectively an ordering of edges incident to v). Consider all maximal edge-intervals of non-cut edges in this ordering of v including the two bounding cut edges. For all such edge-intervals construct a new copy of v and include it in $V(PG)$. E.g., if v is adjacent to k cut edges, the vertex will be copied k times.*
- *Define the "projection" $p : V(PG) \to V(SG)$ that maps $v \in V(PG)$ to the original vertex in $V(SG)$ of which v is a copy.*
- *Connect two vertices in $V(PG)$ when their corresponding edge-intervals contain either (i) the same non-cut edge or (ii) the same cut edge while the edge-intervals are on the appropriate sides of their local embeddings. In other words, the cut edge is the clockwise boundary of one edge-interval and counter-clockwise boundary of the other; or vice versa. This should intuitively correspond to "cutting an edge with scissors".*

*On such a cut graph, denote by **outer nodes** all nodes $v \in V(PG)$ s.t. $|p^{-1}(p(v))| > 1$ and also their projections into SG. The outer nodes in $v \in V(SG)$ are exactly those corresponding to edge-intervals that do not contain the entire neighborhood, or equivalently, those without any cut edges incident to them.*

*Similarly, denote the complement of outer nodes (on both PG and SG) as the **inner nodes**. Note that, by the cutting procedure, there is a one-to-one correspondence between inner nodes on PG and inner nodes on SG.*

Lemma 4.6.2 (Planarization Lemma)**.** *Consider a genus-g graph SG and a spanning tree $T \subseteq SG$. There exists a set of g cycles induced by (adversarially chosen) non-T edges. Call them the **generating cycles**. Let PG be the cut graph of SG with respect to the union of the generating cycles. Then (i) PG is a planar graph, and (ii) outer nodes, as defined in Definition 4.6.1, are on the outer face of PG.*

Proof. Consider the dual graph of SG, denoted by SG^*. The vertices/edges/faces of SG^* correspond one-to-one with faces/edges/vertices of SG, respectively. Then there exists a spanning tree T^* of SG^*, called a co-tree, that is disjoint from T. In other words, there is no edge of T^* that corresponds to an edge of T. This claim is a direct consequence of Lemma 1 in Eppstein [39].

Let C be the set of edges not in either T nor T^*. Then Lemma 2 in Eppstein [39] asserts that the cycles induced by the edges of T generate the fundamental group of the surface on which SG is embedded. In other words, contracting the tree T to a single node and deleting $E(T^*)$ would give us an embedded graph with exactly one node and one face. Denote this face by F. Because SG is 2-cell embedded in the surface, F is homeomorphic to some plane which we denote by Π.

We are now ready to prove the Lemma. Our argument will be that the cut graph can be embedded without intersections in the plane Π. Note that the points that are not on the cut edges have a natural embedding into Π, namely, they have a position in F which corresponds to a point in the plane Π. This fixes the embedding of the inner nodes of PG. However, this also forces the embedding of the outer nodes $v \in V(PG)$. Such nodes correspond to an edge-interval of $w \in V(SG)$. Since the surface points sufficiently close inside the edge-interval and arbitrarily close to v are embedded into Π, we just set the embedding of w to be the limit of such surface points.

Such an embedding has no intersections since non-cut-edges correspond one-to-one with edges on F which do not intersect. Cut edges cannot intersect by the cutting procedure. This proves claim (i). Also, note that the boundary of the face F corresponds to the outer face boundary of Π and that exactly outer nodes get mapped to that boundary. This proves claim (ii). □

4.6.2 Combinatorial Gate in Genus-g graphs

We begin with a strengthening of Lemma 4.4.12, whose proof is immediate following the proof of Lemma 4.4.12.

Lemma 4.6.3. *Let G be a planar graph with a cell partition of diameter d. Fix a planar embedding of G. Then, there is an $36d$-combinatorial gate $\mathcal{S}$, and furthermore, each node on the outer face of G contained in a gate S_i is also in the corresponding fence F_i.*

Now, we present the combinatorial gate proof for bounded genus graphs.

Lemma 4.6.4. *A genus-g graph SG with a diameter-d cell partition $\mathcal{C}_{SG}$ has a $O((g+1)d)$-combinatorial gate.*

Proof. The main idea is to planarize the graph using Lemma 4.6.2, find a combinatorial gate for the planar graph and project it back to the surface graph SG. The details follow.

We construct a rooted spanning tree T of SG by first constructing a spanning tree of each cell, connecting them arbitrarily into a spanning tree, and arbitrarily rooting it. Denote the inter-cell tree edges as **connecting edges**. For concreteness, assume directed tree edges go towards the root.

Planarize the graph SG into $PG = (V(PG), E(PG))$ w.r.t. T using Lemma 4.6.2. Let $p : V(PG) \to V(SG)$ denote the corresponding projection. We define a cell decomposition $\mathcal{C}_{PG}$ that will intuitively match the decomposition on $\mathcal{C}_{SG}$, except that some cells get split in multiple ones to respect the planarization. We formalize it by defining $\mathcal{C}_{PG}$ on the planar graph PG in an implicit manner: we will define them as connected components of a forest of rooted trees $T_{PG} \subseteq PG$ that is defined as follows. For $x, y \in V(PG)$ there is an directed edge $x \to y$ in T_{PG} when all three of (i) $p(x)$ and $p(y)$ are in the same cell of $\mathcal{C}_{SG}$, (ii) $(p(x) \to p(y)) \in T$, and (iii) $\{x, y\} \in E(PG)$ hold. This completely defines T_{PG} and therefore $\mathcal{C}_{PG}$.

Note that (i) T_{PG} is a tree (when ignoring directions) and (ii) each vertex $x \in V(PG)$ has outdegree at most 1. Claim (i) follows because any cycle in T_{PG} would project into a cycle in T; and claim (ii) follows from construction since $p(x) \in T$ has outdegree at most 1 and the projections of neighboring nodes of x in PG are all distinct.

Next, we analyze the cells $\mathcal{C}_{PG}$. Note that cells in $\mathcal{C}_{PG}$ have diameter at most $2d = O(d)$ since traveling via out-edges towards the root will reach it within d steps. This can be argued from the projection of such a travel reaching its root in at most d steps.

Furthermore, we claim that each cell splits into at most $O(g + 1)$ new cells. We can represent each cell $C \in \mathcal{C}_{PG}$ with its root node in T_{PG}. And since every node in PG corresponds to an edge-interval of a vertex in SG (c.f. planarization), we can represent C by an edge-interval. Finally, we say that $C \in \mathcal{C}_{PG}$ splits from $C' \in \mathcal{C}_{SG}$ when the projection of the root of C maps to a node in C'.

We fix a cell $C' \in \mathcal{C}_{SG}$, i.e., on the surface graph, and argue about the number of nodes that split from from C'. Let $C \in \mathcal{C}_{PG}$ be such a cell and let v be the root of its corresponding component of T_{PG}. Note that node v being a root in T_{PG} implies that either (i) its edge-interval does not contain the unique outgoing edge out of $p(v)$ or (ii) it contains an outgoing connecting edge, i.e., unique edge connecting $T \subseteq SG$ with its parent. The number of cases (i) increases by $O(1)$ with each new generating cycle, of which there are $O(g + 1)$. Case (ii) can occur only twice since the

construction implies each edge gets duplicated at most twice. This concludes the argument that there are $|\mathcal{C}_{PG}| \leq O(g+1)|\mathcal{C}_{SG}|$.

To summarize, we have constructed a planar PG and a diameter-$O(d)$ cell partition $\mathcal{C}_{PG}$. We now apply Lemma 4.4.12 to find a $O(d)$-combinatorial gate $\mathcal{S} = \{(F_i, S_i)\}_i$.

In order to construct a combinatorial gate in SG, we project $\mathcal{S}$. To that end, we extend the projection p to work on subsets $2^{V(SG)}$ in the obvious manner: $p(A) = \bigcup_{a \in A} p(a)$. Next, let $\mathcal{S}' = \{p(F_i), p(S_i) \mid (F_i, S_i) \in \mathcal{S}\}_i$. We claim that $\mathcal{S}'$ is an $O((g+1)d)$-combinatorial gate in SG and we prove it by verifying its properties one by one.

(1) $p(F_i) \subseteq p(S_i)$ is clear because p is an increasing function w.r.t. $\subseteq$.

(2) We want to show that $\partial p(S_i) \subseteq p(F_i)$. Let $v' \in \partial p(S_i)$. If v' is an inner node and $w' \in V(SG) \setminus p(S_i)$ is its neighbor outside of $p(S_i)$, then any preimage $w \in p^{-1}(w')$ must be outside of S_i, hence the unique $p^{-1}(v') \in \partial S_i \subseteq F_i$. This implies that $v' \in p(F_i)$ as needed. On the other hand, if v' is not an inner node, then any preimage $v \in p^{-1}(v') \cap S_i$ must be an outer node, hence on the outer face of PG by Lemma 4.4.12. If we go through the proof of Lemma 4.4.12, we observe that construction of the planar combinatorial gate in the lemma has an additional property: if a node on the outer face of the planar graph is contained in a gate S_i, then it is also in the corresponding fence F_i. Thus by construction of $\mathcal{S}$ it is included in F_i, which implies $v' \in p(F_i)$.

(3) Let $\{a', b'\} \in E(SG)$ be an edge whose endpoints are in different cells of $\mathcal{C}_{SG}$. Then $\{p(a'), p(b')\}$ is covered by a gate S in PG. Hence the gate $p(S)$ covers $\{a', b'\}$.

(4) Each gate S' intersects at most 2 cells in $\mathcal{C}_{OG}$ since the projection maps the same cell into the same cell.

(5) Let $v' \in p(S_i) \setminus p(F_i)$. We want to show that there can be at most one such i. On one hand, if $v' \in V(SG)$ is an outer node, then its preimage $p^{-1}(v') \cap S_i$ must be an outer node and hence included in F_i. This is a contradiction since then $v' \notin p(S_i) \setminus p(F_i)$. On the other hand, if $v' \in V(SG)$ is an inner node, then it has a unique preimage $v = p^{-1}(v')$. If $v' \in p(S_i) \setminus p(F_i)$, then $v \in S_i \setminus F_i$. Hence by claim (5) on $\mathcal{S}$ there can be at most one such i.

(6) $\sum_{(F',S') \in \mathcal{S}'} |F'| \leq \sum_{(F,S) \in \mathcal{S}} |p(F)| \leq \sum_{(F,S) \in \mathcal{S}} |F| \leq O(d)|\mathcal{C}_{PG}| \leq O((g+1)d)|\mathcal{C}_{OG}|$ □

4.6.3 Finalizing the Proof

Finally, we extend the combinatorial gate proof for bounded genus graphs to include vortices.

Lemma 4.4.13. *Let G be a genus-g graph with (a possibly unbounded number of) vortices of depth k, and consider a cell partition of diameter d such that no vortex is split between more than one cell. Then, there exists an $O((g+1)kd)$-combinatorial gate of G.*

Proof. For notation, rename the graph G to OG for "original graph". We first replace the Genus-g+Vortex-k OG graph with a tightly related genus-g graph SG by the following method: for each vortex A in OG, remove all the internal vortex nodes and replace it with a star node s_A that

is connected to all nodes in the cycle of the vortex. Name the final graph SG and note that it is genus g by construction.

Next, we construct a corresponding cell partition $\mathcal{C}_{SG}$: all non-star nodes get included in the same cell as they were in OG, while the star node gets included in the cell of its vortex. Note that the preconditions ensure that all vortex nodes are in the same cell. Furthermore, the diameter of the cell partition $\mathcal{C}_{SG}$ is at most $d + 1 = O(d)$, hence we can apply Lemma 4.6.4 on it and obtain a $O((g + 1)d)$-combinatorial gate $\mathcal{S}$.

We now convert $\mathcal{S}$ into a $O((g + 1)kd)$-combinatorial gate $\mathcal{S}'$ on OG, hence completing the theorem. First, define an **expansion** $\mathcal{E} : V(SG) \to 2^{V(OG)}$ in the following manner: if v is a star node, then $\mathcal{E}(v) = \emptyset$; if v is a node on a vortex cycle, then $\mathcal{E}$ contains the set of all internal vortex nodes of OG whose arcs contain v and v itself; finally, if v is neither a star nor vortex cycle node, then $\mathcal{E}(v) = \{v\}$. Note that $\mathcal{E}$ furnishes a one-to-one correspondence between non-star nodes of SG and non-internal vortex nodes of OG.

This allows us to define $\mathcal{S}'$. First, extend the expansion to work on subsets $2^{V(SG)}$ in the obvious manner: $\mathcal{E}(A) = \bigcup_{a \in A} \mathcal{E}(a)$. Then set $\mathcal{S}' := \{(\mathcal{E}(F_i), \mathcal{E}(S_i) \mid (F_i, S_i) \in \mathcal{S}\}$. In other words, we expand the gates and fences in $\mathcal{S}$ to obtain $\mathcal{S}'$.

We prove that $\mathcal{S}'$ is an $O((g + 1)kd)$-combinatorial gate by verifying its properties one by one.

1. It is clear that $F'_i \subseteq S'_i$ for each $(F'_i, S'_i) \in \mathcal{S}$ since $\mathcal{E}$ is an increasing function w.r.t. $\subseteq$.
2. We want to show that $\partial S'_i \subseteq F'_i$. Let $v' \in \partial S'_i$ where $S'_i = \mathcal{E}(S_i), F'_i = \mathcal{E}(F_i)$. We split into three cases, depending on the location of v'.
   - If v' is an internal vortex node, it is enough to prove that F_i contains at least one node in the arc of v', since this would imply that $v' \in F'_i$. Suppose, otherwise, that F_i does not intersect the arc of v'. Knowing that S_i contains at least one node in the arc of v', we conclude that S_i contains the entire arc of v'. Therefore, all neighbors of v' on the vortex boundary are in S'_i. Moreover, every neighbor of v' internal to the vortex must share a boundary vertex in the arc of v', so the neighbor is also in S'_i. Therefore, all neighbors of v' are contained in S'_i, a contradiction.
   - If v' is neither an internal vortex node nor on the vortex cycle, then there is a one-to-one correspondence via $\mathcal{E}$ between its neighborhood and gate incidence as in SG since v' cannot be connected to a vortex internal vertex. Hence the claim follows from the same claim for $\mathcal{S}'$.
   - If $v' \in \partial S'_i$ is on a vortex cycle, let $w' \in N(v') \setminus S'_i$ be its neighbor outside of S'_i. If w' is not internal to the vortex, then this case is equivalent to the previous one. If w' is internal to the vortex, then w' would be in S'_i since $S'_i = \mathcal{E}(S_i) \subseteq \mathcal{E}(\mathcal{E}^{-1}(v')) \ni w'$.
3. An edge $\{a', b'\} \in E(OG)$ whose endpoint are in different cells of $\mathcal{C}_{OG}$ cannot have any of its endpoints as an internal vortex node, hence it has a corresponding edge $\{\mathcal{E}^{-1}(a), \mathcal{E}^{-1}(b)\}$ in $E(SG)$. The claim now follows from the same claim in $\mathcal{S}$.
4. Let S' be a gate we want to prove intersects at most 2 cells. If S' does not intersect the vortex internals, then it has a corresponding gate $\mathcal{E}^{-1}(S')$ in $\mathcal{S}$ from which the claim

follows. If it intersects the vortex, then each internal vortex node belongs to the same cell denoted by c. Furthermore, the cells intersecting S' are exactly the same as those intersecting $\mathcal{E}^{-1}(S')$ together with c. But the cells intersecting $\mathcal{E}^{-1}(S')$ are either an empty set or already include c, hence the claim follows.

5. Let $v' \in S'_i \setminus F'_i$. If v' is not an internal vortex node, then there can be only one such $S' \setminus F' \ni v'$ from the claim for $\mathcal{S}$. On the other hand, if v' is a internal vortex node, then $S'_i \setminus F'_i$ must contain the entire arc of v'. Then the claim follows from the non-internal vortex node case by picking any such node on the arc.

6. We first note that $|\mathcal{E}(X)| \leq k|X|$ since the vortex is of depth k. Then we have

$$\sum_{(F',S')\in\mathcal{S}'} |F'| \leq \sum_{(F,S)\in\mathcal{S}} |\mathcal{E}(F)| \leq k \sum_{(F,S)\in\mathcal{S}} |F| \leq kO((g+1)d)|\mathcal{C}_{SG}| \leq O((g+1)kd)|\mathcal{C}_{OG}|.$$

□

Chapter 5

Network Coding Gaps for Completion-time of Multiple Unicasts

The results of this chapter were published in [69] with Bernhard Haeupler and David Wajc as co-authors. The work was supported in part by NSF grants CCF-1618280, CCF-1814603, CCF-1527110, NSF CAREER award CCF-1750808, and a Sloan Research Fellowship, as well as a DFINITY scholarship.

5.1 Introduction

In this chapter we study the natural mathematical abstraction of what is arguably the most common network communication problem: *multiple unicasts*. In this problem, distinct packets of different size are at different nodes in a network, and each packet needs to be delivered to a specific destination as fast as possible. That is, minimizing the *makespan*, or the time until all packets are delivered.

All known multiple-unicast solutions employ (fractional) *routing* (also known as store-and-forward protocols), i.e., network nodes potentially subdivide packets and route (sub-)packets to their destination via store and forward operations, while limited by edge capacities. The problem of makespan minimization of routing has been widely studied over the years. A long line of work [10, 14, 19, 92, 99, 100, 117, 119, 120, 124, 131, 132, 135], starting with the seminal work of Leighton, Maggs, and Rao [99], studies makespan minimization for routing along fixed paths. The study of makespan minimization for routing (with the freedom to pick paths along which to route) resulted in approximately-optimal routing, first for asymptotically-large packet sizes [14], and then for all packet sizes [135].

It seems obvious at first that routing packets, as though they were physical commodities, is the only way to solve network communication problems, such as multiple unicasts. Surprisingly, however, results discovered in the 2000s [7] suggest that information need not flow through a network like a physical commodity. For example, nodes might not just forward information, but

instead send out XORs of received packets. Multiple such XORs or linear combinations can then be recombined at destinations to reconstruct any desired packets. An instructive example is to look at the XOR $C \oplus M$ of two s-bit packets, C and M. While it is also s bits long, one can use it to reconstruct either all s bits of C or all s bits of M, as long as the other packet is given. Such network coding operations are tremendously useful for network communication problems, but they do not have a physical equivalent. Indeed, the $C \oplus M$ packet would correspond to some s ounces of a magic "café latte" liquid with the property that one can extract either s ounces of milk or s ounces of coffee from it, as long as one has enough of the other liquid already. Over the last two decades, many results demonstrating gaps between the power of network coding and routing have been published (e.g., [7, 8, 25, 28, 43, 57, 62, 63, 76, 87, 97, 103, 138, 139, 140]). Attempts to build a comprehensive theory explaining what is or is not achievable by going beyond routing have given rise to an entire research area called network information theory.

The question asked in this chapter is:

"How much faster than routing can network coding be for any multiple-unicast instance?"

In other words, what is the (multiplicative) *network coding gap* for makespan of multiple unicasts. Surprisingly, no general makespan coding gap bounds were known prior to this work. This is in spite of the vast amount of effort invested in understanding routing strategies for this problem, and ample evidence of the benefits of network coding.

This question was studied in depth for the special case of *asymptotically-large* packet sizes, otherwise known as *throughput* maximization (e.g., [3, 8, 75, 76, 85, 93, 97, 103, 103, 140, 141]). Here, the maximum *throughput* of a multiple-unicast instance can be defined as $\sup_{w\to\infty} w/C(w)$, where $C(w)$ is the makespan of the fastest protocol for the instance after increasing all packet sizes by a factor of w (see Section 5.5). In the throughput setting, no instances are known where coding offers *any* advantage over routing, and this is famously conjectured to be the case for all instances [75, 103]. This conjecture, if true, has been proven to have surprising connections to various lower bounds [3, 5, 41]. Moreover, by the work Afshani et al. [5], a throughput coding gap of $o(\log k)$ for all multiple-unicast instances with k unicast pairs (k-*unicast instances*, for short) would imply explicit *super-linear circuit lower bounds*—a major breakthrough in complexity theory. Such a result is currently out of reach, as the best known upper bound on throughput coding gaps is $O(\log k)$, which follows easily from the same bound on multicommodity flow/sparsest cut gaps [11, 106].

In this work we prove makespan coding gaps for the general problem of *arbitrary* packet sizes. In particular, we show that this gap is at most $O(\log^2 k)$ for any k-unicast instance (for the most interesting case of similar-sized packet sizes). We note that any coding gap upper bound for this more general setting immediately implies the same bound in the throughput setting (Section 5.5), making our general bound only quadratically larger than the best known bound for the special case of throughput. Complementing our results, we prove that there exist k-unicast instances where the network coding gap is $\Omega(\log^c k)$ for some constant $c > 0$.

To achieve our results we develop novel techniques that might be of independent interest. The need for such new tools is due to makespan minimization for general packet sizes needing to

take both source-sink distances as well as congestion issues into account. This is in contrast with the throughput setting, where bounds must only account for congestion, since asymptotically-large packet sizes make distance considerations inconsequential. For our more general problem, we must therefore develop approaches that are both congestion- and distance-aware. One such approach is given by a new combinatorial object we introduce, dubbed the *moving cut*, which allows us to provide a *universally optimal* characterization of the coding makespan. That is, it allows us to obtain tight bounds (up to polylog terms) on the makespan of *any* given multiple-unicast instance. We note that moving cuts can be seen as generalization of prior approaches that were (implicitly) used to prove unconditional lower bounds in distributed computing on specially crafted networks [27, 122]; the fact they provide a *characterization* on all networks and instances is novel. This underlies our main result—a polylogarithmic upper bound on the makespan coding gap for any multiple-unicast instance.

5.1.1 Preliminaries

In this section we define the completion-time communication model. We defer the, slightly more general, information-theoretic formalization to Section 5.6.

A *multiple-unicast instance* $\mathcal{M} = (G, \mathcal{S})$ is defined over a communication network, represented by an connected undirected graph $G = (V, E)$ with capacity $c_e \in \mathbb{Z}_{\geq 1}$ for each edge e. The $k \triangleq |\mathcal{S}|$ *sessions* of $\mathcal{M}$ are denoted by $\mathcal{S} = \{(s_i, t_i, q_i)\}_{i=1}^{k}$. Each session consists of source node s_i, which wants to transmit a packet to the sink t_i, consisting of $q_i \in \mathbb{Z}_{\geq 1}$ sub-packets. Without loss of generality we assume that a uniform sub-packetization is used; i.e., all sub-packets have the same size (think of sub-packets as the underlying data type, e.g., field elements or bits). For brevity, we refer to an instance with k sessions as a k-*unicast* instance.

A *protocol* for a multiple-unicast instance is conducted over finitely-many *synchronous time steps*. Initially, each source s_i knows its packet, consisting of d_i sub-packets. At any time step, the protocol instructs each node v to send a different packet along each of its edges e. The packet contents are computed with some predetermined function of packets received in prior rounds by v or originating at v. *Network coding protocols* are unrestricted protocols, allowing each node to send out any function of the packets it has received so far. On the other hand, *routing protocols* are a restricted, only allowing a node to forward sub-packets which it has received so far or that originate at this node.

We say a protocol for multiple-unicast instance has *completion times* $(T_1, T_2, \ldots, T_k)$ if for each $i \in [k]$, after T_i time steps of the protocol the sink t_i can determine the d_i-sized packet of its source s_i. The complexity of a protocol is determined by functions $\mathcal{C} : \mathbb{R}_{\geq 0}^{k} \to \mathbb{R}_{\geq 0}$ of its completion times. For example, a protocol with completion times $(T_1, T_2, \ldots, T_k)$ has *makespan* $\max_{i \in [k]} T_i$ and *average completion time* $(\sum_{i \in [k]} T_i)/k$. Minimizing these measures is a special case of minimizing weighted ℓ_p norms of completion time, namely minimizing $(\sum_{i \in [k]} w_i \cdot T_i^p)^{1/p}$ for some $\vec{w} \in \mathbb{R}^k$ and $p \in \mathbb{R}_{\geq 0}$.

Since coding protocols subsume routing ones, for any function $\mathcal{C}$ of completion times, and for any multiple-unicast instance, the fastest routing protocol is no faster than the fastest coding

protocol. Completion-time coding gaps characterize how much faster the latter is.

Definition 5.1.1. *(Completion-time coding gaps) For any function $\mathcal{C} : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$ of completion times, the* network coding gap for $\mathcal{C}$ *for a k-unicast instance $\mathcal{M} = (G, \mathcal{S})$ is the ratio of the smallest $\mathcal{C}$-value of any routing protocol for $\mathcal{M}$ and the smallest $\mathcal{C}$-value of any network coding protocol for $\mathcal{M}$.*

We note that the multiple-unicast instance problem can be further generalized, so that each edge has both capacity and *delay*, corresponding to the amount of time needed to traverse the edge. This more general problem can be captured by replacing each edge e with a path with unit delays of total length proportional to e's delay. As we show, despite path length being crucially important in characterizing completion times for multiple-unicast instances, this transformation does not affect the worst-case coding gaps, which are independent of the network size (including after this transformation). We therefore consider only unit-time delays in this chapter, without loss of generality.

5.1.2 Our Contributions

In this work we show that completion-time coding gaps of multiple unicasts are vastly different from their throughput counterparts, which are conjectured to be trivial (i.e., equal to one). For example, while the throughput coding gap is always one for instances with $k = 2$ sessions [81], for makespan it is easy to derive instances with $k = 2$ sessions and coding gap of $4/3$ (based on the butterfly network). Having observed that makespan coding gaps can in fact be nontrivial, we proceed to study the potential asymptotic growth of such coding gaps as the network parameters grow. We show that the makespan coding gap of multiple unicasts with k sessions and packet sizes $\{d_i\}_{i\in[k]}$ is polylogarithmic in the problem parameters, k and $\sum_i d_i / \min_i d_i$, but independent of the network size, n. The positive part of this result is given by the following theorem.

Theorem 5.1.2. *The network coding gap for makespan of any k-unicast instance is at most*

$$O\left(\log(k) \cdot \log\left(\sum_i d_i / \min_i d_i\right)\right).$$

For similarly-sized packets, this bound simplifies to $O(\log^2 k)$. For different-sized packets, our proofs and ideas in [123] imply a coding gap of $O(\log k \cdot \log(nk))$. Moreover, our proofs are constructive, yielding for any k-unicast instance $\mathcal{M}$ a routing protocol which is at most $O(\log k \cdot \log(\sum_i d_i / \min_i d_i))$ and $O(\log k \cdot \log(nk))$ times slower than the fastest protocol (of any kind) for $\mathcal{M}$. We note that our upper bounds imply the same upper bounds for throughput (see Section 5.5). Our bounds thus also nearly match the best coding gap of $O(\log k)$ known for this special case of makespan minimization.

On the other hand, we prove that a polylogarithmic gap as in Theorem 5.1.2 is inherent, by

providing an infinite family of multiple-unicast instances with unit-sized packets ($d_i = 1$ for all $i \in [k]$) exhibiting a polylogarithmic makespan coding gap.

Theorem 5.1.3. *There exists an absolute constant $c > 0$ and an infinite family of k-unicast instances whose makespan coding gap is at least $\Omega(\log^c k)$.*

Building on our results for makespan we obtain similar results to Theorems 5.1.2 and 5.1.3 for average completion time and more generally for any weighted ℓ_p norm of completion times.

5.1.3 Techniques

Here we outline the challenges faced and key ideas needed to obtain our results, focusing on makespan.

Upper Bounding the Coding Gap

As we wish to bound the ratio between the best makespan of any routing protocol and any coding protocol, we need both upper and lower bounds for these best makespans. As it turns out, upper bounding the best makespan is somewhat easier. The major technical challenge, and our main contribution, is in deriving lower bounds on the optimal makespan of any given multiple-unicast instance. Most notably, we formalize a technique we refer to as the *moving cut*. Essentially the same technique was used to prove that distributed verification is hard on one *particular* graph that was designed specifically with this technique in mind [27, 35, 122]. Strikingly, we show that the moving cut technique gives an almost-tight characterization (up to polylog factors) of the coding makespan for *every* multiple-unicast instance (i.e., it gives *universally* optimal bounds).

We start by considering several prospective techniques to prove that no protocol can solve an instance in fewer than T rounds, and build our way up to the moving cut. For any multiple-unicast instance, $\max_{i \in [k]} \text{dist}(s_i, t_i)$, the maximum distance between any source-sink pair, clearly lower bounds the coding makespan. However, this lower bound can be arbitrarily bad since it does not take edge congestion into account; for example, if all source-sink paths pass through one common edge. Similarly, any approach that looks at sparsest cuts in a graph is also bound to fail since it does not take the source-sink distances into account.

Attempting to interpolate between both bounds, one can try to extend this idea by noting that a graph that is "close" (in the sense of few deleted edges) to another graph with large source-sink distances must have large makespan for routing protocols. For simplicity, we focus on instances where all capacities and demands are one, i.e., $c_e = 1$ for every edge e and $d_i = 1$ for all i, which we refer to as *simple* instances. The following simple lemma illustrates such an approach.

Lemma 5.1.4. *Let $\mathcal{M} = (G, \mathcal{S})$ be a simple k-unicast instance. Suppose that after deleting some edges $F \subseteq E$, any sink is at distance at least T from its source; i.e., $\forall i \in [k]$ we have $\text{dist}_{G \setminus F}(s_i, t_i) \geq T$. Then any routing protocol for $\mathcal{M}$ has makespan at least $\min\{T, k/|F|\}$.*

Proof. For any sets of flow paths between all sinks and source, either (1) all source-sink flow paths contain at least one edge from F, incurring a congestion of $k/|F|$ on at least one of these $|F|$ edges, or (2) there is a path not containing any edge from F, hence having a hop-length of at least T. Either way, any routing protocol must take at least $\min\{T, k/|F|\}$ to route along these paths. □

Perhaps surprisingly, the above bound does not apply to general (i.e., coding) protocols. Consider the instance in Figure 5.1. There, removing the single edge $\{S, T\}$ increases the distance between any source-sink pair to 5, implying any routing protocol's makespan is at least 5 on this instance. However, there exists a network coding protocol with makespan 3: Each source s_i sends its input to its neighbor S and all sinks t_j for $i \neq j$ along the direct 3-hop path $s_i - - - t_j$. Node S computes the XOR of all inputs, passes this XOR to T who, in turn, passes this XOR to all sinks t_j, allowing each sink t_j to recover its source s_j's packets by canceling all other terms in the XOR.

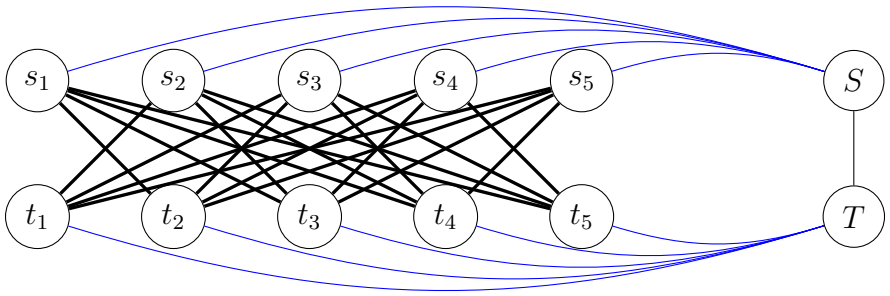


Figure 5.1: A family of instances with $k = 5$ pairs of terminals and makespan coding gap of $5/3$. Thick edges represent paths of 3 hops, while thin (black and blue) edges represent single edges. In other words, each of the k sources s_i has a path of 3 hops (in black and bold) connecting it to every sink t_j for all $j \neq i$. Moreover, all sources s_i neighbor a node S, which also neighbors node T, which neighbors all sinks t_j.

One can still recover a valid general (i.e., coding) lower bound by an appropriate strengthening of Lemma 5.1.4: one has to require that *all* sources be far from *all* sinks in the edge-deleted graph. This contrast serves as a good mental model for the differences between coding and routing protocols.

Lemma 5.1.5. *Let $\mathcal{M} = (G, \mathcal{S})$ be a simple k-unicast instance. Suppose that after deleting some edges $F \subseteq E$, any sink is at distance at least T from **any** source; i.e., $\forall \boldsymbol{i}, \boldsymbol{j} \in [k]$ we have $\text{dist}_{G \setminus F}(\boldsymbol{s_i}, \boldsymbol{t_j}) \geq T$. Then any (network coding) protocol for $\mathcal{M}$ has makespan at least* $\min\{T, k/|F|\}$.

Proof. We can assume all sources can share information among themselves for free (e.g., via a common controlling entity) since this makes the multiple-unicast instance strictly easier to solve; similarly, suppose that the sinks can also share information. Suppose that some coding protocol has makespan $T' < T$. Then all information shared between the sources and the sinks has to pass through some edge in F at some point during the protocol. However, these edges can pass a total

of $|F| \cdot T'$ packets of information, which has to be sufficient for the total of k source packets. Therefore, $|F| \cdot T' \geq k$, which can be rewritten as $T' \geq k/|F|$. The makespan is therefore at least $T' \geq \min\{T, k/|F|\}$. □

Unfortunately, Lemma 5.1.5 is not always tight and it is instructive to understand when this happens. One key example is the previously-mentioned instance studied in the influential distributed computing papers [27, 35, 122] (described in Figure 5.2), where congestion and dilation both play key roles. Informally, this network was constructed precisely to give an $\tilde{\Omega}(\sqrt{n})$ makespan lower bound (leading to the pervasive $\tilde{\Omega}(\sqrt{n} + D)$ lower bound for many global problems in distributed computing [27]). The intuitive way to explain the $\tilde{\Omega}(\sqrt{n})$ lower bound is to say that one either has to communicate along a path of length $\sqrt{n}$ or *all* information needs to shortcut significant distance over the tree, which forces all information to pass through near the top of the tree, implying congestion of $\widetilde{\Omega}(\sqrt{n})$. Lemma 5.1.5, however, can at best certify a lower bound of $\tilde{\Omega}(n^{1/4})$ for this instance. That is, this lemma's (coding) makespan lower bound can be *polynomially* far from the optimal coding protocol's makespan.

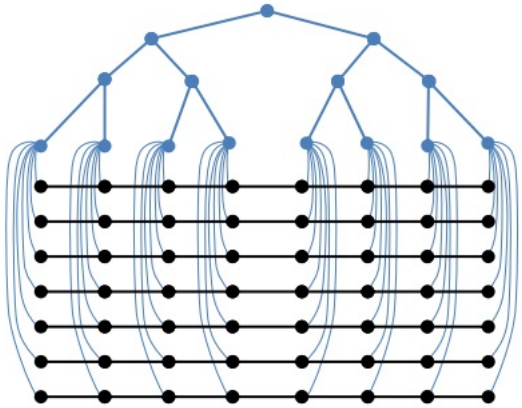


Figure 5.2: The hard instance for distributed graph problems [27, 35, 122], as appears in [49]. The multiple-unicast instance has $\Theta(n)$ nodes and is composed of $\sqrt{n}$ disjoint paths of length $\sqrt{n}$ and a perfectly balanced binary tree with $\sqrt{n}$ leaves. The i^{th} node on every path is connected to the i^{th} leaf in the tree. There are $\sqrt{n}$ sessions with s_i, t_i being the first and last node on the i^{th} path. All capacities and demands are one. The graph's diameter is $\Theta(\log n)$, but its coding makespan is $\tilde{\Omega}(\sqrt{n})$. Figure taken from Ghaffari and Haeupler [49].

A more sophisticated argument is needed to certify the $\tilde{\Omega}(\sqrt{n})$ lower bound for this specific instance. The aforementioned papers [27, 35, 122] prove their results by implicitly using the technique we formalize as our moving cut in the following definition and lemma (proven in Section 5.2.1).

Definition 5.1.6. *(Moving cut) Let $G = (V, E)$ be a communication network with capacities $c : E \to \mathbb{Z}_{\geq 1}$ and let $\{(s_i, t_i) \mid i \in [k]\}$ be source-sink pairs. A **moving cut** is an assignment $\ell : E \to \mathbb{Z}_{\geq 1}$ of positive integer lengths to edges of G. We say the moving cut has **capacity** C, if $\sum_{e \in E} c_e(\ell_e - 1) = C$, and **distance** T, if all sinks and sources are at distance at least T with respect to ℓ; i.e., $\forall i, j \in [k]$ we have $d_\ell(s_i, t_j) \geq T$.*

Lemma 5.1.7. *Let $\mathcal{M} = (G, \mathcal{S})$ be a unicast instance which admits a moving cut ℓ of capacity strictly less than $\sum_{i\in[k]} d_i$ and distance T. Then* any *(coding) protocol for $\mathcal{M}$ has makespan at least T.*

Lemma 5.1.7 can be seen as a natural generalization of Lemma 5.1.5, which can be equivalently restated in the following way: *"Suppose that after increasing each edge e's length from one to $\ell_e \in \{\mathbf{1}, \boldsymbol{T+1}\}$, we have that (1) $\sum_{e\in E} c_e(\ell_e - 1) < \sum_{i=1}^{k} d_i$, and (2) $dist_\ell(s_i, t_j) \geq T$. Then any (coding) protocol $\mathcal{M}$ has makespan at least T"*. Dropping the restriction on ℓ_e recovers Lemma 5.1.7.

Strikingly, the moving cut technique allows us not only to prove tight bounds (up to polylog factors) for the instance of Figure 5.2—it allows us to get such tight bounds *for every multiple-unicast instance*. In order to upper bound the makespan coding gap, we therefore relate such a moving cut with the optimal routing makespan, as follows.

To characterize the optimal routing makespan, we study hop-bounded multicommodity flow, which is an LP relaxation of routing protocols of makespan T. First, we show that a fractional LP solution of high value to this LP implies a routing with makespan $O(T)$. Conversely, if the optimal value of this LP is low, then by strong LP duality this LP's dual has a low-valued solution, which we us to derive a moving cut and lower bound the coding makespan. Unfortunately, the dual LP only gives us bounds on (average) distance between source-sink pairs (s_i, t_i), and not between all sources s_i and sinks t_j (including $j \neq i$), as needed for moving cuts. For this conversion to work, we prove a generalization of the main theorem of Arora, Rao and Vazirani [9] to general metrics, of possible independent interest. (See Section 5.2.3.) This allows us to show that a low-valued dual solution implies a moving cut certifying that no coding protocol has makespan less than $T/O(\log k \cdot \log(\sum_i d_i / \min_i d_i))$. As the above rules out low-valued optimal solutions to the LP for $T = T^* \cdot O(\log k \cdot \log(\sum_i d_i / \min_i d_i))$ with T^* the optimal coding makespan, the LP must have high optimal value, implying a routing protocol with makespan $O(T)$, and thus our claimed upper bound on the makespan coding gap.

Lower Bounding the Coding Gap

To complement our polylogarithmic upper bound on the makespan gap, we construct a family of multiple-unicast instances $\mathcal{M}$ that exhibit a polylogarithmic makespan coding gap. We achieve this by amplifying the gap via graph products, a powerful technique that was also used in prior work to construct extremal throughput network coding examples [15, 18, 110]. Here we outline this approach, as well as the additional challenges faced when trying to use this approach for makespan.

We use a graph product introduced by Braverman et al. [18] (with some crucial modifications). Braverman et al. [18] use their graph product to prove a conditional *throughput* coding gap similar to the one of Theorem 5.1.3, conditioned on the (unknown) existence of a multiple-unicast instance I with non-trivial throughput coding gap. The graph product of [18] takes instances I_1, I_2 and intuitively replaces each edge of I_1 with a source-sink pair of a different copy of I_2. More precisely, multiple copies of I_1 and I_2 are created and interconnected. Edges

of a copy of I_1 are replaced by the same session of different copies of I_2; similarly, sessions of a copy of I_2 replace the same edge in different copies of I_1. This product allows for coding protocols in I_1 and I_2 to compose in a straightforward way to form a fast coding protocol in the product instance. The challenge is in proving impossibility results for routing protocols, which requires more care in the definition of the product graph.

To address this challenge, copies of instances are interconnected along a high-girth bipartite graph to prevent unexpectedly short paths from forming after the interconnection. For example, to prove a throughput routing impossibility result, Braverman et al. [18] compute a dual of the multicommodity flow LP (analogous to our LP, but *without any hop restriction*) to certify a limit on the routing performance. In the throughput setting, a direct tensoring of dual LP solutions of I_1 and I_2 gives a satisfactory dual solution of the product instance. In more detail, a dual LP solution in I assigns a positive length $\ell_I(e)$ to each edge in I; each edge of the product instance corresponds to two edges $e_1 \in I_1$ and $e_2 \in I_2$, and the direct tensoring $\ell_+((e_1, e_2)) = \ell_{I_1}(e_1) \cdot \ell_{I_2}(e_2)$ provides a feasible dual solution with an adequate objective value. To avoid creating edges in the product distance of zero ℓ_+-length, they contract edges assigned length zero in the dual LP of either instance. Unfortunately for us, such contraction is out of the question when studying makespan gaps, as such contractions would shorten the hop length of paths, possibly creating short paths with no analogues in the original instance.

Worse yet, any approach that uses the dual of our T-hop-bounded LP is bound to fail in the makespan setting. To see why, suppose we are given two instances I_1, I_2, both of which have routing makespan at least T and expect that the product instance I_+ to have routing makespan at least T^2 by some construction of a feasible dual LP solution. Such a claim cannot be directly argued since a source-sink path in the product instance that traverses, say, $T - 1$ different copies of I_2 along a path of hop-length $T + 1$ in I_2 could carry an arbitrary large capacity! This is since the hop-bounded LP solution on I_2 only takes *short* paths, of hop-length at most T, into account. Since there is no direct way to compose the dual LP solutions, we are forced to use a different style of analysis from the one of [18], which in turn forces our construction to become considerably more complicated.

To bound the routing *makespan* in the product instance we rely on Lemma 5.1.4: We keep a list of edges F along with each instance and ensure that (i) all source-sink distances in the F-deleted instance are large and that (ii) the ratio of the number of sessions k to $|F|$ is large. We achieve property (i) by interconnecting along a high-girth graph and treating the replacements of edges in F in a special way (hence deviating from the construction of [18]). Property (ii) is ensured by making the inner instance I_2 significantly larger than the outer instance I_1, thus requiring many copies of I_1 and resulting in a large number of sessions in the product graph. To allow for this asymmetric graph product, we need an infinite number of base cases with non-trivial makespan coding gap for our recursive constructions (rather than a single base instance, as in the work of Braverman et al. [18]). This infinite family is fortunately obtained by appropriately generalizing the instance of Figure 5.1.

The main challenge in our approach becomes controlling the size of the product instance. To achieve this, we affix to each instance a relatively complicated set of parameters (e.g., coding makespan, number of edges, number of sessions, etc.) and study how these parameters change

upon applying the graph product. Choosing the right set of parameters is key—they allow us to properly quantify the size escalation. In particular, we show that the coding gap grows doubly-exponentially and the size of the instances grow triply-exponentially, yielding the desired poly-logarithmic coding gap.

5.1.4 Related Work

This work ties in to many widely-studied questions. We outline some of the most relevant here.

Routing multiple unicasts. Minimizing the makespan of multiple unicasts using routing has been widely studied. When packets must be routed along fixed paths, two immediate lower bounds on the makespan emerge: *dilation*, the maximum length of a path, and *congestion*, the maximum number of paths crossing any single edge. A seminal result of Leighton et al. [99] proves one can route along such fixed paths in $O(\text{congestion}+\text{dilation})$ rounds, making the result optimal up to constants. Follow ups include works improving the constants in the above bound [119, 132], computing such protocols [100], simplifying the original proof [131], routing in distributed models [117, 124], and so on. When one has the freedom to choose paths, Bertsimas and Gamarnik [14] gave near-optimal routing solutions for asymptotically-large packet sizes, later extended to all packet sizes by Srinivasan and Teo [135]. The power of routing for multiple unicasts is therefore by now well understood.

Network coding gains. The utility of network coding became apparent after Ahlswede et al. [7] proved it can increase the (single multicast) throughput of a communication network. Following their seminal work, there emerged a vast literature displaying the advantages of network coding over routing for various measures of efficiency in numerous communication models, including for example energy usage in wireless networks [43, 57, 139], delay minimization in repeated single unicast[25, 138], and makespan in gossip protocols [28, 62, 63]. The throughput of a single multicast (i.e., one single node sending to some set of nodes), arguably the simplest non-trivial communication task, was also studied in great detail (e.g., [6, 7, 79, 83, 104, 105]). In particular, Agarwal and Charikar [6] showed that the throughput coding gap for a single multicast equals the integrality gap of natural min-weight Steiner tree relaxations, for which non-trivial bounds were known (see, e.g., [71, 142]). While the throughput coding gap for a single multicast is now fairly well understood, the case of multiple senders seems to be beyond the reach of current approaches.

Throughput gaps for multiple unicasts. The routing throughput for multiple unicasts is captured by multicommodity max-flow, while the coding throughput is clearly upper bounded by the sparsest cut. Known multicommodity flow-cut gap bounds therefore imply the throughput coding gap for k unicasts is at most $O(\log k)$ [11, 106], and less for special families of instances [22, 23, 24, 91, 94, 98, 127]. In 2004 Li and Li [103] and Harvey et al. [75] independently put forward the *multiple-unicast conjecture*, which asserts that the throughput coding

gap is trivial (i.e., it is one). This conjecture was proven true for numerous classes of instances [3, 81, 85, 93, 116]. More interestingly, a positive resolution of this conjecture has been shown to imply unconditional lower bounds in external memory algorithm complexity [3, 41], computation in the cell-probe model [3], and (recently) an $\Omega(n \log n)$ circuit size lower bound for multiplication of n-bit integers [5] (matching an even more recent breakthrough algorithmic result for this fundamental problem [74]). Given this last implication, it is perhaps not surprising that despite attempts by many prominent researchers [8, 76, 85, 97, 103, 140, 141], the conjecture remains open and has established itself as a notoriously hard open problem. Indeed, even improving the $O(\log k)$ upper bound on throughput coding gaps seems challenging, and would imply unconditional *super-linear circuit size lower bounds*, by the work of Afshani et al. [5]. Improving our upper bound on makespan coding gaps to $o(\log k)$ would directly imply a similar improvement for throughput coding gaps, together with these far-reaching implications.

5.2 Upper Bounding the Coding Gap

In this section we prove Theorem 5.1.2, upper bounding the makespan network coding gap. Given a multiple-unicast instance $\mathcal{M}$ we thus want to upper bound its routing makespan and lower bound its coding makespan. To characterize these quantities we start with a natural hop-bounded multicommodity flow LP, $\text{CONCURRENTFLOW}_{\mathcal{M}}(T)$, which serves as a "relaxation" of routing protocols of makespan at most T. The LP, given in Figure 5.3, requires sending a flow of magnitude $z \cdot d_i$ between each source-sink pair (s_i, t_i), with the additional constraints that (1) the combined congestion of any edge e is at most $T \cdot c_e$ where c_e is the capacity of the edge (as only c_e packets can use this edge during any of the T time steps of a routing protocol), and (2) the flow is composed of only *short* paths, of at most T hops. Specifically, for each $i \in [k]$, we only route flow from s_i to t_i along paths in $\mathcal{P}_i(T) \triangleq \{p : s_i \rightsquigarrow t_i \mid |p| \leq T, \, p \text{ is simple}\}$, the set of simple paths of hop-length at most T connecting s_i and t_i in G.

Primal: $\text{CONCURRENTFLOW}_{\mathcal{M}}(T)$		Dual: $\text{CUT}_{\mathcal{M}}(T)$	
maximize	z	minimize	$T \cdot \sum_{e \in E} c_e \ell_e$
subject to:		subject to:	
$\forall i \in [k]$:	$\sum_{p \in \mathcal{P}_i(T)} f_i(p) \geq z \cdot d_i$	$\forall i \in [k], p \in \mathcal{P}_i(T)$:	$\sum_{e \in p} \ell_e \geq h_i$
$\forall e \in E$:	$\sum_{p \ni e} f_i(p) \leq T \cdot c_e$		$\sum_{i \in [k]} d_i h_i \geq 1$
$\forall i \in [k], p$:	$f_i(p) \geq 0$	$\forall e \in E$:	$\ell_e \geq 0$
		$\forall i \in [k]$:	$h_i \geq 0$

Figure 5.3: The concurrent flow LP relaxation and its dual.

A routing protocol solving $\mathcal{M}$ in T rounds yields a solution to $\text{CONCURRENTFLOW}_{\mathcal{M}}(T)$ of value $z = 1$, almost by definition.[1] A partial converse is also true; a feasible solution to $\text{CONCURRENTFLOW}_{\mathcal{M}}(T)$ of value at least $\Omega(1)$ implies a routing protocol for $\mathcal{M}$ in time $O(T)$. This can be proven using standard LP rounding [135] and $O(\text{congestion + dilation})$ path routing [99]. (See Section 5.7.)

[1] Such a protocol must send d_i packets along paths of length at most T between each sour-sink pair (s_i, t_i), and it can send at most c_e packets through each edge e during any of the T rounds, or at most $c_e \cdot T$ packets overall.

Proposition 5.2.1. *Let $z, \{f_i(p) \mid i \in [k], p \in \mathcal{P}_i(T)\}$ be a feasible solution for* $\textsc{ConcurrentFlow}_{\mathcal{M}}(T)$. *Then there exists an integral routing protocol with makespan $O(T/z)$.*

Complementing the above, we show that a low optimal LP value for $\textsc{ConcurrentFlow}_{\mathcal{M}}(T)$ implies that no *coding* protocol can solve the instance in much less than T time.

Lemma 5.2.2. *If the optimal value of* $\textsc{ConcurrentFlow}_{\mathcal{M}}(T)$ *is at most $z^* \leq 1/10$, then the coding makespan for $\mathcal{M}$ is at least $T/(C \cdot \log(k) \cdot \log\left(\sum_i d_i / \min_i d_i\right))$ for some constant $C > 0$.*

Before outlining our approach for proving Lemma 5.2.2, we show why this lemma together with Proposition 5.2.1 implies our claimed upper bound for the makespan network coding gap.

Theorem 5.1.2. *The network coding gap for makespan of any k-unicast instance is at most*

$$O\left(\log(k) \cdot \log\left(\sum_i d_i / \min_i d_i\right)\right).$$

Proof. Fix a multiple-unicast instance $\mathcal{M}$. Let T^* be the minimum makespan for any coding protocol for $\mathcal{M}$. Let $T = (C+1) \cdot T^* \cdot (\log(k) \cdot \log(\sum_i d_i / \min_i d_i))$ for C as in Lemma 5.2.2. Then, the LP $\textsc{ConcurrentFlow}_{\mathcal{M}}(T)$ must have optimal value at least $z^* \geq 1/10$, else by Lemma 5.2.2 and our choice of T, any coding protocol has makespan at least $T/O(\log(k) \cdot \log(\sum_i d_i / \min_i d_i)) > T^*$, contradicting the definition of T^*. But then, by Proposition 5.2.1, there exists a routing protocol with makespan $O(T/z^*) = O(T^* \cdot \log(k) \cdot \log(\sum_i d_i / \min_i d_i)$. The theorem follows. □

The remainder of the section is dedicated to proving Lemma 5.2.2. That is, proving that a low optimal value for the LP $\textsc{ConcurrentFlow}_{\mathcal{M}}(T)$ implies a lower bound on the makespan of any coding protocol for $\mathcal{M}$. To this end, we take a low-valued LP solution to the dual LP $\textsc{Cut}_{\mathcal{M}}(T)$ (implied by strong LP duality) and use it to obtain an information-theoretic certificate of impossibility, which we refer to as a *moving cut*. Section 5.2.1 introduces a framework to prove such certificates of impossibility, which we show *completely characterizes* any instance's makespan (up to polylog terms). We then explain how to transform a low-value dual LP solution to such a moving cut in Section 5.2.2. For this transformation, we prove a lemma reminiscent of Arora et al. [9, Theorem 1] for general metrics, in Section 5.2.3.

5.2.1 Moving Cuts: Characterizing Makespan

In this section we prove that moving cuts characterize the makespan of a multiple unicast instance. For ease of reference, we re-state the definition of moving cuts.

Definition 5.1.6. *(Moving cut) Let $G = (V, E)$ be a communication network with capacities $c : E \to \mathbb{Z}_{\geq 1}$ and let $\{(s_i, t_i) \mid i \in [k]\}$ be source-sink pairs. A* ***moving cut*** *is an assignment*

*$\ell : E \to \mathbb{Z}_{\geq 1}$ of positive integer lengths to edges of G. We say the moving cut has **capacity** C, if $\sum_{e \in E} c_e(\ell_e - 1) = C$, and **distance** T, if all sinks and sources are at distance at least T with respect to ℓ; i.e., $\forall i, j \in [k]$ we have $d_\ell(s_i, t_j) \geq T$.*

We start by proving Lemma 5.1.7, whereby moving cuts with small capacity and large distance imply makespan lower bounds.

Lemma 5.1.7. *Let $\mathcal{M} = (G, \mathcal{S})$ be a unicast instance which admits a moving cut ℓ of capacity strictly less than $\sum_{i \in [k]} d_i$ and distance T. Then* any *(coding) protocol for $\mathcal{M}$ has makespan at least T.*

Proof. We will show via simulation that a protocol solving $\mathcal{M}$ in at most $T - 1$ rounds would be able to compress $\sum_{i=1}^{k} d_i$ random bits to a strictly smaller number of bits, thereby leading to a contradiction. Our simulation proceeds as follows. We have two players, Alice and Bob, who control different subsets of nodes. In particular, if we denote by $A_r \triangleq \{v \in V \mid \min_i \text{dist}_\ell(s_i, v) \leq r\}$ the set of nodes at distance at most r from any source, then during any round $r \in \{0, 1, \ldots, T - 1\}$ all nodes in A_r are "spectated" by Alice. By spectated we mean that Alice gets to see all of these nodes' private inputs and received transmissions during the first r rounds. Similarly, Bob, at time r, spectates $B_r \triangleq V \setminus A_r$. Consequently, if at round r a node $u \in V$ spectated by Alice sends a packet to a node $v \in V$, then Bob will see the contents of that packet if and only if Bob spectates the node v at round $r + 1$. That is, this happens only if $u \in A_r$ and $v \in B_{r+1} = V \setminus A_{r+1}$. Put otherwise, Bob can receive a packet from Alice along edge e during times $r \in [\min_i \text{dist}_\ell(s_i, u), \min_i \text{dist}_\ell(s_i, v) - 1]$. Therefore, the number of rounds transfer can happen along edge e is at most $\min_i \text{dist}_\ell(s_i, v) - \min_i \text{dist}_\ell(s_i, u) - 1 \leq \ell_e - 1$. Hence, the maximum number of bits transferred from Alice to Bob via e is $c_e(\ell_e - 1)$. Summing up over all edges, we see that the maximum number of bits Bob can ever receive during the simulation is at most $\sum_{e \in E} c_e(\ell_e - 1) < \sum_{i=1}^{k} d_i$. Now, suppose Alice has some $\sum_{i=1}^{k} d_i$ random bits. By simulating this protocol with each source s_i having (a different) d_i of these bits, we find that if all sinks receive their packet in T rounds, then Bob (who spectates all t_j at time $T - 1$, as $\min_i \text{dist}_\ell(s_i, t_j) \geq T$ for all j) learns all $\sum_{i=1}^{k} d_i$ random bits while receiving less than $\sum_{i=1}^{k} d_i$ bits from Alice—a contradiction. □

Lemma 5.1.7 suggests the following recipe for proving makespan lower bounds: Prove a lower bound on the makespan of some sub-instance $\mathcal{M}' = (G, \mathcal{S}')$ with $\mathcal{S}' \subseteq \mathcal{S}$ induced by indices $I \subseteq [k]$ using Lemma 5.1.7. As any protocol solving $\mathcal{M}$ solves $\mathcal{M}'$, a lower bound on the makespan of $\mathcal{M}'$ implies a lower bound on the makespan of $\mathcal{M}$. So, to prove makespan lower bounds for $\mathcal{M}$, identify a moving cut for some sub-instance of $\mathcal{M}$. If this moving cut has capacity less than the sum of demands of the sub-instance and distance at least T, then the entire instance, $\mathcal{M}$, has makespan at least T.

By the above discussion, the worst distance of a (low capacity) moving cut over any sub-instance serves as a lower bound on the makespan of any instance. The following lemma, whose proof is deferred to the end of Section 5.2.2, asserts that in fact, the highest distance of a moving cut over any sub-instance is equal (up to polylog terms) to the best routing makespan of $\mathcal{M}$.

Consequently, by Theorem 5.1.2, the strongest lower bound obtained using moving cuts is equal up to polylog terms to the optimal (coding) makespan for $\mathcal{M}$.

Lemma 5.2.3. *If a k-unicast instance $\mathcal{M}$ has no routing protocol with makespan T, then there exists a set of sessions $I \subseteq [k]$ with a moving cut of capacity strictly less than $\sum_{i \in I} d_i$ and distance at least $T/O(\log k \cdot (\sum_i d_i / \min_i d_i))$ with respect to the unicast sub-instance induced by I.*

We now turn to leveraging moving cuts to prove makespan lower bounds. Specifically, Lemma 5.2.2.

5.2.2 From Dual Solution to Moving Cut

Proposition 5.2.1 shows that high objective value for the primal LP, $\textsc{ConcurrentFlow}_{\mathcal{M}}(T)$ implies an upper bound on the routing time for the given instance. In this section we prove the "converse", Lemma 5.2.2, whereby low objective value of the primal LP implies a lower bound on the *coding* time for the given instance.

Our approach will be to prove that a low objective value of the primal LP —implying a feasible dual LP solution of low value—yields a moving cut for some sub-instance. This, by Lemma 5.1.7, implies a lower bound on protocols for this sub-instance, and thus for the entire instance, from which we obtain Lemma 5.2.2. We turn to converting a low-valued dual LP solution to such a desired moving cut.

By definition, a low-value feasible solution to the dual LP, $\textsc{Cut}_{\mathcal{M}}(T)$, assigns non-negative lengths $\ell : E \to \mathbb{R}_{\geq 0}$ such that (1) the c-weighted sum of ℓ-lengths is small, i.e., $\sum_{e \in E} c_e \ell_e = \widetilde{O}(1/T)$, as well as (2) if h_i is the ℓ-length of the T-hop-bounded ℓ-shortest path between s_i and t_i, then $\sum_{i \in [k]} d_i \cdot h_i \geq 1$. Property (1) implies that appropriately scaling the lengths ℓ yields a moving cut given by lengths $\tilde{\ell}$ of bounded capacity, $\sum_e c_e \tilde{\ell}_e$. For this moving cut to be effective to lower bound the makespan of some sub-instance using Lemma 5.1.7, the cut must have high distance w.r.t. this sub-instance. As a first step to this end, we use Property (2) to identify a subset of source-sink pairs $I \subseteq [k]$ with pairwise $\tilde{\ell}$-distance at least $\tilde{\Omega}(T)$. Claim 5.2.4, proven in Section 5.7 using a "continuous" bucketing argument, does just this. The claim introduces a loss factor α_{gap} that we use throughout this section.

Claim 5.2.4. *Given sequences $h_1, \ldots, h_k, d_1, \ldots, d_k \in \mathbb{R}_{\geq 0}$ with $\sum_{i \in [k]} d_i \cdot h_i \geq 1$ there exists a non-empty subset $I \subseteq [k]$ with $\min_{i \in I} h_i \geq \frac{1}{\alpha_{\text{gap}} \cdot \sum_{i \in I} d_i}$ for $\alpha_{\text{gap}} \in \left[1, O\left(\log \frac{\sum_{i \in [k]} d_i}{\min_{i \in [k]} d_i}\right)\right]$.*

Scaling up the ℓ lengths yields a sub-instance induced by pairs $I \subseteq [k]$ and moving cut with bounded capacity and with $\tilde{\ell}$-distance between every source and its sink of at least $\tilde{\Omega}(T)$, i.e., $d_{\tilde{\ell}}(s_i, t_i) \geq \tilde{\Omega}(T)$ for all $i \in I$. However, Lemma 5.1.7 requires $\tilde{\ell}$-distance $\tilde{\Omega}(T)$ between *any* source and sink, i.e., $d_{\tilde{\ell}}(s_i, t_j) \geq \tilde{\Omega}(T)$ for all $i, j \in I$. To find a subset of source-sink pairs with such distance guarantees, we rely on the following metric decomposition lemma, whose proof is deferred to Section 5.2.3.

Lemma 5.2.5. *Let (X, d) be a metric space. Given n pairs $\{(s_i, t_i)\}_{i\in[n]}$ of points in X with at most k distinct points in $\bigcup_i \{s_i, t_i\}$ and pairwise distances at least $d(s_i, t_i) \geq T$, there exists a subset of indices $I \subseteq [n]$ of size $|I| \geq \frac{n}{9}$ such that $d(s_i, t_j) \geq \frac{T}{O(\log k)}$ for all $i, j \in I$. Moreover, such a set can be computed in polynomial time.*

We are now ready to construct the moving cut.

Lemma 5.2.6. *If the optimal value of $\textsc{ConcurrentFlow}_{\mathcal{M}}(T)$ is at most $z^* \leq 1/10$, then there exists $\tilde{I} \subseteq [k]$ and a moving cut $\tilde{\ell}$ of capacity strictly less than $\sum_{i\in\tilde{I}} d_i$ and distance at least $T/O(\alpha_{\text{gap}} \log k)$ with respect to the unicast sub-instance induced by $\tilde{I}$ (i.e., $\tilde{\mathcal{M}} = (G, \tilde{\mathcal{S}})$ where $\tilde{\mathcal{S}} = \{(s_i, t_i, d_i)\}_{i\in\tilde{I}}$).*

Proof. By strong duality, the dual LP $\textsc{Cut}_{\mathcal{M}}(T)$ has a feasible solution $\{h_i, \ell_e \mid i \in [k], e \in E\}$ to $\textsc{Cut}_{\mathcal{M}}(T)$ with objective value $T\sum_{e\in E} c_e\ell_e = z^*$. Fix such a solution. Let $I \subseteq [k]$ be a subset of indices as guaranteed by Claim 5.2.4. Define $\tilde{\ell}_e \triangleq 1 + \lfloor \ell_e \cdot T \cdot \sum_{i\in I} d_i \rfloor$ for all $e \in E$ and note that $\tilde{\ell}_e \in \mathbb{Z}_{\geq 1}$. By definition of $\tilde{\ell}$ and $T\sum_e c_e\ell_e = z$, we get a bound on the capacity of $\tilde{\ell}$:

$$\sum_{e\in E} c_e(\tilde{\ell}_e - 1) \leq \sum_e c_e\ell_e \cdot T \cdot \sum_{i\in I} d_i = z^* \cdot \sum_{i\in I} d_i \leq \frac{1}{10}\sum_{i\in I} d_i < \frac{1}{9}\sum_{i\in I} d_i.$$

We now show that $d_{\tilde{\ell}}(s_i, t_i) > T/\alpha_{\text{gap}}$ for all $i \in I$. Consider any simple path p between $s_i \rightsquigarrow t_i$. Denote by $\tilde{\ell}(p)$ and $\ell(p)$ the length with respect to $\tilde{\ell}$ and ℓ, respectively. It is sufficient to show that $\tilde{\ell}(p) > T/\alpha_{\text{gap}}$. If $p \notin \mathcal{P}_i(T)$, i.e., the hop-length of p (denoted by $|p|$) is more than T. Then $\tilde{\ell}(p) \geq |p| > T \geq T/\alpha_{\text{gap}}$, since $\tilde{\ell}_e \geq 1\ \forall e \in E$. Conversely, if $p \in \mathcal{P}_i(T)$, then by our choice of I as in Claim 5.2.4 and the definition of h_i, we have that $\ell(p) \geq h_i \geq \frac{1}{\alpha_{\text{gap}}\sum_{i\in I} d_i}$, hence

$$\tilde{\ell}(p) \geq \ell(p) \cdot T \cdot \sum_{i\in I} d_i = \frac{1}{\alpha_{\text{gap}}\sum_{i\in I} d_i} \cdot T \cdot \sum_{i\in I} d_i = T/\alpha_{\text{gap}}.$$

Finally, we choose a subset $\tilde{I} \subseteq I$ s.t. $d_{\tilde{\ell}}(s_i, t_j) > T/O(\alpha_{\text{gap}} \log k)$ for all $i, j \in \tilde{I}$. By Lemma 5.2.5 applied to the graphic metric defined by $\tilde{\ell}$ and each pair (s_i, t_i) repeated d_i times, there exists a multiset of indices $\tilde{I} \subseteq I \subseteq [k]$ such that $d_{\tilde{\ell}}(s_i, t_j) \geq T/O(\alpha_{\text{gap}} \log k)$ for all $i, j \in \tilde{I}$ and such that $|\tilde{I}| \geq \sum_i d_i/9$. Therefore, taking each pair (s_i, t_i) indexed by $\tilde{I}$ at least once, we find a subset of sessions $\tilde{I} \subseteq [k]$ such that $\sum_{i\in\tilde{I}} d_i \geq \frac{1}{9}\sum_{i\in[k]} d_i > \sum_e c_e \cdot (\tilde{\ell}_e - 1)$ and $d_{\tilde{\ell}}(s_i, t_j) \geq T/O(\alpha_{\text{gap}} \log k)$ for all $i, j \in \tilde{I}$. In other words, $\tilde{\ell}$ is a moving cut of capacity strictly less than $\sum_{i\in\tilde{I}} d_i$ and distance $T/O(\alpha_{\text{gap}} \log k)$ with respect to the sub-instance induced by $\tilde{I}$. □

Combining Lemma 5.2.6 with Lemma 5.1.7, we obtain this section's main result, Lemma 5.2.2.

Lemma 5.2.2. *If the optimal value of* $\textsc{ConcurrentFlow}_{\mathcal{M}}(T)$ *is at most* $z^* \leq 1/10$*, then the coding makespan for* $\mathcal{M}$ *is at least* $T/(C \cdot \log(k) \cdot \log\left(\sum_i d_i / \min_i d_i\right))$ *for some constant* $C > 0$*.*

Remark 1. We note that the $\log k$ term in Lemma 5.2.2's bound is due to the $\log k$ term in the bound of Lemma 5.2.5. For many topologies, including genus-bounded and minor-free networks, this $\log k$ term can be replaced by a constant (see Section 5.2.3), implying smaller makespan gaps for such networks.

Remark 2. Lemma 5.2.3, which states that a lower bound on routing makespan implies the existence of a moving cut of high distance with respect to some sub-instance, follows by Lemma 5.2.6 and Proposition 5.2.1, as follows. By Proposition 5.2.1, $\mathcal{M}$ having no routing protocol with makespan T implies that for some constant $c > 0$, the LP $\textsc{ConcurrentFlow}_{\mathcal{M}}(c \cdot T)$ has objective value at most $1/10$. Lemma 5.2.6 then implies the existence of the moving cut claimed by Lemma 5.2.3.

5.2.3 From Pairwise to All-Pairs Distances

This section is dedicated to a discussion and proof of the following Lemma that seems potentially useful beyond the scope of this chapter.

Lemma 5.2.5. *Let* (X, d) *be a metric space. Given* n *pairs* $\{(s_i, t_i)\}_{i \in [n]}$ *of points in* X *with at most* k *distinct points in* $\bigcup_i \{s_i, t_i\}$ *and pairwise distances at least* $d(s_i, t_i) \geq T$*, there exists a subset of indices* $I \subseteq [n]$ *of size* $|I| \geq \frac{n}{9}$ *such that* $d(s_i, t_j) \geq \frac{T}{O(\log k)}$ *for all* $i, j \in I$*. Moreover, such a set can be computed in polynomial time.*

We note that the above lemma is similar to the main Theorem of Arora et al. [9]. Our result holds for general metrics with a factor of $O(\log k)$ in the distance loss, while their holds for ℓ_2^2 metrics with a factor of $O(\sqrt{\log k})$. The results are incomparable and both are tight. (The tightness of Lemma 5.2.5 can be shown to be tight for graph metrics, for example in graph metrics of constant-degree expanders.)

To prove Lemma 5.2.5 we rely on *padded decompositions* [61]. To define these, we introduce some section-specific notation. Let (X, dist) be a metric space. Let the (weak) diameter of a set of points $U \subseteq X$ be denoted by $\text{diam}(U) \triangleq \max_{x,y \in U} \text{dist}(x, y)$. We say a partition $P = \{X_1, X_2, \ldots, X_t\}$ of X is Δ-*bounded* if $\text{diam}(X_i) \leq \Delta$ for all i. Next, for $U \subseteq X$ and a partition P as above, we denote by $U \subseteq P$ the event that there exists a part $X_i \in P$ containing U in its entirety; i.e., $U \subseteq X_i$. Let $B(x, \rho) \triangleq \{y \in X \mid \text{dist}(x, y) \leq \rho\}$ denote the ball of radius $\rho \geq 0$ around $x \in X$.

Definition 5.2.7. *Let $(X, dist)$ be a metric space. We say that a distribution $\mathcal{P}$ over Δ-bounded partitions of X is* (β, Δ)-padded *if, for some universal constant δ, it holds that for every $x \in X$ and $0 \leq \gamma \leq \delta$,*

$$\Pr_{P \sim \mathcal{P}}[B(x, \gamma\Delta) \not\subseteq P] \leq \beta\gamma.$$

In words, each part of the partition has diameter at most Δ and the probability of any point x in the metric being at distance less than $\gamma\Delta$ from a different part than its own part is at most $\beta\gamma$. Such decompositions were presented, for example, by Gupta et al. [61].

Lemma 5.2.8 ([61])**.** *Any metric $(X, dist)$ on k points admits a (β, Δ)-padded decomposition, for any $\Delta > 0$ and some $\beta = O(\log k)$. Such a decomposition can be computed in polynomial time.*

We are now ready to prove Lemma 5.2.5.

Proof of Lemma 5.2.5. First note that we can focus on the metric space induced by the k distinct points. Let $\mathcal{P}$ be a Δ-bounded β-padded decomposition with $\Delta = T - 1$ and $\beta = O(\log k)$. We first note that for all $i \in [k]$, s_i and t_i are contained in different parts since the diameter of each part X_i is at most $\Delta = T - 1$ and $\text{dist}(s_i, t_i) \geq T$. Furthermore, letting $\gamma = \frac{1}{2\beta}$, we have that $\Pr[B(s_i, \gamma\Delta) \subseteq P] \geq \frac{1}{2}$. Let $I' \subseteq [n]$ be the subset of indices i with $B(s_i, \gamma\Delta) \subseteq P$. Then we have $\Pr[i \in I'] \geq \frac{1}{2}$ for all $i \in [n]$.

Flip a fair and independent coin for each part in P. Let $U \subseteq X$ be the set of points in parts whose coin came out heads, and $V \subseteq X$ be the analogous set for tails. Then for each $i \in I'$ we have that $\Pr[s_i \in U \text{ and } t_i \in V] = \frac{1}{4}$. Let $I \subseteq I'$ be the subset of indices i with $s_i \in U$ and $t_i \in V$, giving $\Pr[i \in I] = \Pr[i \in I'] \cdot \Pr[i \in I \mid i \in I'] = \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8}\ \forall i \in [n]$. We also have that $\text{dist}(s_i, t_j) > \rho = \frac{T-1}{2\beta}$ for all $i, j \in I$, since $B(s_i, \rho) \subseteq U$ for all $i \in I \subseteq I'$ and $\{t_j\}_{j \in I} \cap U = \emptyset$. Therefore, this random process yields a subset of indices $I \subseteq [n]$ such that $\text{dist}(s_i, t_j) > \frac{T-1}{2\beta}$ for all $i, j \in I$, of expected size at least $\mathbb{E}[|I|] \geq \sum_{i \in [n]} \Pr[i \in I] \geq \frac{n}{8}$. As $n - \mathbb{E}[|I|]$ is a non-negative random variable, Markov's inequality implies that with constant probability $n - |I| \leq \frac{64}{63} \cdot (n - \mathbb{E}[|I|]) \leq \frac{8n}{9}$. The lemma follows. □

Remark: The $O(\log k)$ term in Lemma 5.2.5's bound is precisely the smallest possible β for which (β, Δ)-padded decompositions of the metric exist. For many graphic metrics, such as those of minor-excluded, bounded-genus, and bounded-doubling-dimension networks, padded decompositions with smaller β exist [2, 61, 98]. This improves the bounds of Lemma 5.2.5 and thus Lemma 5.2.2 by $(\log k)/\beta$, implying the same improvement for our makespan coding gaps for such networks.

5.3 Chapter Appendix: Polylogarithmic Coding Gap Instances

In this section we construct a family of multiple-unicast instances with polylogarithmic makespan coding gap. More precisely, we construct instances where the coding gap is at least $(5/3)^{2^i}$ and the size (both the number of edges and sessions) is bounded by $2^{2^{O(2^i)}}$. Here we give a bird's eye view of the construction and leave the details to subsequent subsections. We clarify that all big-O bounds like $f = O(g)$ mean there exists a universal constant $c > 0$ s.t. $f \leq c \cdot g$ for all admissible values (in particular, there is no assumption on f or g being large enough).

We use the graph product of [18] as our main tool. Given two multiple-unicast instances I_1, I_2 (called the *outer* and *inner instance*, resp.) we create a new instance I_+ where the coding gap is the product of the coding gaps of I_1 and I_2. The product is guided by a *colored bipartite graph* $B = (V_1, V_2, E)$ where each edge is labeled by $(\chi_1, \chi_2) =$ (edge in I_1, session in I_2). Precisely, we create $|V_1|$ copies of I_1, $|V_2|$ copies of I_2 and for each edge $(a, b) \in E(B)$ with label (χ_1, χ_2) we replace the edge χ_1 in the a^{th} copy of I_1 with session χ_2 in the b^{th} copy of I_2.

To prove a lower bound on the coding gap, one needs to upper bound the coding makespan and lower bound the coding makespan. The former is easy: the coding protocols nicely compose. The latter, however, is more involved. Our main tool is Lemma 5.1.4, which necessitates (i) keeping track of *cut edges* F along each instance I such that all source-sink pairs of I are well-separated after edges in F are deleted, and (ii) keeping the ratio $r \triangleq \frac{k}{|F|}$, number of sessions to cut edges, high. We must ensure that the properties are conserved in the product instance I_+. For (i), i.e., to disallow any short paths from forming as an unexpected consequence of the graph product, we choose B to have *high girth*. Also, we replace edges F in the outer instances with paths rather than connecting them to a session in the inner instance. Issue (ii) is somewhat more algebraically involved but boils down to ensuring that the ratio of sessions to cut edges (i.e., r) in the inner instance is comparable to the size (i.e., number of edges) of the outer instance itself. Note that makes the size of the outer instance I_1 insignificant when compared to the size of the inner instance I_2.

We recursively define a family of instances by parametrizing them with a "level" $i \geq 0$ and a lower bound on the aforementioned ratio r, denoting them by $I(i, r)$. We start for $i = 0$ with the $5/3$ instance of Figure 5.1 where we can control the ratio the aforementioned ratio r by changing the number of sessions k (at the expense of increasing the size). Subsequently, an instance on level i is defined as a product two of level $i-1$ instances with appropriately chosen parameters r. One can show that the coding makespan for a level i instance is at most 5^{2^i} and routing makespan is at least 3^{2^i}, hence giving a coding gap of $(5/3)^{2^i}$. Furthermore, we show that the size of $I(i, r)$ is upper bounded by $r^{2^{O(2^i)}}$, giving us the full result.

Finally, we note an important optimization to our construction and specify in more detail how $I(i, r)$ is defined. Specifically, it is defined as the product of $I_1 \triangleq I(i-1, 3r)$ being the outer instance and $I_2 \triangleq I(i-1, m_1/f_1)$ being the inner instance, where m_1 and f_1 are the number of total and cut edges of I_1. This necessitates the introduction and tracking of another parameter $u \triangleq m/f$ to guide the construction. We remark that this might be necessary since if one uses a looser construction of $I_2 \triangleq I(i-1, m_1)$ the end result $I(i, r)$ would be of size $r^{2^{O(i \cdot 2^i)}}$ and give

a coding gap of $\exp\left(\frac{\log\log k}{\log\log\log k}\right)$, just shy of a polylogarithm.

5.3.1 Gap Instances and Their Parameters

In this section we formally define the set of instance parameters we will track when combining the instances.

A **gap instance** $I = (G, \mathcal{S}, F)$ is a multiple-unicast instance $\mathcal{M} = (G, \mathcal{S})$ over a connected graph G, along with an associated set of **cut edges** $F \subseteq E(G)$. We only consider gap instances where the set of terminals is disjoint, i.e., $s_i \neq s_j, s_i \neq t_j, t_i \neq t_j$ for all $i \neq j$. Furthermore, edge capacities and demands are one; i.e., $c_e = 1\ \forall e \in E(G)$, and $d_i = 1\ \forall (s_i, t_i, d_i) \in \mathcal{S}$. A gap instance $I = (G, \mathcal{S}, F)$ has **parameters** (a, b, f, k, m, r, u) when:

- $\mathcal{M}$ admits a network coding protocol with makespan at most a.
- Let $\text{dist}_{G\setminus F}(\cdot,\cdot)$ be the hop-distance in G after removing all the cut edges F. Then for all terminals $i \in [k]$ we have that $\text{dist}_{G\setminus F}(s_i, t_i) \geq b$.
- The number of cut edges is $f = |F|$.
- The number of sessions is $k = |\mathcal{S}|$.
- The graph G has at most m edges; i.e., $|E(G)| \leq m$.
- r is a lower bound on the ratio between number of sessions and cut edges; i.e., $k/f \geq r$.
- u is an upper bound on the ratio between number of total edges and cut edges; i.e., $m/f \leq u$.

We note that the parameters of a gap instance immediately imply a lower bound on the optimal routing makespan via Lemma 5.1.4. Indeed, all packets transmitted in the first $b-1$ rounds must pass through F, and thus at most $f \cdot (b-1)$ packets can be sent between any source and its sink in the first $b-1$ rounds, implying that under any routing protocol, most sessions have completion time at least b.

Observation 5.3.1. *Let I be a gap instance with parameters (a, b, f, k, m, r, u) and $b \leq r$. Then the routing makespan for $(G, \mathcal{S})$ is at least b. Moreover, for any routing protocol of I, at least $k \cdot (1 - \frac{b-1}{r})$ sessions have completion time at least b.*

As an application of the above observation, we obtain another proof of the lower bound of the routing makespan for the family of instances of Figure 5.1. More generally, letting the cut edges be the singleton $F = \{(S, T)\}$, we obtain a family of gap instances with the following parameters.

Fact 5.3.2. *The family of gap instances of Figure 5.1 have parameters $(3, 5, 1, k, \theta(k^2), k, \theta(k^2))$ for $k \geq 5$.*

The above family of gap instances will serve as our base gap instances in a recursive construction which we describe in the following section.

5.3.2 Graph Product of Two Gap Instances

In this section we present the graph product that combines two instances to obtain one a with higher coding gap.

Definition 5.3.3. ***Colored bipartite graphs*** *are families of bipartite graphs $\mathcal{B}(n_1, n_2, m, k, g)$. Graphs $B = (V_1, V_2, E) \in \mathcal{B}(n_1, n_2, m, k, g)$ are bipartite graphs with $|V_1| = n_1$ (resp. $|V_2| = n_2$) nodes on the left (resp., right), each of degree m (resp., k), and these graphs have girth at least g. In addition, edges of B are colored using two color functions,* ***edge color*** *$\chi_1 : E(B) \to [m]$ and* ***session color*** *$\chi_2 : E(B) \to [k]$, which satisfy the following.*

- *$\forall v \in V_1$, the edge colors of incident edges form a complete set $\{\chi_1(e) \mid e \ni v\} = [m]$.*
- *$\forall v \in V_2$, the session colors of incident edges form a complete set $\{\chi_2(e) \mid e \ni v\} = [k]$.*
- *$\forall v \in V_1$, the session colors of incident edges are unique $|\{\chi_2(e) \mid e \ni v\}| = 1$.*
- *$\forall v \in V_2$, the edge colors of incident edges are unique, i.e, $|\{\chi_1(e) \mid e \ni v\}| = 1$.*

The size of the colored bipartite graphs will determine the size of the derived gap instance obtained by performing the product along a colored bipartite graph. The following gives a concrete bound on the size and, in turn, allows us to control the growth of the gap instances obtained this way.

Lemma 5.3.4 ([18])**.** *$\forall r, m, g \geq 3$, there exists a colored bipartite graph $B \in \mathcal{B}(n_1, n_2, m, k, 2g)$ with $n_1, n_2 \leq (9mk)^{g+3}$.*

Performing the product along a colored bipartite graph. Having defined colored bipartite graphs, we are now ready to define the graph product of I_1 and I_2 along B.

For $i \in \{1, 2\}$ let $I_i = (G_i, \mathcal{S}_i, F_i)$ be a gap instance with parameters $(a_i, b_i, f_i, k_i, m_i, r_i, u_i)$ and let $B \in \mathcal{B}(n_1, n_2, 2(m_1 - f_1), k_2, g)$ be a colored bipartite graph with girth $g \triangleq 2b_1b_2$. We call I_1 the **outer instance** and I_2 the **inner instance**. Denote the **product** gap **instance** $I_+ \triangleq T(I_1, I_2, B)$ by the following procedure:

- Replace each non-cut edge $e = \{u, v\} \in E(G_1) \setminus F_1$ with two anti-parallel arcs $\vec{e} = (u, v), \overleftarrow{e} = (v, u)$ and let $\vec{E}(G_1) = \{\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_{2(m_1 - f_1)}\}$ be the set of all such arcs.
- Construct n_1 copies of $(V(G_1), \vec{E}(G_1))$ and n_2 copies of G_2. Label the i^{th} copy as $G_1^{(i)}$ and $G_2^{(i)}$.
- Every cut edge $e \in F_1$ and every $i \in [n_1]$ replace edge e in $G_1^{(i)}$ by a path of length a_2 with the same endpoints. Let $f_e^{(i)}$ be an arbitrary edge on this replacement path.
- For every $(i, j) \in E(B)$ where $i \in [n_1], j \in [n_2]$ with edge color χ_1 and session color χ_2 do the following. Let $\vec{e}_{\chi_1}^{(i)} = (x, y)$ be the $\chi_1{}^{\text{th}}$ arc in $G_1^{(i)}$ and let (s, t) be the $\chi_2{}^{\text{th}}$ terminal pair in $G_2^{(j)}$. Merge x with s and y with t; delete $\vec{e}_{\chi_1}^{(i)} = (x, y)$ from $G_1^{(i)}$.

- For each session in the outer instance $(s_i, t_i, d_i = 1) \in \mathcal{S}_1$ add a new session $(s_i^{(j)}, t_i^{(j)}, 1)$ in $G_1^{(j)}$ for $j \in [n_1]$ to the product instance.
- The cut edges F_+ in the product instance I_+ consist of the the union of the following: (i) *one* arbitrary (for concreteness, first one) edge from all of the a_2-length paths that replaced cut edges in $G_1^{(i)}$, i.e., $\{f_e^{(i)} \mid e \in F_1, i \in [n_1]\}$, and (ii) all cut edges in copies of G_2, i.e., $\{e^{(i)} \mid e \in F_2, i \in [n_2]\}$.

We now give bounds on how the parameters change after combining two instances. First, we note that by composing network coding protocols for I_1 and I_2 in the natural way yields a network coding protocol whose makespan is at most the product of these protocols' respective makespans.

Lemma 5.3.5. *(Coding makespan) The product instance I_+ admits a network coding protocol with makespan at most $a_1 a_2$.*

Less obviously, we show that if we choose a large enough girth g for the colored bipartite graph, we have that the b parameter of the obtained product graph is at least the product of the corresponding b parameters of the inner and outer instances.

Lemma 5.3.6 (Routing makespan)**.** *Let $I_+ = (G_+, \mathcal{S}_+, F_+)$ be the product instance using a colored bipartite graph B of girth $g \triangleq 2b_1b_2$ and let $dist_{G_+ \setminus F_+}(\cdot, \cdot)$ be the hop-distance in G_+ with all the edges of F_+ deleted. We have that $dist_{G_+ \setminus F_+}(s_i, t_i) \geq \min(b_1b_2, \frac{g}{2}) = b_1b_2$ for all $(s_i, t_i, d_i) \in \mathcal{S}_+$.*

Proof. Let p be a path in $G_+ \setminus F_+$ between some terminals $s_i \rightsquigarrow t_i$ that has the smallest hop-length among all $(s_i, t_i, d_i) \in \mathcal{S}_+$. We want to show that $|p| \geq \min(b_1b_2, \frac{g}{2})$.

First, let q be the path in the colored bipartite graph B that corresponds to p. There are some technical issues with defining q since merging vertices in the graph product has the consequence that some $v \in V(G_+)$ belong to multiple nodes $V(B)$. To formally specify q, we use the following equivalent rephrasing of the graph product that will generate an "expanded instance" G'_+. Instead of "merging" two vertices u, v as in G_+, connect then with an edge e of hop-length $h(e) = 0$ and add e to G'_+. Edges from G_+ have hop-length $h(e) = 1$ and are analogously added to G'_+. The path p can be equivalently specified as the path between $s_i \rightsquigarrow t_i$ in $G'_+ \setminus F_+$ that minimizes the distance $\text{dist}_h(s_i, t_i)$. Now, each vertex $V(G'_+)$ belongs to exactly one vertex $V(B)$, hence the path q in B corresponding to p is well-defined. Note that p is a closed path in G_+ and q is a closed path in B.

Suppose that q spans a non-degenerate cycle in B. Then $|p| \geq \frac{|q|}{2} \geq \frac{g}{2}$, where the last inequality $|q| \geq g$ is due to the girth of G. The first inequality $|p| \geq \frac{|q|}{2}$ is due to the fact that when q enters a node $v \in V_2(B)$, a node representing an inner instance, the corresponding path p had to traverse at least one inner instance edge before its exit since the set of terminals is disjoint and a path can enter/exit inner instances only in terminals.

Suppose now that q does not span a cycle in B, therefore the set of edges in q span a tree $\mathcal{T}$ in B and q is simply the (rotation of the unique) Eulerian cycle of that tree. Notation-wise, let $v \in V_1(B)$ be the node in the colored bipartite graph B that contains the critical terminals s_i and

t_i and suppose that $\mathcal{T}$ is rooted in v. If the depth of $\mathcal{T}$ is 1 (i.e., consists only of $v \in V_1(B)$ and direct children $w_1, \ldots, w_t \in V_2(B)$), then p must correspond to a $s_i \rightsquigarrow t_i$ walk in v, where each (non-cut) edge traversal is achieved by a non-cut walk in the inner instance w_j between a set of inner terminals. Note that every $s_i \rightsquigarrow t_i$ non-cut walk has hop-length at least b_1 and each non-cut walk in the inner instance has hop-length at least b_2, for a cumulative $b_1 \cdot b_2$.

Finally, suppose that $\mathcal{T}$ has depth more than 1, therefore there exists two $v, w \in V(\mathcal{T})$ and $v, w \in V_1(B)$. Since $\mathcal{T}$ is traversed via an Eulerian cycle, the path p passes through two terminals of $(s_j, t_j, \cdot) \in \mathcal{S}_+$. Let p' be the natural part of p going from $s_j \rightsquigarrow t_j$, e.g., obtained by clipping the path corresponding to the subtree of q in $\mathcal{S}$. Furthermore, let p'' be the part of the p connecting v and w and is disjoint from p''. From the last paragraph we know that $|p''| \geq 1$ since it passes through at least one $u \in V_2(B)$. Also, by minimality of $s_i \rightsquigarrow t_i$ we have that $\text{dist}_h(s_j, t_j) \geq \text{dist}_h(s_i, t_i)$. Now we have a contradiction since $\text{dist}_h(s_i, t_i) = |p| \geq |p'| + |p''| \geq 1 + \text{dist}_h(s_j, t_j)$. □

Combining Lemmas 5.3.5 and 5.3.6 together with some simple calculations (deferred to Section 5.8), we find that the product instance is a gap instance with the following parameters.

Lemma 5.3.7. *For $i \in \{1, 2\}$ let $I_i = (G_i, \mathcal{S}_i, F_i)$ be a gap instance with parameters $(a_i, b_i, f_i, k_i, m_i, r_i, u_i)$ with $\frac{m_i}{f_i} \geq 2$ and $a_i \geq 2$; let $B \in \mathcal{B}(n_1, n_2, 2(m_1 - f_1), k_2, 2b_1b_2)$ be a colored bipartite graph. Then $I_+ \triangleq T(G_1, G_2, B)$ is a gap instance with parameters $a_+ \triangleq a_1a_2$, $b_+ \triangleq b_1b_2$, $f_+ \triangleq n_1f_1 + n_2f_2$, $k_+ \triangleq n_1k_1$, $m_+ \triangleq a_2n_1f_1 + n_2m_2$, $r_+ \triangleq r_1 \frac{1}{1+2u_1/r_2}$, $u_+ \triangleq u_2 \frac{1+a_2/2}{1+r_2/(2u_1)}$. Moreover, $\frac{m_+}{f_+} \geq 2$ and $a_+ \geq 2$.*

5.3.3 Iterating the Graph Product

Having bounded the parameters obtained by combining two gap instances, we are now ready to define a recursive family of gap instances from which we obtain our polylogarithmic makespan network coding gap.

Definition 5.3.8. *We recursively define a collection of gap instances $(I(i, r))_{i \geq 0, r \geq 5}$, and denote its parameters by $(a_{i,r}, b_{i,r}, f_{i,r}, k_{i,r}, m_{i,r}, r_{i,r}, u_{i,r})$. For the base case, we let $I(0, r)$ be the gap instance of Fact 5.3.2 with parameters $(3, 5, 1, r, \theta(r^2), r, \theta(r^2))$. For $i + 1 > 0$ we define $I(i + 1, r) \triangleq T(I_1, I_2)$. Here, $I_1 \triangleq I(i, 3r)$ and $I_2 \triangleq I(i, u_{i,3r})$, with parameters $(a_1, \ldots, u_1)$ and $(a_2, \ldots, u_2)$, respectively.*

In other words, I_1 is defined such that $r_1 = 3r_+$ and I_2 such that $r_2 = u_1$. In Section 5.8 we study the growth of the parameters of our gap instance families. Two parameters that are easy to bound for this construction are the following.

Observation 5.3.9. *For any $i \geq 0$ and $r \geq 5$, we have $a_{i,r} = 3^{2^i}$ and $b_{i,r} = 5^{2^i}$.*

A less immediate bound, whose proof is also deferred to Section 5.8, is the following bound on the number of edges of the gap instances..

Lemma 5.3.10. *We have that* $\log m_{i,r} \leq 2^{O(2^i)} \log r$ *for all* $i \geq 0, r \geq 5$.

5.3.4 Lower Bounding the Coding Gap

We are now ready to prove this section's main result – a polylog(k) makespan coding gap.

Theorem 5.1.3. *There exists an absolute constant* $c > 0$ *and an infinite family of* k*-unicast instances whose makespan coding gap is at least* $\Omega(\log^c k)$.

Proof. For each $i \geq 0$ and $r \triangleq 5^{2^i}$, consider $I_{i,r}$ as defined above. By Observation 5.3.9 this gap instance has coding makespan at most $a_{i,r} = 3^{2^i}$. Moreover, also by Observation 5.3.9, this instance has $b_{i,r} = 5^{2^i}$, and so by Observation 5.3.1 its routing makespan is at least 5^{2^i}. Hence the makespan coding gap of $I_{i,r}$ is at least $(5/3)^{2^i}$. It remains to bound this gap in terms of $k \triangleq k_{i.r}$.

As the terminals of $I_{i,r}$ are disjoint, we have that k is upper bounded by the number of nodes of $I_{i,r}$, which is in turn upper bounded by $m_{i,r}$, as $I_{i,r}$ is connected and not acyclic. That is, $k \leq m_{i,r}$. But by Lemma 5.3.10, we have that $\log m_{i,r} \leq 2^{O(2^i)} \cdot \log r = 2^{O(2^i)} \cdot O(2^i) \leq 2^{O(2^i)} \leq 2^{c' \cdot 2^i}$, for some universal constant $c' > 0$. Therefore, stated in terms of k, the makespan coding gap is at least

$$(5/3)^{2^i} = 2^{2^i \log 5/3} = (2^{c' 2^i})^{\frac{\log 5/3}{c'}} \geq (\log m_{i,r})^c \geq \log^c k,$$

where $c \triangleq \frac{\log 5/3}{c'} > 0$ is a universal constant, as claimed. □

5.4 Coding Gaps for Other Functions of Completion Times

In this section we extend our coding gap results to other time complexity measures besides the makespan. For our upper bounds, we show that our coding gaps for ℓ_∞ minimization of the completion times (makespan) implies similar bounds for a wide variety of functions, including all weighted ℓ_p norms; proving in a sense that ℓ_∞ is the "hardest" norm to bound. The following lemma underlies this connection.

Lemma 5.4.1. *Let* α *be an upper bound on the coding gap for completion times'* ℓ_∞ *norm (makespan). Then, if multiple-unicast instance* $\mathcal{M}$ *admits a coding protocol with completion times* $(T_1, T_2, \ldots, T_k)$*, there exists a routing protocol for* $\mathcal{M}$ *with completion times placewise at most* $(4\alpha \cdot T_1, 4\alpha \cdot T_2, \ldots, 4\alpha \cdot T_k)$.

Proof. Let $\mathcal{M}$ be a multiple-unicast instance. Let $(T_1, T_2, \ldots, T_k)$ be the vector of completion times of some coding protocol. Without loss of generality, assume $T_1 \leq T_2 \leq \cdots \leq T_k$. Next,

for any $j \in \mathbb{Z}$, denote by $\mathcal{M}_j$ the sub-instance of $\mathcal{M}$ induced by the unicasts with completion time $T_i \in [2^j, 2^{j+1})$. Then, there exists a network coding protocol for each $\mathcal{M}_j$ with makespan at most 2^{j+1}. Consequently, there exists a routing protocol for $\mathcal{M}_j$ with makespan at most $\alpha \cdot 2^{j+1}$. Scheduling these protocols in parallel, in order of increasing $j = 0, 1, 2, \ldots$, we find that a unicast with completion time $T_i \in [2^j, 2^{j+1})$ in the optimal coding protocol has completion time in the obtained routing protocol which is at most

$$\sum_{j' \leq j} \alpha \cdot 2^{j'+1} \leq 2\alpha \cdot 2^{j+1} \leq 4\alpha \cdot T_i. \qquad \square$$

Note that unlike our routing protocols for makespan minimization of Theorem 5.1.2, the proof here is non-constructive, as it assumes (approximate) knowledge of the completion times of each unicast in the optimal coding protocol. Nonetheless, this proof guarantees the existence of a protocol, which suffices for our needs. In particular, applying Lemma 5.4.1 to the coding protocol minimizing a given weighted ℓ_p norm, we immediately obtain the following.

Corollary 5.4.2. *Let α be an upper bound on the coding gap for completion times' ℓ_∞ norm (makespan). Then the coding gap for any weighted ℓ_p norm of the completion times is at most 4α.*

Plugging in our coding gap upper bound of Theorem 5.1.2, we therefore obtain a generalization of Theorem 5.1.2 to any weighted ℓ_p norm, as well as average completion time (which corresponds to ℓ_1).

Theorem 5.4.3. *The network coding gap for any weighted ℓ_p norms of completion times is at most*

$$O\left(\log(k) \cdot \log\left(\sum_i d_i / \min_i d_i\right)\right).$$

Note that similar bounds hold even more generally. In particular, for any sub-homogeneous function of degree d (i.e., $f(c \cdot \vec{x}) \leq c^d \cdot f(\vec{x})$, Lemma 5.4.1 implies a coding gap of at most $(4\alpha)^d$, where α is the best upper bound on the coding gap for makespan minimization.

Lower bounds. As with makespan minimization, a polylogarithmic dependence in the problem parameters as in Theorem 5.4.3, as we prove below. Crucially, we rely on our makespan coding gap's examples displaying the property that under coding nearly all unicast sessions' completion time is at least polylogarithmically larger than under the best coding protocol.

Theorem 5.4.4. *There exists an absolute constant $c > 0$ and an infinite family of k-unicast instances whose ℓ_p-coding gap is at least $\Omega(\log^c k)$.*

Proof. We follow the proof of Theorem 5.1.3 and consider $I_{i,r}$, this time setting $r \triangleq (5^{2^i})^2 = 5^{2^{i+1}}$. This does not change the parameters $a_{i,r} = 3^{2^i}$ and $b_{i,r} = 5^{2^i}$, nor the bound $\alpha \triangleq (5/3)^{2^i} \geq$

$\log^c k$ for some absolute constant $c > 0$. This instance has a coding protocol with completion times $(a_{i,r}, \ldots, a_{i,r})$, and so this coding protocol's completion times' ℓ_p value is $a_{i,r}$. On the other hand, by Observation 5.3.1, at least $k \cdot (1 - \frac{b_{i,r}}{r}) \geq (1 - o(1)) \cdot k$ pairs have routing completion time at least $b_{i,r}$, where $o(1)$ tends to 0 as $i \to \infty$. Consequently, the ℓ_p-value of any routing protocol's completion times is at least $(1 - o(1)) \cdot b_{i,r}$. Since $b_{i,r}/a_{i,r} = \alpha \geq \log^c k$, we obtain the required coding gap. □

Acknowledgements The authors would like to thank Mohsen Ghaffari for suggesting an improvement to Theorem 5.1.2 which resulted in a coding gap independent of n, Anupam Gupta for pointing out a simplification of Lemma 5.2.5 and the Lemma's similarity to [9, Theorem 1], and Paritosh Garg for bringing [18] to our attention.

5.5 Chapter Appendix: Completion Time vs. Throughput

In this section we argue why network coding upper bounds for makespan imply coding gaps for throughput maximization. We first introduce the standard definitions of the throughput maximization model [3, 18]. The differences between the throughput and completion-time model (see Section 5.1.1) are highlighted in blue.

Throughput maximization model. A *multiple-unicast instance* $\mathcal{M} = (G, \mathcal{S})$ is defined over a communication network, represented by an undirected graph $G = (V, E, c)$ with capacity $c_e \in \mathbb{Z}_{\geq 1}$ for each edge e. The $k \triangleq |\mathcal{S}|$ *sessions* of $\mathcal{M}$ are denoted by $\mathcal{S} = \{(s_i, t_i, d_i)\}_{i=1}^k$. The (maximum) *throughput* of $\mathcal{M}$ is the supremum $r > 0$ such that there exists a sufficiently large $b > 0$ where the following problem has a correct protocol. Each session consists of source node s_i, which wants to transmit a packet to its sink t_i, consisting of $\lceil r \cdot b \cdot d_i \rceil$ sub-packets (e.g., an element of an underlying field). A *protocol* for a multiple-unicast instance is conducted over finitely-many *synchronous time steps*. Initially, each source s_i knows its packet, consisting of d_i sub-packets. At any time step, the protocol instructs each node v to send a different packet along each of its edges e. The packet contents are computed with some predetermined function of packets received in prior rounds by v or originating at v. The total number of sub-packets sent through an edge e over the duration of the entire protocol is at most $b \cdot c_e$. We differentiate the maximum throughput achievable by coding and routing protocols as r^C and r^R, respectively. The throughput coding gap is the largest ratio r^C/r^R that can be achieved for any instance.

Relating completion times and throughput. The throughput maximization intuitively corresponds to the makespan minimization of an instance with asymptotically-large packet sizes. More formally, we modify a multiple unicast instance $\mathcal{M}$ by increasing its demands by a factor of w while keeping the capacities the same. This causes the makespan to increase. We argue that the slope of the increase with respect to w is exactly the throughput of $\mathcal{M}$.

Definition 5.5.1. *Given a multiple-unicast instance $\mathcal{M}$ we define $C^C(w)$ and $C^R(w)$ to be the makespan of the fastest coding and routing protocols when the all demands are multiplied*

by a common factor w.

Observation 5.5.2. *Let $\mathcal{M}$ be a multiple-unicast instance. The maximum throughput r corresponding to $\mathcal{M}$ is equal to $\sup_{w\to\infty} w/C(w)$ for both coding and routing. Formally, $r^C = \sup_{w\to\infty} w/C^C(w)$ and $r^R = \sup_{w\to\infty} w/C^R(w)$.*

Proof. We drop the R/C superscripts since the proof holds for both without modification. Let $L \triangleq \sup_{w\to\infty} w/C(w)$. We first argue that $L \geq r$, i.e., we can convert a throughput protocol to a makespan-bounded one. For simplicity, we will assume that $b = 1$ (b from the throughput definition); when this is not the case one needs to appropriately re-scale the sub-packets for the completion-time protocol.

Let $\mathcal{T}$ be a protocol of throughput at least $r - o(1)$ and let T be the total number of rounds $\mathcal{T}$ uses (note that in the throughput setting T has no impact on the quality of $\mathcal{T}$). Let $w \in \mathbb{Z}$ be a sufficiently large number. We use pipelining by scheduling $w' \triangleq w/(r - o(1))$ independent copies of $\mathcal{T}$: the first one starting at time 1, second at time 2, ..., last one at time w'. Each copy operates on a separate set of sub-packets, with the pipelined protocol being able to transmit $(r-o(1))\cdot d_i \cdot w' = d_i \cdot w$ sub-packets across the network (in line with Definition 5.5.1) in at most $w' + T$ rounds. Note that $\mathcal{T}$ sends at most c_e sub-packets over an edge e over its entire execution, hence the pipelined version of $\mathcal{T}$ never sends more than c_e sub-packets during any one round. In other words, we have that $C(w) \leq w' + T$. Letting $w \to \infty$ (which implies $w' \to \infty$), we have that

$$L \geq \frac{w}{C(w)} \geq \frac{w}{w' + T} = (r - o(1))\frac{w'}{w' + T} = r - o(1).$$

We now argue the converse $r \geq L$, i.e., we can convert a completion-time protocol into a throughput protocol with the appropriate rate. The result essentially follows by definition. By assumption, for some sufficiently large $w > 0$ there exists a protocol with makespan at most $C(w) \leq w/(L - o(1))$. The protocol sends a total of at most $C(w)c_e$ sub-packets over an edge e. Furthermore, by construction of $C(w)$, each source-sink pair successfully transmits $w \cdot d_i$ sub-packets. By noting that $w \cdot d_i = (L - o(1))C(w)d_i$, we conclude that by using $b \triangleq C(w)$ we get a protocol with rate $L - o(1)$. □

Corollary 5.5.3. *Suppose that the makespan coding gap is at most α (over all instances). Then the throughput coding gap is at most α.*

Proof. Consider some multiple unicast instance $\mathcal{M}$, with coding throughput r. By Observation 5.5.2, for sufficiently large w there is a coding protocol $\mathcal{P}_1$ satisfying $w/C^C(w) \geq r - o(1)$, i.e., $C^C(w) \leq w/(r - o(1))$. By the makespan coding gap assumption, there exists a routing protocol $\mathcal{P}_2$ implying that $C^R(w) \leq \alpha \cdot w/(r - o(1))$. Furthermore, following the proof of Observation 5.5.2, protocol $\mathcal{P}_2$ implies a routing throughput of r' for the original instance $\mathcal{M}$, satisfying

$$r' \geq w/C^R(w) \geq (r - o(1))/\alpha = r/\alpha - o(1).$$

In other words, $r/r' \geq \alpha + o(1)$ and we are done. □

5.6 Chapter Appendix: Network Coding Model for Completion Time

In this section we formalize the k-session unicast communication problem and the notion of completion time for it. We note that our model is not new—e.g., it is equivalent to the models of Chekuri et al. [25], Wang and Chen [138] that study delay in communication networks.

The input to a *k-session unicast problem* $(G, \mathcal{S})$ consists of a graph $G = (V, E, c)$, where edges have capacities $c : E \to \mathbb{R}_{\geq 0}$, and a set of k sessions $\mathcal{S} = \{(s_i, t_i, d_i)\}_{i=1}^{k}$. Each triplet (s_i, t_i, d_i) corresponds to the source $s_i \in V$, sink $t_i \in V$ and the demand $d_i \in \mathbb{R}_{\geq 0}$ of session i. The graph G can be either directed or undirected, where in the latter case we model an undirected edge e as two directed edges $\vec{e}, \overleftarrow{e}$ where both of them have equal capacity $c(\vec{e}) = c(\overleftarrow{e}) = c_e$.[2]

Each source s_i is privy to an input message $m_i \in M_i$ generated by an arbitrary stochastic source with entropy at least d_i, hence the entropy of the random variable m_i is d_i. The sources corresponding to different sessions are independent.

A T-round *network coding computation* consists of a set of $|E| \times T$ coding functions $\{f_{\vec{e},r} : M \to \Gamma\}_{\vec{e} \in E, 1 \leq r \leq T}$, where Γ is some arbitrary alphabet and $M \triangleq \prod_i M_i$. These functions satisfy the following properties:

- The entropy of any coding function $f_{\vec{e},r}$ never exceeds the edge capacity $c(\vec{e})$, i.e., $H(f_{\vec{e},r}) \leq c(\vec{e})$ for all $\vec{e} \in E, 1 \leq r \leq T$.
- For each directed edge $\vec{e} = (u, v) \in E$ and round $1 \leq r \leq T$ the function $f_{\vec{e},r}$ is computable from communication history received strictly before round r at node u. In other words, let the communication history $Y_{u,r}$ be defined as $\{m_i \mid i \in [k], s_i = u\} \cup \{f_{(x,y),r'} \mid y = u \text{ and } r' < r\}$, then $H(f_{(u,v),r} | Y_{u,r}) = 0$.
- The completion times of a network coding computation are $(T_1, T_2, \ldots, T_k) \in \mathbb{Z}_{\geq 0}^{k}$ when the following holds. For every session i, the message m_i of the session (s_i, t_i, d_i) must be computable from the sink t_i's history after T_i rounds are executed, i.e., $H(m_i | Y_{t_i, T_i + 1}) = 0$.

Remark: The above "bare-bones" formalization is sufficient for all of our results to hold. However, such a formalization can be unwieldy since a complete instance description would also need to specify a stochastic distribution corresponding to each source s_i. A standard way of avoiding this issue is to simply assume the sources generate a uniformly random binary string of length d_i (forcing d_i to be an integer). Without going into too much detail, we mention this assumption can be made without loss of generality if we allow for (1) an arbitrarily small decoding error $\varepsilon > 0$, (2) slightly perturbing the edge capacities c_e and source entropies d_i by ε, and (3) scaling-up both c_e's and d_i's by a common constant $b > 0$; this approach is standard in the literature (e.g., see [3, 18]).

[2] Papers such as Adler et al. [3] often impose an alternative condition $c(\vec{e}) + c(\overleftarrow{e}) = c(e)$, which would make our proofs slightly heavier on notation. However, their convention can only impact the results up to a factor of 2, which we typically ignore in this chapter.

5.7 Chapter Appendix: Deferred Proofs of Section 5.2

In this section we provide proofs deferred from Section 5.2, starting with the proof of Proposition 5.2.1, restated here for ease of reference.

Proposition 5.2.1. *Let $z, \{f_i(p) \mid i \in [k], p \in \mathcal{P}_i(T)\}$ be a feasible solution for* ConcurrentFlow$_{\mathcal{M}}(T)$. *Then there exists an integral routing protocol with makespan $O(T/z)$.*

To prove the above, we rely on the celebrated $O(\text{congestion + dilation})$ packet scheduling theorem of Leighton et al. [99]. In particular, we use the solution to ConcurrentFlow$_{\mathcal{M}}(T)$ to obtain a collection of short paths with bounded congestion (i.e., bounded maximum number of paths any given edge belongs to). We then route along these paths in time proportional to these paths' maximum length and congestion. The issue is that the feasible LP solution provides fractional paths, hence requiring us to round the LP solution. Independent rounding would result in paths of length T and congestion $T/z + O(\log n)$ (with high probability). To avoid this additive dependence on n, we rely on the following theorem of Srinivasan and Teo [135].

Lemma 5.7.1 ([135], Theorem 2.4, paraphrased)**.** *Let $\mathcal{M}$ be a multiple-unicast instance and for each $i \in [k]$ let $\mathcal{D}_i$ be a distribution over $s_i \rightsquigarrow t_i$ paths of hop-length at most L. Suppose that the product distribution $\prod \mathcal{D}_i$ has expected congestion for each edge at most L. Then there exists a sample $\omega \in \prod \mathcal{D}_i$ (i.e., a choice of a from $\mathcal{D}_i$ between each $s_i \rightsquigarrow t_i$) with (maximum) congestion $O(L)$.*

Using the above lemma to round the LP and using Leighton et al. [99] path routing to route along the obtained paths yields Proposition 5.2.1.

Proof of Proposition 5.2.1. Consider an optimal solution to this ConcurrentFlow$_{\mathcal{M}}(T)$. Clearly, picking for each pair (s_i, t_i) some d_i paths in $P_i(T)$ with each $p \in \mathcal{P}_i(T)$ picked with probability $f_i(p) \cdot d_i / \sum_{p \in \mathcal{P}_i(T)} f_i(p) \leq f_i(p)/z$ yields an expected congestion at most $T \cdot c_e/z$ for each edge e. That is, thinking of G as a multigraph with c_e copies per edge, each such parallel edge has congestion T/z. On the other hand, each such path has length at most $T \leq T/z$ (since $z \leq 1$). Therefore, by Lemma 5.7.1, there exist choices of paths for each pair of (maximum) congestion and hop-bound (i.e., dilation) at most $O(T/z)$. But then, using $O(\text{congestion + dilation})$ routing [99] this implies an integral routing protocol with makespan $O(T/z)$, as claimed. □

Here we prove Claim 5.2.4, restated here for ease of reference.

Claim 5.2.4. *Given sequences $h_1, \ldots, h_k, d_1, \ldots, d_k \in \mathbb{R}_{\geq 0}$ with $\sum_{i \in [k]} d_i \cdot h_i \geq 1$ there exists a non-empty subset $I \subseteq [k]$ with $\min_{i \in I} h_i \geq \frac{1}{\alpha_{\text{gap}} \cdot \sum_{i \in I} d_i}$ for $\alpha_{\text{gap}} \in \left[1, O\left(\log \frac{\sum_{i \in [k]} d_i}{\min_{i \in [k]} d_i}\right)\right]$.*

Proof. Suppose (without loss of generality) that $h_1 \geq h_2 \geq \ldots h_k$ and assume for the sake of contradiction that none of the sets $[1], [2], \ldots, [k]$ satisfy the condition. In other words, if we let $d([j]) \triangleq \sum_{i=1}^{j} d_j$, then $h_i < \frac{1}{\alpha} \cdot \frac{1}{d([i])}$ for all $i \in [k]$. Multiplying both sides by d_i and

summing them up, we get that $1 \leq \sum_{i=1}^{k} d_i h_i < \frac{1}{\alpha} \sum_{i=1}^{k} \frac{d_i}{d([i])}$. Reordering terms, this implies $\sum_{i=1}^{k} \frac{d_i}{d([i])} > \alpha$.

Define $f(x)$ as $1/d_1$ on $[0, d_1)$; $1/(d_1 + d_2)$ on $[d_1, d_1 + d_2)$; ...; $1/d([i])$ on $[d([i-1]), d([i]))$ for $i \in [k]$. Now we have

$$\int_0^{d([k])} f(x) = \frac{d_1}{d_1} + \frac{d_2}{d_1 + d_2} + \frac{d_3}{d_1 + d_2 + d_3} + \ldots + \frac{d_k}{d([k])}.$$

However, since $f(x) \leq 1/x$

$$\begin{aligned}\int_0^{d([k])} f(x) &= \int_0^{d_1} f(x)\,dx + \int_{d_1}^{d([k])} f(x)\,dx \\ &\leq 1 + \int_{d_1}^{d([k])} \frac{1}{x}\,dx = 1 + \ln \frac{d([k])}{d_1}.\end{aligned}$$

Hence by setting $\alpha \triangleq 1 + \ln \frac{d([k])}{d_1}$ we reach a contradiction and finish the proof. □

5.8 Chapter Appendix: Deferred Proofs of Section 5.3

Here we provide the deferred proofs of lemmas of Section 5.3, restated below for ease of reference.

Lemma 5.3.5. *(Coding makespan) The product instance I_+ admits a network coding protocol with makespan at most $a_1 a_2$.*

Proof. Suppose there exists a network coding protocol with makespan $t_i \leq a_i$ that solves $(G_i, \mathcal{S}_i)$ for $i \in \{1, 2\}$. Functionally, each round in the outer instance $(G_1, \mathcal{S}_1)$ consists of transmitting c_e bits of data from u to v for all arcs (u, v) where $\{u, v\} \in E(G_1)$. This is achieved by running the full t_2 rounds of the inner instance protocol over all copies of the instances which pushes d_i bits from s_i to t_i for all (s_i, t_i, d_i) and all copies of the inner instance. The reason why such inner protocol pushes the information across each arc (u, v) is because u is merged with some s_i, v is merged with t_i, and with $d_i = c_e$ for some $(s_i, t_i, d_i) \in \mathcal{S}_2$ and some copy of the inner instance. In conclusion, by running t_1 outer rounds, each consisting of t_2 inner rounds, we get a $t_1 t_2 \leq a_1 a_2$ round protocol for the product instance. □

Lemma 5.3.7. *For $i \in \{1, 2\}$ let $I_i = (G_i, \mathcal{S}_i, F_i)$ be a gap instance with parameters $(a_i, b_i, f_i, k_i, m_i, r_i, u_i)$ with $\frac{m_i}{f_i} \geq 2$ and $a_i \geq 2$; let $B \in \mathcal{B}(n_1, n_2, 2(m_1 - f_1), k_2, 2b_1 b_2)$ be a colored bipartite graph. Then $I_+ \triangleq T(G_1, G_2, B)$ is a gap instance with parameters $a_+ \triangleq a_1 a_2$, $b_+ \triangleq b_1 b_2$, $f_+ \triangleq n_1 f_1 + n_2 f_2$, $k_+ \triangleq n_1 k_1$, $m_+ \triangleq a_2 n_1 f_1 + n_2 m_2$, $r_+ \triangleq r_1 \frac{1}{1 + 2u_1/r_2}$, $u_+ \triangleq u_2 \frac{1 + a_2/2}{1 + r_2/(2u_1)}$. Moreover, $\frac{m_+}{f_+} \geq 2$ and $a_+ \geq 2$.*

Proof of Lemma 5.3.7. First, the set of terminals in the product instance I_+ is disjoint, as distinct terminals of copies of the outer instance I_1 have their edges associated with distinct terminals source-sink pairs of the inner instance I_2. Consequently, no two terminals of the outer instance are associated with the same node of the same copy of an inner instance. The capacities and demands of I_+ are one by definition. We now turn to bounding the gap instance's parameters.

Parameters a_+ and b_+ are directly argued by Lemma 5.3.5 and Lemma 5.3.6. Furthermore, f_+, k_+, m_+ are obtained by direct counting, as follows.

Recall that the cut edges of the outer instance get replaced with a path of length a_2. Since there are n_1 copies of outer instances, each having f_1 cut edges, this contributes $a_2 n_1 f_1$ edges to m_+. The non-cut edges of the outer instance get deleted and serve as a merging directive, hence they do not contribute to m_+. Finally, each edge of the inner instance gets copied into I_+, contributing $n_2 m_2$ as there are n_2 copies of the inner instance.

For r_+ we need to show it is a lower bound on k_+/f_+. We note that $|E(B)| = n_1 \cdot 2(m_1 - f_1) = n_2 k_2$ and proceed by direct calculation:

$$\begin{aligned}\frac{k_+}{f_+} &= \frac{n_1 k_1}{n_1 f_1 + n_2 f_2} = \frac{k_1}{f_1} \cdot \frac{1}{1 + \frac{n_2}{n_1}\frac{f_2}{f_1}} \geq \frac{k_1}{f_1} \cdot \frac{1}{1 + \frac{2(m_1 - f_1)}{k_2}\frac{f_2}{f_1}} \\ &\geq \frac{k_1}{f_1} \cdot \frac{1}{1 + \frac{2m_1}{f_1}\frac{f_2}{k_2}} \geq \frac{k_1}{f_1} \cdot \frac{1}{1 + \frac{2u_1}{r_2}} = r_1 \cdot \frac{1}{1 + \frac{2u_1}{r_2}} = r_+.\end{aligned}$$

For u_+ we need to show it is an upper bound on m_+/f_+. Note that $k_2 \leq m_2$ since the set of terminals is disjoint and the graph is connected.

$$\begin{aligned}\frac{m_+}{f_+} &= \frac{n_2 m_2 + a_2 n_1 f_1}{n_2 f_2 + n_1 f_1} = \frac{m_2}{f_2} \cdot \frac{1 + a_2 \frac{n_1}{n_2}\frac{f_1}{m_2}}{1 + \frac{n_1}{n_2}\frac{f_1}{f_2}} \leq u_2 \cdot \frac{1 + a_2 \frac{k_2}{2(m_1 - f_1)}\frac{f_1}{m_2}}{1 + \frac{k_2}{2(m_1 - f_1)}\frac{f_1}{f_2}} \\ &\leq u_2 \cdot \frac{1 + a_2 \frac{k_2}{2(m_1/f_1 - 1)}\frac{1}{m_2}}{1 + \frac{f_1}{2m_1}\frac{k_2}{f_2}} \leq u_2 \cdot \frac{1 + a_2 \cdot \frac{1}{2} \cdot 1}{1 + \frac{1}{2}\frac{r_2}{u_1}} = u_+.\end{aligned}$$

Here the last inequality relies on $m_1/f_1 \geq 2$ and on $k_2 \leq m_2$, which follows from the set of terminals being disjoint and the graph G_2 being connected.

For the final technical conditions, note that $a_+ \geq 2$ is clear from $a_+ = a_1 a_2 \geq 4 \geq 2$. Finally, $\frac{m_+}{f_+} \geq 2$ follows from the following.

$$\frac{m_+}{f_+} = \frac{a_2 n_1 f_1 + n_2 m_2}{n_1 f_1 + n_2 f_2} = a_2 \frac{n_1 f_1}{n_1 f_1 + n_2 f_2} + \frac{m_2}{f_2}\frac{n_2 f_2}{n_1 f_1 + n_2 f_2} \geq 2\left(\frac{n_1 f_1}{n_1 f_1 + n_2 f_2} + \frac{n_2 f_2}{n_1 f_1 + n_2 f_2}\right) = 2. \qquad \square$$

5.8.1 Upper Bounding $m_{i,r}$

For readability, we sometimes write $u(i,r)$ instead of $u_{i,r}$ and similarly for $m(i,r)$. Also, we note that the technical conditions $a_{i,r} \geq 2$ and $\frac{m_{i,r}}{f_{i,r}} \geq 2$, which clearly hold for $i = 0$, hold for all $i > 0$, due to Lemma 5.3.7. Finally, we note that by Lemma 5.3.7, since $r_2 = u_1$ and $a_2 \geq 1$, we have that $u_+ \geq u_2$ and so for all gap instances in the family we have $u_{i,r} \geq u_{i-1,u(i-1,3r)} \geq 5$.

Lemma 5.8.1. *The parameter of $I(i,r)$ for any $i \geq 0, r \geq 5$ satisfy the following.*

(i) $\frac{k_{i,r}}{f_{i,r}} \geq r$,

(ii) $u_{i+1,r} \leq 3^{2^i} \cdot u_{i,u(i,3r)}$ *and*

(iii) $\log m(i+1,r) \leq O(5^{2^{i+1}}) \cdot \log(m_{i,3r} \cdot m_{i,u(i,3r)})$.

Proof. Claim (i) follows from Lemma 5.3.7, as follows.

$$\frac{k_{i+1,r}}{f_{i+1,r}} \geq r_{i+1,r} = r_{i,3r} \cdot \frac{1}{1 + 2u_{i,3r}/r_{i,u(i,3r)}} \geq 3r \cdot \frac{1}{1 + 2u_{i,3r}/u_{i,3r}} = 3r \cdot \frac{1}{3} = r.$$

We now prove claims (ii) and (iii). Fix i, r and define $I_1 \triangleq I(i, 3r)$ (with parameters $(a_1, \ldots, u_1)$) and $I_2 \triangleq I(i, u_{i,r})$ (with parameters $(a_2, \ldots, u_2)$). We have $u(i+1,r) = u_2 \frac{1+a_2/2}{1+r_2/(2u_1)} \leq u_2 \frac{1+a_2/2}{1+1/2} \leq u_2 \cdot a_2$ (Lemma 5.3.7), with $a_2 = a_{i,u(i,3r)} = 3^{2^i}$ and $u_2 = u(i, u(i, 3r))$ from the iterated tensoring process. Therefore, we conclude that $u(i+1,r) \leq 3^{2^i} \cdot u_{i,u(i,3r)}$, as claimed.

We now prove Claim (iii). The corresponding colored bipartite graph $B \in \mathcal{B}(n_1, n_2, 2(m_1 - f_1), k_2, 2b_1b_2)$ used to produce the product $I_{i,r}$ has $\max(n_1, n_2) \leq (2(m_1 - f_1)k_2)^{O(b_1b_2)}$, by Lemma 5.3.4. Therefore, as $k_2 \leq m_2$, we have that $\max(n_1, n_2) \leq (m_1 \cdot m_2)^{O(b_1b_2)}$. This implies the following recurrence for $m_{i,r}$.

$$m_{i+1,r} = a_2 n_1 f_1 + n_2 m_2 \leq a_2 \max(n_1, n_2) m_1 m_2.$$

Taking out logs, we obtain the desired bound.

$$\begin{aligned}
\log m_{i+1,r} &\leq \log a_2 + \log \max(n_1, n_2) + \log m_1 m_2 \\
&= O(2^i) + O(b_1 b_2) \log(m_1 m_2) + \log(m_1 m_2) \\
&= O(2^i) + O(5^{2^i}) \log(m_1 m_2) \\
&= O(5^{2^i}) \cdot \log(m_{i,3r} \cdot m_{i,u(i,3r)}).
\end{aligned}$$

□

Given Lemma 5.8.1 we obtain the bound on $u_{i,r}$ in terms of i and r.

Lemma 5.8.2. *We have that* $\log u_{i,r} \leq 2^{O(2^i)} \log r$ *for all* $i \geq 0, r \geq 5$.

Proof. By Lemma 5.8.1, we have the recursion $u(i+1,r) \leq 3^{2^i} \cdot u(i, u(i, 3r))$ with initial condition $u(0,r) = O(r^2)$, by Fact 5.3.2. Taking out logs, we obtain $\log u(i+1,r) \leq O(2^i) + \log u(i, u(i, 3r))$ and $\log u(0,r) = O(\log r)$. We prove via induction that $\log u(i,r) \leq \frac{1}{c}(c^2)^{2^i} \cdot \log r$ for some sufficiently large $c > 0$. In the base case $\log u(0,r) = O(\log r) \leq \frac{1}{c}(c^2) \log r =$

$c \log r$. For the inductive step we have:

$$\begin{aligned}
\log u(i+1, r) &\leq O(2^i) + \log u(i, u(i, 3r)) \\
&\leq O(2^i) + \frac{1}{c}(c^2)^{2^i} \log u(i, 3r) \\
&\leq O(2^i) + \frac{1}{c}(c^2)^{2^i} \frac{1}{c} c^{2^i} \log 3r \\
&\leq O(2^i) + \frac{1}{c^2}(c^2)^{2^{i+1}} \log 3r \\
&\leq \frac{1}{c}(c^2)^{2^{i+1}} \log r,
\end{aligned}$$

where the last inequality holds for $i \geq 1$ and $r \geq 5$ and a sufficiently large $c > 0$. □

Plugging in the bound of Lemma 5.8.2 and Lemma 5.8.1 we can prove inductively the upper bound on the number of edges of $I_{i,r}$ in terms of i and r given by Lemma 5.3.10, restated here.

Lemma 5.3.10. *We have that* $\log m_{i,r} \leq 2^{O(2^i)} \log r$ *for all* $i \geq 0, r \geq 5$.

Proof. By Lemma 5.8.1, we have the recursion $\log m(i+1, r) \leq O(5^{2^{i+1}}) \cdot \log(m_{i,3r} \cdot m_{i,u(i,3r)})$ with initial condition $\log m(0, r) = O(\log r)$, by Fact 5.3.2. We prove via induction that $\log m_{i,r} \leq c^{2^i} \log r$ for a sufficiently large universal constant $c > 0$. In the base case $\log m_{0,r} \leq O(\log r) \leq c \log r = c^{2^0} \log r$. For the inductive step, using Lemma 5.8.2 to bound $\log u(i, 3r)$, we have:

$$\begin{aligned}
\log m_{i+1,r} &\leq O(5^{2^{i+1}}) \cdot (\log m_{i,3r} + \log m_{i,u(i,3r)}) \\
&\leq O(5^{2^{i+1}}) \cdot \left(c^{2^i} \log 3r + c^{2^i} \log u(i, 3r) \right) \\
&\leq O(5^{2^{i+1}}) \cdot \left(c^{2^i} \log 3r + c^{2^i} 2^{O(2^i)} \log 3r \right) \\
&= c^{2^i} \cdot O(5^{2^{i+1}}) \cdot 2^{O(2^i)} \log 3r \\
&\leq c^{2^i + 1} \log r,
\end{aligned}$$

where the last inequality holds for $i \geq 1$ and $r \geq 5$ and a sufficiently large $c > 0$. □

Chapter 6

Shortcuts are a Universal Lower Bound for Distributed Optimization

The results of this chapter were published in [70] with Bernhard Haeupler and David Wajc as co-authors. The work was supported in part by NSF grants CCF-1910588, CCF-1814603, CCF-1618280, CCF-1527110, NSF CAREER award CCF-1750808, and a Sloan Research Fellowship.

6.1 Introduction

Much of modern large-scale graph processing and network analysis is done using systems like Google's Pregel [111], Facebook's Giraph [26, 72], or Apache's Spark GraphX [58] which implement synchronous message-passing algorithms, in which nodes send (small) messages to their neighbors in each round.[1]

This has motivated a recent, broad, concentrated, and highly-successful effort to advance our theoretical understanding of message-passing algorithms for fundamental network optimization problems, such as minimum-spanning trees (MST) [34, 37, 89, 96, 118], shortest paths [36, 44, 78, 80, 82, 84, 101, 102, 113], flows [55], and cuts [49, 114]. As a result, many fundamental optimization problems now have worst-case-optimal message-passing algorithms, running in $\tilde{\Theta}(\sqrt{n}+D)$ rounds on every n-node network with diameter D.[2] In general, these running times cannot be improved due to unconditional lower bounds [27, 122] showing *some* pathological n-node topologies with small diameter on which any non-trivial optimization problem requires $\tilde{\Omega}(\sqrt{n})$ rounds. This type of worst-case optimality is also called *existential optimality*.

Unfortunately, while network diameters tend to be small in practice, a $\tilde{\Theta}(\sqrt{n})$ round complexity is completely impractical. On the other hand, existential optimality says little about the dis-

[1]For the sake of concreteness, we limit message sizes to $O(\log n)$ bits where n is the number of network nodes. This is exactly the classic CONGEST model of distributed computation [121]. Throughout this paper we use the terms distributed, message-passing, or CONGEST algorithm interchangeably.

[2]Throughout, we use $\tilde{O}, \tilde{\Omega}$ and $\tilde{\Theta}$ to suppress $\operatorname{poly}\log n$ terms. E.g., $\tilde{O}(f(n)) = O(f(n)\log^{O(1)} n)$.

tributed complexity of optimization problems on non-worst-case topologies. Indeed, evidence suggests [49, 52, 56, 65, 66, 67, 68, 90] that most, if not all, networks of interest allow for exponentially faster $\tilde{O}(1)$-round optimization algorithms.

"*The interesting question that arises is therefore whether it is possible to identify the inherent graph parameters associated with the distributed complexity of various fundamental network problems, and develop* universally-optimal *algorithms[, that are as fast as possible on* every *topology.]*" [46]

These two fundamental questions have a 25+ year old history. In fact, the above is a verbatim quote from the influential FOCS'93 paper of Garay, Kutten and Peleg, which started and majorly shaped the area of distributed optimization algorithms. Despite this, both questions have remained wide open. Indeed, besides universal optimality seeming out of reach of current techniques, it is unclear whether this notion should even be possible at all. After all, universal optimality asks for a uniform algorithm running on an unknown topology G to be competitive with the best non-uniform algorithm $\mathcal{A}_G$, which is specifically designed for a topology G known to it.

The closest thing towards algorithms for non-worst-case topologies has been the low-congestion shortcut framework, introduced by Ghaffari and Haeupler in [49] and further developed in [52, 54, 64, 65, 66, 68, 90].[3] This framework identifies partwise aggregation as a simple communication primitive which is sufficient to efficiently solve many distributed optimization problems, including $(1+\varepsilon)$-min-cuts, various approximate shortest-path problems, and MST. It furthermore introduces low-congestion shortcuts as a graph structure that can be used to solve partwise aggregation quickly. This leads to distributed optimization algorithms with $\tilde{O}(Q(G)+T(G))$ running times, where $Q(G) = \text{SHORTCUTQUALITY}(G)$ is the best shortcut quality admitted by G, and $T(G)$ is the time to construct an approximately-optimal shortcut, of quality $\widetilde{O}(Q(G))$. It is furthermore shown that bounding some graph parameters such as genus, tree-width [66], expansion [52], or the largest clique-minor [68] results in topologies with good and efficiently constructable [65] shortcuts. These graph parameters are therefore sufficient to imply fast distributed optimization algorithms and give some insights into which topologies are "easy". On the other hand, no lower bound relating any "hard" feature of a general topology to a non-trivial $\Omega(D)$ distributed complexity is known for any optimization problem.

6.1.1 New Results and Contributions

This chapter makes significant progress on the above two questions of [46] by proving the first universal lower bounds on the distributed complexity of many important optimization problems. Most importantly, we prove that $\text{SHORTCUTQUALITY}(G)$, which is the key parameter in the running time of shortcut-based distributed optimization algorithms for these same problems, is itself a universal lower bound. This means that the different lower bounds provided in this

[3]The k-broadcast algorithm of [48] also goes into the direction of universal-optimality except that its running time is competitive with the best routing schedule in G (and not the best CONGEST algorithm which might be much faster). Some time to preprocess the topology is also required.

chapter are likely the tight and long thought-after characterizations which tightly determine the complexity of distributed optimization for any topology. It also implies that the only ingredient missing for achieving provably universally-optimal algorithms for many important distributed optimization problems is an improved shortcut construction or approximation algorithm.

Generalizing [27] to worst-case subnetworks in general topologies

To achieve our results we give a robust definition of a *worst-case subnetwork*, which generalizes the pathological worst-case topology of the existential lower bound of Das Sarma et al. [27] to subnetworks in general graphs. This generalization builds on insights and crucial definitions from a recent work [69] which connects the lower bound of [27] to network coding gaps for multiple unicasts. Once the new definition is in place it is easy to verify that the proof of [27] generalizes to our worst-case subnetworks. Defining $\text{WC}\textsc{SubNetwork}(G)$ to be the size of the largest such worst-case subnetwork in G then gives a lower bounds for *any* network, instead of just a single graph that is carefully chosen to facilitate the lower bound proof.[4] One particularly nice aspect of this universal lower bound is that it brings the full strength and generality of the lower bound of [27] to general topologies. In particular, it applies to a myriad of different optimization and verification problems, holds for deterministic and randomized algorithm alike, holds regardless of whether the input topology is known or not, and extends in full strength to any non-trivial approximations.

Lemma 6.1.1. *For any topology $G = (V, E)$, any message-passing algorithm $\mathcal{A}_G$ that determines for any given subgraph $H \subseteq G$ whether it is a connected spanning subgraph has a running time of at least $\widetilde{\Omega}(\text{WC}\textsc{SubNetwork}(G) + D)$ rounds. This holds even if $\mathcal{A}_G$ is randomized and knows G.*

By the simple reductions to the connected spanning subgraph problem described in detail in [27], this extends to lower bounds for MST, cut, min-cut, s-source distance, shallow light trees, min-routing cost trees and many other problems as well as to any non-trivial approximations for these problems.

Shortcut quality is a universal lower bound, and implications for universal optimality

A priori, it is not clear how strong or interesting the $\text{WC}\textsc{SubNetwork}(G)$ lower bound is. By definition, it only applies to networks with subnetworks displaying similar characteristics to the pathological worst-case topology from [27], which seems very specific. Surprisingly, however, the main contribution of this chapter proves several other universal lower bounds, including and most importantly $\textsc{ShortcutQuality}(G)$, by showing them to be equivalent to $\text{WC}\textsc{SubNetwork}(G)$.

[4] Indeed, Das Sarma et al. [27] state concerning their existential lower bound that "The choice of graph G is critical."

Theorem 6.1.2. *For any graph G,*

$$\textsc{ShortcutQuality}(G) = \tilde{\Theta}(\textsc{WCSubnetwork}(G) + D).$$

As a corollary of Lemma 6.1.1 and Theorem 6.1.2, the parameter $\tilde{\Theta}(\textsc{ShortcutQuality}(G))$ is also a lower bound for the complexity of the very same optimization problems for which the low-congestion framework has already established algorithmic results with running times mostly depending on $\textsc{ShortcutQuality}(G)$. This strongly suggests that all our lower bound parameters are not just equivalent to each other, but indeed serve as tight characterizations (up to $\tilde{O}(1)$ terms) of the inherent distributed complexity of a wide variety of optimization problems.

This also stops just short of completely resolving the 25+ year long quest for a universally-optimal MST algorithm. All that is needed is an efficient shortcut construction. More precisely, a distributed algorithm which for every topology G computes $\tilde{O}(1)$-approximately optimal shortcuts of quality $\widetilde{O}(\textsc{ShortcutQuality}(G))$ in $\tilde{O}(D + \textsc{ShortcutQuality}(G))$ time. Using such an algorithm as a shortcut-construction subroutine in the existing shortcut-based optimization algorithms would immediately give universally-optimal algorithms for MST, the various approximate shortest path type problems in [64], $(1+\epsilon)$-approximate min-cuts, connectivity, and several other problems. This would mark a tremendous achievement and crowning victory for the intensive decades-long research efforts which have contributed to this algorithmic understanding of distributed optimization.

We remark that initially it may look circular and too good to be true to ask for the efficiency of the construction algorithm to be upper bounded by the quality of the near-optimal shortcuts it is supposed to find: How can the algorithm profit from merely the existence of the unknown high-quality shortcut it is supposed to find? This barrier, however, has already been overcome by the recursive approach of the shortcut-construction algorithm in [65]. Indeed, this algorithm's running time and approximation guarantees have exactly the desired form, except that the algorithm in [65] only approximates a somewhat restricted form of shortcuts. We are hopeful that the ideas from [65] can be used nonetheless to create a shortcut construction algorithm without this restriction.

Different characterizations of a topology's inherent distributed complexity

Assuming the existence of a good shortcut-construction algorithm, proving that $\textsc{ShortcutQuality}(G)$ is a universal lower bound is sufficient for certifying universal optimality of algorithms within the low-congestion shortcut framework. However, as mentioned before, identifying, understanding, and characterizing the aspects of a topology that influence and determine the complexity of distributed optimization problems is in itself a worthwhile goal. Indeed, there are a multitude of reasons why a detailed understanding of the relationship between topology and complexity is important. Among other reasons, it (a) can be important for the design of good networks, (b) might give important leads for understanding the structure of existing natural and artificial networks occurring in society, biology, and other areas, and (c) is necessary to provide quantitative and provable running time guarantees for universally-optimal algorithms run on a known topology

G, beyond a simple "it runs as fast as possible".

As such, another important contribution of the tight lower bounds proven in this chapter consists of giving different characterizations and ways to think about what makes a topology hard (or easy). For example, while WCSUBNETWORK(G) and SHORTCUTQUALITY(G) are both quantitatively equal, the fact that they both characterize the complexity of distributed optimization lends itself to very different interpretations and conclusions.

Indeed, SHORTCUTQUALITY(G) can be seen as the best routing schedules for the partwise aggregation problem, which is the very natural communication primitive underlying distributed divide-and-conquer style algorithms (see, e.g., [65]). Shortcut quality being a tight universal lower bound further demonstrates the key role partwise aggregation plays for distributed optimization algorithms, even to the extent that the complexity of many very different optimization tasks is dominated by how fast this simple aggregation procedure can be performed on a given topology.

The tightness of WCSUBNETWORK(G) as a lower bound, on the other hand, points to the pathological network structure identified by Peleg and Rubinovich [27, 122] as indeed the only way in which a topology can be hard for optimization. Put otherwise, a topology is exactly as hard as the worst obstruction of this type within a network.

As part of our proof of Theorem 6.1.2 we identify, define, and expose several other graph parameters which similarly characterize the complexity of a topology G, such as, MOVINGCUT(G), ROUTING(G) and others. Many of these parameters have very different flavors. For example the MOVINGCUT(G) parameter can be seen as identifying crucial communication bottlenecks within a topology via a sequence of cuts. It is also known [69] to characterize the time needed to solve a simple multiple unicast communication problem which requires information to be sent between different sender-receiver pairs in the network. ROUTING(G) relates to the same communication problem, but with the restriction that information is routed (without any coding) which, by Leighton, Maggs, and Rao [99], is equivalent to the best congestion and dilation of paths connecting the sender-receiver pairs. We give precise definitions and further explanations for these and other equivalent universal lower bound parameters in the technical sections of this chapter. We hope that they will help to further illuminate different aspects of the topology-complexity interplay.

6.2 Preliminaries

In this work we study universal optimality, achieved by uniform algorithms whose running time is competitive with the fastest algorithm tailor-made for the given graph, for *any* graph. More precisely, if we denote by $T_{\mathcal{A}}(G)$ the worst-case running time of algorithm $\mathcal{A}$ for problem Π on any input with topology G (i.e., the maximum number of rounds $\mathcal{A}$ takes over all possible inputs), then universal optimality is defined as follows.

Definition 6.2.1. *An algorithm $\mathcal{A}$ for problem Π is **universally optimal** if for every graph G and algorithm $\mathcal{A}_G$ for Π, we have*

$$T_{\mathcal{A}}(G) = \tilde{O}(1) \cdot T_{\mathcal{A}_G}(G).$$

An alternative view would be to say $\mathcal{A}$ must be competitive with the best algorithm that knows the topology G, but not the *problem input* (e.g., H in the spanning connected subgraph problem). We note that a similar notion of competitiveness with the fastest tailor-made algorithm for any *input* is hopeless, since any input has a zero-round algorithm: just output the solution!

Shortcuts and partwise aggregation. In the *partwise aggregation* problem, the input consists of disjoint subsets of nodes in G, denoted by $S_1, \ldots, S_k$ and called *parts*, inducing *connected* subgraphs $G[S_i]$. Each node has some private input, and the goal is to compute some simple aggregate function (e.g., the minimum) of these private inputs, and send this value to all nodes in each part. This problem appears frequently in numerous distributed graph algorithms, showing that $\tilde{O}(1)$ invocations of algorithms for this problem allow to solve problems such as MST, approximate SSSP and global min-cut, and numerous graph verification problems (see [27, 34, 45, 46, 47, 49, 51, 55, 64, 96, 114]).

Partwise aggregation can be solved by a routing-based approach. We remind the reader that we say a set of subgraphs $H_i \subseteq G$ are *q-quality shortcuts* for $S_1, \ldots, S_k$ if each edge is contained in at most q subgraphs $G[S_i] + H_i$ and the diameter of each such subgraph $G[S_i] + H_i$ is at most q. Extending the random delay method of [99], Ghaffari and Haeupler [49] propose a framework for solving partwise aggregation using q-quality shortcuts in $\tilde{O}(q)$ time. Given the wide applicability of partwise aggregation, this framework has unsurprisingly found numerous applications in distributed algorithms. The following notation is useful when assessing the usefulness of this shortcut-based approach to partwise aggregation. For parts $P = (S_1, \ldots, S_k)$ as above, we denote by $\text{SHORTCUTQUALITY}(P)$ the minimum q such that there exist q-quality shortcuts for P. The shortcut quality of a graph G, $\text{SHORTCUTQUALITY}(G) = \max_P \text{SHORTCUTQUALITY}(P)$, is the worst-case shortcut quality over all such $\{S_i\}_i$ in G. As we show, this graph parameter is intimately related to the time needed to solve many distributed problems.

6.2.1 Moving cuts

In this section we reinterpret the *moving cuts* of Chapter 5 in order to facilitate distributed lower bounds. Moving cuts represent an important tool for giving distributed information-theoretic lower bounds. Moving cuts are used to lift strong unconditional lower bounds from the classic communication complexity setting into the distributed setting. This approach was used to prove existentially-optimal lower bounds in Das Sarma et al. [27], and moving cuts can be seen as a generalization of their techniques. However, moving cuts were only explicitly defined formally Chapter 5, which used them to relate the maximum worst-case time gap between coding and store-and-forward protocols for pairwise communication. Before defining moving cuts, we

briefly discuss the communication complexity model and distributed function computation problems.

Distributed computation of a Boolean function f is a problem (in the CONGEST model). Two distinguished (multi-)sets of nodes $\{s_i \in V\}_{i=1}^k$ and $\{t_i \in V\}_{i=1}^k$ are given. For a given function $f : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$ and inputs $x, y \in \{0,1\}^k$, we want every node in G to learn $f(x,y)$. However, x_i and y_i are given only to s_i and t_i as their respective private inputs. Nodes in G have access to shared random coins. We are interested in the worst-case running time to complete the above task. The problem is motivated by the (classic) **communication complexity model**, which is the special case of CONGEST with a two-node, single-edge graph, where Alice controls one node (with k input bits) and Bob controls the other node (ditto), and *single-bit* messages are sent in each round. The time to compute f in this model is referred to as its communication complexity. In this chapter we are mostly interested in the k-bit *disjointness function*, $\mathrm{disj} : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$, given by $\mathrm{disj}(x,y) = 1$ if for each $i \in [k]$, we have $x_i \cdot y_i = 0$, and $\mathrm{disj}(x,y) = 0$ otherwise. I.e., if x and y are indicator vectors of sets, this function indicates whether these sets are disjoint. This function is known to have communication complexity $\Theta(k)$ [27, 128].

Having defined the distributed function computation model, we are now ready to define our lower-bound certificate on the time to compute a function between nodes $\{s_i\}_{i=1}^k$ and $\{t_i\}_{i=1}^k$: a *moving cut*. We restate the moving-cut definition.

Definition 6.2.2 ([69])**.** *Let $S = \{(s_i, t_i)\}_{i=1}^k$ be a set of source-sink pairs in a graph $G = (V, E)$. A **moving cut** for S is an assignment of positive integer edge lengths $\ell : E \to \mathbb{Z}_{\geq 1}$. We say that:*

*(i) ℓ has **capacity** $\gamma := \sum_{e \in E}(\ell_e - 1)$;*

*(ii) ℓ has **distance** β when $\mathrm{dist}_\ell(\{s_i\}_{i=1}^k, \{t_j\}_{j=1}^k) \geq \beta$, i.e., the ℓ-distance between all sinks and sources is at least β.*

The utility of moving cuts is showcased by the following lemma.

Lemma 6.2.3. *If G contains a moving cut for k pairs $S = \{(s_i, t_i)\}_{i=1}^k$ with distance at least β and capacity strictly less than k, then distributed computation of disj between $\{s_i\}_{i \in [k]}$ and $\{t_i\}_{i \in [k]}$ takes $\widetilde{\Omega}(\beta)$ time. This lower bound holds even for bounded-error randomized algorithms that know G and S.*

Broadly, Lemma 6.2.3 follows from a simulation argument. Given a sufficiently-fast $\widetilde{O}(\beta)$-time distributed algorithm for disj between $\{s_i\}_{i=1}^k$ and $\{t_i\}_{i=1}^k$ and a moving cut for $\{(s_i, t_i)\}_{i=1}^k$ of capacity less than k and distance β, we show how to obtain a communication complexity protocol with a sufficiently small complexity $O(k)$ to contradict the classic $\Omega(k)$ communication complexity lower bound for disjointness. This yields the lower bound. The full proof follows the arguments (implicitly) contained in [27] and Chapter 5, and is given in Section 6.6 for completeness.

Lemma 6.2.3 motivates the search for moving cuts of large distance and bounded capacity. For a fixed set of k pairs S, we denote by $\text{MOVINGCUT}(S)$ the largest distance β of a moving cut for S of capacity strictly less than k.

6.2.2 Relation of moving cuts to communication

Consider the simple communication problem for pairs $S = \{s_i, t_i\}$, termed *multiple unicasts*. In this problem, each s_i has a single-bit message x_i it wishes to transmit to t_i. We denote by $\text{COMMUNICATING}(S)$ the time of the fastest algorithm for this problem which knows G and S (but not the messages). One natural way to solve this problem is to store-and-forward (or "route") the messages x_i through the network. We denote the fastest such algorithm's running time by ROUTING(S). While faster solutions can be obtained by encoding and decoding messages in intermediary nodes, prior work has shown the gap between the fastest routing and unrestricted (e.g., coding-based) algorithms is at most $\widetilde{O}(1)$ [69].

Furthermore, the following lemma shows that moving cuts characterize the time required to complete multiple unicasts. As the model and terminology of [69] is slightly different from ours, we provide a proof of this lemma in Section 6.6.

Lemma 6.2.4. *([69]) For any set of pair S, we have that*

$$\text{MOVINGCUT}(S) = \tilde{\Theta}(\text{COMMUNICATING}(S)) = \tilde{\Theta}(\text{ROUTING}(S)).$$

Furthermore, routing algorithms (and, by extension, moving cuts) are intricately related to shortcuts. We first extend shortcuts to (not necessarily connected) pairs in the straightforward way: given a set of pairs $S = \{\{s_i, t_i\}\}_{i=1}^{k}$ we say that a set of paths $\{H_i\}_{i=1}^{k}$ with endpoints $\{s_i, t_i\}$ are q-quality shortcuts if both their dilation (longest path) and congestion (maximum number of paths containing the same edge) are at most q. We define $\text{SHORTCUTQUALITY}_2(S)$ as the minimum shortcut quality achievable for S. The seminal work of Leighton et al. [99] relates (pairwise) shortcuts and routing algorithms.

Lemma 6.2.5. *([99]) For any set of pairs S, we have that*

$$\text{ROUTING}(S) = \tilde{\Theta}(\text{SHORTCUTQUALITY}_2(S)).$$

Note that the above statements hold for all sets of pairs. However, in this chapter we will mostly be concerned with sets of pairs S for which there exist vertex-disjoint paths connecting them, which we refer to as *connectable* pairs. We argue that worst-case connectable pairs characterize distributed optimization, hence we define $\text{MOVINGCUT}(G) := \max\{\text{MOVINGCUT}(S) \mid S \text{ is connectable}\}$, and analogously for $\text{COMMUNICATING}(G)$, $\text{ROUTING}(G)$ and $\text{SHORTCUTQUALITY}_2(G)$.

6.3 Our Lower Bound

In this section we present our proof of our universal lower bounds in terms of shortcut quality. In particular, this section is dedicated to proving the following theorem.

Theorem 6.3.1. *The time to solve the spanning connected subgraph problem in any graph G by a randomized algorithm which knows G is at least*

$$T_{conn}(G) = \tilde{\Omega}(\text{SHORTCUTQUALITY}(G)).$$

We defer most proofs of this section to Section 6.4 and Section 6.7, focusing on a high-level overview. We start by introducing *disjointness gadgets*, which are pathological sub-graphs for distributed optimization, and outline their use in proving distributed lower bounds, in Section 6.3.1. In order to obtain informative lower bounds from these gadgets, we then relate the worst such subgraph to the highest distance of any moving cut in G, $\text{MOVINGCUT}(G)$, in Section 6.3.2. This is the technical meat of the paper, and Section 6.4 is dedicated to proving this relation. We then relate the obtained lower bounds to shortcut quality in Section 6.3.3. Finally, we conclude with the proof of Theorem 6.3.1, as well as discussions of its implications to other distributed problems, in Section 6.3.4.

6.3.1 Lower bound witnesses

In this section we define *β-disjointness gadgets*, a structure that connects together information-theoretic bounds with higher-level distributed optimization problems like MST. The structure can be seen as a generalization of previous *existential* lower bounds that show many distributed problems cannot be solved faster than $\widetilde{\Omega}(D + \sqrt{n})$ on a specific graph family [27, 35, 122]. We argue that β-disjointness gadgets are the "right" way to generalize their approaches to arbitrary graphs.

Definition 6.3.2. *A* β-disjointness gadget (P, T, ℓ) *in graph G consists of a set of vertex-disjoint paths $P \neq \emptyset$, each of length at least three; a tree $T \subseteq G$ which intersects each path in P exactly at its endpoint vertices; and a moving cut of capacity strictly less than $|P|$ and distance β with respect to the pairs $\{(s_i, t_i)\}_{i=1}^{|P|}$ of endpoints of paths $p_i \in P$.*

As we show, such disjointness gadgets are precisely the worst-case subgraphs which cause distributed verification (and optimization) to be hard. In particular, denoting by WCSUBNETWORK(G) the highest value of β for which there exists a β-disjointness gadget in G (or zero, if none exists). This quantification the most pathological subgraph in G. We prove the following.

Lemma 6.3.3. *The time to solve spanning connected subgraph verification algorithm in a graph G by any algorithm (even one which knows G) is at least*

$$T_{conn}(G) = \widetilde{\Omega}(\text{WCSUBNETWORK}(G) + D).$$

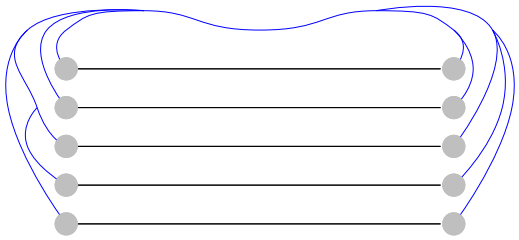


Figure 6.1: A disjointness gadget's path and tree, given by straight and rounded blue lines, respectively.

The non-trivial part of this lemma is the lower bound $T_{conn}(G) = \widetilde{\Omega}(\text{WCSUBNETWORK}(G))$. Our proof of this bound (see Section 6.7) follows the approach implicit in [27]. Broadly, we use a disjointness gadget (P, T, ℓ) to construct a subgraph H determined by private $|P|$-bit inputs x, y for the endpoints of the paths, such that H is a spanning and connected subgraph of G if and only if $\text{disj}(x, y) = 1$. Combined with Lemma 6.2.3, this equivalence and the moving cut ℓ yield a lower bound on subgraph connectivity in a graph G containing a β-disjointness gadget. In following sections we show how to use this bound to prove lower bounds for this problem in *any* graph G.

6.3.2 Disjointness gadgets in any graph

The first challenge in deriving an informative lower bound on the time for spanning connected subgraph verification from Lemma 6.3.3 is that graphs need not contain disjointness gadgets. For example, as disjointness gadgets induce cycles, trivially no such gadgets exist in a tree. Consequently, for trees Lemma 6.3.3 only recreates the trivial lower bound of $\widetilde{\Omega}(D)$.

The following theorem implies that for any graphs where $\text{MOVINGCUT}(G)$ is sufficiently larger than D, disjointness gadgets *do* exist. More precisely, we prove the following theorem.

Theorem 6.3.4. *For any graph G,*

$$\text{WCSUBNETWORK}(G) + D = \tilde{\Theta}(\text{MOVINGCUT}(G)).$$

Theorem 6.3.4 is the technical core of this chapter, and Section 6.4 is dedicated to its proof. At a (very) high level, what we prove there is that, while disjointness gadgets do not always exist, some relaxation of them always does. In particular, we show that for any graph G and set of connectable pairs S in G, some relaxed notion of disjointness gadgets always exists for a subset $S' \subseteq S$ of size $|S'| = \Omega(|S|)$. The majority of Section 6.4 is dedicated to proving the existence of such relaxed disjointness gadgets. We then show how to extend a moving cut of distance $\beta \geq 9D$ on S to (strict) disjointness gadgets: construct a relaxed disjointness gadget on a large subset of S (since S are connectable), then clean-up the structure using $\beta \geq 9D$ to transform it to a (strict) β-disjointness gadget.

6.3.3 Relating MOVINGCUT(G) to SHORTCUTQUALITY(G)

So far we have shown that (up to polylog multiplicative terms and additive $O(D)$ terms), the time to solve subgraph connectivity is at least the length of the worst moving cut in G, which we denote by MOVINGCUT(G). More precisely, so far we proved that

$$T_{conn}(G) \geq \tilde{\Theta}(\text{WCSUBNETWORK}(G) + D) = \tilde{\Theta}(\text{MOVINGCUT}(G)).$$

In this section we show that the above terms we have proven to be equivalent (up to polylog factors) are in turn equivalent to the graph's shortcut quality.

Indeed, by lemmas 6.2.4 and 6.2.5, we have that $\text{MOVINGCUT}(G) = \Theta(\text{SHORTCUTQUALITY}_2(G))$. The following lemma proves an equivalence (up to polylog factors) between shortcut quality for pairs to the graph's shortcut quality (for parts).

Lemma 6.3.5. *For any graph G,*

$$\text{SHORTCUTQUALITY}(G) = \tilde{\Theta}(\text{SHORTCUTQUALITY}_2(G)).$$

Broadly, we use heavy-light decompositions [133] of spanning trees of parts, to show how to obtain shortcuts for parts by gluing together a polylogarithmic number of shortcuts for connected pairs. The overall dilation and congestion of the obtained shortcuts for the parts are at most polylogarithmically worse than those of the shortcuts for the pairs. (See Section 6.7.2 for proof.)

6.3.4 Putting it all together

In this section we review our main result, whereby shortcut quality serves as a universal lower bound for the spanning connected subgraph problem, as well as numerous other problems.

Theorem 6.3.1. *The time to solve the spanning connected subgraph problem in any graph G by a randomized algorithm which knows G is at least*

$$T_{conn}(G) = \tilde{\Omega}(\text{SHORTCUTQUALITY}(G)).$$

Proof. Putting all the lemmas above together, we have

$$\begin{aligned}
T_{conn}(G) &\geq \tilde{\Theta}(\text{WCSUBNETWORK}(G) + D) && Lemma\ 6.3.3 \\
&= \tilde{\Theta}(\text{MOVINGCUT}(G)) && Theorem\ 6.3.4 \\
&= \tilde{\Theta}(\text{COMMUNICATING}(G)) && Lemma\ 6.2.4 \\
&= \tilde{\Theta}(\text{ROUTING}(G)) && Lemma\ 6.2.4 \\
&= \tilde{\Theta}(\text{SHORTCUTQUALITY}_2(G)) && Lemma\ 6.2.5 \\
&= \tilde{\Theta}(\text{SHORTCUTQUALITY}(G)) && Lemma\ 6.3.5
\end{aligned}$$ □

Known reductions presented in Das Sarma et al. [27] extend the same universal lower bounds of Theorem 6.3.1 to numerous problems such as the MST, shallow-light tree, SSSP, min-cut and others. The reductions hold for both non-trivial approximation factors as well as randomized algorithms.

6.4 Constructing Disjointness Gadgets

The goal of this section is to prove Theorem 6.3.4, namely that WCSUBNETWORK$(G) + D$ and MOVINGCUT(G) are equivalent, up to $\widetilde{O}(1)$ factors. At the heart of this proof is a lemma that extends a moving cut between connectable pairs to a disjointness gadget. The proof is fairly involved—which is why we first provide an abbreviated summary before presenting the formal argument.

6.4.1 Technical overview

At a high level, the proof defines two auxiliary structures: *crowns* and *relaxed disjointness structures* (both defined below). Given k connectable pairs, we first show that one can always construct a crown on a large, $\Omega(k)$-sized, subset of these pairs. Next, we show that one can always construct relaxed disjointness gadgets on an $\Omega(k)$-sized subset of the pairs, by converting a crown to a relaxed disjointness gadget. Finally, to obtain a (strict) disjointness gadget, which also requires a moving cut, we show how to construct a disjointness gadget by considering a moving cut between connectable pairs and then adapting a (relaxed) disjointness structure on a large fraction of those pairs.

Crowns. As crowns have a somewhat technical definition, we start by motivating their definition.

Given k vertex-disjoint paths $\{\mathrm{p}_i\}_{i=1}^{k}$ in G, we call the p_i's **part-paths** and the indices i **parts**. Suppose that the following (false) statement holds: "One can always find a connected subgraph $T \subseteq E(G)$ that touches $\widetilde{\Omega}(k)$ part-paths". By touching we mean that exactly one node and zero edges lie in the intersection of p_i and T. Such a structure would be highly interesting—it would show, in an analogous fashion to Lemma 6.3.3, that a moving cut on k connectable pairs can be extended to a universal lower bound for the SSSP problem, by reducing disjointness to SSSP using this structure, and then appealing to Lemma 6.2.3.

Unfortunately, the statement as written does not hold (e.g., when G is a path), but can be relaxed in a way that is both true and does not break the reduction: we allow the intersection of T and p_i to be coverable by a sub-path of p_i of length at most D—the graph diameter. This makes the path example trivial and changes the reduction up to a multiplicative constant and additive $\widetilde{O}(D)$ factors, both of which are insignificant in the context of this chapter. This relaxed structure is a (global) crown. However, constructing global crowns is challenging; our definition is essentially a local version of the above relaxed structure.

Definition 6.4.1 (Crown)**.** *Let $\{\mathrm{p}_i\}_{i=1}^{k}$ be a set of vertex-disjoint paths in a graph G of diameter D. A triplet (T, A, U), where $T \subseteq G$ is a connected subset of edges in G, and $U \subseteq A \subseteq [k]$ are two subsets of parts, is a* ***crown*** *if the following properties hold:*

1. $|U| \geq \frac{1}{4}|A| + 2$.
2. *T only intersects parts in A. More precisely, $V(T) \cap V(\mathrm{p}_i) = \emptyset$ for all $i \in [k] \setminus A$.*
3. *T intersects each part $i \in U$, and this (non-empty) intersection, $V(T) \cap V(\mathrm{p}_i)$, can be covered by a single sub-path of p_i of length at most D.*

We use the following crown terminology for expressiveness (see Figure 6.2). We say that part i *belongs* to crown (T, A, U) if $i \in A$; i is *useful* if $i \in U$; i is *sacrificial* if $i \in A \setminus U$ (note: A stands for "all", U stands for "useful"). While not a part of the definition, for the crowns we consider, the sacrificial parts will always be fully contained in T, i.e., if $i \in A \setminus U$, then $E(\mathrm{p}_i) \subseteq T$.

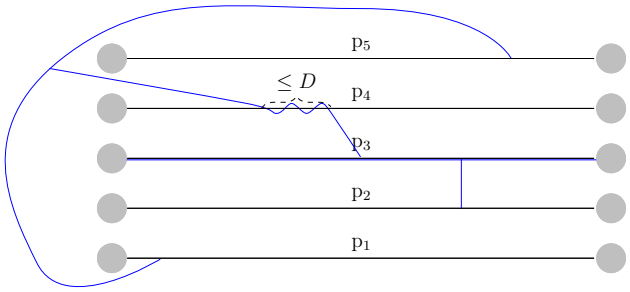


Figure 6.2: A crown $(T, \{1, 2, 3, 4, 5\}, \{1, 2, 4, 5\})$. T is depicted in blue. Note that part 3 is sacrificial, hence is contained in T. The intersection of T and p_4 is covered by a sub-path of length at most D. Intersections with other useful part-paths are covered by a trivial sub-path of length 0.

We say that two crowns (T_1, A_1, U_1) and (T_2, A_2, U_2) are *disjoint* if $A_1 \cap A_2 = \emptyset$ (though T_1 and T_2 can intersect arbitrarily). A set of crowns is disjoint if every pair is disjoint. The definition of crowns implies that a set of disjoint crowns can be easily merged into a single crown, as follows. Consider a path q between two crowns (T_1, A_1, U_1) and (T_2, A_2, U_2) that does not touch any other crown (such a path must exist). Merge the two crowns via $(T_1 \cup T_2 \cup E(q), A_1 \cup A_2, U_1 \cup U_2)$ and declare all part-paths that q touches sacrificial (at most 2). All crown properties continue to hold.

We take a step back and compare the crown definition with the above SSSP motivation. In order for the reduction to work, it is sufficient to prove that "for every k part-paths one can construct a crown (T, A, U) with $|A| \geq \Omega(k)$' (while only useful parts have a bounded intersection and can be used in the reduction, Property 1 of crowns relaxes this requirement to the previous statement). Combining this with the merging property, it is sufficient to show that "for every set of k part-paths one can partition $\Omega(k)$ of them into crowns". We argue this is true by considering two

sub-cases, phrased in terms of the *contraction graph* R, whose vertices are parts $[k]$ whose edges $\{i, j\}$ corresponds to pairs of indices with a path (in G) between p_i and p_j that does not touch any other part-path. In the *(high-degree case)*, $\Omega(k)$ part-paths are adjacent to at least three other part-paths, and in the *(low-degree case)* when almost all part-paths are adjacent to at most two other part-paths.

High-degree case. We consider the case when $\Omega(k)$ parts have R-degree at least three. Pick a part i and seed (i.e., start) a crown from it: declare i sacrificial and all of its R-neighbors useful. Continue growing the crown as long as a useful part j with at least two "unused" R-neighbors exists, in which case declare j sacrificial and its neighbors to the useful parts to the crown. When we cannot grow the crown anymore we add it to the collection, delete the used parts from the graph and repeat the process with another seed i that has three unused neighbors. It is easy to argue that $\Omega(k)$ parts belong to some crown: an unused part i of R-degree three or more has at least one neighboring part j in some crown (otherwise i would start its own crown), and j is a useful part in its crown (otherwise j's crown would absorb i). However, useful parts belonging to some crown can have only one unused neighbor (otherwise they would grow a crown), hence we can charge each unused part to a unique useful part in a crown. This shows that only a small fraction of parts of degree at least three can be unused.

Low-degree case. We illustrate our techniques on the case when *all* vertices in R have degree two (the formal proof handles the case when *most* nodes have this property). We decompose R into cycles and paths and construct crowns on a constant fraction of each. For an R-path, we show one can construct a crown on any three consecutive parts (a, b, c), thereby proving one can construct a crown on a constant fraction of a R-path (of length at least three). Fix (a, b, c) and consider the shortest path f from any node in p_a to any node in p_c; the length of f is at most the diameter of G, namely D. Note that f intersects p_b, but if the intersection is coverable by a sub-path of length at most D then one can make a crown $(E(f), \{a, b, c\}, \{a, b, c\})$. If this is not the case, we can "replace" the part of p_b between the first and last intersection with f with the appropriate part of f, forcing the intersection to be coverable by a short sub-path; this proves one can construct a crown on $\{a, b, c\}$. Cycles can be handled similarly. Combining both cases, we conclude that for every set of k part-paths, there exists a crown on some $\Omega(k)$-sized subset of them.

Converting a crown into a relaxed disjointness gadget. Relaxed disjointness gadgets are an intermediate step between crowns and disjointness gadgets. Relaxed disjointness gadgets require both endpoints of part-paths to be included in T (like a disjointness gadget), but also allow for three exceptional sub-paths of length at most D on each part-path (unlike crowns that allow only one). Moreover, relaxed disjointness gadgets do not require a moving cut.

Definition 6.4.2. *A **relaxed disjointness gadget** (P, T) in graph G of diameter D consists of vertex-disjoint paths $P = \{\mathrm{p}_i\}_{i=1}^{k}$ and a connected subset of edges $T \subseteq E(G)$ which*

intersects all paths at their endpoint vertices, and such that for each $i \in |P|$, $V(T) \cap V(\mathrm{p}_i)$ can be covered by at most three sub-paths of p_i of length at most D. We say that the ***endpoints*** *of (P, T) are the endpoints of $\{\mathrm{p}_i\}_{i=1}^{k}$.*

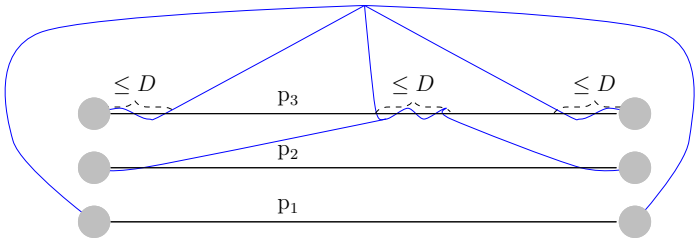


Figure 6.3: A relaxed disjointness gadget. The edges of the paths and T are horizontal and blue lines, respectively. The intersection of p_3 and T is covered by three sub-paths of length at most D.

We show that "for every crown (T, A, U) on k parts, one can construct a relaxed disjointness gadget on $\Omega(k)$ parts". The proof sketch is fairly simple: we need to "connect" the endpoints $\{s_i, t_i\}$ of $\Omega(k)$ parts to T. Fix some i and find the shortest path from each endpoint of p_i to T. If both s_i and t_i can connect to T without intersecting another part-path p_j, we include the connections in T and continue to the next i. If this is not the case, we can either connect i and sacrifice (at most two) other part-paths, or not include i. One can always choose a constant fraction of i's that can be simultaneously connected to T: consider the "interference graph" that has a directed edge $i \to j$ if connecting i sacrifices j. The graph has out-degree at most two, hence total (in+out) average degree at most four; by Turan's theorem, there exists an independent set of size $k/5$, which can be simultaneously connected and, conceptually, we are done. There are significant technical challenges that lie beyond this simple sketch: the connections from s_i and t_i could arbitrarily intersect p_i (i.e., not be coverable by a sub-path of length at most D). This can be dealt with by path-replacement strategies such as the ones in the low-degree crown construction. However, a replacement could start to intersect other connections which are not accounted for in the interference graph, requiring special care.

Converting a relaxed disjointness gadget to a disjointness gadget. To prove Theorem 6.3.4 we need to bring together moving cuts and relaxed disjointness structures. Suppose we are given a moving cut ℓ between k connectable pairs. As previously argued, we can construct a relaxed disjointness gadget $(\{p_i\}_i, T)$ on $\Omega(k)$ of such pairs. The moving cut ℓ ensures that the ℓ-length of each p_i is at least β (assume $\beta \geq 9D$). One can assume WLOG that $\ell(e) = 1$ for each edge on each part-path. For each p_i we exclude the sub-paths that intersect T (total exclusion has ℓ-length and length at most $3D$); this partitions p_i into two sub-paths, one of which (denoted by p_i') has ℓ-length at least $(\beta - 3D)/2$. Adding $\bigcup_i p_i \setminus p_i'$ to T, we maintain the property that the endpoints of p_i' connect to T and that p_i' does not intersect T internally. This almost completes the disjointness structure construction. Denote the endpoints of p_i' with s_i', t_i'. The final property we need to ensure is that $\mathrm{dist}_\ell(s_{\boldsymbol{i}}', t_{\boldsymbol{j}}') \geq \widetilde{\Omega}(\beta)$ for all i, j; instead, we only ensured that $\mathrm{dist}_\ell(s_i', t_i') \geq \widetilde{\Omega}(\beta)$ for all i. However, going between one such distance guarantee and the other can be done via a

structural lemma from prior work by losing an extra $O(\log n)$ factor in β [69, Lemma 2.5]. This completes the full proof.

6.4.2 Constructing crowns

In this section we show that large crowns exist in all graphs (see Definition 6.4.1). We first describe the crown merging procedure and show how to construct them. The construction is partitioned into the high-degree case and the low-degree case. We start by defining some notation we will be using throughout this section.

Notation specific to Section 6.4. We denote the disjoint union via $A \sqcup B$, where we guarantee $A \cap B = \emptyset$. For a graph H, we denote by $V(H)$ and $E(H)$ the vertices and edges of H, respectively. For a set of edges $T \subseteq E(H)$, we denote by $V(T) := \bigcup_{e \in T} e$ the set of endpoints of edges in T. Similarly, for a path p we denote by $V(p)$ the set of nodes on the path (including its endpoints).

We denote the degree of a vertex v in graph H by $\deg_H(v)$, and its neighborhood in H by $\Gamma_H(v)$. We extend the neighborhood definition to sets of vertices $X \subseteq V(H)$, letting $\Gamma_H(X) := \bigcup_{v \in X} \Gamma_H(v)$ (i.e., the set of vertices in $V(H)$ that have a neighbor in X). Furthermore, we denote the inclusive and exclusive neighborhoods by $\Gamma^+_H(X) := \Gamma_H(X) \cup X$ and $\Gamma^-_H(X) := \Gamma_H(X) \setminus X$.

A *walk* in H is a sequence of vertices $w = (w_0, w_1, \ldots, w_\ell)$, where $w_i \in V(H)$ and $\{w_i, w_{i+1}\} \in E(H)$ for all $0 \leq i \leq \ell - 1$. The indices i are often called *steps*. The length of the walk is denoted by $|w| := \ell$. Given two steps $i \leq j$ we denote by $w_{[i,j]}$ the *subwalk* $(w_i, w_{i+1}, \ldots, w_j)$. Furthermore, we define $E(w)$ as the set of edges the walk traverses over and we define $V(w) := \bigcup_{i=0}^{\ell} \{w_i\}$. Given two nodes $u, v \in V(H)$ the distance $\mathrm{dist}_H(u, v)$ is the length of the shortest walk between them. We extend the distance definition to sets $X, Y \subseteq V(H)$ in the natural way, letting $\mathrm{dist}_H(X, Y) := \min_{x \in X, y \in Y} \mathrm{dist}_H(x, y)$.

We define walk *clipping*. For a walk w in graph H, and two sets $A, B \subseteq V(H)$, we denote the subwalk of w from the last step $i := \max\{i \mid w_i \in A\}$ corresponding to a vertex in A to the first step $j := \min\{j \mid w_j \in B, j > i\}$ corresponding to a vertex in B by $\mathrm{clip}(w, A, B) := w_{[i,j]}$. We note that the clipping operation is well defined only when after each step i where $w_i \in A$ there exists a step $j \geq i$ such that $w_j \in B$. This will always be the case when we use this operation. (This is, for instance, true if $w_0 \in A$ and $w_\ell \in B$.)

The definition of crowns allows for disjoint crowns to merged rather directly, by sacrificing at most one part in each crown in order to merge the trees. See Section 6.8 for a full proof.

Lemma 6.4.3. *(Crown merging) Let $\{(T_i, A_i, U_i)\}_{i \in I}$ be a set of disjoint crowns of vertex-disjoint paths $\{\mathrm{p}_i\}_{i=1}^k$. Then there exists a single crown (T_*, A_*, U_*) of $\{\mathrm{p}_i\}_{i \in A_*}$ such that $A_* = \bigsqcup_{i \in I} A_i$.*

Contraction graph

In this section we define the contraction graph which we will extensively used in the crown construction.

Definition 6.4.4 (Contraction graph)**.** *A **contraction graph** R with respect to k fixed vertex-disjoint paths $\{\mathrm{p}_i\}_{i=1}^{k}$ has vertex set $V(R) = [k]$. The edges of R consist of all pairs $\{i, j\}$ such that there exists a simple path $(v_0, v_1, \ldots, v_\ell)$ in G where $v_0 \in V(\mathrm{p}_i)$, $v_\ell \in V(\mathrm{p}_j)$, and the internal nodes do not belong to any part-path, i.e. $\bigcup_{x=1}^{\ell-1}\{v_x\} \cap \bigcup_{y=1}^{k} V(\mathrm{p}_y) = \emptyset$. We denote this path by $E_G(\{i, j\})$.*

Closely related to the contraction graph, is the **projection** $\pi_{G\to R}$, which is a mapping between walks $w = (v_0, v_1, \ldots, v_\ell)$ in G and walks in R. We first fix a $v \in V(G)$ and define $\pi_{G\to R}(v) = x$ if $v \in \mathrm{p}_x$ and $\pi_{G\to R}(v) = \perp$ if v does not belong to any part-path (note that our definition is well-defined because the part-paths are disjoint). We now extend the function to walks $w = (v_0, \ldots, v_\ell)$ on G. We define $w' := (\pi_{G\to R}(v_0), \pi_{G\to R}(v_1), \ldots, \pi_{G\to R}(v_\ell))$. Furthermore, we let w'' be the sequence w' with elements $\perp$ removed, and consecutive duplicate elements merged down to a single element. For instance, if $w' = (5, 1, \perp, 1, 1, \perp, 2, 2, \perp, 2, 3\}$, then $w'' = (5, 1, 2, 3)$. The projection $\pi_{G\to R}$ maps $w \mapsto w''$. It is easy to see that w'' is a walk on R.

Observation 6.4.5. *The contraction graph R is connected.*

Proof. Let $a, b \in V(R)$ be parts and let a', b' be arbitrary nodes satisfying $a' \in \mathrm{p}_a, b' \in \mathrm{p}_b$. Since G is connected, there exists a walk w connecting a' and b'. Projecting the walk to R, $\pi_{G\to R}(w)$ is a walk in R connecting a and b, proving the claim. □

Constructing crowns in the high-degree case

In this section we consider the case where a large fraction of parts either have R-degree of at least three or a neighbor with this property. In this case, combining with the crown-merging Lemma 6.4.3, we can successfully construct a global crown on a constant fraction of parts.

Lemma 6.4.6. *Let R be a contraction graph R, and let $H = \{v \in V(R) \mid \deg_R(v) \geq 3\}$. If $|\Gamma_R^+(H)| \geq \frac{1}{10}k$, then there exists a collection of disjoint crowns $\mathcal{C} = \{(T_i, A_i, U_i)\}_{i\in I}$, with $\sum_{i\in I} |A_i| \geq \frac{1}{70}k$.*

Proof. We perform an iterative procedure to find the required collection. Let be $\mathcal{C}$ an (initially empty) list of disjoint valid crowns. Initially, we let the set of "available parts" be $L \leftarrow V(R)$ (i.e., all parts).

The procedure repeatedly "seeds" a new crown and then it proceeds to "grow" it until no longer possible. We describe the seeding procedure. Find a part $v \in L$ with at least three C-neighbors in L (i.e. $|\Gamma_R(v) \cap L| \geq 3$); the procedure stops when no viable v can be found. We seed a new crown (T, A, U) from v: We define $A \leftarrow \Gamma_R^+(v) \cap L$, and $L \leftarrow L \setminus A$. The part v is sacrificial

and the R-neighbors of v are useful, i.e., $U \leftarrow A \setminus \{v\}$. We assign $T \subseteq E(G)$ to the union of $E(\mathrm{p}_v)$ (edges in G corresponding to the part-path p_v) and $\bigcup_{w \in U} E_G(\{v, w\})$ (all of the simple paths corresponding to R-edges between v and its useful neighbors). This completes the seeing portion.

We proceed to "grow" the new crown. Find a useful part $w \in U$ in the current crown with at least two R-neighbors in L (i.e., $|\Gamma_R(w) \cap L| \geq 2$). Let $X \leftarrow \Gamma_R(w) \cap L$. We add these neighbors to the crown and sacrifice w: $A \leftarrow A \cup X$, $U \leftarrow U \setminus \{w\} \cup X$ and $L \leftarrow L \setminus A$. We add to T the part-path corresponding to w and the simple paths connecting w to the new useful parts $T \leftarrow T \cup E(\mathrm{p}_w) \cup \bigcup_{x \in X} E_G(\{w, x\})$. We repeat the growing step until no viable w can be found. At that point we add (T, A, U) to $\mathcal{C}$ and the seeding procedure is restarted.

Suppose the above procedure yields crowns $\mathcal{C} = \{(T_i, A_i, U_i)\}_{i \in I}$. We proceed to analyze it. Note that by construction $V(R) = L \sqcup \left(\bigcup_{i \in I} A_i\right)$ where $\sqcup$ denoted disjoint union. Let us define $H = H_{in} \sqcup H_{out}$ (disjoint union) where H_{in} are the parts of H that are inside some crown ($H_{in} = H \cap (\bigcup_i A_i)$) and H_{out} outside $H_{out} = H \cap L$, therefore we have $|H| = |H_{in}| + |H_{out}|$.

Subclaim: we argue that $|\bigcup_{i \in I} A_i| \geq \frac{1}{2}|H|$. However, we first establish that $|H_{out}| \leq |\bigcup_{i \in I} A_i|$ via a charging argument from H_{out} to $\bigcup_{i \in I} A_i$. Let $h \in H_{out}$. Since $h \in L$ it must have at least one neighbor v that belongs to some crown (T, A, U) (choose an arbitrary v if multiple neighbors fit the condition); if this were not the case then $|\Gamma_R(h)| \geq 3$ would ensure that a crown would be seeded from h. Furthermore, it must be that $v \in U$, since sacrificial parts absorb all of their available neighbors when joining a crown. We "charge" h to v. On the other hand, fix a part c in some crown. Note that at most one $h \in H_{out}$ can charge itself to c: if two parts in H_{out} charge themselves to c, we would grow the crown via c (note that v must be a useful part in its crown if anyone charges to it since sacrificial parts absorb their available neighbors). Since each $h \in H_{out}$ can be charged to a unique part in $\bigcup_{i \in I} A_i$ we conclude that $|\bigcup_{i \in I} A_i| \geq |H_{out}|$. We complete the subclaim by noting that $|\bigcup_{i \in I} A_i| \geq |H_{in}| = |H| - |H_{out}| \geq |H| - |\bigcup_{i \in I} A_i|$, which can be rewritten as $|\bigcup_{i \in I} A_i| \geq \frac{1}{2}|H|$.

Let use define $M := \Gamma_R(H) \setminus H$. Furthermore, partition $M = M_{in} \sqcup M_{out}$ where $M_{in} = M \cap (\bigcup_{i \in I} A_i)$ and $M_{out} = M \cap L$. Subclaim: we prove that $|H| \geq \frac{1}{2}|M_{out}|$ by charging elements of M_{out} to H in such a way that at most two elements of M_{out} get charged to the same H. Fix $v \in M_{out} \subseteq \Gamma_R(H) \cap L$. By definition of $\Gamma_R(H)$, there exists $h \in \Gamma_R(v) \cap H$; we charge v to h. We now argue that at most two different parts can charge the same h. First, note that a part $h \in H$ that is being charged to by at least one part must be either available (i.e., in L), or a useful part in some crown (since sacrificial parts in a crown consume all of their available neighbors). Assume, for the sake of contradiction, that a node h is charged more than two times. Then the construction would either seed a crown (if $h \in L$) or grow a crown via h. We conclude that $|H| \geq \frac{1}{2}|M_{out}|$.

We finalize the proof using the two subclaims and $M_{in} \subseteq \bigcup_{i \in I} A_i$:

$$\begin{aligned}\frac{1}{10}k \leq |\Gamma_R^+(H)| &= |H| + |M_{in}| + |M_{out}| \\ &\leq |H| + |\bigcup_{i \in I} A_i| + 2|H| \leq (2 + 1 + 4)|\bigcup_{i \in I} A_i| = 7|\bigcup_{i \in I} A_i|.\end{aligned}$$

This can be rewritten as $|\bigcup_{i\in I} A_i| = \sum_{i\in I} |A_i| \geq \frac{1}{70}k$. □

Constructing crowns in the low-degree case

In this section we consider the second case, where a large fraction of parts have R-degree at most two.

We start by introducing the notion of **minimal part-paths** with endpoints $S := \{(s_i, t_i)\}_{i=1}^{k}$. A set of vertex-disjoint paths $\{\mathrm{p}_i\}_{i=1}^{k}$ with endpoints S is called *minimal* if they minimize $\sum_{i\in[k]} |\mathrm{p}_i|$ among all such sets with endpoints S.

Lemma 6.4.7. *Let R be a contraction graph with respect to some minimal part-paths, and let $H = \{v \in V(R) \mid \deg_R(v) \geq 3\}$. If $|\Gamma_R^+(H)| \leq \frac{1}{10}k$, then there exists a collection of disjoint crowns $\mathcal{C} = \{(T_i, A_i, U_i)\}_{i\in I}$ with $\sum_{i\in I} |A_i| \geq \frac{3}{100}k$.*

The rest of this section will function as a proof of Lemma 6.4.7. This will allow us to break down its complexity into smaller pieces.

We define $M_0 := \Gamma_R^+(H)$, $M_i := \Gamma^-(M_{i-1})$ for $i \in \{1, \ldots, 8\}$, and $M := \bigcup_{i=0}^{8} M_i$. In particular, for each $m \in M_i$ ($i \in \{0, \ldots, 8\}$) we have that $\mathrm{dist}_R(m, H) = i + 1$.

Claim 6.4.8. *$|M_i| \leq |M_{i-1}|$ for $i = 1, \ldots, 8$.*

Proof. By definition, each $m \in M_i$ is a neighbor of some $m' \in M_{i-1}$ of degree at most two (there might be two choices for m', but in this case we choose arbitrarily). We charge m to m'. At most one part can be charged to m': otherwise m' could not be with distance i from H since then $i = \mathrm{dist}_G(m', H) = \mathrm{dist}_G(\Gamma_G(m'), H) + 1 = i + 2$. □

We now define $O := V(R) \setminus M$. We have that $|M_8| \leq \ldots \leq |M_0| \leq \frac{1}{10}k$, implying that $|M| \leq \frac{9}{10}k$, and giving us that $|O| \geq k - \frac{9}{10}k = \frac{1}{10}k$.

Consider the induced subgraph $R' := R[V(R) \setminus H]$. R is connected (Observation 6.4.5). By definition of H, all parts in this subgraph have R-degree at most two, hence we can decompose R' into paths. Note that cycles are not allowed since they would be an isolated connected component of R. There are two special cases: $k \leq 2$ which is a trivial case, or the entire graph being a cycle in which case we redefine $R' = R[V(R) \setminus \{v\}]$ where v is an arbitrary part and the rest of the proof remains unchanged. Let $\{q_i\}_{i\in J}$ be the collection of paths in R' of length at least 9 (discard all shorter paths), in other words, $|V(q_i)| \geq 10$.

Claim 6.4.9. *$\sum_{i\in J} |V(q_i)| \geq \frac{1}{10}k$.*

Proof. We argue that $O \subseteq \bigcup_{i\in J} V(q_i)$ which would, together with disjointness of $V(q_i) \cap V(q_j) = \emptyset$, imply that $\sum_{i\in J} |V(q_i)| = |\bigcup_{i\in J} V(q_i)| \geq |O| \geq \frac{1}{10}k$. Fix a part $v \in O$. By definition, we have that $\mathrm{dist}_R(v, H) \geq 10$, hence the R'-connected-component that includes v must be a path of length at least 9, leading to the required conclusion. □

We introduce some notation: we say that a crown (T, A, U) is *supported* on a subset of parts $X \subseteq [k]$ if $A \subseteq X$. Similarly, for a path f in R we say that a (T, A, U) is supported on the path if $A \subseteq V(f)$ (note that $V(f) \subseteq V(R) \subseteq [k]$).

Claim 6.4.10. *Let f be a simple path in R with $|V(f)| \geq 10$ and the extra condition that for all $v \in V(f)$ we have $\deg_R(v) \leq 2$. Then, there exists a collection of disjoint crowns supported on f such that $\mathcal{C} = \{(T_i, A_i, U_i)\}_{i \in I'}$ with $\sum_{i \in I'} |A_i| \geq \frac{3}{10}|V(f)|$.*

Proof. Suppose that $f = (f_0, f_1, f_2, \ldots, f_\ell)$ with $\ell = |V(f)| - 1 \geq 9$, where $f_i \in V(R)$ are parts. Let $x = 3\lceil \ell/9 \rceil - 1$. We note that f_0, f_x and f_{2x} are all valid parts since $2x = 6\lceil \ell/9 \rceil - 2 \leq 6\ell/9 + 6 - 2 = \frac{2}{3}\ell + 4 \leq \ell$, where the last inequality holds for $\ell \geq 12$ and can be manually checked for $\ell = 9, 10, 11$ (giving $2x = 4, 10, 10$, respectively). Furthermore, we note that $x + 1 \geq \frac{3}{10}|V(f)|$ for $|V(f)| \geq 10$ since $x + 1 = 3\lceil \ell/9 \rceil \geq \ell/3 = \frac{|V(f)|-1}{3} = \frac{0.9|V(f)|+0.1|V(f)|-1}{3} \geq \frac{0.9|V(f)|}{3} = \frac{3}{10}|V(f)|$.

Subclaim: there exists a simple path w' in G (note: not R) of length at most D whose projection $\pi_{G \to R}$ is supported on f and intersects exactly $x+1$ part-paths. More precisely, $V(\pi_{G \to R}(w')) \subseteq V(f)$ and $|V(\pi_{G \to R}(w'))| = x + 1$. Let $a, b \in V(G)$ be arbitrary nodes on the part-paths corresponding to f_x and f_{2x-1}, respectively. Since the diameter of G is at most D, there exists a simple path $w = (w_0 = a, w_1, \ldots, w_d = b)$ in G from a to b of length at most $d \leq D$. Consider $w' := \mathrm{clip}(w, V(\mathrm{p}_{f_x}), V(\mathrm{p}_{f_0}) \cup V(\mathrm{p}_{f_{2x}}))$; the clipping is well-defined because the walk starts on p_{f_x} and ends on f_{2x}. Furthermore, we claim that $\pi_{G \to R}(w')$ is supported on f: either $\pi_{G \to R}(w')$ is supported on f and the statement is immediate or w' exists f, in which case it must go through p_{f_0} or $\mathrm{p}_{f_{2x}}$; in both cases the path is clipped only to the subwalk supported on f. We conclude that f intersects exactly $x+1$ part-paths since it starts on p_{f_x} and ends in p_{f_0} or $\mathrm{p}_{f_{2x}}$. It must cover all parts in between due to the projection $\pi_{G \to R}$ being a walk on R and cannot cover anything outside do to clipping. This completes the subclaim.

Let $\pi_{G \to R}(w')$ intersect exactly $f' := \{f'_0, f'_1, \ldots, f'_x\}$, where f'_i and f'_{i+1} are consecutive parts on the path f and w' start (resp., end) on a node corresponding to $\mathrm{p}_{f'_0}$ (resp., $\mathrm{p}_{f'_x}$).

We construct crowns on f' by partitioning f' into triplets $(f'_0, f'_1, f'_2), (f'_3, f'_4, f'_5), \ldots, (f'_{x-2}, f'_{x-1}, f'_x)$ (note that $3 \mid x + 1$, i.e., $x + 1$ is divisible by 3). Fix a triplet $(f'_i, f'_{i+1}, f'_{i+2})$ for $3|i$ and construct a crown on it as follows.

Let $w^i := \mathrm{clip}(w', V(\mathrm{p}_{f'_i}), V(\mathrm{p}_{f'_{i+2}}))$ and note that $|w^i| \leq D$. The clipping is well defined because since every subwalk of w' that passes through $\mathrm{p}_{f'_i}$ must eventually cross over $\mathrm{p}_{f'_{i+2}}$ before ending at $\mathrm{p}_{f'_x}$. Let $\tilde{w}^i$ be the subwalk of w^i between the first and last occurrence of a node that is on $\mathrm{p}_{f'_{i+1}}$; this is well-defined since w^i intersects $\mathrm{p}_{f'_{i+1}}$. Let u, v be the first and last node of $\tilde{w}^i$. Note that, because of path minimality, $\mathrm{p}_{f'_{i+1}}$ connects u and v via some shortest path in the subgraph of G that excludes the other part-paths (otherwise we could shorten the p_i); call this sub-path q. This is because each sub-path of the shortest path (e.g., q) is also a shortest path. On the other hand, $\tilde{w}^i$ is some walk connecting u and v that does not touch any other part-path outside of $V(\mathrm{p}_{f'_{i+1}})$. Therefore, the length of $\tilde{w}^i$ cannot be smaller than q. This allows us to swap the subwalk of w^i corresponding to $\tilde{w}^i$ with q without increasing the length of the walk w^i. In the reminder we assume we have performed this replacement.

We construct a crown (T, A, U) by assigning $T \leftarrow w^i$, $A \leftarrow U \leftarrow \{f'_i, f'_{i+1}, f'_{i+2}\}$ and verify it is a valid crown. T is connected since it is a walk. Property 1 is satisfied since $3 = |U| \geq \frac{1}{4}|A|+2 = \frac{3}{4}+2$. Property 2: considering $\pi_{G\to R}(w^i) \subseteq \{f'_i, f'_{i+1}, f'_{i+2}\}$ we conclude that no part-path outside of A is intersected. Property 3: by the clipping, T intersects $\mathrm{p}_{f'_i}$ and $\mathrm{p}_{f'_{i+2}}$ in a single node. Furthermore, T intersects $\mathrm{p}_{f'_{i+1}}$ in at most D consecutive nodes due to the replacement; the property is satisfied.

This concludes the proof because we constructed valid disjoint crowns containing (in union) exactly $x + 1 \geq \frac{3}{10}|V(f)|$ parts. □

We finalize the proof of the main result of this section by applying Claim 6.4.10 to all $\{q_i\}_{i\in J}$ and concatenating the collections of crowns constructed this way (they are clearly disjoint since they are supported on disjoint paths of R). We establish a disjoint collection of crowns $\mathcal{C} = \{(T_i, A_i, U_i)\}_{i\in I}$ with $\sum_{i\in I} |A_i| \geq \frac{1}{10}k \cdot \frac{3}{10} = \frac{3}{100}k$. This completes the proof of Lemma 6.4.7. ■

Finalizing the crown construction

Combining the high-degree case (Lemma 6.4.6) and the low-degree case (Lemma 6.4.7) with crown merging (Lemma 6.4.3) we conclude the crown construction with the following result.

Lemma 6.4.11. *For every set of k minimal part-paths $\{\mathrm{p}_i\}_{i=1}^k$, there always exists a crown (T, A, U) with respect to $\{\mathrm{p}_i\}_{i\in A}$, where $U \subseteq A \subseteq [k]$ and $|U| \geq \frac{1}{280}k$.*

Proof. Let R be the contraction graph of $\{\mathrm{p}_i\}_{i=1}^k$. By applying Lemma 6.4.6 and Lemma 6.4.7 to R, we can find a set of disjoint crowns $\mathcal{C} = \{(T_i, A_i, U_i)\}_{i\in I}$ where $\sum_{i\in I} |A_i| \geq \min(\frac{1}{70}k, \frac{3}{100}k) = \frac{1}{70}k$. Merging the crowns via Lemma 6.4.3 we construct a single crown (T_*, A_*, U_*) of $\{\mathrm{p}_i\}_{i\in A_*}$ satisfying $|A_*| = \sum_{i\in I} |A_i| \geq \frac{1}{70}k$. We deduce that $|U_*| \geq \frac{1}{4}|A_*| + 2 \geq \frac{1}{280}k$ by Property 1 of crowns. □

6.4.3 Converting crowns into relaxed disjointness gadget

In this section we prove the following result.

Lemma 6.4.12. *Given a set of k connectable pairs $\{(s_i, t_i)\}_{i=1}^k$, there always exists a subset $U \subseteq [k]$ of size $|U| \geq \frac{1}{1400}k$ and a relaxed disjointness gadget (P, T) with endpoints $\{(s_i, t_i)\}_{i\in U}$.*

We prove this result by the following lemma, which converts a large crown (in terms of number of parts belonging to a it) into a large relaxed disjointness gadget (Definition 6.4.2).

Lemma 6.4.13. *Let (T, A, U) be a crown with respect to $\{\mathrm{p}_i\}_{i=1}^k$ with endpoints $\{(s_i, t_i)\}_{i=1}^k$. There exists a subset $U' \subseteq U$ of size $|U'| \geq \frac{1}{5}|U|$ and a relaxed disjointness gadget (P, T)*

with endpoints $\{(s_i, t_i)\}_{i \in U'}$.

Proof. We build a directed "interference graph" I: the vertices correspond to parts U and the outgoing edges of a part $i \in U$ are defined by the following.

For every $i \in U$ let ps_i be the shortest walk in G from s_i to the closest vertex in $V(T)$. If ps_i touches another part-path, we clip-off anything beyond the first touch, i.e., $\mathrm{ps}_i \leftarrow \mathrm{clip}(\mathrm{ps}_i, \{s_i\}, \bigcup_{j \in U, j \neq i} V(\mathrm{p}_j))$. In this case, let j be the part index ps_i touches. We add the directed edge $i \to j$ to the interference graph I. Finally, we "associate" the walk ps_i to part i (regardless of whether it touches another part or not).

We repeat the exact same steps for t_i: let pt_i be the shortest walk from t_i to $V(T)$; we clip-off a suffix and add an edge $i \to j'$, if needed; finally, associate pt_i with part i. Exactly two walks are associated with each part.

Let I' be the undirected version of I (directed edges $i \to j$ are transformed to undirected edges $\{i, j\}$). Since the out-degree of I is at most two, the average degree of I' is at most 4. Then, by Turan's theorem, there exists an independent set $U' \subseteq U$ in I' with $|U'| \geq \frac{1}{1+d}k \geq \frac{1}{5}k$ where $d \leq 4$ is the average degree. We will call the parts $U \setminus U'$ (i.e., outside of the independent set) "sacrificial".

We initialize T' with by adding all part-paths of sacrificial parts to T, i.e., $T' \leftarrow T \cup \bigcup_{j \in U \setminus U'} E(\mathrm{p}_j)$. Note that this T' is connected since the crown ensures T touches each part-path. We will later adjust T' and we maintain its connectivity.

As warm-up, suppose that for all $i \in U'$ (called "useful" parts) all walks associated with i do not intersect p_i (except at the starting endpoint of the walk). In this case, for each $i \in U'$ (in arbitrary order) we do the following. Consider each of the two walks f associated with i (in arbitrary order). We reassign $T' \leftarrow T' \cup f$, ensuring $\{s_i, t_i\} \ni f_0 \in V(T')$ (i.e., T' touches an endpoint of p_i). We also maintain the connectivity of T' because either $f_{|f|} \in V(T)$ (i.e., no clipping occurred, f connects to T, in which case the claim is obvious) or it connects to another part-path p_j, which necessarily must be sacrificial (i.e., $j \notin U'$) and therefore added to T since $i \in U'$ and U' is an independent set. After the process is finished for all $i \in U'$ we have that $(\{\mathrm{p}_i\}_{i \in U'}, T')$ is a relaxed disjointness gadget: we already argued that connectivity of T' is maintained; $s_i, t_i \in V(T')$ since the two walks associated with $i \in U'$ contain s_i and t_i as endpoints and always get added to T'. Lastly, we argue about covering $V(T') \cap V(\mathrm{p}_i)$: we have that $T' = T \cup \bigcup_{j \in U \setminus U'} E(\mathrm{p}_j) \cup \bigcup_{j \in J} f_j$ where $\{f_j\}_{j \in J}$ is a collection of walks associated with parts in U' (i.e., useful parts). For some $i \in U'$ we have:

$$\begin{aligned} V(T') \cap V(\mathrm{p}_i) &= (V(T) \cap V(\mathrm{p}_i)) \cup \left(\bigcup_{j \in U \setminus U'} V(\mathrm{p}_j) \cap V(\mathrm{p}_i) \right) \cup \left(\bigcup_{j \in J} V(p_j) \cap V(\mathrm{p}_i) \right) \\ &\subseteq (V(T) \cap V(\mathrm{p}_i)) \cup \emptyset \cup \{s_i, t_i\}. \end{aligned}$$

Here we used that $\mathrm{p}_i, \mathrm{p}_j$ are vertex-disjoint for $i, j \in U, i \neq j$; walks $\{f_j\}_{j \in J}$ do not intersect useful part-paths except in endpoints $\{s_i, t_i\}$; $V(T) \cap V(\mathrm{p}_i)$ can be covered by one path of length at most D (from Property 3 of crowns). In conclusion, $V(T') \cap V(\mathrm{p}_i)$ can be covered by

$\{s_i\}, \{t_i\}$ and one sub-path of length at most D; (T', U') is a disjointness gadget. This completes the warm-up.

We now consider general f, i.e., we allow f associated with i to intersect p_i. To achieve this, we will need to iteratively adjust the part-paths. We initialize $\mathrm{p}'_i \leftarrow \mathrm{p}_i,\ \forall i \in U'$ and adjust $\{\mathrm{p}'_i\}_{i \in U'}$ as needed. We will maintain the following invariant: for all vertices $v \in V$ that, at any point, are in the set $v \in \bigcup_{j \in U'} V(\mathrm{p}'_j)$ and $v \notin \bigcup_{j \in U'} V(\mathrm{p}_j)$ (i.e., were not in the same set for the original definition of part-paths), it will hold that $v \in V(T')$.

For each $i \in U'$ (in arbitrary order) we do the following. Consider each walk f associated with part i (in arbitrary order). We first examine whether internal nodes of f intersect $V(T')$ and apply $f \leftarrow \mathrm{clip}(f, \{f_0\}, V(T'))$ if this is the case.

Let t be the largest step such that $f_t \in \mathrm{p}'_i$ (e.g., $t = 0$ in the warm-up scenario). We replace the prefix/suffix of p'_i between f_0 and f_t with $f_{[0,t]}$. Note that this does not change the endpoints of p'_i. We finally update $T' \leftarrow T' \cup f$. This maintains the invariant that all vertices added to p'_i are in $V(T')$ since $V(f_{[0,t]}) \subseteq V(f) \subseteq V(T')$.

We argue this maintains connectivity of T'. We discuss a few possibilities. If internal nodes of f intersected $V(T')$ the connectivity of $T' \cup f$ is clear. If it did not intersect T' then we have the same possibilities as in the warm-up: either $f_{|f|} \in T$, or the endpoint could belong to a sacrificed part (as in the warm-up). In all of these cases f is connected to T' and the connectivity of T' is maintained.

After the process is finished for all $i \in U'$, we argue that $(\{\mathrm{p}'_i\}_{i \in U'}, T')$ is a relaxed disjointness gadget. The connectivity of T' is satisfied as previously argued. We have that $s_i, t_i \in V(T')$ since the two walks associated with i contain s_i and t_i as endpoints (this property is maintained after potential clipping) and get added to T'.

Finally, we argue that $V(T') \cap V(\mathrm{p}'_i)$ for $i \in U'$ can be covered by at most three sub-paths of p'_i of length at most D. By construction, $\mathrm{p}'_i = f^i_{[0,t]} \circ (\mathrm{p}_i)_{[a,b]} \circ q^i_{[t',0]}$ (a, b, t, t' depend on i but we will drop this for notational simplicity) where f^i, q^i are the two walks associated with part i (possibly empty), $q^i_{[t',0]}$ represents the walk in reverse order $(q^i_{t'}, q^i_{t'-1}, \ldots, q^i_0)$ and $\circ$ concatenates walks with matching endpoints. The construction stipulates that $T' = T \cup \bigcup_{j \in U \setminus U'} E(\mathrm{p}_j) \cup \bigcup_{j \in J} f_j$ where $\{f_j\}_{j \in J}$ is a collection of walks associated with parts in U' (i.e., useful parts).

By construction, $V(f^i_{[0,t]}) \subseteq V(T')$. Since f^i is a shortest path in G and the diameter of G is at most D we have that $|f^i| \leq D$, hence $|f^i_{[0,t]}| \leq D$. In other words, the intersection $V(f^i_{[0,t]}) \cap V(T') = V(f^i_{[0,t]})$ can be covered by a sub-path of length at most D. The same holds for q^i.

The intersection of $V(T') \cap V((\mathrm{p}_i)_{[a,b]}) = V(T) \cap V((\mathrm{p}_i)_{[a,b]})$ for $i \in U'$: this follows from $T' = T \cup \bigcup_{j \in U \setminus U'} E(\mathrm{p}_j) \cup \bigcup_{j \in J} f_j$, the original part-paths being vertex-disjoint, and $\{f_j\}_{j \in J}$ are (clipped versions of) walks that do not intersect $(\mathrm{p}_i)_{[a,b]}$ of any useful part i (they do not intersect non-associated useful parts and the restriction to $[a, b]$ avoids associated walks). By Property 3, the intersection of $V(T) \cap V(\mathrm{p}_i)$ can be covered by one sub-path of length at most D. Thereby, restricting to $(\mathrm{p}_i)_{[a,b]}$, we conclude that $V(T') \cap V((\mathrm{p}_i)_{[a,b]})$ can be covered by one sub-path of length at most D.

In conclusion, remembering that $\mathrm{p}'_i = f^i_{[0,t]} \circ (\mathrm{p}_i)_{[a,b]} \circ q^i_{[t',0]}$, we conclude that $V(\mathrm{p}'_i) \cap V(T')$ can be

covered by at most three sub-paths of length at most D (one for each factor in the concatenation). This concludes the proof. □

We now complete the proof of Lemma 6.4.12.

Proof. Let $\{\mathrm{p}_i\}_{i=1}^k$ be minimal part-paths with endpoints $\{(s_i,t_i)\}_{i=1}^k$. Using Lemma 6.4.11, we find a crown (T_*, A_*, U_*) of $\{\mathrm{p}_i\}_{i\in A_*}$ with $|U_*| \geq \frac{1}{280}k$. Furthermore, Lemma 6.4.13 guarantees the existence of a relaxed disjoint gadget (P,T) with endpoints $\{(s_i,t_i)\}_{i\in U}$ satisfying $|U| \geq \frac{1}{5}|U_*| = \frac{1}{1400}k$. □

6.4.4 Finalizing the disjointness gadget

In this section we use the developed tools to extend every moving cut to a disjointness gadget.

Lemma 6.4.14. *If a set of k connectable pairs S in G admit a moving cut with capacity $\widetilde{O}(k)$ and distance $\beta \geq 9D$, then G contains an $\widetilde{\Omega}(\beta)$-disjointness gadget.*

Proof. Let $S = \{(s_i,t_i)\}_{i=1}^k$ be a set of connectable pairs, admitting an (α,β)-moving which we denote by ℓ. We can assume WLOG that the capacity is at most $k/30001$ due to Fact 6.6.1 which allows us to scale-down both the capacity and distance of ℓ by $\mathrm{poly}\log n$ factors (note that this lemma is insensitive to such factors).

By Lemma 6.4.12, there exists a relaxed disjointness gadget (P,T) with respect to some subset of these pairs, $S \subseteq \{(s_i,t_i)\}_{i=1}^k$, of size $|S| \geq \frac{1}{1400}k$. We will further focus on the subset $S' \subseteq S$ of pairs (s_i,t_i) whose path $p_i : s_i \rightsquigarrow t_i$ in P is made up entirely of edges e with ℓ-length precisely one (smallest possible). The capacity of ℓ is at most $\sum_e(\ell_e - 1) < k/30000 < |S|/2$. Consequently, since the paths p_i are disjoint and each edge e with $\ell_e > 1$ contributes at least one to $\sum_e(\ell_e-1)$, the set $S' \setminus S$ contains at most $k/30000$ pairs, and so $|S'| \geq |S| - k/30000 \geq k/3000$. Note that the distance and capacity (upper bound) of ℓ when move from S to its subset S'.

Now, consider a source-sink pair $(s_i,t_i) \in S'$, connected by path $p_i \in P$. By the definition of a relaxed disjointness gadget, the intersection of p_i with T is covered by a set Φ_i of at most 3 sub-paths of p_i, of hop-length at most D. Note that, by definition of relaxed disjointness gadget, sub-paths in Φ_i always cover endpoints $\{s_i,t_i\}$. By our choice of S', the sub-paths in Φ_i, which have hop-length at most D, also have ℓ-length at most D (since each of their edges' ℓ-length is precisely one). Now, consider the set Ψ_i of (one or two) sub-paths of p_i obtained by removing the sub-paths Φ_i from p_i. For each such sub-path $p \in \Psi_i$, denote by $s(p)$ and $t(p)$ the first and last node of p. Then, since ℓ has distance β for S, we have in particular that

$$\sum_{p\in\Psi_i} d_\ell(s(p),t(p)) + \sum_{p\in\Phi_i} |p| \geq d_\ell(s_i,t_i) \geq \beta.$$

Now, since $|p| \leq D$ for each $p \in \Phi_i$, and therefore $\sum_{p\in\Phi_i}|p| \leq 3D$, and since Ψ_i contains no more than two sub-paths of p_i, there must be some sub-path $p_i' \in \Psi_i$ of p_i, with endpoints

$s'_i := s(p'_i)$ and $t'_i := t(p'_i)$ for which

$$d_\ell(s(p'_i), t(p'_i)) \geq (\beta - 3D)/2. \tag{6.1}$$

By our choice of S', all edges in $p_i : s_i \rightsquigarrow t_i$ for $(s_i, t_i) \in S'$ have ℓ-length of one. Consequently, by Equation (6.1), all such paths p'_i have hop-length at least $|p'_i| = \ell(p'_i) \geq (\beta - 3D)/2 \geq 3$, since $\beta \geq 9D$. We now show that this implies that (P', T) induces a (strict) disjointness gadget for the endpoints of these sub-paths $P' := \{p'_i\}_i$. Indeed, the subgraph $H := T \cup \bigcup_{i:(s_i,t_i)\in S'}(p_i \setminus p'_i)$ is connected, and only intersects the paths p'_i (of length $|p'_i| \geq 3$) at their first and last vertices. Consequently, for T' a spanning tree of H, the pair (P', T') is a disjointness gadget with endpoints S'.

Denote by $S' = \{(s(p'_i), t(p'_i))\}_{i\in I'}$ where I' is a set of indices. Here we note that we have shown, via construction, that $\text{dist}_\ell(s(p'_{\boldsymbol{i}}), t(p'_{\boldsymbol{i}})) \geq (\beta - 3D)/2 = \Omega(\beta)$ for all $i \in I'$. However, the moving cut requires a lower bound on $\text{dist}_\ell(s(p'_{\boldsymbol{i}}), t(p'_{\boldsymbol{j}}))$ for all $i, j \in I'$, which might not be true in general. We invoke a structural lemma [69, Lemma 2.5]: given k source-sink pairs $\{(s_i, t_i)\}_{i=1}^k$ in a (general) metric space with $\text{dist}(s_i, t_i) \geq \beta$, one can find a subset $I \subseteq [k]$ of size $|I| \geq k/9$ such that $\text{dist}(s_i, t_j) \geq \beta/O(\log k)$ for all $i, j \in I$. Applying this result, we find a subset $\tilde{I} \subseteq I'$ of size $|\tilde{I}| \geq |I'|/10 \geq k/30000$ where $\text{dist}_\ell(s(p'_i), t(p'_i)) \geq \widetilde{\Omega}(\beta)$ for all $i, j \in \tilde{I}$. Therefore, ℓ has capacity at most $k/30001 < |\tilde{I}|$ and distance at most $\widetilde{\Omega}(\beta)$ with respect to $\{(s_i, t_i)\}_{i\in\tilde{I}}$. Hence $(\{p'_i\}_{i\in\tilde{I}}, T', \ell)$ is a $\widetilde{\Omega}(\beta)$-disjointness gadget. □

Armed with Lemma 6.4.14, we may finally prove Theorem 6.3.4, restated below.

Theorem 6.3.4. *For any graph G,*

$$\text{WC}\textsc{subnetwork}(G) + D = \tilde{\Theta}(\text{M}\textsc{ovingCut}(G)).$$

Proof. If $\text{M}\textsc{ovingCut}(G) < 9D$, then trivially $\text{WC}\textsc{subnetwork}(G) \leq \text{M}\textsc{ovingCut}(G) < 9D$. Therefore, since $\text{M}\textsc{ovingCut}(G) \geq D$ always (a length assignment of one to all edges yields a moving cut of capacity 0 and distance D for any pair of maximally-distant nodes), we have that

$$\text{WC}\textsc{subnetwork}(G) + D = \Theta(D) = \Theta(\text{M}\textsc{ovingCut}(G)).$$

If, conversely, $\text{M}\textsc{ovingCut}(G) \geq 9D$, then by Lemma 6.4.14, there exists an $\widetilde{\Omega}(\text{M}\textsc{ovingCut}(G))$-disjointness gadget, and therefore $\text{WC}\textsc{subnetwork}(G) = \widetilde{\Omega}(\text{M}\textsc{ovingCut}(G)) \geq \widetilde{\Omega}(D)$. On the other hand, since we trivially have that $\text{WC}\textsc{subnetwork}(G) \leq \text{M}\textsc{ovingCut}(G)$, we find that

$$\text{WC}\textsc{subnetwork}(G) + D = \tilde{\Theta}(\text{WC}\textsc{subnetwork}(G)) = \tilde{\Theta}(\text{M}\textsc{ovingCut}(G)). \qquad \square$$

6.5 Chapter Appendix: Further Related Work

Probably the most well-studied optimization problem in the distributed message-passing literature, and the one that best illustrates the search for universal optimality, is the MST problem. We

discuss some of the literature on this problem here in more detail.

The MST problem was first studied in a distributed setting in the seminal work of Gallager, Humblet, and Spira [45], who gave an $O(n \log n)$-time MST algorithm. This was later improved by Awerbuch [12] to $O(n)$ time, which is existentially optimal, due $\Omega(n)$-diameter graphs. Garay et al. [46], advocating for more refined analysis, moved closer to the universal lower bound of $\Omega(D)$, giving an $\tilde{O}(D+n^{0.613})$-time algorithm. This was later improved to $\tilde{O}(D+\sqrt{n})$ [96], with example networks proving this bound is also existentially optimal given by Peleg and Rubinovich [122]. These networks were then used to prove lower bounds for approximate MST by Elkin [35], and many other problems by Das Sarma et al. [27]. The existentially-optimal $\widetilde{O}(D + \sqrt{n})$ upper bound was obtained by numerous algorithms over the years [34, 37, 118], including using the shortcuts framework in [49].

Restricted topologies. The above works have motivated much work on studying other graph parameters which allow for improved running time for this problem. One example is restricting the diameter even further. For example, for graphs of diameter 1 (i.e., the congested clique model), a sequence of works [53, 77, 107] culminated in an $O(1)$-time algorithm [86]. For higher diameter, Lotker et al. [108] gave an $O(\log n)$ algorithm for diameter-2 graphs, and $\tilde{\Omega}(\sqrt[3]{n})$ and $\tilde{\Omega}(\sqrt[4]{n})$ lower bounds for graphs of diameter 3 and 4, with algorithms matching these bounds recently given by [90] using the shortcuts framework of [49]. Indeed, the shortcuts framework has been the driving force behind numerous improved results for restricted graph families [49, 52, 56, 65, 66, 67, 68, 90]. For most of these results, the worst-case shortcut quality a graph in the graph family serves as an upper bound for these algorithms' running time. Our lower bounds imply that these algorithms are existentially optimal (for their respective graph families). Moreover, they hint that refined bounds can be obtained by one uniform algorithm, without having to design tailor-made algorithms for graph families of interest.

Message Complexity. Another measure of message-passing algorithms is their *message complexity*, i.e., the number of messages they send during their execution. Awerbuch et al. [13] showed that any MST algorithm (under some conditions) in m-edge graphs must send $\Omega(m)$ messages—matched by many algorithms. Whether existentially-optimal time and message complexities are achievable simultaneously was open, until a recent breakthrough of Pandurangan et al. [118], who gave a randomized algorithm achieving just that. This was then shown to be achievable deterministically, by Elkin [37], and then shown to be achievable within the shortcuts framework, in [67]. This leaves open the possibility of algorithms which have both universally-optimal time complexity, and optimal message complexity.

Universal Optimality. The notion of universal optimality (or *instance-optimality*, as it is called in some fields) is a well-studied notion, and algorithms achieving this desired property are known in many computational models; these include aggregation algorithms for database systems [40], shared memory distributed algorithms [33], geometric algorithms [4], and distribution testing and learning algorithms [136, 137]. Indeed, the entire field of online algorithms and competitive analysis precisely concerns itself with notions of instance optimality. For other models, instance-optimal algorithms were long sought after, but remain elusive. For example, one of the oldest open questions in computer science is the dynamic optimality conjecture of Sleator and

Tarjan [134], which states that splay trees are instance-optimal binary search trees. Closer to our problem, the closest result to a non-trivial instance-optimal message-passing algorithm is the k-broadcast algorithm of Ghaffari [48], which is optimal among routing-based algorithm, and unfortunately requires additional preprocessing time (or alternatively, knowledge of the topology). The challenge in achieving universally-optimal message-passing algorithms seems to be a lack of strong universal lower bounds to compare algorithms' running time to. We present a family of universal lower bounds for numerous problems, which are plausibly tight, thus making a step towards achieving universally-optimal message-passing algorithms.

6.6 Chapter Appendix: Deferred Proofs of Section 6.2

In this section we present proof of lemmas deferred from Section 6.2, restated for ease of reference. We first state and prove two auxiliary lemmas: first one being a simple scaling of the capacity and distance of moving cuts, the other about using moving cuts for simulations.

Fact 6.6.1. *A moving cut with capacity γ and distance β into a moving cut with capacity γ/c and distance $\beta/(1+c)$, for any $c \geq 1$.*

Proof. Fix a moving cut ℓ of capacity γ and distance β between $S = \{(s_i, t_i)\}_{i=1}^k$.

Consider a new moving cut $\hat{\ell}_e := 1 + \lfloor \frac{\ell_e - 1}{c} \rfloor$. This assignment $\hat{\ell}$ has capacity at most $\sum_e \hat{\ell}_e - 1 \leq \sum_e \frac{\ell_e - 1}{c} < \frac{\gamma}{c}$. So $\hat{\ell}_e$ indeed has capacity γ/c.

We analyze the distance $\hat{\beta}$ of $\hat{\ell}$. Consider a source s_i and sink t_j in S. For any path $p : s_i \rightsquigarrow t_j$ of hop-length $|p| \geq \frac{1}{1+c} \cdot \beta$, clearly we have that this path's $\hat{\ell}$-length is at least $\hat{\ell}(p) \geq \sum_{e \in p} 1 = |p| \geq \frac{\beta}{1+c}$. For any path of hop-length $|p| \leq \frac{\beta}{1+c}$, we have that this path's $\hat{\ell}$-length is at least

$$\hat{\ell}(p) = \sum_{e \in p} \hat{\ell}_e = \sum_{e \in p} 1 + \left\lfloor \frac{\ell_e - 1}{c} \right\rfloor \geq \sum_{e \in p} \frac{\ell_e - 1}{c} \geq \frac{\ell(p) - |p|}{c} \geq \frac{\beta - \frac{1}{1+c} \cdot \beta}{c} = \frac{1}{1+c} \cdot \beta,$$

where the last inequality relied on $|p| \leq \frac{1}{1+c} \cdot \beta$ and on ℓ having distance β. As this argument holds for any source s_i and sink t_j, the distance of $\hat{\ell}$ is at least $\mathrm{dist}_{\hat{\ell}}(\{s_i\}_{i=1}^k, \{t_j\}_{j=1}^k) \geq \frac{\beta}{1+c}$. We conclude that $\hat{\ell}$ has indeed capacity γ/c and distance $\beta/(1+c)$ for S. □

The following lemma allows us to transform an efficient distributed algorithm into an efficient communication complexity solution. The proof follows the arguments (implicitly) contained in [27, 69].

Lemma 6.6.2. *Let $\mathcal{A}$ be a distributed computation of $f : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ between $\{s_i\}_{i=1}^k$ and $\{t_i\}_{i=1}^k$ with running time at most T. If there exists a moving cut ℓ between $\{s_i\}_{i=1}^k$ and $\{t_i\}_{i=1}^k$ of capacity γ and distance at least $2T$ then there is an $O(\gamma \log n)$-*

communication complexity protocol $\mathcal{C}$ for f. The error probability of the protocol is the same as the distributed algorithm's error probability.

Proof. We use a simulation argument to convert $\mathcal{A}$ into an efficient protocol $\mathcal{C}$, using the moving cut ℓ as a guide. Naturally, both Alice and Bob know G, ℓ and $\mathcal{A}$, but not each others' private inputs $\{x_i\}_{i\in[k]}$ and $\{y_i\}_{i\in[k]}$.

We introduce some proof-specific notation. For node $v \in V$, we denote the shortest ℓ-distance from v to any node in $\{s_i\}_i$ by $d_v := \mathrm{dist}_\ell(\{s_i\}_{i\in[k]}, v)$. Moreover, we denote by $L_{<t} := \{v \in V \mid d_v < t\}$ the nodes within distance t of some node in $\{s_i\}_i$, and similarly we let $L_{>t} := \{v \in V \mid d_v > t\}$. We label the message-sending rounds of $\mathcal{A}$ with $\{1, 2, \ldots, T\}$, and define a *timestep* $t \in \{0, 1, \ldots, T\}$ to be the moments of "inactivity" (during which the message history of each node is constant) after round t. So, timestep 0 is the moments before any message is sent; timestep $t > 0$ is the moments between round t and $t + 1$; timestep T is the moments after $\mathcal{A}$ completed.

The simulation proceeds as follows. At timestep $t \in \{0, 1, \ldots, T\}$ Alice will simulate all nodes in $L_{<2T-t}$ and Bob will simulate all nodes in $L_{>t}$. By simulating a node v, we mean that the player knows all messages received by v up to that timestep and the private input of v. (Note that only $\{s_i\}_i$ and $\{t_i\}_i$ have private inputs.) Such a simulation at timestep $t = T$ would imply in particular then Alice and Bob will know the answer $f(x, y)$, since $\mathcal{A}$ completed by timestep $t = T$, and so all nodes know the answer, including $\{s_i\}_{i\in[k]} \subseteq L_{<2T-T}$ and $\{t_i\}_{i\in[k]} \subseteq L_{>T}$ (by the moving cut's distance). We now construct a communication complexity protocol $\mathcal{C}$ by sequentially extending a valid simulation at timestep $t - 1$ to one at timestep $t \leq T$ (i.e., we simulate the exchange of round-t messages of $\mathcal{A}$). We make this argument only for Alice, while Bob's side follows analogously. Initially, Alice can simulate $L_{<2T-0}$ without any messages from Bob, since no nodes have received any messages and she knows the private inputs $\{x_i\}_{i\in[k]}$ of $\{s_i\}_{i\in[k]} \subseteq L_{<2T-0}$ and $\{t_i\}_{i\in[k]} \cap L_{<2T-0} = \emptyset$ (again, by the moving cut's distance); vice versa for Bob.

Next, let v be a node simulated by Alice in timestep $t \in (0, T]$. Since $v \in L_{<2T-t} \subseteq L_{<2T-(t-1)}$, Alice simulated v in timestep $t - 1$, and thus at timestep t, Alice knows the messages sent to v in rounds $1, 2, \ldots, t - 1$. So, all she needs to learn to simulate v at timestep t are the messages sent to v at round t in $\mathcal{A}$. If such a message comes from a neighbor w simulated by Alice in the prior timestep, $w \in L_{<2T-(t-1)}$, then Alice has all the information needed to construct the message herself. Conversely, if $w \notin L_{<2T-(t-1)}$, then we argue that w was simulated by Bob in timestep $t - 1$. This is because $w \notin L_{<2T-(t-1)}$ implies $d_w \geq 2T - t + 1 \geq T + 1 > t > t - 1$ (due to $t \leq T$), i.e., $w \in L_{>t-1}$. When this is the case, we say the edge $\{v, w\}$ is *active*. For each active edge of round t, Bob sends Alice the $O(\log n)$-bit message sent via $\mathcal{A}$ from w to v, by serializing it over $O(\log n)$ one-bit communication complexity rounds and we append those rounds to $\mathcal{C}$. So, denoting by $\mathcal{C}_t$ be the set of messages that Bob needs to send to Alice to transition from timestep $t - 1$ to timestep t (in some pre-determined order), then $\mathcal{C}$ is the concatenation of $\mathcal{C}_1, \mathcal{C}_1, \ldots, \mathcal{C}_T$. By construction, after receiving the messages of $\mathcal{C}_t$, Alice can simulate timestep t of $\mathcal{A}$. Finally, it is guaranteed that Alice learns the answer by timestep $t = T$ when $\mathcal{A}$ completes.

It remains to prove that the communication complexity protocol $\mathcal{C}$ uses only $O(\gamma \log n)$ bits. For

each edge $e = \{v, w\} \in E$, Bob sends at most $O(\log n)$ bits in $\mathcal{C}$ for every timestep e is active. By definition, this edge is active when $v \in L_{<2T-t}$ and $w \notin L_{<2T-(t-1)}$. Equivalently, $\{v, w\}$ is active when $d_v < 2T - t$ and $d_w \geq 2T - t + 1$, or, $2T + 1 - d_w \leq t < 2T - d_v$. Therefore, e is active during $2T - d_v - (2T + 1 - d_w) = d_w - d_v - 1$ rounds. We have that $d_w \leq d_v + \ell_e$ since $d_{(\cdot)}$ are distances with respect to ℓ. Therefore, an edge is active at most $d_w - d_v - 1 \leq \ell_e - 1$ rounds. Adding over all edges $e \in E$ we have that the total round-complexity of $\mathcal{C}$ is $O(\log n) \cdot \sum_{e \in E}(\ell_e - 1) = O(\gamma \log n)$. We note that this accounts only for the information sent from Bob to Alice. The analysis for the messages sent from Alice to Bob is completely analogous and requires the same amount of rounds. Interlacing these rounds increases the communication complexity by a multiplicative factor of 2, leading to a final $O(\gamma \log n)$-round communication complexity protocol $\mathcal{C}$ for f, as claimed. □

Note that the simulation argument even applies to distributed algorithms with shared randomness between the nodes.

We now show that Lemma 6.6.2 implies a lower bound on distributed disjointness computation in the face of a moving cut.

Lemma 6.2.3. *If G contains a moving cut for k pairs $S = \{(s_i, t_i)\}_{i=1}^{k}$ with distance at least β and capacity strictly less than k, then distributed computation of* disj *between $\{s_i\}_{i \in [k]}$ and $\{t_i\}_{i \in [k]}$ takes $\widetilde{\Omega}(\beta)$ time. This lower bound holds even for bounded-error randomized algorithms that know G and S.*

Proof. Let $C > 0$ be a sufficiently large constant. Suppose there is a T-time distributed computation $\mathcal{A}$ for disj with $T \leq \frac{\beta}{2(1+C \log n)}$. We can transform the moving cut to have capacity $\frac{k}{C \log n}$ and dilation $\frac{\beta}{1+C \log n}$ via Fact 6.6.1. Applying the simulation Lemma 6.6.2 we construct a $O(k)/C$-communication complexity solution for disj. However, disj is known to have communication complexity $\Omega(k)$, even for constant-probability error protocols [128], leading to a contradiction for a sufficiently large constant C. Thus we have that $T \geq \Omega(\beta / \log n)$ and we are done. □

Here we explain why [69] implies that moving cuts for pairs of nodes provide a characterization (up to polylog factors) for the time to solve multiple unicasts for these pairs.

Lemma 6.2.4. *([69]) For any set of pair S, we have that*

$$\textsc{MovingCut}(S) = \tilde{\Theta}(\textsc{Communicating}(S)) = \tilde{\Theta}(\textsc{Routing}(S)).$$

Proof. In the terminology of [69], each edge has some capacity c_e denoting the number of bits transmittable across an edge in one round (in our case $c_e = O(\log n)$). Moreover, each of the k pairs (s_i, t_i) has some demand d_i, corresponding to a message size of d_i bits to send from s_i to t_i. In our setting, $d_i = 1$ for all pairs in the set S.

The authors show in [69, Lemma 1.7] that the existence of a moving cut with capacity strictly less than k and distance β for S implies $\textsc{Communicating}(S) \geq \beta$. On the other hand, by [69,

Lemma 2.3], if such a moving cut does not exist, then $\textsc{Routing}(S) = \widetilde{O}(\beta)$. Combining these bounds together, we obtain the following chain of inequalities, implying the lemma.

$$\textsc{MovingCut}(S) \leq \textsc{Communicating}(S) \leq \textsc{Routing}(S) = \widetilde{O}(\textsc{MovingCut}(S)). \quad \square$$

6.7 Chapter Appendix: Deferred Proofs of Section 6.3

In this section we provide the proofs of lemmas deferred from Section 6.3, restated below for ease of reference.

6.7.1 β-disjointness gadgets as lower bounds certificates

Here we prove that β-disjoint gadgets are indeed witnesses of a lower bound on the time to solve subgraph spanning connectivity.

Lemma 6.3.3. *The time to solve spanning connected subgraph verification algorithm in a graph G by any algorithm (even one which knows G) is at least*

$$T_{conn}(G) = \widetilde{\Omega}(\textsc{WCSubnetwork}(G) + D).$$

Proof. Let (P, T, ℓ) be a β-disjointness gadget. Let the endpoints of the paths $\mathrm{p}_i \in P$ be $\{(s_i, t_i)\}_{i=1}^k$. We denote the edges of p_i by $E(\mathrm{p}_i)$, and their strictly internal edges (i.e., edges not incident to s_i or t_i) by $E^\circ(\mathrm{p}_i)$.

Suppose we want to perform distributed computation of the disjointness function between $\{s_i\}_i$ and $\{t_i\}_i$. For each $i \in [k]$, s_i and t_i know a private bit x_i and y_i, respectively. We define a subgraph H which is spanning and connected if and only if $\mathrm{disj}(x, y) = 1$. For a set $F \subseteq E$ let G/F denote G with edges F contracted. Let H_1 be some spanning tree of $G/(T \cup \bigcup_{i\in[k]} E(\mathrm{p}_i))$, which must exist since G is connected. Let $H_2 = T \cup \bigcup_{i\in[k]} E^\circ(\mathrm{p}_i)$. We construct H_3: for every path p_i, the edge of p_i incident to s_i (resp., t_i) belongs to H_3 iff $x_i = 0$ (resp., $y_i = 0$). Finally, let $H := H_1 \cup H_2 \cup H_3$. Each node can precompute H_1 and H_2 without extra information (they know G and $\{\mathrm{p}_i\}_i$), and each node can use its private info to compute which of its incident edges is in H_3.

It is easy to check that H is connected if and only if $\mathrm{disj}(x, y) = 1$ (i.e., $x_i \cdot y_i = 0$ for all $i \in [k]$). Thus, a T-round algorithm for spanning subgraph connectivity immediately yields a T-round distributed computation of disj between $\{s_i\}_{i=1}^k$ and $\{t_i\}_{i=1}^k$. By Lemma 6.2.3, combining this algorithm and the moving cut of the β-disjointness gadget implies $T = \Omega(\beta)$. $\square$

6.7.2 Relating shortcuts for pairs and for parts

In this section we show an equivalence (up to polylog factors) between shortcuts for connectable pairs of nodes and for parts.

Lemma 6.3.5. *For any graph G,*

$$\textsc{ShortcutQuality}(G) = \tilde{\Theta}(\textsc{ShortcutQuality}_2(G)).$$

The non-trivial part of the above lemma is the following upper bound on $Q(G)$.

Lemma 6.7.1. *For any graph G,*

$$\textsc{ShortcutQuality}(G) = \widetilde{O}(\textsc{ShortcutQuality}_2(G)).$$

Proof. Let $S_1, S_2, \ldots, S_k$ be a partitioning of V into subsets of nodes inducing connected subgraph $G[S_i]$. We show that shortcuts for this partition can be obtained by combining any (pairwise) shortcuts for some polylogarithmic number of pairs of vertices in $\mathcal{C}$, implying the lemma.

For our proof, we make use of heavy-light tree decompositions [133], which decompose a tree on n nodes into sub-paths, with each root-to-leaf path in the tree intersecting $O(\log n)$ of these sub-paths. For each $i \in [k]$, we consider some heavy-light decomposition of T_i. Note that since the parts are disjoint, these trees are disjoint, and consequently, so are the obtained sub-pats of the decomposition. Letting $q := \textsc{ShortcutQuality}_2(G)$, we first show that the partition with parts $V(p)$ for all sub-paths p of some tree decomposition of some T_i admits shortcuts of quality $O(q \cdot \log n)$.

Indeed, shortcuts of quality $O(q \cdot \log n)$ for parts P whose parts are disjoint paths, as above, can be defined recursively, as follows. For a path $p \in P$, we denote by $s(p), m(p)$ and $t(p)$ the first, median, and last node on this path p. We note that using sub-paths of these p, we find that the pairs $\{(s(p), m(p)), (m(p), t(p)) \mid p \in P\}$ are connectable. Thus, these pairs admit shortcuts of quality q. We use these shortcuts (and more edges which we will choose shortly) as the shortcuts for parts $V(p)$. We then consider recursively the two sub-paths of all such paths p, one starting at $s(p)$ and ending at $m(p)$, and the other starting at $m(p)$ and ending at $t(p)$, noting that these sub-paths for all paths $p \in P$ are also connectable. We recursively compute shortcuts for all such sub-paths and add the shortcuts for each sub-path of p to the shortcuts for each part $V(p)$. We show that the union of shortcuts for all sub-paths at each of the $O(\log n)$ levels of recursion are shortcuts for the parts P above. First, as these shortcuts for P are the union of $O(\log n)$ shortcuts for pairs of quality q, then the congestion of these shortcuts is trivially $O(q \cdot \log n)$. In order to bound dilation, we prove by induction that for each node v in one of these sub-paths of length $|p| \leq 2^k$, the shortcuts of the sub-paths of p constructed this way contain a path of length at most $q \cdot k$ from v to $s(p)$, and likewise to $t(p)$. Indeed, consider the subpath p' containing v and $m(p)$, which has length at most $|p|/2 \leq 2^{k-1}$. Then, since $m(p)$ is either $s(p')$ or $t(p')$, there exits a path of length at most $q \cdot (k-1)$ from v to $t(p)$. Concatenating this path with the shortcut path of length (at most) q from $m(p)$ to $s(p)$, and likewise to $t(p)$, yields the desired result. Therefore, any two nodes in a path p can be connected by a walk of length at most $q \cdot 2\log_2 n$ contained in the shortcuts of its recursively-defined sub-paths. Thus $O(q \cdot \log n)$ upper bounds the dilation of these shortcuts for p. We conclude that any partition P whose parts are disjoint paths admits shortcuts of quality $O(q \cdot \log n)$.

So far, we have shown that the partition given by parts induced by the sub-paths of tree decompo-

sitions of spanning trees T_i of parts S_i admit shortcuts of quality $O(q \cdot \log n)$. We now show that the union of these sub-paths' shortcuts, where each part has as shortcuts the union of the shortcuts of its' tree decomposition's sub-paths, are shortcuts of quality $O(q \cdot \log^2 n)$ for the partition $S_1, S_2, \ldots, S_k$. First, the congestion of these shortcuts is $O(q \cdot \log^2 n)$, since each edge belongs to at most $O(q \cdot \log n) = O(q \cdot \log^2 n)$ sub-paths' shortcuts, and therefore it belongs to the shortcuts of at most $O(q \cdot \log^2 n)$ distinct parts. Second, since any two nodes u, v in S_i have that their T_i-path consists of at most $O(\log n)$ sub-paths, using the shortcuts for these $O(\log n)$ sub-paths induce a path of length $O(q \cdot \log n) \cdot O(\log n) = O(q \cdot \log^2 n)$ between u and v. Thus, each partition $S_1, S_2, \ldots, S_k$ admits $O(q \cdot \log^2 n) = \tilde{O}(q) = \tilde{O}(\text{SHORTCUTQUALITY}_2(G))$-quality shortcuts, as claimed. □

The following simple lemma yields the complementary "opposite" inequality.

Lemma 6.7.2. *For any graph G,*

$$\text{SHORTCUTQUALITY}_2(G) \leq \text{SHORTCUTQUALITY}(G).$$

Proof. Let $S \in \mathcal{C}$ be a set of connectable pairs, and let P be a set of disjoint paths connecting these pairs. Now, we consider these paths (or more precisely, their vertices) as parts of a partition with one more part for the remaining nodes, $V \setminus \bigcup_{p \in P} v(p)$. Then, by definition of $\text{SHORTCUTQUALITY}(G)$, there exist shortcuts for these parts of quality at most $\text{SHORTCUTQUALITY}(G)$. Consequently, these shortcuts contain each edge at most $\text{SHORTCUTQUALITY}(G)$ times. Moreover, for each i, these shortcuts contain a path of length at most $\text{SHORTCUTQUALITY}(G)$ connecting each pair of nodes (s_i, t_i), as these nodes are in a common part. Put otherwise, the best shortcuts for the parts defined above imply the existence of shortcuts of quality no worse than $\text{SHORTCUTQUALITY}(G)$ for S. □

Lemma 6.3.5 follows directly from lemmas 6.7.1 and 6.7.2. We note, however, that the claim is existential and does not yield a constructive and distributed method to convert between the quantities. The following result addresses this shortcoming.

Before proving the claim, we establish some technical algorithmic terminology. In distributed optimization it is often infeasible for each node to know the entire input, output or intermediary data. Therefore, data is represented in a distributed way, which we here define. We now disambiguate the specific way a data is distributedly stored. For a node function $f : V_G \to X$ we say that f is "**distributedly known**" by the nodes if v knows $f(v)$. Similarly, a edge function $f : E_G \to X$ is distributedly known if each $v \in V_G$ knows $f(e)$ for each edge e incident to v. A set of parts $\mathcal{P} = (S_1, \ldots, S_k)$ is distributedly known if each node v knows the part ID it belongs to (or $\perp$ if none). A set of pairs $\mathcal{S} = \{(s_i, t_i)\}_{i=1}^k$ is distributedly known if for each pair (s_i, t_i) both s_i and t_i know the pair (s_i, t_i) (i.e., both nodes). Finally, a shortcut $(H_1, \ldots, H_k$ is distributedly known if each node v knows, for each incident edge e the set of part IDs that contain e.

Lemma 6.7.3. *Suppose that, in a graph G, one is given a black-box oracle takes as input a set of connectable pairs $\mathcal{S}$ (the input is distributed among the nodes) and (distributedly)*

output shortcuts of $\mathcal{S}$ with quality $\text{SHORTCUTQUALITY}_2(G)$ in at most T rounds. Then there is a randomized distributed algorithm that takes a set of parts $\mathcal{P} = (P_1, \ldots, P_k)$ as input and constructs a shortcut of quality $\tilde{O}(\text{SHORTCUTQUALITY}_2(G))$ in $\tilde{O}(T)$ rounds with high probability.

Proof. We introduce some helpful notation. Consider a function $f : E_G \to \{\rightarrow, \leftarrow, \perp\}$. We denote by $G(f)$ the directed graph (V_G, E') where the edge $e \in E_G$ is either not in E' if $f(e) = \perp$, and is otherwise directed one way or the other (the specific meaning of $\rightarrow$ and $\leftarrow$ is arbitrary).

We prove a sequence of intermediary results.

Subclaim 1: Suppose that $f : E_G \to \{\rightarrow, \leftarrow, \perp\}$ is such that $G(f)$ is precisely a union of (maximal) vertex-disjoint directed paths $P_1 \cup P_2 \ldots \cup P_q$. We argue there is a distributed $\tilde{O}(T)$-round algorithm that takes the distributed knowledge of f as input and (distributedly) learns for each node v (i) the unique path ID of the maximal path $P_i \ni v$ it is contained in, the size of the path $|P_i|$ and the first node on the path (ii) the depth (distance of v from the start of the P_i), (iii) a shortcut with respect to $(P_1, P_2, \ldots, P_q)$ of quality $\text{SHORTCUTQUALITY}_2(G)$. Moreover, given a distributed function $g : V_G \to [n^{O(1)}]$, each node can also learn (iv) the sum $\sum_{u \in R(v)} g(u)$ where $R(v)$ is the set of nodes u reachable from v via the directed edges of $G(f)$ (i.e., they are on the same path P_I, but "below" v).

Proof of subclaim 1: The algorithm starts with initially singleton clusters and then grows the clusters for $O(\log n)$ steps. In the first step, each node is its own cluster. In each subsequent step we assume we have maintained the above data (i)–(iv) for each node v with respect to the cluster (i.e., subpath) that v is in. We show how to merge neighboring clusters while maintaining the validity of this data. First, each cluster flips an independent random heads/tails coin (the entire cluster flips a common coin without communication via shared randomness). Intuitively, tails will merge into heads and only if "heads subpath" has a directed edge towards the "tails subpath". The merging is fairly simple. For each head subpath A (i.e., a cluster that flipped heads) and each tail subpath B where there is a directed "connecting" edge of $G(f)$ from A to B, we do the following. We pass the unique path ID of the head subpath via the connecting edge into the tail subpath B and disseminate it throughout the nodes of B via the computed shortcut in $\tilde{O}(T)$ rounds. Similarly, we can disseminate the sizes $|A|, |B|$ and the starting node of the path A to correctly all values stipulated by (i). For (ii), We add the value of $|A|$ to the depths of each node in the B via the shortcut of B. For (iii), we add the shortcut between the start of A to the start of B (note that pairs of connecting clusters do this operating in parallel, but all of these paths are connectable, hence the operation is valid; we argue in the next paragraph the quality suffices). Finally, for (iv), we simply calculate the sum of g's in B via the shortcut, propagate this sum σ to A and add σ to the sum of g's in each node of A.

Clearly, each pair of neighboring subpaths (i.e., neighboring clusters) will merge with probability at least $1/4$ in each step. Furthermore, on each maximal path P_i each cluster will have at least one other neighboring cluster unless the entire path is a single cluster. Therefore, after $O(\log n)$ merging steps the entire path is a single cluster with high probability via standard arguments. Finally, we argue about the shortcut quality of the clusters. First, since the shortcut is constructed by $O(\log n)$ calls to the pairwise oracle, the congestion is clearly $\tilde{O}(\text{SHORTCUTQUALITY}_2(G))$.

We argue the same dilation bound by consider an arbitrary node v and arguing that after i merging steps it can reach the start of its own cluster in $i \cdot \text{SHORTCUTQUALITY}_2(G)$ hops: in each merging step the subpath v belongs to either is not merged (we do not do anything), it is a head subpath being merged (we do not do anything), or it is a tail subpath B being merged with subpath A. The the last case we can move v to the start of B in $i \cdot \text{SHORTCUTQUALITY}_2(G)$ hops, and then to the start of A in another $\text{SHORTCUTQUALITY}_2(G)$, for a cumulative $(i + 1)\text{SHORTCUTQUALITY}_2(G)$ hops, proving the shortcut dilation claim since $i = O(\log n)$. This completes the proof of subclaim 1.

Before we proceed, we remind the reader about the "**heavy-light decomposition**" of a tree. Consider a (rooted) tree with at most n nodes. Define the "subtree size" of a node v as the number of nodes that are in the subtree of v. For each node v we label the edge connecting v to its child node of larger subtree size as "light", and the remaining edges as "heavy". It is easy to show that each root-leaf path contains $O(\log n)$ heavy edges and that the light edges form a collection of vertex-disjoint paths. This fact is known as the heavy-light tree decomposition.

Subclaim 2: Suppose that $f : E_G \to \{\rightarrow, \leftarrow, \perp\}$ is such that $G(f)$ is precisely a union of (maximal) vertex-disjoint rooted **trees** $P_1 \cup P_2 \ldots \cup P_q$. We argue there is a distributed $\tilde{O}(T)$-round algorithm that takes the distributed knowledge of f as input and (distributedly) learns for each node v that is in the (maximal) tree P_i (i) the unique tree ID of P_i, the root of P_i, and the parent of its root in $G(f)$ if any, (ii) the subtree size with respect to P_i, (iii) the (distributed) heavy-light edge labelling of P_i, (iv) the number of heavy edges on the path between the root of P_i and v.

Proof of subclaim 2: We first note that, given vertex-disjoint clusters (the clusters are subtrees of P_i) where values (i)–(iv) are maintained, one can easily construct shortcuts on each cluster (and perform partwise aggregation on it). First, we construct shortcut via subclaim 1 on all light paths of all clusters combined: we ignore all non-light edges and note that light edges can be partitioned into vertex-disjoint paths, hence the preconditions of subclaim 1 are valid. Then the heavy edges of each cluster are added to this shortcut of that cluster. Since every root-leaf path has at most $\tilde{O}(1)$ heavy edges and the set of all light paths have $\tilde{O}(\text{SHORTCUTQUALITY}_2(G))$-quality shortcuts, this shortcut can clearly be argued to have $\tilde{O}(\text{SHORTCUTQUALITY}_2(G))$ quality with respect to the set of all clusters.

Furthermore, given a (distributed) node function $g : V_G \to [n^{O(1)}]$ and a set of vertex-disjoint clusters (i.e., the clusters are subtrees of P_i), for each node u we can calculate the sum of the values $g(v)$ for all nodes v in the subtree of u with respect to the cluster u is in. First, using subclaim 1, we partition all clusters into vertex-disjoint light paths and calculate for each node v the sum of all values g from nodes reachable from v in its light path. Then, for $O(\log n)$ step we calculate the sum of g's on the entire path and forward this sum to the parent path, which then disseminates this value via the shortcut of subclaim 1 to all of its nodes, increasing subtree sum. After $O(\log n)$ step, all light paths have the correct value since any root-to-leaf path in the cluster has $O(\log n)$ heavy edges.

Finally, we prove the subclaim in a similar manner to subclaim 1. The algorithm starts with initially singleton clusters and then grows the clusters for $O(\log n)$ steps. In the first step, each node is its own cluster. In each subsequent step we assume we have maintained the above data (i)–(iv)

for each node v with respect to the cluster (i.e., subtree) that v is in. We show how to merge neighboring clusters while maintaining the validity of this data. First, each cluster flips an independent random heads/tails coin (the entire cluster flips a common coin without communication via shared randomness). Intuitively, tails will merge into heads and only if "heads subtree" has a directed edge towards the "tails subtree". The merging is fairly simple. Consider a head subtree with its incident tail subtrees (possibly multiple ones). We can easily update the values (i) by construct a shortcut for each cluster via the above remark and disseminating the appropriate data, like in Subclaim 1. To compute (ii), we pass from the (root of the) tail subtrees the sizes of these trees to the root's parent node in the head subtree. Suppose that each node v in the head subtree receives a cumulative sum of values $g(v) \geq 0$ via this process. We set up $g^\dagger(v) \leftarrow g(v) + 1$ and calculate for each node u the sum of $g^\dagger$'s in the cluster's subtree of u via the above remark. This value is exactly the updated subtree size of a node u with respect to its updated cluster. The value (iii) can be directly calculated from the subtree sizes we computed in (ii). And finally, (iv) can be computed in an analogous way as (ii). This completes the proof of the subclaim.

Completing the proof of Section 6.7.2. Fix a (distributed) vertex partition $\mathcal{P} = (P_1, \ldots, P_k)$. The algorithm starts with initially singleton clusters and then grows the clusters for $O(\log n)$ steps. In the first step, each node is its own cluster. Each cluster is a subset of some part P_i and in the end the the cluster will be equal to the entire P_i. In each step, there is some partition into clusters, and for that step we maintain a distributed function $f : E_G \to \{\rightarrow, \leftarrow, \perp\}$ such that $G(f)$ can be partitioned into maximal rooted spanning trees that correspond (in a one-to-one fashion) with the spanning trees of clusters (in other words, each cluster has a unique spanning tree in $G(f)$ and each maximal spanning tree in $G(f)$ determines a cluster). First, each cluster flips an independent random heads/tails coin (the entire cluster flips a common coin without communication via shared randomness). Intuitively, tails will merge into heads and only if "heads cluster" has a directed edge towards the "tails cluster". The only issue we encounter is that the "orientation" of the head cluster spanning tree might not match the orientation of the tail cluster spanning tree. To this end, we need to "fix" each tail cluster B in the following way. Suppose that the head cluster A has a directed edge from some node in A to a node b in B. We need to reverse the orientation of edges on the root (of B)-to-b path. However, this can be easily done via subclaim 2 (note that the tail components in isolation satisfy subclaim 2 and hence we can invoke the subclaim upon them). After that we add the directed edge between A and B to f and we are done, the merged components are correctly oriented and satisfy the stipulations of subclaim 2. After $O(\log n)$ rounds of merging the procedure completes and clusters exactly correspond to $P_1, P_2, \ldots, P_k$ with high probability. At this moment we use subclaim 2 to build a shortcut on $\{P_1, \ldots, P_k\}$ and we are done. □

6.8 Chapter Appendix: Deferred proofs of Section 6.4

We start by proving the simple crown-merging lemma, whereby disjoint crowns can be merged to form a single crown on the union of the crowns' parts.

Lemma 6.4.3. *(Crown merging) Let $\{(T_i, A_i, U_i)\}_{i\in I}$ be a set of disjoint crowns of vertex-disjoint paths $\{\mathrm{p}_i\}_{i=1}^{k}$. Then there exists a single crown (T_*, A_*, U_*) of $\{\mathrm{p}_i\}_{i\in A_*}$ such that $A_* = \bigsqcup_{i\in I} A_i$.*

Proof. Consider the shortest path connecting two vertices on part-paths belonging to different crowns. More precisely, let $V(A_i) := \bigcup_{x\in A_i} V(\mathrm{p}_x)$. Then we consider $\min_{i,j\in I; i\neq j} \mathrm{dist}_G(V(A_i), V(A_j))$. Let q be this shortest path and suppose its endpoints are $u \in V(\mathrm{p}_x)$ and $v \in V(\mathrm{p}_y)$, with $x \in A_i$ and $y \in A_j$. By the minimality of q we know that the internal vertices of q do not belong to any crown, where we say that a vertex u *belongs* to a crown (T, A, U) if $u \in V(\mathrm{p}_i)$ and $i \in A$. We now merge the crowns (T_i, A_i, U_i) and (T_j, A_j, U_j) into a single crown (T_*, A_*, U_*). We remove crowns i and j from the list of crowns I, add $*$ to the list, and recurse until there is only one crown in the list. The following construction ensures that $A_* = A_i \sqcup A_j$ and that the new crown $*$ is disjoint from all other crowns in the list I. The final crown (T_*, A_*, U_*) will clearly have the property that $A_* = \bigsqcup_{i\in I} A_i$ and will be defined with respect to $\{\mathrm{p}_i\}_{i\in A_*}$.

To help with properly defining the merge, we define auxiliary sets $(T_i', A_i', U_i') \leftarrow (T_i, A_i, U_i)$ and $(T_j', A_j', U_j') \leftarrow (T_j, A_j, U_j)$. If the endpoint of q, belonging to crown i, is in a useful part x (i.e., $q \in \mathrm{p}_x$ and $x \in U_i$), then we declare the part x that q intersects sacrificial and add p_x to T_i'. More precisely, if $x \in U_i$ then we define $U_i' \leftarrow U_i \setminus \{x\}$ and $T_i' \leftarrow T_i \cup E(\mathrm{p}_x)$. Note that T_i' is still connected since $V(T_i')$ intersects $V(\mathrm{p}_x)$. We analogously do the same for crown j: if the endpoint of q belonging to crown j is in a useful part, we declare that part sacrificial and add its path to T_j'.

Define the crown $(T_*, A_*, U_*) \leftarrow (T_1' \cup T_2' \cup E(q), A_1 \cup A_2, U_1' \cup U_2')$, add it to the set of remove (T_i, A_i, U_i) and (T_j, A_j, U_j). We check it is a proper crown. T_* is connected since T_1' and T_2' are connected and q is a path connecting them. Property 1: $|U_*| \geq |U_i| + |U_j| - 2 \geq \frac{1}{4}|A_i| + 2 + \frac{1}{4}|A_j| + 2 - 2 = \frac{1}{4}|A_*| + 2$. Properties 2 and 3 are satisfied because they were satisfied in (T_i', A_i', U_i') and (T_j', A_j', U_j') and (by minimality of q) no internal nodes of q intersect a part-path belonging to a crown. □

Chapter 7

Near-Optimal Distributed Known-Topology Shortcut Construction

This chapter was done in collaboration with Bernhard Haeupler and Mohsen Ghaffari. The chapter's contents are unpublished at the time of writing.

7.1 Introduction

The *low-congestion shortcut* framework has been shown to be an indispensable tool in distributed optimization. Shortcuts of good *quality* can be used to provide efficient distributed primitives which can, in turn, be used to design fast algorithms for various distributed optimization problems. While the technique was originally conceived for designing efficient algorithms for special graph classes, subsequent work has shown that shortcuts offer deep insights into the complexity of distributed computing.

It was shown that for a graph G if Q_G is the best quality shortcut in G, any correct distributed algorithm for problems such as the MST, SSSP, or min-cut requires $\tilde{\Omega}(Q_G)$ rounds of CONGEST to complete on G. In other words, the shortcut quality is a universal lower bound for many distributed optimization problems. Moreover, this lower bound holds even in the known-topology setting (making the lower bound stronger). In this setting nodes know the graph G upfront, but not the input to the problem (e.g., for the MST problem they know G, but not the edge costs). The known-topology setting is well-motivated even on the algorithmic side: there is a practical need to solve multiple distributed optimization instances on the same network, hence potentially amortizing the preprocessing overhead required to learn the graph topology before the input is given.

In this chapter we prove the converse to the above universal lower bound in the known-topology setting: if Q_G is the optimal shortcut quality of a graph G, then one can solve the MST (and other distributed optimization problems) in $\tilde{O}(Q_G)$ known-topology CONGEST rounds. By prior work, the question can be reduced to the following one.

Problem 7.1.1 (Pairwise shortcut construction)**.** *Let $Q_G > 0$ and let $\{(s_i, t_i)\}_{i=1}^k$ be k pairs of nodes in a graph G where each node belongs to at most one pair. A (pairwise) shortcut of* ***quality*** *Q_G is a set of paths $\{p_1, \ldots, p_k\}$ where p_i has endpoints s_i and t_i such that the $\text{dilation} + \text{congestion} \leq Q_G$. The* ***dilation*** *is the maximum number of hops of any path p_i, and the* ***congestion*** *is the maximum number of paths any edge e is in. Furthermore, the pairs are distributed: a node v only knows the one or zero pairs that contain v.*

One can see the pairwise shortcuts are a relaxation of the (standard) low-congestion shortcuts (Definition 1.2.3). Comparing the definitions between pairwise and (partwise) low-congestion shortcuts, the latter corresponds to shortcuts with parts P_i being set to (s_i, t_i). We note that low-congestion shortcuts require the parts to be connected, a requirement which we drop here.

For the sake of simplicity, we can assume Q_G is given as input; the reader should think of Q_G as the smallest value for which any instance of pairwise shortcut construction has a feasible solution. Using the terminology of Chapter 6, this value is exactly $Q_G = \textsc{ShortcutQuality}_2(\{(s_i, t_i)\}_{i=1}^k)$. We solve the problem of pairwise shortcut construction and immediately prove the existence of *universally-optimal distributed algorithms* in the distributed known-topology setting for the MST problem (for other problems the result follows analogously).

Given $\{(s_i, t_i)\}_{i=1}^k$, we consider all possible sets of paths $\{p_i\}_{i=1}^k$ (that connect those endpoints) and evaluate all possible values of $\text{dilation} + \text{congestion}$. The smallest achievable value is called the *offline optimum*. However, we note the important fact that the pairs $\{(s_i, t_i)\}_{i=1}^k$ are given in a distributed manner: generally, no single node is privy to the entire set of pairs. This raises the need to choose the paths (nearly) "*obliviously*" to the set of other pairs in the graph: ideally, each pair (s_i, t_i) would choose some path p_i solely based on s_i and t_i; this is in contrast to the centralized select performed by the offline optimum. However, we require that these obliviously chosen paths have $\text{dilation} + \text{congestion}$ within a polylogarithmic factor of the offline optimum. This problem has been defined in prior work as "$\text{dilation} + \text{congestion}$ *oblivious routing*".

The main technical contribution of this chapter is to prove that every graph has an $\text{dilation} + \text{congestion}$ oblivious routing. Known oblivious routing strategies [126] only take the congestion into account: such structures are highly interesting when one is only interested in preserving the congestion close to the optimal, allowing the dilation to change arbitrarily. However, our distributed computing application requires us to control for both parameters. In fact, the construction of such $\text{congestion} + \text{dilation}$ oblivious routings is an interesting open problem that spurred interest in the special case of particular graphs like mesh graphs and geometric networks [20, 21]. Interestingly, $\text{congestion} + \text{dilation}$ oblivious routings for general graphs with polylogarithmic approximation ratios were believed to be impossible due to an impossibility result of Räcke [125, pg. 59]. We side-step this barrier by using a slightly more general definition of oblivious routing that is still usable in the distributed setting. One of the relaxations we make is to "decouple" the $\text{congestion} + \text{dilation}$ objective in the following way: suppose that there exists a set of (secret, or witness) paths $\{p_1, \ldots, p_k\}$ connecting the endpoints $\{(s_i, t_i)\}_{i=1}^k$ such that each path has at most Q hops (i.e., dilation is at most Q) and the congestion is OPT. Our oblivious routing is given an input parameter Q and is obliviously given the pairs; its objective is to find paths of dilation at most $\tilde{O}(Q)$ while approximating OPT (up to polylogarithmic factors).

The optimal input parameter Q can subsequently be binary searched. Due to this decoupling, we call our structure the"*hop-bounded oblivious routing*".

To construct the new oblivious routing structure we introduce and construct a new version of Hierarchical Separated Trees (HST). Classic HST applications typically embed some weighted graph G into a distribution of trees while keeping the expected cost between two nodes within a logarithmic factor of the original weight. We propose a novel HST that embeds only $(1-p)$-fraction of the nodes in a graph into a tree (instead of all nodes). However, this relaxation allows us to control the maximum number of hops in a path between any two embedded nodes in each tree to be $\tilde{O}(p^{-1})$. Furthermore, we can still recover a similar distance guarantee to the regular HST: for every two nodes neighboring nodes u, v the expected value of the weight between u, v in the tree taken over all trees where u, v are embedded (and ignoring the rest) is $\tilde{O}(p^{-1})$ times larger than the weight of the edge between u and v. We believe that both the new HST construction and the hop-bounded oblivious routing will be of independent interest to the approximation algorithms community.

This chapter is organized as follows. In Section 7.1.1 we give an overview of the results and technical contributions of this chapter. In Section 7.2 we introduce the notation used in the chapter. In Section 7.3 we describe a decomposition lemma used to construct our HSTs. In Section 7.4 we define and construct our novel HSTs. In Section 7.5 we define and construct hop-bounded oblivious routings. In Section 7.6 we describe a useful distributed subroutine that we can be used to recover a (pairwise) shortcut of good quality from a set of paths containing both the shortcut and adversarially inserted paths. Finally, in Section 7.7 we distributedly construct pairwise shortcuts using the hop-bounded oblivious routing and the subroutine of Section 7.6.

7.1.1 Overview of results

We contribute a distributed known-topology approximate scheme for pairwise shortcut construction. More precisely, we prove the following theorem.

Theorem 7.1.2 (Pairwise shortcut construction)**.** *Let $P = \{(s_i, t_i)\}_{i=1}^{k}$ be k pairs of nodes in G that admit a shortcut of* $\text{quality} = \text{congestion} + \text{dilation} \leq \text{OPT}$. *There is a randomized distributed known-topology CONGEST algorithm that constructs shortcuts of quality $\tilde{O}(\text{OPT})$ in $\tilde{O}(\text{OPT} + D)$ rounds with high probability. A node v initially distributedly knows P and at termination will distributedly know the computed shortcut.*

As a direct corollary of the above theorem, we get the following corollary as our main result.

Corollary 7.1.3. *There exists a universally optimal distributed known-topology CONGEST algorithm for the MST problem.*

Proof. We reuse the notation of Chapter 6: let $\text{SHORTCUTQUALITY}_2(G, \{(s_i, t_i)\}_{i=1}^{k})$ be the optimal shortcut quality of a set of k paths with endpoints $\{(s_i, t_i)\}_{i=1}^{k}$. Let $\text{SHORTCUTQUALITY}_2(G)$ be the minimum $\text{SHORTCUTQUALITY}_2(G, S)$ over all *connectable* set of pairs S. $\text{SHORTCUTQUALITY}(G)$

is defined analogously for standard shortcut quality. Let $Q = \text{SHORTCUTQUALITY}_2(G)$ and note that $Q \geq D$, where D is the diameter of G. Therefore, we can construct pairwise shortcuts with respect to any set of pairs in $\tilde{O}(Q)$ via Theorem 7.7.2.

By Theorem 6.3.1, we get that any correct distributed algorithm solving the MST problem requires $\tilde{\Omega}(Q)$ rounds. We now argue we can solve the MST problem in $\tilde{O}(Q)$ rounds. By Fact 1.2.2, it is sufficient to solve some part-wise aggregation instance in $\tilde{O}(Q)$ rounds. Fix such an instance $\mathcal{P} = (P_1, \ldots, P_k)$. We use Section 6.7.2 to reduce the question of constructing shortcuts on $\mathcal{P}$ to $\tilde{O}(1)$ calls to the pairwise shortcut construction. The pairwise shortcuts can be constructed in $\tilde{O}(Q)$ via Theorem 7.7.2, giving us a $\tilde{O}(Q)$-quality shortcut on $\mathcal{P}$ in $\tilde{O}(Q)$ rounds.

Finally, applying Fact 1.2.5, we distributedly solve the partwise aggregation problem on $\mathcal{P}$ in $\tilde{O}(Q)$ rounds. Since $\tilde{\Theta}(Q)$ is both an upper bound and a lower bound for the problem, we prove the existence of universally optimal algorithms in the distributed known-topology setting. □

7.2 Definitions and notations

General. We denote by $[k] = \{1, 2, \ldots, k\}$ for some non-negative integer k and $A \sqcup B$ denotes the disjoint union of A and B. We often use the Iverson bracket notation $\mathbb{I}[\text{condition}]$ which evaluates to 1 when the condition is true and 0 otherwise. We assume that all graphs are undirected and we typically assume the existence of a graph $G = (V, E_G)$ with $n := |V|$. The (unweighted) diameter (also called hop-diameter) of G is denoted by D_G, or usually just D.

Weighted Graphs. Let $G = (V, E_G)$ be a weighted graph and let $w = (w_0, w_1, \ldots, w_\ell)$ be a path (or more generally, a walk) in G. We denote the number of hops in w with $|w| := \ell$ and the sum of weights of the edges in the walk with $\text{dist}_G(w)$ or just $\text{dist}(w)$ if there are no ambiguities. We denote by $\text{dist}_G^{(k)}(u, v) := \min\{\text{dist}_G(w) \mid \text{walks } w \text{ between } u, v \text{ with } |w| \leq k\}$.

Routing via a tree. Given a tree $T = (V, E_T)$ of $G = (V, E_G)$, we define the T-route between $u, v \in V$ in the following way. Note that T might not be a subgraph of G, i.e., generally $E_T \not\subseteq E_G$. If u and v are connected with a tree edge we let T_{uv} be the path corresponding to that edge. For other u, v consider the unique tree path $(u = w_0, w_1, \ldots, w_\ell = v)$ connecting u and v and concatenate all paths $T_{w_i, w_{i+1}}$. This results in a path between u and v in G, which we denote by T_{uv}. Note that if G is weighted, we can access the weight of this path by $\text{dist}_G(T_{uv})$.

Flow. A flow f is a non-negative vector indexed over the edges of the underlying graph G, i.e., $f \in \mathbb{R}_{\geq 0}^{E_G}$. When talking about flows we also consider paths p to be vectors $p \in \mathbb{R}_{\geq}^{E_G}$ where $p_e = \mathbb{I}[\text{p goes through e}]$. Given a flow f, we call f_e the maximum **congestion** on edge e and we call $\max(f)$ the (maximum) **congestion** of f.

Demands. Let $d \in \mathbb{Z}_{\geq 0}^{V \times V}$ be called a **demand vector**. We define the **total demand** $|d|$ of d as $\sum_{s,t \in V \times V} d(s, t)$ and we say that (s, t) is a **demand pair** if $d_{s,t} > 0$. The multiset of demand pairs of d is the multiset where the pair (s, t) occurs $d(s, t)$ times.

Set of paths. Let $H = \{p_1, p_2, \ldots, p_k\}$ be a set of paths in a graph G. We define $\text{dilation}(H) = \max_{p \in H} |p|$ to be the maximum length of any path, $\text{congestion}(H, e) = (\sum_{p \in H} p)_e$ to be the congestion over an edge e, and $\text{congestion}(H) = \max_{e \in E_g} \text{congestion}(H, e)$ to be the (maximum)

congestion over all edges.

7.3 Decomposition Lemma

In this section, we prove a useful graph decomposition lemma that can handle multidimensional weights on edges. Notably, the Lemma will furnish dealing with $\text{congestion} + \text{dilation}$ constraints (or hop-bounded constraints) by reserving one dimension for hops and another dimension for edge costs.

For clarity, (only) in this section we will denote vector quantities in bold (like $\boldsymbol{w} \in \mathbb{R}^t$). We focus on multi-dimensional weights, $\boldsymbol{w} : E_G \to \mathbb{R}^t$, which is simply the concatenation of t one-dimensional weights, $\boldsymbol{w}_i : E_G \to \mathbb{R}$ for all $i \in [t]$. As with single-dimensional weights, the $\boldsymbol{w}$ weight of a path p is simply the sum of weights of its edges, $\boldsymbol{w}(p) \triangleq \sum_{e \in p} \boldsymbol{w}(e)$. We use $\preceq$ to denote the partial order among vectors given by element-wise domination. That is, for any two t-dimensional vectors $a = (a_1, a_2, \ldots, a_t) \in \mathbb{R}^t$ and $b = (b_1, b_2, \ldots, b_t) \in \mathbb{R}^t$, we use $a \preceq b$ as shorthand for $a_i \leq b_i \; \forall i \in [t]$. Slightly abusing this notation, we use $\mathbf{dist}_{\boldsymbol{w}}(u, v) \preceq b$ to denote the existence of a path $p : u \rightsquigarrow v$ such that $\boldsymbol{w}(p) \preceq b$. Furthermore, given a vertex partition $V = P_1 \sqcup P_2 \ldots P_q$ and a path p we say that "p is **preserved**" if there exists $i \in [q]$ that contains all vertices of p.

Lemma 7.3.1. *Let $G = (V, E)$ be an undirected graph with $n := |V|$ and $\boldsymbol{w}$ be a collection of t edge weights. For any $\boldsymbol{b} \in \mathbb{R}^t_{>0}$ and $0 \leq \gamma \leq 1$ there exists a (poly-time computable) distribution over partitions $P_1 \sqcup P_2 \sqcup \ldots \sqcup P_q = V$ and subsets of the parts $C_i \subseteq P_i$ for $i = 1, \ldots, q$, where*

1. *For each i and $u, v \in P_i$, we have that $\mathbf{dist}_{\boldsymbol{w}}(u, v) \preceq \boldsymbol{b} \cdot O(t \log n)$ (with probability 1).*
2. *For each i, $u \in C_i, v \notin P_i$, we have that $\mathbf{dist}_{\boldsymbol{w}}(u, v) \not\preceq \boldsymbol{b} \cdot \gamma$ (with probability 1).*
3. *For each $v \in V$, we have that $\Pr[v \notin \bigcup_{i \in [q]} C_i] \leq \gamma$.*
4. *For each path p of weight $\boldsymbol{w}$, we have that $\Pr[p \text{ is not preserved in } P_1 \sqcup \ldots \sqcup P_q] \leq \frac{1}{t} \sum_{i=1}^{t} \boldsymbol{w}_i / \boldsymbol{b}_i$.*

To prove Lemma 7.3.1 we rely on *padded decompositions* [61], a structural result for metric spaces. We first introduce some section-specific notation. Given a metric space (V, dist), we define $B_{\text{dist}}(x, \rho) \triangleq \{y \in V \mid \text{dist}(x, y) \leq \rho\}$ denote the ball of radius $\rho \geq 0$ around $x \in V$. Next, for $U \subseteq V$ and a partition P as above, we denote by $U \subseteq P$ the event that there exists a part $V_i \in P$ containing U in its entirety; i.e., $U \subseteq V_i$.

Definition 7.3.2 (Padded Decompositions)**.** *Let $(V, \text{dist} : V \times V \to \mathbb{R})$ be a metric space. We say that a distribution $\mathcal{P}$ over partitions $P = P_1 \sqcup P_2 \sqcup \ldots \sqcup P_q = V$ is* (β, Δ)-padded *if the following holds:*

1. *For each* $i \in [q]$ *we have that* $\max_{u \in P_i, v \in P_i} \mathrm{dist}(u, v) \leq \Delta$.
2. *For some constant* $\delta > 0$*, we have that for every* $v \in V$ *and every* $0 \leq r \leq \delta \cdot \Delta$

$$\Pr_{P \sim \mathcal{P}}[B_{\mathrm{dist}}(v, r) \not\subseteq P] \leq \frac{r\beta}{\Delta}.$$

In words, each part of the partition has diameter at most Δ and the probability of any point v in the metric being at distance at most r from a node in a different part is at most $\frac{r\beta}{\Delta}$. Such decompositions were presented, for example, by [61].

Lemma 7.3.3 ([61])**.** *Any metric* $(V, dist : V \times V \to \mathbb{R})$ *on* n *points admits a* (β, Δ)*-padded decomposition, for any* $\Delta > 0$ *and some* $\beta = O(\log n)$*. Such a decomposition can be computed in polynomial time.*

We are now ready to prove the main result of this section.

Proof of Lemma 7.3.1. We define a metric space $(V, \mathrm{dist}' : V \times V \to \mathbb{R})$ as the metric space induced by the (metric completion) of the following graph metric on G. The dist'-length of an edge $e \in E_G$ is defined as $\sum_{j=1}^{t} \frac{\boldsymbol{w}_j(e)}{\boldsymbol{b}_j}$; since dist' is a metric completion, it is clearly a metric space.

Setting $\Delta \triangleq O(t \log n)$ and using Lemma 7.3.3 we find a (distribution over) $(O(\log n), \Delta)$-padded decompositions $P = P_1 \sqcup P_2 \sqcup \ldots \sqcup P_q$ using dist' as the underlying metric. Fix $u, v \in V$. Since every P_i has diameter at most Δ, there exists a path $p = (e_1, e_2, \ldots, e_\ell)$ between u and v whose dist'-length is at most $O(t \log n)$. For all $j' \in [t]$,

$$O(t \log n) = \Delta \geq \sum_{i=1}^{\ell} \sum_{j=1}^{t} \frac{\boldsymbol{w}_j(e_i)}{\boldsymbol{b}_j} \geq \frac{1}{\boldsymbol{b}_{j'}} \sum_{i=1}^{\ell} \boldsymbol{w}_{j'}(e_i).$$

In other words, for all $j' \in [t]$ the $w_{j'}$-cost of p is $\boldsymbol{b}_j \cdot O(t \log n)$, implying that $\mathbf{dist}_{\boldsymbol{w}}(u, v) \preceq \boldsymbol{b} \cdot O(t \log n)$.

Next, we construct $C_i \subseteq P_i$ by removing from P_i all vertices $v \in P_i$ where $B_{\mathrm{dist}'}(v, \gamma t) \not\subseteq P_i$. We first note that $\Pr[v \notin \bigcup_{i=1}^{q} C_i] \leq \frac{\gamma \cdot t \cdot O(\log n)}{O(t \log n)} \leq \gamma$, i.e., we can drive the probability of this event down to γ by choosing a sufficiently large O-notation constant in the definition of Δ. In other words, $\Pr[v \notin \bigcup_{i=1}^{q} C_i] \leq \gamma$ for each vertex $v \in V$, as stipulated.

For $u \in C_i$ and $v \notin P_i$ we argue that $\mathbf{dist}_{\boldsymbol{w}}(u, v) \not\preceq \gamma b$. Since, by definition of C_i, we have that $B_{\mathrm{dist}'}(u, \gamma t) \subseteq P_i$, we conclude that $\mathrm{dist}'(u, v) > \gamma t$. Assuming the opposite, $\mathbf{dist}_{\boldsymbol{w}}(u, v) \preceq \gamma \cdot \boldsymbol{b}$ implies $\mathrm{dist}'(u, v) \leq \gamma t$, leading to a contradiction.

Finally, consider a path of weight $\boldsymbol{w}$ starting at a node u and let $d := \sum_{j=1}^{t} \boldsymbol{w}_i / \boldsymbol{b}_i$. We know that $\Pr[B_{\mathrm{dist}'}(u, d) \subseteq P] \leq \frac{dO(\log n)}{\Delta} \leq d/t$ (by choosing a sufficiently large O-notation constant in the definition of Δ). Since $B_{dist'}(u, d) \subseteq P$ implies that the path is preserved in P, the final property follows and we are done. □

7.4 Hop-Bounded HSTs

In this section, we introduce and construct our novel tree embeddings (or HSTs). The following definition formalizes the encapsulating structure which will be used as our HST.

Definition 7.4.1 (Labeled Tree embedding)**.** *A tree embedding of a graph $G = (V, E_G)$ is a (rooted) tree $T = (V, E_T)$ on the same set of vertices as G where each tree edge $\{u, v\} \in E_T$ is associated with a path in G between u and v. A **labeled tree embedding** consists of a pairs (T, U) where T is a tree embedding on G and $U \subseteq V$ is a subset of (so-called) **good** nodes.*

We note the following rephrasing of the decomposition lemma, using the terminology of hop-bounded paths.

Corollary 7.4.2 (Rephrasing of Lemma 7.3.1)**.** *Let $G = (V, E_G)$ be an undirected graph with $n := |V|$ and weights $w : E_G \to \mathbb{R}_{\geq 0}$. For any $Q \geq 0, b \geq 0, \gamma \leq 1$, there exists a distribution over partitions $P_1 \sqcup P_2 \sqcup \ldots \sqcup P_q = V$ and subsets of the parts $C_i \subseteq P_i$ for $i = 1, \ldots, q$, where*

1. *For each i and $u, v \in P_i$, we have that $\mathrm{dist}_w^{(Q)}(u, v) \leq b$ (with probability 1).*
2. *For each i, $u \in C_i, v \notin P_i$, we have that $\mathrm{dist}_w^{(Q\frac{\gamma}{O(\log n)})}(u, v) \geq b \cdot \frac{\gamma}{O(\log n)}$ (with probability 1).*
3. *For each $v \in V$, we have that $\Pr[v \notin \bigcup_{i \in [q]} C_i] \leq \gamma$.*
4. *For each path p of at most h hops and distance (sum of weights) at most ℓ, we have that $\Pr[p$ is not preserved in $P_1 \sqcup \ldots \sqcup P_q] \leq (h/Q + \ell/b) \cdot O(\log n)$.*

7.4.1 Hop-bounded HST construction

In this section, we construct the HSTs with the properties required to build hop-bounded oblivious routings.

Lemma 7.4.3. *Let $q \geq 1, 0 < p < 1$. Given a weighted graph $G = (V, E_G)$ of hop-diameter $O(p^{-1} q \log n)$ with polynomially bounded weights there exists a polynomial-time sampling of a distribution over labeled tree embeddings (T, U) of G where for all pairs of nodes u, v we have that $|T_{uv}| \leq O(p^{-1} q \log^2 n)$ (always), $\mathbb{E}[\mathbb{I}[u, v \in U] \cdot \mathrm{dist}(T_{uv})] \leq O(\log^2 n) \cdot d_G^{(q)}(u, v)$, and the probability that $\Pr[v \in U] \geq 1 - p$.*

Proof. Let $Q := O(p^{-1} q \log n)$ (for a sufficiently large hidden O-constant).

We describe a recursive tree embedding construction procedure that takes a set of nodes $P \subseteq V$ of (weak) diameter Δ and returns an embedding (T, U). We maintain the invariant that for all $u, v \in P$ we have that $\mathrm{dist}_G^{(Q)}(u, v) \leq \Delta$. The construction is initially invoked with $P = V$ and with diameter $\Delta = \max_{u,v} \mathrm{dist}_G^{(Q)}(u, v) \leq \mathrm{poly}(n)$.

We use the decomposition of Corollary 7.4.2 with hop-diameter Q, distance-diameter $\Delta/2$ and $\gamma := \frac{O(q \log n)}{Q}$ (for a sufficiently large hidden O-constant) to obtain $\{(C_i, P_i)\}_{i=1}^k$ where for each $i, j \in [k], j \neq i$ we have that:

$$\max_{u,v \in P_i} \mathrm{dist}_G^{(Q)}(u, v) \leq \Delta/2 \tag{7.1}$$

$$\mathrm{dist}_G^{(q)}(C_i, P_j) \geq \mathrm{dist}_G^{(Q\frac{\gamma}{O(\log n)})}(C_i, P_j) \geq \Delta \frac{\gamma}{O(\log n)} \geq \Delta \frac{q}{Q} \tag{7.2}$$

$$\Pr[v \notin \bigcup_i C_i] \leq \frac{O(q \log n)}{Q} = \frac{p}{O(\log n)}. \tag{7.3}$$

We recursively construct k tree embeddings $(T_1, U_1), \ldots, (T_k, U_k)$ by calling the same procedure with parameters $\Delta' \leftarrow \Delta/2$ on sets $P_1, \ldots, P_k$. The tree embedding T is constructed by connecting the roots of $T_2, \ldots, T_k$ to the root of T_1 via a tree edge. Specifically, let r be the root of T_1 and r' be the root of T_i. The tree edge $\{r, r'\}$ (i) corresponds to the shortest Q-hop-bounded path in G between r and r', (ii) we label the tree edge with a "hop-label" Q and a "distance-label" Δ. Since $|T_{rr'}| \leq Q$ and $\mathrm{dist}_G(T_{rr'}) \leq \Delta$, this will maintain the following invariant: for each $u, v \in V$ let h be the sum of hop-labels and d be the sum of distance labels on the unique path between u and v in the tree. Then $|T_{uv}| \leq h$ and $\mathrm{dist}_G(T_{uv}) \leq d$. Finally, we let $U := (U_1 \cup \ldots \cup U_k) \cap (C_1 \cup \ldots \cup C_k)$ and return (T, U). The procedure is stopped when the set of nodes P is a singleton.

We now analyze the above procedure. Fix some nodes $u, v \in V$. Clearly, there can be only $O(\log n)$ levels of recursion since the weights of G are polynomially bounded, and the diameter Δ drops by a factor of 2 in each level. Since each tree-edge corresponds to a path of at most Q hops we have that $|T_{uv}| \leq O(Q \log n) = O(p^{-1} q \log^2 n)$. Note that this directly implies $\mathrm{dist}_G(T_{uv}) \geq \mathrm{dist}_G^{(|T_{uv}|)}$ since T_{uv} corresponds to a valid path in G between u and v.

Furthermore, each node $u \in V$ is in a unique recursive call on each level of the recursion (since the sets P among different recursions on the same level are disjoint). In each recursive call with $v \in P$ and decomposition $\{(C_i, P_i)\}_{i=1}^k$, the probability $\Pr[v \notin \bigcup_{i=1}^k C_i] \leq \frac{O(q \log n)}{Q}$ and there are $O(\log n)$ levels, hence we conclude via a union bound the probability that there exists a level i at which v is not in any $\bigcup_{i=1}^k C_i$ is at most $O(\gamma \log n) \leq p$. Hence $\Pr[v \in U] \geq 1 - p$.

Next, let d_{uv} (or just d) be the heaviest distance-label on the tree path between u and v. Note that distance-labels are decreasing exponentially (or faster) on any root-leaf path. Therefore, $\mathrm{dist}_G(T_{uv}) \leq O(d_{uv})$. Let Δ be the (hop-bounded) diameter of G and let $\Delta_i = \Delta/2^i$ be the (hop-bounded) diameter of the parameter P at any recursive call at level i (level $i = 1$ is the root level, up to $i = O(\log n)$ for the lowest, leaf, level of the recursion). This edge of distance-label d_{uv} was created via a recursive call on some level i parameters $P \subseteq V$ and $d_{uv} = 2\Delta_i$ (hence the invariant $\mathrm{dist}_G^{(Q)}(u, v) \leq 2\Delta_i = d_{uv} \leq \mathrm{dist}_G(T_{uv})$). The recursive call yields $\{(C_i, P_i)\}_{i=1}^k$ with the decomposition $P = P_1 \sqcup \ldots \sqcup P_k$ where $u \in P_i$ and $v \in P_j$, $i \neq j$.

By definition of $\mathrm{dist}_G^{(q)}(u, v) < \infty$, there exists a path p in G between u and v of at most q hops and weight $\ell := \mathrm{dist}_G^{(q)}(u, v)$. When $u, v \in U$, we know since $u \in C_i \subseteq P_i$ and $v \notin P_i$ that $\ell \geq \mathrm{dist}_G^{(q)}(u, v) \geq \Delta_i \frac{q}{Q}$. In other words, $\frac{q\Delta_i}{Q} = O(\ell)$.

We give an upper bound on the expectation of d_{uv} (jointly with both u, v being good).

$$\begin{aligned}
\mathbb{E}[d_{uv} \cdot \mathbb{I}[u, v \in U]] &\leq \sum_{i=1}^{O(\log n)} \Pr[p \text{ not preserved at level } i] \cdot \Delta_i \cdot \mathbb{I}[u, v \in U] \\
&\leq \sum_{i=1}^{O(\log n)} O(\log n) \left(\frac{q}{Q} + \frac{\ell}{\Delta_i}\right) \Delta_i \mathbb{I}[u, v \in U] \\
&\leq O(\log n)\mathbb{I}[u, v \in U] \sum_{i=1}^{O(\log n)} \left(\frac{q\Delta_i}{Q} + \ell\right) \\
&\leq O(\log n) \sum_{i=1}^{O(\log n)} O(\ell) \\
&= O(\log^2 n) \cdot \ell = O(\log^2 n) \cdot \mathrm{dist}_G^{(q)}(u, v).
\end{aligned}$$

This completes the proof by noting that $\mathrm{dist}_G(T_{uv}) = O(d_{uv})$. □

7.5 Hop-bounded oblivious routings

In this section, we define and construct novel hop-bounded oblivious routings, as well as give a result that showcases the usage of these structures.

Let $\mathcal{T}$ be a distribution over labeled tree embeddings (T_j, U_j) of $G = (V, E_G)$. In other words, we can represent $\mathcal{T}$ as $\{(\lambda_1, T_1, U_1), (\lambda_1, T_2, U_2), \ldots\}$ where (T_j, U_j) are labeled tree embeddings and $\lambda_i \geq 0, \sum_i \lambda_i = 1$ are normalized weights. If the vector $\lambda = (\lambda_i)_i$ is not immediately clear, we sometimes write $\mathcal{T}(\lambda)$. We define a **good-only routing** on a pair (s_i, t_i) (with respect to $\mathcal{T}$) as the following flow f_i: for each $j \in [|\mathcal{T}|]$ we add a path of value λ_i to the flow along $(T_j)_{s_i,t_i}$ if both $s_i \in U_j$ and $t_i \in U_j$ (and otherwise nothing is added). Similarly, good-only routing on a demand $d \in \mathbb{Z}_{\geq 0}^{V \times V}$ (of total demand $k := |d|$) is a sequence of flows $(f_i)_{i=1}^k$ obtained by collecting the good-only routings of each demand pair (where the pair (s, t) is repeated $d(s, t)$ times).

We distinguish a specific demand called D_1 which has a demand pair of value 1 for each edge E_G. In other words, $D_1(s, t) = \sum_{\{u,v\} \in E_G, u<v} \mathbb{I}[u = s \text{ and } v = t]$.

We now formally define hop-bounded oblivious routings (or HBORs) as a distribution over labeled tree embeddings $\mathcal{T}$ that satisfies three properties. First, any path routed along the tree must have a small hop-length. Second, the total congestion induced by routing the D_1 demand using good-only routings must be small. And third, for each path in G of length at most Q, all of the nodes on the path are good nodes in at least $1/2$-fraction of (T_j, U_j) (w.r.t. weights).

Definition 7.5.1. *A (Q, h, w)-**hop-bounded oblivious routing (HBOR)** of $G = (V, E_G)$ is a distribution over labeled tree embeddings $\mathcal{T} = \{(\lambda_1, T_1, U_1), (\lambda_2, T_2, U_2), \ldots\}$ where (1) for each tree T_j and any two nodes $u, v \in V$ we have that $|T_{uv}| \leq Qh$, (2) let $(f_1, \ldots, f_{|E_G|})$*

be the good-only routing of the D_1 demand, then $\max(\sum_{i=1}^{k} f_i) \leq w$, (3) for each path $p = (v_0, v_1, \ldots, v_\ell)$ in G of length at most Q we have that $\Pr_{(T,U)\sim\mathcal{T}}[\forall i \in [\ell],\ v_i \in U] \geq 1/2$.

We note in the following that the hop-diameter condition is technical and can be removed via a suitable "forest embedding" generalization of tree embeddings, which would somewhat complicate notation.

Lemma 7.5.2. *Given $Q \geq 1$ and graph G of (unweighted) diameter, $O(Q \log n)$ there exists a $(Q, O(\log^2 n), O(\log^2 n))$-**HBOR**.*

Proof. We write an LP to minimize the (maximum) congestion induced by routing the D_1 demand while satisfying other HBOR properties. Let $(T_1, U_1), (T_2, U_2), \ldots$ the (finite) set of possible labeled tree embeddings of G. We denote by Λ the set of vectors $\lambda = (\lambda_i)_i$ such that $\mathcal{T}(\lambda) = \{(\lambda_j, T_j, U_j)\}_j$ is a probability distribution that satisfies properties (1) and (3) of the HBOR definition. The optimization is performed over the set $\Lambda \subseteq \{\lambda \mid \lambda \geq 0, \sum_j \lambda_j = 1\}$.

$$\begin{aligned} \min_{\alpha,\lambda}. \quad & \alpha \\ \text{such that} & \\ \forall e \in E_G \quad & \mathbb{E}_{(T,U)\sim\mathcal{T}(\lambda)}\left[\text{load}(D_1, e, T, U)\right] \leq \alpha \\ & \lambda \in \Lambda \end{aligned}$$

The $\text{load}(D_1, \{u, v\}, T, U) := f_e$ where f is the good-only routing of D_1 with respect to $\{(1, T, U)\}$. In other words, $\text{load}(D_1, \{u, v\}, T, U)$ represents the congestion of the good-only routing induced by routing the D_1 demand over the edge e by routing only over the tree embedding (T, U).

Dualizing this, we get the following:

$$\begin{aligned} \max_{\beta,\ell}. \quad & \beta \\ \text{such that} & \\ \forall \lambda \in \Lambda \quad & \mathbb{E}_{(T,U)\sim\mathcal{T}(\lambda)}\left[\sum_{e\in E_G} \ell_e \cdot \text{load}(D_1, e, T, U)\right] \geq \beta \\ & \ell \geq 0, \sum_{e\in E_G} \ell_e = 1 \end{aligned}$$

In other words, it is sufficient to show for every distribution $(\ell_e)_{e\in E_G}$ there exists a $\lambda \in \Lambda$ (distribution satisfying (1) and (3)) over tree embeddings (T, U) with small expected β.

We apply Lemma 7.4.3 with $q = 1$ and $p = 1/(2(Q+1))$ on G with weights (lengths) ℓ. Specifically, We construct tree embedding distribution $\mathcal{T} = \{(\lambda_i, T_i, U_i)\}_i$ (where λ_i are the probabilities) where (i) for all edges $e = \{u, v\} \in E_G$ we have that $\mathbb{E}[\mathbb{I}[u, v \in U]\text{dist}_G(T_{uv})] \leq O(\log^2 n)\text{dist}_G^{(1)}(u, v) \leq O(\log^2 n)\ell_e$, (ii) for all $u, v \in V$ we have that $|T_{uv}| \leq O(Q \log^2 n)$, (iii) for all $v \in V$ we have that $\Pr[v \in U] \geq 1 - \frac{1}{2(Q+1)}$.

Note that property (1) is immediately satisfied by property (ii), and property (3) follows from (iii) via a union bound: each node is good with probability at least $1 - \frac{1}{2(Q+1)}$ so nodes on any path of length Q are preserved with probability at least $1/2$.

Property (2): we have that

$$\mathbb{E}_{(T,U)\sim\mathcal{T}(\lambda)}[\mathrm{load}(D_1, e, T, U)] = \sum_{(s,t)\in D_1} \mathbb{E}_{(T,U)\sim\mathcal{T}(\lambda)}[\mathrm{load}(\{(s,t)\}, e, T, U)].$$

Furthermore, $\mathbb{E}_{(T,U)\sim\mathcal{T}(\lambda)}[\mathrm{load}(\{s,t\}, e, T, U)] \leq O(\log^2 n) \cdot \ell_{\{s,t\}}$ since the load is 0 if $s \notin U$ or $t \notin U$ and otherwise it holds by property (i). Therefore, $\mathbb{E}_{(T,U)\sim\mathcal{T}(\lambda)}[\mathrm{load}(D_1, e, T, U)] = \sum_{(s,t)\in D_1} O(\log^2 n) \cdot \ell_{\{s,t\}} \leq O(\log^2 n) \sum_{(s,t)\in D_1} \ell_{\{s,t\}} = O(\log^2 n)$. □

The following lemma demonstrates how to use HBORs. Notably, the structure can approximate the congestion of the optimal set of paths while controlling for the dilation. However, the useful paths are "hidden" within a larger set of paths. The user of HBORs is required to be able to identify the useful paths and discard the rest.

Lemma 7.5.3. *Let $H^* = \{p_1^*, \ldots, p_k^*\}$ be paths in G (such that $|H^*| \leq \mathrm{poly}(n)$) with endpoints $\{(s_i, t_i)\}_{i=1}^k$ (respectively). Let $\mathcal{T}(\lambda)$ be a (Q, h, w)-HBOR with $Q \geq \mathrm{dilation}(H^*)$ and $w \gg \log n$. Suppose that for each $i \in [|H^*|]$ we independently sample a tree embedding $(T(i), U(i))$ (proportionally to the weights λ) and set $p_i := T(i)_{s_i,t_i}$. Then there exists a set $H' \subseteq \{p_1, \ldots, p_k\}$ with $\mathbb{E}[|H'|] \geq |H^*|/4$, $\mathrm{congestion}(H') \leq \widetilde{O}(\mathrm{congestion}(H^*) \cdot w)$, and with $\mathrm{dilation}(H') \leq Qh$.*

Proof. Let $H = \{p_1, \ldots, p_k\}$ be the paths selected via the HBOR and call $H^* = \{p_i^*\}_{i=1}^k$ the "witness paths". Property (1) of HBOR ensures that the hop-bound is sufficiently small, i.e., $|p_i| \leq Qh$, hence the dilation property holds for every subset of H.

We construct a subset $H' \subseteq H$ with the required properties. We write $p_i^* \subseteq U(i)$ when all nodes on p_i^* are in $U(i)$. For each (pair) $i \in [k]$ we define the (random) variable $R_i := \mathbb{I}[p_i^* \subseteq U(i)]$ and the set $H' = \{p_i \mid R_i = 1\} \subseteq H$. We have that $\mathbb{E}[R_i] \geq 1/2$ because $Q \geq \mathrm{dilation}(H^*)$ via property (3) of HBOR.

We now argue that H' has small congestion. As a reminder, we consider paths as vectors in $\mathbb{R}_{\geq 0}^{E_G}$. First, we define a (random) flow $F \in \mathbb{R}_{\geq 0}^{E_G}$ as $F := \sum_{i=1}^k R_i \cdot p_i$. By definition, $\sum_{p\in H'} p = F$; F contains exactly $|H'|$ many paths. The expectation $\mathbb{E}[F]$ can, also by definition, be expressed in the following way: For each $i \in [k]$ and each (λ_j, T_j, U_j) from the HBOR, if all nodes of p_i^* are in U_j, then we add $\lambda_i \cdot (T_j)_{s_i,t_i}$ to F (i.e., a path between s_i and t_i in T with weight λ_i). Second, we define $f \in \mathbb{R}_{\geq 0}^{E_G}$ as

$$f := \sum_{i=1}^{k} \mathbb{E}_{(T,U)\sim T} \left[\mathbb{I}[p_i^* \subseteq U] \cdot \sum_{\{u,v\}=e^*\in p_i^*} T_{uv} \right]$$

and we show that $f \geq \mathbb{E}[F]$ (coordinate-wise). In other words, f is a flow constructed as follows: For each $i \in [k]$ and each (λ_j, T_j, U_j) from the HBOR, if all nodes in p_i^* are in U_j, we iterate

over each edge e^* in p_i^* and add to f a path between the endpoints of e^* induced by the tree T_j with weight λ_i (i.e., $\lambda_i(T_j)_{uv}$ where u, v iterate over endpoints of each edge in p_i^*). It is clear that since $\mathbb{E}[F]$ is (the sum of) tree route(s) between s_i and t_i, and f is the (sum of) sum of routes of edges on a path between s_i and t_i, that $f \geq \mathbb{E}[F]$.

We note, by construction, that each path added to f is a good-only routing over its corresponding tree. Furthermore, each edge e in G appears in at most $\text{congestion}(H^*)$ many different witness paths p_i^*, hence we conclude that f is the flow obtained by (fractionally) good-only routing a demand that is (coordinate-wise) smaller than the $\text{congestion}(H^*) \cdot D_1$ demand. Therefore, from the property (2) of HBOR, we conclude that $\max(\mathbb{E}[F]) \leq \max(f) \leq \text{congestion}(H^*) \cdot w$.

Finally, we fix and edge $e \in E_G$ and note that the congestion on e of H' is F_e, hence its expectation is at most $\text{congestion}(H^*) \cdot w$. Since each (s_i, t_i) pair independently samples a path, the congestion of H' over e can be seen as a sum of independent $\{0, 1\}$ variables, hence we conclude it is $O(\text{congestion}(H^*) \cdot w)$ via a standard Chernoff bound argument with high probability. Using a union bound over all edges E_G we conclude that $\text{congestion}(H') = O(\text{congestion}(H^*) \cdot w)$ with high probability.

We now argue about $|H'|$. We have that $|H'| = \sum_{i=1}^{k} R_i$, hence $\mathbb{E}[|H'|] \geq k/2$. However, since $k \leq \text{poly}(n)$ we conclude that in events where the above properties hold with high probability we have $\mathbb{E}[|H'|] \geq (1/2 - 1/n)|H^*| \geq |H^*|/4$. □

7.6 Routing with Noise

In this section, we propose a useful subroutine that can identify in a distributed manner a useful set of paths (i.e., of small congestion+dilation) from a larger set of possibly uncontrolled dilation or congestion. This routine demonstrates how to use HBORs in a distributed setting.

Let $H = \{p_1, p_2, \ldots, p_k\}$ be a set of paths in G. A classic routing result states we can simultaneously send packets via the paths H in $O(\text{dilation}(H) + \text{congestion}(H))$ rounds (in this model each edge can forward one packet per round) [100].

We now consider an extension of the result where we have a set of paths M that can be routed efficiently due to $\text{dilation}(M) + \text{congestion}(M) \leq Q$. Consider an adversary that blows up the congestion and dilation by inserting at most $\alpha|M|$ new paths. We show one can still recover a distributed algorithm that routes a good fraction of the original paths in at most $\tilde{O}(Q/\alpha)$ rounds.

Lemma 7.6.1. *Given a set of k paths $\{p_1, \ldots, p_k\}$ in G, suppose there exists a (secret) subset of indices $M \subseteq [k]$ of size $|M| \geq \alpha k$ such that $\text{congestion}(\{p_i\}_{i \in M}) + \text{dilation}(\{p_i\}_{i \in M}) \leq Q$. There exists a (randomized) distributed CONGEST algorithm that will successfully complete the routing of at least $|M|/2$ paths in M in $O(Q\alpha^{-1} \log n)$ rounds with high probability. Every node needs to know Q, α, and to be able to compute the next hop of any packet it received; however, they do not need to know M.*

Proof. The issue is dealing with the congestion: We can simply augment packets with a counter

denoting the number of times it has been forwarded and drop any packet that continues its propagation after it has been forwarded Q times. This will enforce the dilation constraint.

We group the rounds into $\kappa \log n$ blocks called "super rounds" ($\kappa > 0$ is a sufficiently large constant). Furthermore, we set $c := 2Q/\alpha$ and delay the start of the propagation of each p_i by an independent and uniformly random number of super rounds in $[c]$. In each super round we do the following: each edge (locally) considers the amount of packets going over it. If this number is at most $\kappa \log n$, we send all of those packets through the edge in $\kappa \log n$ rounds. If this is not the case, we deactivate the edge (hence deactivating all packets wanting to pass through the edge in this super round or a future one).

We now analyze the above algorithm. Each packet, after its initial delay, in each super round either gets deactivated or forwarded. Hence after $c + Q$ super rounds, all packets are either delivered to their destination or deactivated. Hence we stop the procedure after $O(Q \cdot \alpha^{-1})$ super rounds or $O(Q\alpha^{-1} \log n)$ rounds.

Suppose that an edge e got deactivated in some round. Let $t = t_m + t_b$ be the total number of packets that want to go through the edge during or after deactivation, where t_m of them are in the set M and t_b (for "bad") are not from M. Using random delays, we know that in each super round the number of messages waiting on e is at most $x := 2t/c + O(\log n)$ with high probability. If $x > \kappa \log n$, then $t > 1/2 \cdot (\kappa - O(1)) \cdot (c \log n) = \kappa' Q\alpha^{-1} \log n \geq \kappa'' Q\alpha^{-1}$ (for some sufficiently large $\kappa', \kappa'' > 0$) with high probability. By assumption we have that $t_m \leq Q$, therefore $t_b \geq t - Q \geq \kappa'' Q\alpha^{-1} \geq 2Q\alpha^{-1}$. In other words, we deactivate at most Q paths in M and at least $2Q\alpha^{-1}$ paths not in M.

Consider the potential function $\phi = 2|A \cap M| - \alpha|A \setminus M|$ where A is the set of active (non-deactivated) paths. In the beginning $\phi_{\text{start}} \geq 2|M| - \alpha k \geq |M|$. On each deactivation the change in $\phi_{i+1} - \phi_i = -2t_m + \alpha t_b \geq -2Q + \alpha 2Q\alpha^{-1} = 0$, hence ϕ is never decreasing. Since in the end we have $\phi_{\text{end}} \geq \phi_0 \geq |M|$, we have that $|A \cap M| \geq \frac{1}{2}M$. In other words, $|M|/2$ messages got delivered. □

7.7 Distributed and Oblivious Shortcut Construction

In this section, we present how to construct pairwise shortcuts using HBORs and the "routing with noise" subroutine of Section 7.6.

In distributed optimization, it is often infeasible for each node to know the entire input or output (e.g., transmitting $\Theta(n)$ to the same node might take a prohibitively long amount of time). Therefore, the input/output is often represented in a distributed way, which we here define. We say that a distributed algorithm **distributedly knows** a set of pairs $P = \{(s_i, t_i)\}_{i=1}^k$ where $s_i, t_i \in V$ if every node v knows all pairs that contain v. In other words, v knows all pairs $\{(s, t) \in P \mid s = v \text{ or } t = v\}$. Furthermore, an algorithm **distributedly knows** a set of paths $H = \{p_i\}_{i=1}^k$ if for each edge $\{u, v\} = e \in E_G$ both u and v know all (indices of) all paths that contain e.

The following lemma stipulates that one can use HBORs to identify a shortcut for a constant

fraction of pairs, even if a good-quality shortcut exists only for a subset of the pairs (albeit this subset can be at most a constant fraction of all pairs).

Lemma 7.7.1. *Let $P = \{(s_i, t_i)\}_{i=1}^{k}$ be k pairs of nodes in G such that there exists a (secret) subset $P^* = \{(s_i, t_i)\}_{i \in I} \subseteq P$ of pairs with $|P^*| \geq \Omega(k)$ (for any arbitrary constant) such that pairs P^* admit a shortcut of* $\text{quality} = \text{congestion} + \text{dilation} \leq \text{OPT}$. *There is a randomized distributed known-topology CONGEST algorithm that constructs shortcuts of quality $\tilde{O}(\text{OPT})$ for some subset of pairs $P' \subseteq P$ of size $|P'| \geq |P^*|/2$ in $\tilde{O}(\text{OPT} + D_G)$ rounds with high probability. The nodes must initially know* OPT, $|P^*|$, *and distributedly know P (but not P^* nor its secret shortcut). Upon termination, the nodes distributedly know P' and the shortcut paths for P'.*

Proof. We first prove the claim assuming the hop-diameter $D_G = O(\text{OPT} \log n)$ (note that computing a 2-approx to D_G can easily be done in $O(D_G)$ rounds by building a BFS tree and computing its diameter).

For concreteness, we will assume $|P^*| \geq k/100$, but the analysis holds accordingly for other constants. Since we are in the known-topology setting, all nodes can agree on a tree embedding distribution $\mathcal{T}$ which is an $(\text{OPT}, O(\log^2 n), O(\log^2 n))$-HBOR of G using Lemma 7.5.2. For each pair (s_i, t_i) the node s_i chooses a path p_i via the procedure described in Lemma 7.5.3 and (using the same claim, but only for the secret subset of pairs P^*) there exists a (secret) subset of paths $H^s \subseteq \{p_1, \ldots, p_k\}$ with $\mathbb{E}[|H^s|] \geq \frac{1}{4}|P^*| \geq \frac{1}{400}k$, $\text{congestion}(H^s) = O(\text{OPT} \log^2 n)$, and $\text{dilation}(H^s) \leq O(\text{OPT} \log^2 n)$. Furthermore, since $|H^s| \in [0, k]$ and $\mathbb{E}[|H^s|] \geq k/400$ we conclude by Markov that

$$\Pr[|H^s| \geq \frac{k}{800}] \leq \frac{k/400 - k/800}{k - k/800} = \frac{1/800}{1 - 1/800} = \Omega(1).$$

We call such events "lucky". In case of a lucky event, applying Lemma 7.6.1 on $\{p_1, \ldots, p_k\}$ with a secret subset H^s with $|H^*| \geq k/800$, we have that there is a distributed CONGEST $\tilde{O}(\text{OPT})$-round algorithm that can successfully route $|H^s|/2 \geq k/1600$ packets via a subset of the paths $\{p_1, \ldots, p_k\}$ in $\tilde{O}(\text{OPT})$ rounds with high probability. We note that we augment the packets sent from s_i to t_i with the random seeds used to choose the path p_i in order for intermediate nodes to be able to compute the next step. Those pairs (s_i, t_i) for which s_i successfully delivers a packet to t_i are called "successful" (at this point only t_i know whether a pair was successful). However, reversing the routing procedure both pair nodes can be made aware whether the pair was successful.

Note that the set of paths corresponding to successful pairs have both congestion and dilation $\tilde{O}(\text{OPT})$ (since otherwise delivering them in $\tilde{O}(\text{OPT})$ rounds would be impossible). Hence we can add the successful pairs to P', paths of successful pairs to the computed shortcut, and delete those pairs from consideration. In case of a lucky event, we delete $k/1600$ pairs from consideration with high probability. The procedure above is repeated (on the pairs that were never successful) until the number of pairs that were ever successful (i.e., $|P'|$) is at least $|P^*|/2$. Since an iteration is lucky with constant probability, repeating $O(\log n)$ times yields the $|P'| \geq |P^*|/2$ and will blow up the congestion by at most a negligible $O(\log n) = \tilde{O}(1)$ factor. Note that the

procedure could not be repeated until $|P'| \geq |P^*|$ due to the requirement of Lemma 7.6.1 that the secret subset H^s be large relative to H; in further iterations, the constants slightly change, but they remain constants until $|P'| \leq (1 - \Omega(1))|P^*|$ (we leave the details out for simplicity of exposition). Finally, we note that the algorithm does not know whether an event was lucky, but can calculate the total number of successful pairs in $O(D_G)$ rounds and repeat the process until $|P'| \leq |P^*|/2$, which will work as argued. This completes the proof in the case of $D_G = O(\text{OPT} \log n)$.

Finally, we assume general $G = (V, E_G)$ (with larger hop-diameters). By known results of (e.g., [112]), there exists a partition a (randomized) partition of V into $Z_1 \sqcup Z_2 \sqcup \ldots \sqcup Z_q$ such that (1) the hop-diameter of Z_i is $O(\text{OPT} \log n)$ for all i (this holds when Z_i is taken in isolation from the rest of the graph, i.e., it is a "strong" diameter), and (2) for each path p in G of at most OPT hops (i.e., $|p| \leq \text{OPT}$) we have that $\Pr[p \text{ is preserved in } Z_1 \sqcup \ldots \sqcup Z_q] \geq 1/2$ ("preserved" as in the terminology of Lemma 7.3.1). Since we are in the known-topology setting all nodes can agree on the partition without distributed communication (note: this decomposition could be distributedly computed in $\tilde{O}(\text{OPT})$ rounds using the random shifts technique introduced in [112] for PRAM and adapted in, e.g., [67] to the CONGEST model).

Let I be is the (secret) set of pair indices such that those indices are in P^*, i.e., such that $P^* = \{(s_i, t_i)\}_{i \in I} \subseteq P$. Let $H^* = \{p_i^*\}_{i \in I}$ be the (secret) shortcut of $\text{congestion} + \text{dilation} \leq \text{OPT}$ with endpoints P^*. For each partition $j \in [q]$, we define U_j to be the number of parts $i \in [k]$ such that both endpoints s_i and t_j are in Z_j. Similarly, for each partition $j \in [q]$, we define R_j to be the number of parts $i \in [k]$ such that both $i \in I$ and the entire path p_i^* is contained in Z_j.

By the partition properties, we have that $\mathbb{E}[\sum_{j=1}^{q} R_j] \geq |P^*|/2$. Call a partition Z_j "happy" if $R_j \geq U_j/1000$. We have that:

$$\begin{aligned}
|P^*|/2 &\leq \mathbb{E}[\sum_j R_j] \\
&= \mathbb{E}[\sum_j R_j \mathbb{I}[j \text{ happy}] + \sum_j R_j \mathbb{I}[j \text{ not happy}]] \\
&\leq \mathbb{E}[\sum_j R_j \mathbb{I}[j \text{ happy}]] + \sum_j U_j/1000 \\
&\leq \mathbb{E}[\sum_j R_j \mathbb{I}[j \text{ happy}]] + k/1000.
\end{aligned}$$

Therefore, we have that $\mathbb{E}\left[\sum_j R_j \mathbb{I}[j \text{ happy}]\right] \geq |P^*|/2 - k/1000 \geq |P^*|/4 \geq k/400$ (since $k/1000 \leq |P^*|/4$), hence its value is at least $|P^*|/8$ with constant probability (by Markov, the random variable is bounded above by k). Call this event "lucky", i.e., when $\sum_j R_j \mathbb{I}[j \text{ happy}] \geq |P^*|/8$.

We now describe the algorithm. We apply this claim on each partition Z_j that has hop-diameter at most $O(OPT \log n)$, and the claim is already proven in this case. Note that the algorithm can compute U_j by aggregating the values within each partition Z_j withing $D_{Z_j} = \tilde{O}(\text{OPT})$ rounds, but it cannot compute R_j due to H^* being secret. Therefore, in the recursive call, we set the (necessary input parameter) $|P^*|$ to be $U_j/1000$ (this is a quantity that the nodes must

know during initialization); it is clear that nodes know all other inputs that are required. Every happy partition will, with high probability, find shortcuts for at least $U_j/2000 \geq R_j/2000$ pairs (since $R_j \leq U_j$ by definition). Therefore, (conditioning on a lucky even), we find shortcuts for $\sum_j \frac{R_j}{2000}\mathbb{I}[j \text{ is happy}] \geq \frac{1}{2000} \cdot \frac{|P^*|}{4} \geq \frac{|P^*|}{8000}$ many pairs. Repeating the procedure $O(\log n)$ many times on pairs for which have not found shortcut (in a similar manner as in the bounded-diameter case), we get the final results. □

Finally, we show one can use HBORs to construct near-optimal-quality pairwise shortcuts.

Theorem 7.7.2 (Pairwise shortcut construction)**.** *Let $P = \{(s_i, t_i)\}_{i=1}^k$ be k pairs of nodes in G that admit a shortcut of* $\text{quality} = \text{congestion} + \text{dilation} \leq \text{OPT}$. *There is a randomized distributed known-topology CONGEST algorithm that constructs shortcuts of quality $\tilde{O}(\text{OPT})$ in $\tilde{O}(\text{OPT} + D)$ rounds with high probability. A node v initially distributedly knows P and at termination will distributedly know the computed shortcut.*

Proof. We can assume we know the value of OPT up to a factor of 2 by guessing the smallest power-of-2 larger Q than OPT and verifying if the procedure succeeded. Checking whether a shortcut has quality q can be easily done in $\tilde{O}(q)$ CONGEST rounds with high probability by subsampling the pairs (s_i, t_i) with probability $\frac{\log n}{q}$ (non-sampled pairs are ignored) and checking whether each edge in G has congestion $O(\log n)$, which will be the case with high probability (note: this verification step can be avoided, but we keep it for simplicity of exposition).

We apply Lemma 7.7.1 on G with $P^* = P$ and with our guess Q for OPT. The algorithm constructs shortcut paths for $k/2$ pairs. We remove those pairs and iterate the procedure $O(\log n)$ times, after which all pairs will have a shortcut with high probability. □

Chapter 8

Conclusion and Open Questions

8.1 Summary

In this thesis, we explore efficient distributed algorithms for network optimization. Our primary objective is to challenge the notion in distributed optimization that one should not look beyond $\tilde{\Theta}(\sqrt{n})$-round algorithms simply because there exists a pathological worst-case topology where a lower bound of $\tilde{\Omega}(\sqrt{n})$ applies. We argue that a significantly more helpful answer can be given: many topologies of interest do not share the properties of pathological graphs and should not be bound by their worst-case barriers. We believe that our understanding of distributed algorithmics beyond worst-case topologies has worthwhile theoretical and potentially practical benefits. Our pursuit leads to the resolution of several long-standing open problems in theoretical computer science. Furthermore, since performing a practical network optimization on a worst-case topology would be prohibitively expensive on large networks, improving our understanding might lead to future real-world practical improvements in network design and network optimization.

Our contribution can be summarized in two key topics. First, we develop a toolbox that can be utilized to design clean and efficient algorithms for many distributed optimization problems that provably outperform traditional $\tilde{O}(\sqrt{n})$-round solutions. The core of this toolbox is the tree-restricted shortcut framework that provides a distributed algorithm that can be executed without modification on various topologies (i.e., uniform algorithms) and yields ultra-fast runtimes on many networks of interest.

Second, we develop a theory that explains how a network topology influences distributed optimization runtimes. We prove that the *shortcut quality* parameter (or several equivalent quantities) presents an efficiency barrier on many distributed tasks. This proves the first non-trivial universal lower bound for distributed optimization, a result that combines various new information-theoretic and combinatorial techniques. Furthermore, we show that in the known-topology setting, it is possible to solve distributed optimization problems in *shortcut quality* time. This immediately proves the existence of a distributed algorithm that is as fast as possible on a given topology, i.e., so-called universally optimal algorithms. This resolves an open problem of Garay, Kutten and Peleg from an influential FOCS'93 paper [46]. The result builds on top of novel

graph-theoretic structures that include a $\text{congestion} + \text{dilation}$ oblivious routing with polylogarithmic approximation ratio, resolving an open problem by Räcke [125] about the existence of such oblivious routing structures.

8.2 Open Problems and Future Work

8.2.1 The shortcut framework

The tree-restricted shortcut framework matured to a well-rounded algorithmic technique that can provide efficient distributed algorithms for a large variety of graph classes (culminating even in the excluded-minor class of graphs) and for many network optimization problems. However, the set of problems one can apply the framework on has not yet reached a satisfactory level: this set currently includes the (exact) MST, $n^{o(1)}$-approximate SSSP, and $(1+\varepsilon)$-approximate min-cut (see Section 2.1.3 for a survey). One would ideally add other techniques to the list more such techniques. The most straightforward way to improve on these techniques would be to reduce some new optimization problems to multiple applications of the partwise aggregation oracle.

Open Problem 8.2.1. *Extend the (tree-restricted) shortcut framework to other problems in distributed computing. These might include the exact SSSP, $s-t$ maximum flow, exact min-cut, etc.*

Furthermore, we note that the shortcut framework is highly tied to the CONGEST model of communication: e.g., nodes can send different messages to different neighbors, which is accounted for by controlling the number of different parts that can communicate over each edge. On the other hand, a model such as broadcast CONGEST, where a node broadcasts the same $O(\log n)$-bit message to each one of its neighbors would benefit more from a node-centered congestion definition where we control the number of different parts that are allowed to communicate via each node. Other models warrant similar investigation.

Open Problem 8.2.2. *Develop an appropriate shortcut construction that gives efficient distributed algorithms for problems in the broadcast CONGEST or multi-hop radio network models.*

Furthermore, other measures of complexity might require combinatorial structures different from shortcuts. Low-congestion shortcuts are mostly tied to the question of minimizing the running time and, via the results of [67], also the message complexity of distributed tasks. One the other hand, minimizing the energy consumption might require different constructions.

Open Problem 8.2.3. *Develop an appropriate shortcut construction that gives distributed algorithms that control alternative complexity measures such as the energy consumption in addition to the round complexity.*

8.2.2 Coding gaps for completion-time

The material mostly contained in Chapter 5 studies completion-time coding gaps; i.e., the ratio for a given multiple-unicast instance, of the fastest routing protocol's completion time to the fastest coding protocol's completion time. It provides a strong characterization of these gaps in the worst case, showing they can be polylogarithmic in the problem parameters, but no greater. The chapter raises a few exciting questions and research directions.

Probably the most natural question is to close our upper and lower bounds. We show that the network coding gap is polylogarithmic, but what polylog? A particular question is can we replace the term that (logarithmically) depends on the demand aspect ratio, namely $O\left(\log \frac{\sum_i d_i}{\min_i d_i}\right)$, with a $O(\log k)$?

Open Problem 8.2.4. *Determine the correct bound on the completion-time network coding gap.*

Implications for other fields. Recall that the multiple-unicast conjecture of Harvey et al. [75], Li and Li [103] asserts that the throughput coding gap is *one* for multiple unicasts. In addition to being considered "arguably the most important open problem in the field of network coding" [3], this conjecture has also been connected to other seemingly unrelated areas of theoretical computer science. For example, a positive resolution of this conjecture has been shown to imply (1) an answer to a long-standing open question in external memory algorithm complexity [3, 41], (2) improved lower bounds for computation in the cell-probe model [3], and (3) (very recently) an $\Omega(n \log n)$ circuit size lower bound for multiplication of n-bit integers [5] (matching an even more recent breakthrough algorithmic result for this fundamental problem [74]). This conjecture has therefore found applications in proving (conditional) lower bounds. As discussed in Section 5.1 and Section 5.1.4, the conjectured non-existence of throughput coding gaps for multiple unicast has been used to prove (conditional) lower bounds in many seemingly-unrelated problems. It would be interesting to see whether our upper and lower bounds on the coding gap for multiple-unicast completion times can be used to prove *unconditional* lower bounds for other models of computation. Perhaps the most exciting direction would be to investigate whether completion-time coding gaps imply new results in circuit complexity for depth-bounded circuits.

Open Problem 8.2.5. *Explore the possibility of the completion-time coding gap results implying unconditional lower bounds in other models (such as depth-bounded circuit complexity).*

8.2.3 Universal optimality

We demonstrate the existence of universally optimal algorithms in the distributed known-topology setting. The most pressing open question is whether a similar result holds in the general setting (when the topology is not known).

Open Problem 8.2.6. *Show the existence of universally optimal algorithms in the distributed CONGEST setting.*

This thesis makes great strides towards this problem. Many of the techniques used to prove the known-topology setting can be reused for the fully distributed result. Specifically, the following open problem would imply Open Problem 8.2.6.

Open Problem 8.2.7. *Design a distributed algorithm that constructs a* $(Q, \tilde{O}(1), \tilde{O}(1))$*-hop-bounded oblivious routing in* $\tilde{O}(Q)$ *rounds.*

Bibliography

[1] Ittai Abraham and Cyril Gavoille. Object location using path separators. In *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 188–197. ACM, 2006. 4.1, 4.1.2

[2] Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. *SIAM Journal on Computing (SICOMP)*, 48(3):1120–1145, 2019. 5.2.3

[3] Micah Adler, Nicholas JA Harvey, Kamal Jain, Robert Kleinberg, and April Rasala Lehman. On the capacity of information networks. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 241–250. Society for Industrial and Applied Mathematics, 2006. 5.1, 5.1.4, 5.5, 5.6, 2, 8.2.2

[4] Peyman Afshani, Jérémy Barbay, and Timothy M Chan. Instance-optimal geometric algorithms. *Journal of the ACM (JACM)*, 64(1):1–38, 2017. 6.5

[5] Peyman Afshani, Casper Benjamin Freksen, Lior Kamma, and Kasper Green Larsen. Lower bounds for multiplication via network coding. In *Proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 10:1–10:12, 2019. 5.1, 5.1.4, 8.2.2

[6] Amit Agarwal and Moses Charikar. On the advantage of network coding for improving network throughput. In *Information Theory Workshop*, pages 247–249, 2004. 5.1.4

[7] Rudolf Ahlswede, Ning Cai, S-YR Li, and Raymond W Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000. 5.1, 5.1.4

[8] Ali Al-Bashabsheh and Abbas Yongaçoglu. On the k-pairs problem. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pages 1828–1832, 2008. 5.1, 5.1.4

[9] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2):5, 2009. 5.1.3, 5.2, 5.2.3, 5.4

[10] Friedhelm Meyer auf der Heide and Berthold Vöcking. Shortest-path routing in arbitrary networks. *Journal of Algorithms*, 31(1):105–131, 1999. 5.1

[11] Yonatan Aumann and Yuval Rabani. An o (log k) approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing (SICOMP)*, 27(1):291–301, 1998. 5.1, 5.1.4

[12] Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree,

counting, leader election, and related problems. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 230–240, 1987. 1.1.1, 6.5

[13] Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *Journal of the ACM (JACM)*, 37 (2):238–256, 1990. 6.5

[14] Dimitris Bertsimas and David Gamarnik. Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999. 5.1, 5.1.4

[15] Anna Blasiak, Robert Kleinberg, and Eyal Lubetzky. Lexicographic products and the power of non-linear network coding. In *Proceedings of the 52nd Symposium on Foundations of Computer Science (FOCS)*, pages 609–618, 2011. 5.1.3

[16] Hans L Bodlaender. Nc-algorithms for graphs with small treewidth. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 1–10. Springer, 1988. 4.3

[17] Hans L Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing (SICOMP)*, 27(6):1725–1746, 1998. 3.4

[18] Mark Braverman, Sumegha Garg, and Ariel Schvartzman. Coding in undirected graphs is either very helpful or not helpful at all. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 18:1–18:18, 2017. 5.1.3, 5.3, 5.3.4, 5.4, 5.5, 5.6

[19] Costas Busch, Malik Magdon-Ismail, Marios Mavronicolas, and Paul Spirakis. Direct routing: Algorithms and complexity. In *Proceedings of the 12th Conference on Computability in Europe (CiE)*, pages 134–145, 2004. 5.1

[20] Costas Busch, Malik Magdon-Ismail, and Jing Xi. Oblivious routing on geometric networks. In *Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 316–324, 2005. 3, 7.1

[21] Costas Busch, Malik Magdon-Ismail, and Jing Xi. Optimal oblivious path selection on the mesh. *IEEE Transactions on Computers*, 57(5):660–671, 2008. 3, 7.1

[22] Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 183–192, 2005. 5.1.4

[23] Chandra Chekuri, Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Embedding k-outerplanar graphs into l1. *SIAM Journal on Discrete Mathematics*, 20(1): 119–136, 2006. 5.1.4

[24] Chandra Chekuri, F Bruce Shepherd, and Christophe Weibel. Flow-cut gaps for integer and fractional multiflows. *Journal of Combinatorial Theory, Series B*, 103(2):248–273, 2013. 5.1.4

[25] Chandra Chekuri, Sudeep Kamath, Sreeram Kannan, and Pramod Viswanath. Delay-constrained unicast and the triangle-cast problem. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 804–808. IEEE, 2015. 5.1, 5.1.4, 5.6

[26] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proceedings of the VLDB*

Endowment, 8(12):1804–1815, 2015. 1, 6.1

[27] 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 Journal on Computing (SICOMP)*, 41(5):1235–1265, 2012. (document), 1, 1.1.1, 1.1.2, 1.2.2, 2.1.1, 2.1.3, 5.1, 5.1.3, 5.1.3, 5.2, 5.1.3, 6.1, 6.1.1, 6.1.1, 6.1.1, 4, 6.1.1, 6.2, 6.2.1, 6.2.1, 6.3.1, 6.3.1, 6.3.4, 6.5, 6.6

[28] Supratim Deb, Muriel Médard, and Clifford Choute. Algebraic gossip: A network coding approach to optimal multiple rumor mongering. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2486–2507, 2006. 5.1, 5.1.4

[29] Erik D Demaine, Fedor V Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893, 2005. 4.1.2

[30] Erik D Demaine, Mohammad Taghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *Proceedings of the 46th Symposium on Foundations of Computer Science (FOCS)*, pages 637–646. IEEE, 2005. 4.1.2

[31] Matt DeVos, Guoli Ding, Bogdan Oporowski, Daniel P Sanders, Bruce Reed, Paul Seymour, and Dirk Vertigan. Excluding any graph as a minor allows a low tree-width 2-coloring. *Journal of Combinatorial Theory, Series B*, 91(1):25–41, 2004. 4.1.2

[32] Vida Dujmović, Pat Morin, and David R Wood. Layered separators in minor-closed graph classes with applications. *Journal of Combinatorial Theory, Series B*, 2017. 4.4.1

[33] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a byzantine environment i: crash failures. In *Theoretical Aspects of Reasoning about Knowledge*, pages 149–169, 1986. 6.5

[34] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. *Journal of Computer and System Sciences*, 72(8):1282–1308, 2006. 6.1, 6.2, 6.5

[35] Michael Elkin. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM Journal on Computing (SICOMP)*, 36 (2):433–456, 2006. (document), 5.1.3, 5.1.3, 5.2, 5.1.3, 6.3.1, 6.5

[36] Michael Elkin. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 757–770, 2017. 1, 6.1

[37] Michael Elkin. A simple deterministic distributed mst algorithm, with near-optimal time and message complexities. 2017. 1, 6.1, 6.5

[38] David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000. 4.4.1

[39] David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 599–608. Society for Industrial and Applied Mathematics, 2003. 3.6.5, 4.6.1

[40] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for mid-

dleware. *Journal of computer and system sciences*, 66(4):614–656, 2003. 6.5

[41] Alireza Farhadi, MohammadTaghi Hajiaghayi, Kasper Green Larsen, and Elaine Shi. Lower bounds for external memory integer sorting via network coding. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, page To Appear, 2019. 5.1, 5.1.4, 8.2.2

[42] Paola Flocchini and Flaminia L Luccio. Routing in series parallel networks. *Theory of Computing Systems*, 36(2):137–157, 2003. 4.1

[43] Christina Fragouli, Jörg Widmer, and Jean-Yves Le Boudec. Efficient broadcasting using network coding. *IEEE/ACM Transactions on Networking (TON)*, 16(2):450–463, 2008. 5.1, 5.1.4

[44] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1162, 2012. 1, 6.1

[45] 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*, 5(1):66–77, 1983. 1.1.1, 6.2, 6.5

[46] Juan A Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing (SICOMP)*, 27(1):302–316, 1998. 1.1.2, 6.1, 6.1.1, 6.2, 6.5, 8.1

[47] Mohsen Ghaffari. Near-optimal distributed approximation of minimum-weight connected dominating set. In *Proceedings of the 41nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 483–494, 2014. 6.2

[48] Mohsen Ghaffari. Distributed broadcast revisited: Towards universal optimality. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 638–649. Springer, 2015. 1.1.2, 2, 3, 6.5

[49] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 202–219, 2016. (document), 1.2.2, 1.2.2, 1.2.5, 1.3, 2.1.1, 2.1.2, 2.1.2, 2.1.3, 2.1.3, 2.1.3, 2.2.2, 2.2.4, 2.2.2, 2.2.3, 2.2.4, 2.3.3, 3.1, 3.2, 3.6.2, 5.2, 6.1, 6.2, 6.5

[50] Mohsen Ghaffari and Bernhard Haeupler. Low-congestion shortcuts for graphs excluding dense minors. 2020. 2.1.3, 1, 4

[51] Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *Proceedings of the 27th International Symposium on Distributed Computing (DISC)*, pages 1–15, 2013. 1, 6.2

[52] Mohsen Ghaffari and Jason Li. New Distributed Algorithms in Almost Mixing Time via Transformations from Parallel Algorithms. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, pages 31:1–31:16, 2018. 2.1.3, 6.1, 6.5

[53] Mohsen Ghaffari and Merav Parter. MST in log-star rounds of congested clique. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*,

pages 19–28, 2016. 6.5

[54] Mohsen Ghaffari and Merav Parter. Near-Optimal Distributed DFS in Planar Graphs. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC)*, pages 21:1–21:16, 2017. 6.1

[55] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 81–90. ACM, 2015. 1, 6.1, 6.2

[56] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed mst and routing in almost mixing time. pages 131–140, 2017. 6.1, 6.5

[57] Ashish Goel and Sanjeev Khanna. On the network coding advantage for wireless multicast in euclidean space. In *Proceedings of the 7th international conference on Information processing in sensor networks*, pages 64–69. IEEE Computer Society, 2008. 5.1, 5.1.4

[58] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. Graphx: Graph processing in a distributed dataflow framework. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 599–613, 2014. 1, 6.1

[59] Martin Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003. 4.1.2

[60] Hermann Gruber. On balanced separators, treewidth, and cycle rank. *arXiv preprint arXiv:1012.1344*, 2010. 3.7

[61] Anupam Gupta, Robert Krauthgamer, and James R Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS)*, page 534, 2003. 5.2.3, 5.2.3, 5.2.8, 5.2.3, 7.3, 7.3, 7.3.3

[62] Bernhard Haeupler. Simple, fast and deterministic gossip and rumor spreading. *Journal of the ACM (JACM)*, 62(6):47, 2015. 5.1, 5.1.4

[63] Bernhard Haeupler. Analyzing network coding (gossip) made easy. *Journal of the ACM (JACM)*, 63(3):26, 2016. 5.1, 5.1.4

[64] Bernhard Haeupler and Jason Li. Faster Distributed Shortest Path Approximations via Shortcuts. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, pages 33:1–33:14, 2018. 1.2.2, 2.1.3, 2.1.3, 6.1, 6.1.1, 6.2

[65] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Low-congestion shortcuts without embedding. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 451–460, 2016. 2, 2.1.3, 3.6.2, 6.1, 6.1.1, 6.1.1, 6.5

[66] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, pages 158–172, 2016. 3, 4.4.1, 4.4.5, 4.5, 6.1, 6.5

[67] Bernhard Haeupler, D Ellis Hershkowitz, and David Wajc. Round-and message-optimal distributed graph algorithms. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 119–128, 2018. 2.1.3, 2.1.3, 2.1.4, 2.1.3, 6.1,

6.5, 7.7, 8.2.1

[68] Bernhard Haeupler, Jason Li, and Goran Zuzic. Minor excluded network families admit fast distributed algorithms. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 465–474. ACM, 2018. 1, 4, 6.1, 6.5

[69] Bernhard Haeupler, David Wajc, and Goran Zuzic. Network coding gaps for completion times of multiple unicasts. *arXiv preprint arXiv:1905.02805*, 2019. 5, 6.1.1, 6.1.1, 6.2.2, 6.2.2, 6.2.4, 6.4.1, 6.4.4, 6.6, 6.6, 6.2.4, 6.6

[70] Bernhard Haeupler, David Wajc, and Goran Zuzic. Shortcuts are universal lower bounds for distributed optimization. *in submission*, 2020. 6

[71] Eran Halperin, Guy Kortsarz, Robert Krauthgamer, Aravind Srinivasan, and Nan Wang. Integrality ratio for group steiner trees and directed steiner trees. *SIAM Journal on Computing (SICOMP)*, 36(5):1494–1511, 2007. 5.1.4

[72] Minyang Han and Khuzaima Daudjee. Giraph unchained: Barrierless asynchronous parallel execution in pregel-like graph processing systems. *Proceedings of the VLDB Endowment*, 8(9):950–961, 2015. 1, 6.1

[73] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing (SICOMP)*, 13(2):338–355, 1984. 4.3

[74] David Harvey and Joris Van Der Hoeven. Polynomial multiplication over finite fields in time $O(n \log n)$. *HAL preprint hal-02070816*, 2019. 5.1.4, 8.2.2

[75] Nicholas J Harvey, Robert D Kleinberg, and April Rasala Lehman. Comparing network coding with multicommodity flow for the k-pairs communication problem. Technical report, MIT, CSAIL, 2004. 5.1, 5.1.4, 8.2.2

[76] Nicholas JA Harvey, Robert Kleinberg, and April Rasala Lehman. On the capacity of information networks. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2345–2364, 2006. 5.1, 5.1.4

[77] James W Hegeman, Gopal Pandurangan, Sriram V Pemmaraju, Vivek B Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 91–100, 2015. 6.5

[78] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. An almost-tight distributed algorithm for computing single-source shortest paths. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, 2016. 1, 6.1

[79] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006. 5.1.4

[80] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 355–364, 2012. 1, 6.1

[81] T Chiang Hu. Multi-commodity network flows. *Operations research*, 11(3):344–360, 1963. 5.1.2, 5.1.4

[82] Chien-Chung Huang, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed exact weighted all-pairs shortest paths in õ (nˆ{5/4}) rounds. In *Proceedings of the 58th Symposium on Foundations of Computer Science (FOCS)*, pages 168–179. IEEE, 2017. 1, 6.1

[83] Jiaqing Huang, Xunrui Yin, Xiaoxi Zhang, Xu Du, and Zongpeng Li. On space information flow: Single multicast. In *2013 International Symposium on Network Coding (NetCod)*, pages 1–6. IEEE, 2013. 5.1.4

[84] Taisuke Izumi and Roger Wattenhofer. Time lower bounds for distributed distance oracles. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 60–75, 2014. 1, 6.1

[85] Kamal Jain, Vijay V Vazirani, Raymond Yeung, and Gideon Yuval. On the capacity of multiple unicast sessions in undirected graphs. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2805–2809, 2006. 5.1, 5.1.4

[86] Tomasz Jurdziński and Krzysztof Nowicki. MST in $O(1)$ rounds of congested clique. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2620–2632, 2018. 6.5

[87] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: Practical wireless network coding. In *ACM SIGCOMM computer communication review*, volume 36, pages 243–254. ACM, 2006. 5.1

[88] Ken-ichi Kawarabayashi and Paul Wollan. A simpler algorithm and shorter proof for the graph minor decomposition. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 451–458. ACM, 2011. 4.1, 4.1.2

[89] Maleq Khan and Gopal Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*, pages 355–369, 2006. 1, 6.1

[90] Naoki Kitamura, Hirotaka Kitagawa, Yota Otachi, and Taisuke Izumi. Low-congestion shortcut and graph parameters. In *Proceedings of the 33rd International Symposium on Distributed Computing (DISC)*, pages 25:1–25:17, 2019. 6.1, 6.5

[91] Philip Klein, Serge A Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993. 5.1.4

[92] Ronald Koch, Britta Peis, Martin Skutella, and Andreas Wiese. Real-time message routing and scheduling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 217–230. Springer, 2009. 5.1

[93] Gerhard Kramer and Serap A Savari. Edge-cut bounds on network coding rates. *Journal of Network and Systems Management*, 14(1):49, 2006. 5.1, 5.1.4

[94] Robert Krauthgamer, James R Lee, and Havana Rika. Flow-cut gaps and face covers in planar graphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 525–534, 2019. 5.1.4

[95] Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and

applications. In *Proceedings of the 14th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 238–251, 1995. 1.1.1, 2.1.1

[96] Shay Kutten and David Peleg. Fast distributed construction of smallk-dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998. 6.1, 6.2, 6.5

[97] Michael Langberg and Muriel Médard. On the multiple unicast network coding, conjecture. In *47th Annual Allerton Conference on Communication, Control, and Computing*, pages 222–227, 2009. 5.1, 5.1.4

[98] James R Lee and Anastasios Sidiropoulos. Genus and the geometry of the cut graph:[extended abstract]. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 193–201, 2010. 5.1.4, 5.2.3

[99] 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. 1.2.2, 5.1, 5.1.4, 5.2, 5.7, 5.7, 6.1.1, 6.2, 6.2.2, 6.2.5

[100] Tom Leighton, Bruce Maggs, and Andrea W Richa. Fast algorithms for finding o (congestion+ dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999. 5.1, 5.1.4, 7.6

[101] Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 153–162, 2015. 1, 6.1

[102] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 375–382, 2013. 1, 6.1

[103] Zongpeng Li and Baochun Li. Network coding: The case of multiple unicast sessions. In *Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, volume 16, page 8, 2004. 5.1, 5.1.4, 8.2.2

[104] Zongpeng Li and Baochun Li. Network coding in undirected networks. In *Conference on Information Systems and Sciences (CISS)*, 2004. 5.1.4

[105] Zongpeng Li, Baochun Li, and Lap Chi Lau. A constant bound on throughput improvement of multicast network coding in undirected networks. *IEEE Transactions on Information Theory*, 55(3):1016–1026, 2009. 5.1.4

[106] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. 5.1, 5.1.4

[107] Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in $O(\log \log n)$ communication rounds. *SIAM Journal on Computing (SICOMP)*, 35(1):120–131, 2005. 6.5

[108] Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for constant diameter graphs. *Distributed Computing*, 18(6):453–460, 2006. 6.5

[109] László Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43 (1):75–86, 2006. 4.1.3, 4.1.7, 4.1.3

[110] Shachar Lovett. Linear codes cannot approximate the network capacity within any constant factor. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 21, page 141, 2014. 5.1.3

[111] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146, 2010. 1, 6.1

[112] Gary L Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 196–203, 2013. 7.7

[113] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 565–573, 2014. 1, 6.1

[114] Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 439–453, 2014. 1, 6.1, 6.2

[115] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1):3–36, 2001. 1.2.2, 2.2.4

[116] Haruko Okamura and Paul D Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory, Series B*, 31(1):75–81, 1981. 5.1.4

[117] Rafail Ostrovsky and Yuval Rabani. Universal $O(congestion + dilation + \log^{1+\epsilon} n)$ local control packet switching algorithms. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, volume 29, pages 644–653, 1997. 5.1, 5.1.4

[118] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. A time- and message-optimal distributed algorithm for minimum spanning trees. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 743–756, 2017. 6.1, 6.5

[119] Britta Peis and Andreas Wiese. Universal packet routing with arbitrary bandwidths and transit times. In *Proceedings of the 13th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 362–375, 2011. 5.1, 5.1.4

[120] Britta Peis, Martin Skutella, and Andreas Wiese. Packet routing: Complexity and algorithms. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA)*, pages 217–228, 2009. 5.1

[121] David Peleg. *Distributed computing*, volume 5. 2000. 1, 1.2.1, 2.2.1, 1

[122] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM Journal on Computing (SICOMP)*, 30(5):1427–1442, May 2000. (document), 1, 1.1.1, 5.1, 5.1.3, 5.1.3, 5.2, 5.1.3, 6.1, 6.1.1, 6.3.1, 6.5

[123] Serge Plotkin and Éva Tardos. Improved bounds on the max-flow min-cut ratio for multicommodity flows. *Combinatorica*, 15(3):425–434, 1995. 5.1.2

[124] Yuval Rabani and Éva Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, volume 96, pages 366–375, 1996. 5.1, 5.1.4

[125] Harald Räcke. *Data Management and Routing in General Networks*. PhD thesis, University of Paderborn, 2003. 3, 7.1, 8.1

[126] Harald Räcke. Survey on oblivious routing strategies. In *5th*, pages 419–429. Springer, 2009. 3, 7.1

[127] Satish Rao. Small distortion and volume preserving embeddings for planar and euclidean metrics. In *Proceedings of the 15th Symposium on Computational geometry (SoCG)*, pages 300–306, 1999. 5.1.4

[128] Alexander A Razborov. On the distributional complexity of disjointness. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 249–253. Springer, 1990. 6.2.1, 6.6

[129] Neil Robertson and Paul D Seymour. Graph minors. v. excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986. 4.1, 4.1.2

[130] Neil Robertson and Paul D Seymour. Graph minors. xvi. excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43–76, 2003. 4.1, 4.1.2

[131] Thomas Rothvoß. A simpler proof for $O(Congestion + Dilation)$ packet routing. In *Proceedings of the 16th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 336–348, 2013. 5.1, 5.1.4

[132] Christian Scheideler. *Universal routing strategies for interconnection networks*, volume 1390. Springer, 2006. 5.1, 5.1.4

[133] Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. 6.3.3, 6.7.2

[134] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985. 6.5

[135] Aravind Srinivasan and Chung-Piaw Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing (SICOMP)*, 30(6):2051–2068, 2001. 5.1, 5.1.4, 5.2, 5.7, 5.7.1

[136] Gregory Valiant and Paul Valiant. Instance optimal learning of discrete distributions. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 142–155, 2016. 6.5

[137] Gregory Valiant and Paul Valiant. An automatic inequality prover and instance optimal identity testing. *SIAM Journal on Computing (SICOMP)*, 46(1):429–455, 2017. 6.5

[138] Chih-Chun Wang and Minghua Chen. Sending perishable information: Coding improves delay-constrained throughput even for single unicast. *IEEE Transactions on Information Theory*, 63(1):252–279, 2016. 5.1, 5.1.4, 5.6

[139] Yunnan Wu, Philip A Chou, and Sun-Yuan Kung. Minimum-energy multicast in mobile ad hoc networks using network coding. *IEEE Transactions on communications*, 53(11):

1906–1918, 2005. 5.1, 5.1.4

[140] Tang Xiahou, Zongpeng Li, Chuan Wu, and Jiaqing Huang. A geometric perspective to multiple-unicast network coding. *IEEE Transactions on Information Theory*, 60(5): 2884–2895, 2014. 5.1, 5.1.4

[141] Xunrui Yin, Zongpeng Li, Yaduo Liu, and Xin Wang. A reduction approach to the multiple-unicast conjecture in network coding. *IEEE Transactions on Information Theory*, 64(6):4530–4539, 2018. 5.1, 5.1.4

[142] Leonid Zosin and Samir Khuller. On directed steiner trees. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 59–63, 2002. 5.1.4