Compact Representations of Graphs and Their Metrics

D Ellis Hershkowitz

CMU-CS-22-134
August 2022

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

Thesis Committee:
Bernhard Haeupler (Co-Chair)
R. Ravi (Co-Chair)
Anupam Gupta
Michel Goemans (MIT)
Ola Svensson (EPFL)

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

This research was supported by NSF grants CCF-1527110, CCF-1618280, CCF-1814603, CCF-1910588, NSF CAREER award CCF1750808, a Sloan Research Fellowship, funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 949272), the Swiss National Foundation (project grant 200021-184735), the Office of Naval Research under award number N000141812099 and the Air Force Office of Scientific Research under award number FA9550-20-1-0080.

Keywords: Theoretical Computer Science, Metric Embeddings, Approximation Algorithms, Online Algorithms, Distributed Algorithms, Tree Embeddings, Hop-Constrained Flows, Steiner Point Removal.

For my grandparents.

Abstract

Graphs and metrics are two of the most ubiquitous, versatile and powerful tools in modern computing. Both are general enough to be widely applicable but structured enough to facilitate efficient algorithms. Furthermore, the modern proliferation of data has led to graphs and metrics of practical importance which are of unprecedented size. For this reason, now more than ever, understanding how to sparsify or otherwise effectively reduce the size of a graph or metric is of great importance. We give new results in graph and metric sparsification using tools from combinatorial optimization, metric embeddings, approximation algorithms and online algorithms.

In the first part of this thesis we provide two new so-called tree embeddings which represent relevant aspects of an arbitrary input graph by a tree. The first embeds the points in a metric into a single tree containing a small number of copies of each vertex. Using these embeddings we give the first non-trivial deterministic approximation algorithms for online group Steiner tree and the online covering Steiner tree problem. The second is the first embedding of the hop-constrained distances in a graph into a distribution over trees; the hop-constrained distance between two nodes is defined as the minimum weight of a connecting path consisting of at most some hop constraint many edges. Using these embeddings we give the first poly-log bicriteria algorithms for the hop-constrained version of many classic network design problems.

In the second part of this thesis we provide new algorithms for a primitive at the heart of recent advances in hop-constrained expander decompositions for graphs. Specifically, we give the first algorithms for $(1-\epsilon)$-approximate h-length flows running in sequential time $\tilde{O}(m \cdot \mathsf{poly}(h, \frac{1}{\epsilon}))$, parallel time $\tilde{O}(\mathsf{poly}(h, \frac{1}{\epsilon}))$ with m processors and distributed CONGEST time $\tilde{O}(2^{O(\sqrt{\log n})} \cdot \mathsf{poly}(h, \frac{1}{\epsilon}))$. Notably, these algorithms are also deterministic. We give a variety of applications including simplified distributed and deterministic constructions of expander decompositions, efficient algorithms for computing h-length cutmatches which form the backbone of recent work in hop-constrained expander decompositions and what is to our knowledge the first non-trivial distributed $(1-\epsilon)$-approximation for b-matching.

In the last part of this thesis we investigate how graph structure can make metric sparsification easier. In particular, we study the Steiner point removal problem where we must vertex-sparsify a graph from a structured family. We give the first $O(1)$ distortion Steiner point removal solutions on series-parallel graphs as well as new metric decompositions for series-parallel graphs.

Acknowledgments

Grad school has given me the opportunity to dabble in the cosmic alchemy that is theoretical computer science. I have spent countless hours squeezing arcane ideas into my head and what I have found on the other end—brain bursting at the seams—is great joy. I am now never at a loss for something fascinating to turn over; I am now never bored. This joy is something I hope to carry with me the rest of my life and it would surely not have been possible without the incredible support network that gave form to my 6 years at CMU.

Above all this would not be possible without my advisors, Bernhard and Ravi.

Bernhard, I came to CMU with little more in mathematical training than a faint notion of what a Chernoff bound is. Despite this you took me on as your student; eagerly. You navigated me along a steep learning curve, always unwavering in your support and always excited to generously share vivid digestions of otherwise impenetrable ideas. Your talent for mining simplicity from complexity invariably gave me a sense of "this is something that I too can do." I will fondly look back on many memories of you blasting into meetings with some fun new idea, scooter under foot. Beyond technical guidance—and perhaps more importantly—you fiercely advocated for balance in my grad school life. What is all too often some abstract institutional support for "grad student work-life balance" is something you took on as a personal mission of pressing concern. Notably, you led by example: I don't know of any other grad students who have stories of both doing acroyoga and floating down the Limmat with their advisors; maybe one but certainly not both.

Ravi, you buoyed me in the middle years of grad school when classes are done but research is still half-baked. I could count on coming out of meetings with you feeling better than I went in; always full of that addictive sense of electricity that we all chase in research. You would invariably meet me halfway, earnestly engaging with and building on the ideas I brought to our meetings, however inchoate. If I bring to my own future mentorship just a small fraction of the warmth, cheer, enthusiasm and snacks that you brought to our meetings then I will have succeeded as a mentor. Beyond our meetings, you have been an advisor in every sense of the word, not just sharing with me your expansive technical knowledge but seriously advocating for me professionally. I am ever thankful for the extent to which you have really taken your role as my advisor to heart.

I hope to continue to learn from both of you into the future but I also hope to maintain these friendships.

Thank you to Anupam—from whom I feel I learned a great deal despite a short collaboration—for agreeing to be on my committee, always having an open door and some really supportive last-minute job advice. Thank you to my entire committee for thoughtful comments and suggestions and especially to Ola and Michel for graciously agreeing to join along.

An additional thanks to my many mentors who preceded grad school: thank you especially to Stefanie Tellex for introducing me to research, Michael Littman for patiently introducing me to more theoretically-inclined research and Anna Lysyanskaya for encouraging me to pursue my interest in theory.

Thank you to all the admins who have deftly shielded me from so much rigmarole and help to make grad school really about the research; especially thank you to Deb for patiently answering every tiny question I have ever had and Charlotte for the many dinosaur stickers.

Thank you to all my collaborators: Keren Censor-Hillel, Nathan Klein, David Wajc, Arnold Filtser, Jason Li, Laxman Dhulipala, Thatchaphol Saranurak, Sahil Singla, Cam Allen, Yuu Jinnai, George Konidaris, Michael Littman, Hossein Bateni, Rajesh Jayaram, Kuba Lacki, Yiting Wang, Rico Zenklusen, Goran Zuzic, Bernhard Haeupler, R. Ravi, Christoph Grunau, Václav Rozhoň, Anson Kahng, Dominik Peters, Gregory Kehne, Ariel Procaccia and Dave Abel; I could not have done the research or had half as much fun doing it without all of you.

This has been an unusually isolating time to complete a PhD and I am beyond grateful to the many friends that populate my Rolodex. Greg, you've become a staple of my social life. I could not have made it through these past few years without our regular exchange of math chortles, your patient listening or your quick wit. Dave, I can't think of a friend who has so positively influenced how I approach my own life; I hope to continue to osmose that inimitable Dave-like wonder, excitement and thrill of discovery for many years to come. Thank you to Laxman for only almost always beating me at chess, unparallelled thoughtfulness and many meandering walks. Thank you to Anson for an ever-solid friendship, inspiringly understated intensity and genuinely being a lovely collaborator. Thank you to Alex for tolerating my rants, pro bono ombudswork and excellent hugs. Thank you to Roie for good jokes, refreshingly contrarian opinions and (one day I'm sure) a fun collaboration. Thank you to Other Greg for your big grin and your breadth of intellect that it's not fair for anybody as mathematically sharp as you to have. Thank you to Marina for a hail mary friendship of unusual depth and reminding me why I enjoy philosophy. Thank you to Niko for tolerating many Esplanade walks and life talks despite a broken back. Thank you to Will for being an unwaveringly supportive friend over nearly two decades. Thank you to Anthony for always going out of his way to put time into our friendship and willingness to talk. Thank you to Katherine for bringing art into my life and so eagerly putting together so many lovely yet formidable adventures. Thank you to Yaffe for being a bastion of calm takes and clarity of thought. Thank you to you all. I have spent hours and hours wandering through the woods with nothing but my phone. I have never once felt lonely.

Many thanks to the many other characters that have made up my time in grad school. Thank you to Goran for Croatian Wisdom and becoming that mythical friend-collaborator that we all hope for in grad school. Thank you to Erika for supporting me through an uncommonly stressful time in my life and just generally cultivating an unmatched aura of fun, thoughtfulness and self-improvement. Thank you to Anna and Ellen for helping to provide the activation energy that is required to get grad students to go backpacking. Thank you to Javier for teaching me the folly of powerlifting and the merits of yoga. Also thank you to Sam for literally never not having a smile on his face except when his talking about super serious research stuff. Thank you to Nathan Fulton for dealing with Linux idiosyncrasies on my machine that I have no right not knowing how to fix myself at this point. A special thank you to David and Naama for really invaluable advice in the job search and just generally being lovely people. Lastly, a thanks to the many other people who have made my

time in grad school what it is: Vijay, Nic, Jalani, Mark, Kevin, Pallavi, Mansur, Chaitanya, Piyumi and I'm sure many that I'm missing.

Thank you to my family who, owing to present pandemic, have made up an unusually large portion of my social life in grad school. Thank you to Cole, Sveta and Maryna for moving their lives cross-country and bringing so much more family into my life these past few months. Thank you to Luka for teaching me the value of an unshakable fixation on a narrow swath of human knowledge; in his case trucks; in my case this thesis. Thank you to Ivan for being the only other bald nuclear family member (for now). Thank you to Jeannie for reminding me that the river also has things to say. Above all, thank you to my parents for filling my life with the joy of learning new things, gourmand-worthy dinners, extensive plant knowledge and literally a lifetime of support.

Contents

1 Introduction **1**
1.1 Overview . . . 2
1.1.1 New Tree Embeddings (Part I) . . . 2
1.1.2 New Primitives for Graph Decompositions (Part II) . . . 4
1.1.3 Steiner Point Removal (Part III / Chapter 6) . . . 5
1.2 Notation and Conventions . . . 5

I New Tree Embeddings **9**

2 Tree Embedding Background **11**
2.0.1 Network Design and Group Steiner Problems . . . 12

3 Copy Tree Embeddings **15**
3.1 Introduction . . . 15
3.1.1 Our Contributions . . . 16
3.2 Copy Tree Embedding Constructions . . . 19
3.2.1 From Padded Hierarchical Decompositions to Copy Tree Embeddings . . . 20
3.2.2 Deterministically Constructing Padded Hierarchical Decompositions . . . 24
3.2.3 Construction 2: Merging FRT Support . . . 30
3.3 Online Covering Steiner . . . 31
3.3.1 Online Covering Steiner on a Tree . . . 32
3.3.2 Online Covering Steiner on General Graphs . . . 34
3.4 Deterministic Online Group Steiner Reductions . . . 36
3.4.1 Deterministic Online Group Steiner Tree . . . 36
3.4.2 Deterministic Online Group Steiner Forest . . . 38
3.5 Conclusion and Future Work . . . 39

4 Hop-Constrained Tree Embeddings **41**
4.1 Introduction . . . 41
4.1.1 Our Contributions . . . 42
4.2 Hop-Constrained Network Design Related Work . . . 44
4.3 Approximating Hop-Constrained Distances . . . 45
4.3.1 Hop-Constrained Distances Are Inapproximable by Metrics . . . 45
4.3.2 Distances Induced by Distributions Over Partial Metrics . . . 46
4.3.3 Approximating Hop-Constrained Distances with Partial Tree Metrics . . . 47

4.4 h-Hop Partial Tree Embeddings . . . 52
4.4.1 Defining h-Hop-Partial Tree Embeddings . . . 52
4.4.2 Projecting From The Graph to h-Hop Partial Tree Embeddings . . . 54
4.5 Applications of h-Hop Partial Tree Embeddings . . . 60
4.5.1 Oblivious Hop-Constrained Steiner Forest . . . 60
4.5.2 Hop-Constrained Group Steiner Tree . . . 63
4.5.3 Hop-Constrained k-Steiner Tree . . . 66
4.5.4 Hop-Constrained Oblivious Network Design . . . 67
4.6 h-Hop Copy Tree Embeddings . . . 70
4.7 Applications of h-Hop Copy Tree Embeddings . . . 71
4.7.1 Hop-Constrained Group Steiner Tree . . . 71
4.7.2 Online Hop-Constrained Group Steiner Tree . . . 71
4.7.3 Hop-Constrained Group Steiner Forest . . . 72
4.7.4 Online Hop-Constrained Group Steiner Forest . . . 73
4.8 Conclusion and Future Work . . . 74
4.9 Deferred Proofs of Section 4.3 . . . 74

II New Primitives for Graph Decompositions 77

5 Length-Constrained Flows 79
5.1 Introduction . . . 79
5.1.1 Our Contributions . . . 80
5.2 Chapter-Specific Notation and Conventions . . . 83
5.3 Length-Constrained Flows, Moving Cuts and Main Result . . . 84
5.4 Intuition and Overview of Approach . . . 86
5.4.1 Using Lightest Path Blockers for Multiplicative Weights . . . 86
5.4.2 Length-Weight Expanded DAG to Approximate h-Length Lightest Paths 88
5.4.3 Deterministic Integral Blocking Flows Paths via Flow Rounding . . . 89
5.4.4 Overview of Chapter . . . 89
5.5 Preliminaries . . . 90
5.5.1 Deterministic CONGEST Maximal and Maximum Independent Set . . . 90
5.5.2 Deterministic Low Diameter Decompositions . . . 90
5.5.3 Sparse Neighborhood Covers . . . 91
5.5.4 Cycle Covers . . . 92
5.6 Path Counts for h-Layer S-T DAGs . . . 93
5.7 Randomized Blocking Integral Flows in h-Layer DAGs . . . 94
5.8 Deterministic and Distributed Near Eulerian Partitions . . . 97
5.8.1 High-Girth Cycle Decompositions . . . 98
5.8.2 Efficient Algorithms for Computing Near Eulerian Partitions . . . 101
5.9 Deterministic Blocking Integral Flows in h-Layer DAGs . . . 102
5.9.1 Iterated Path Count Flows . . . 103
5.9.2 Deterministic Rounding of Flows in h-Layer DAGs . . . 106
5.9.3 Deterministic Blocking Integral Flows . . . 111
5.10 h-Length $(1+\epsilon)$-Lightest Path Blockers . . . 112
5.10.1 Length-Weight Expanded DAG . . . 113

5.10.2 Decongesting Flows . . . 116
5.10.3 Computing h-Length $(1+\epsilon)$-Lightest Path Blockers . . . 117
5.11 Computing Length-Constrained Flows and Moving Cuts . . . 119
5.12 Application: Maximal and Maximum Disjoint Paths . . . 123
5.12.1 Maximal and Maximum Disjoint Path Variants . . . 123
5.12.2 Reducing Among Variants . . . 124
5.12.3 Maximal Disjoint Path Algorithms . . . 126
5.12.4 Maximum Disjoint Path Algorithms . . . 127
5.12.5 On the Hardness of Maximum Disjoint Paths . . . 127
5.13 Application: Simple Distributed Expander Decompositions . . . 128
5.14 Application: $(1-\epsilon)$-Approximate Distributed Bipartite b-Matching . . . 129
5.15 Application: Length-Constrained Cutmatches . . . 130
5.16 Conclusion and Future Work . . . 131
5.17 Generalizing Our Results to Multi-Commodity . . . 132
5.17.1 Multi-Commodity Flows, Cutmatches and Results . . . 132
5.17.2 Computing Multi-Commodity Length-Constrained Flows and Moving Cuts 134
5.17.3 Computing Multi-Commodity Length-Constrained Cutmatches . . . 138

III Steiner Point Removal **141**

6 Series-Parallel Steiner Point Removal **143**
6.1 Introduction . . . 143
6.1.1 Our Contributions . . . 144
6.2 Related Work . . . 146
6.3 Preliminaries . . . 147
6.3.1 Characterizations of Series-Parallel Graphs . . . 147
6.3.2 Scattering Partitions . . . 148
6.4 Intuition and Overview of Techniques . . . 149
6.4.1 General Approach . . . 149
6.4.2 Scattering Chops . . . 150
6.4.3 Hammock Decompositions and How to Use Them . . . 151
6.5 Chapter-Specific Notation and Conventions . . . 152
6.6 Perturbing KPR and Scattering Chops . . . 153
6.6.1 Perturbing KPR . . . 153
6.6.2 Scattering Chops . . . 156
6.7 Hammock Decompositions . . . 157
6.7.1 Trees of Hammocks . . . 157
6.7.2 Hammock Decompositions . . . 162
6.8 Hammock Decompositions for Series-Parallel Graphs . . . 163
6.8.1 Initial Hammocks $\hat{\mathcal{H}}$ by Connecting Equivalence Classes . . . 165
6.8.2 Extending $\hat{\mathcal{H}}$ to $\bar{\mathcal{H}}$ by Hammock-Joining Paths . . . 170
6.8.3 Extending $\bar{\mathcal{H}}$ to $\tilde{\mathcal{H}}$ by LCA Paths . . . 175
6.8.4 Extending $\tilde{H}$ to $\mathcal{H}$ by Adding Dangling Subtrees . . . 180
6.9 Scattering Chops via Hammock Decompositions . . . 182
6.10 Future Work . . . 186

6.11 Ear Decompositions from Hammock Decompositions 186
6.12 Hammock Decomposition Construction Figures 188

IV Conclusion **191**

7 Conclusion **193**

Bibliography **195**

List of Figures

3.1 Illustration of our first construction where we merge $O(\log n)$ partial tree embeddings. 17

3.2 Illustration of our second construction where we merge the $O(n \log n)$ trees in the FRT support. 17

3.3 Illustration of a hierarchical decomposition $\mathcal{H}$ with $h = 4$ with $n = 7$. Each part in each $\mathcal{P}_i \in \mathcal{H}$ is colored according to i; singleton parts not pictured. We give α-padded nodes in green and all other nodes in red where we illustrate why the node on the far left is α-padded and the node on the far right is not by drawing $B(v, \alpha \cdot 2^i)$ for $i \geq 1$ in colors according to i for these two nodes. 20

3.4 How to turn a hierarchical decomposition into a partial tree embedding. We color nodes from the input metric in green if they are padded and red otherwise. Remaining nodes colored according to their corresponding hierarchical decomposition part. r is the node on the far left of the tree. 22

3.5 Solution our algorithm gives after one group of groups, $\mathcal{G}_1$, is revealed where $r_1 = 2$. Nodes in groups in $\mathcal{G}_1$ outlined in green and nodes colored according to the group of $\mathcal{G}_1$ which contains them. Saturated edges given in blue and edges with $0 < x_e < w_e$ annoted with "x_e/w_e". All other edges labeled by w_e. 33

4.1 An illustration of the top-level recursive call of the embedding of Theorem 37 on graph G (edges omitted from illustration). Vertices in the partial vertex partition of Theorem 40 given in purple. Vertices removed from the process given as empty circles and all other vertices given as filled-in circles. 49

4.2 A counter-example to the naive charging argument for $T(H, h)$ where $\Theta(k) = \Theta(n)$. Edges labeled with their weights, vertices of $V(T)$ given as solid black circles and vertices of $V \setminus V(T)$ given as white-filled circles. Paths colored according to their corresponding pair. 56

4.3 An illustration of an h-hop connector with congestion 1 and hop stretch 2 on a graph G for a vertex set $W \subseteq V(G)$ with $h = 3$. Vertices of W given as solid black circles; all other vertices of G given as white circles. Edges in $(W, \mathcal{P})$ and paths in $\mathcal{P}$ colored according to their correspondence. 57

4.4 An illustration of how to compute an h-hop connector on an arbitrary graph G for $W \subseteq V(G)$ with $h = 3$. Vertices of W given as solid black circles; roots of F given as black squares; all other vertices of G given as white circles. Paths in $\mathcal{P}_1$ and $\mathcal{P}_2$ colored to correspond to their edges in $(W, \mathcal{P})$. 58

5.1 An illustration of the first two iterations of our multiplicative-weights-type algorithm where $h = 5$, $S = \{s\}$ and $T = \{t\}$ and capacities are all 1. Each arc is labelled with the value we multiply its initial weight by (initialized to $w_0 := 1+\epsilon$) then length then flow. Our h-length shortest path blockers are in blue. 87
5.2 A digraph D with $S = \{s\}$ and $T = \{t\}$ where the 5-length lightest S-T paths do not induce a DAG. 5.2a gives D where each arc is labeled with its weight (in black) and length (in green). 5.2b shows how all lightest S-T paths have weight 2 and induce a DAG. 5.2c shows how the two 5-length lightest S-T paths (in blue and red) have weight 6 and induce a digraph with a cycle. 88
5.3 An illustration of a near Eulerian partition $\mathcal{H}$ and H^+ for each $H \in \mathcal{H}$. 5.3a gives $\mathcal{H}$ which consists of one cycle and two paths. 5.3b gives the orientation of $\mathcal{H}$ where the source of each path is in blue. 5.3c gives H^+ (in green) and $H \setminus H^+$ (in red) for each $H \in \mathcal{H}$. 107
5.4 An example of our flow rounding algorithm on digraph D with unit capacities. 5.4a gives the input flow where arcs are labelled with their flow and vertices are labelled with their deficit. 5.4b gives $D^{(2)}$, the graph induced by all arcs with flow value .5. 5.4c gives our oriented near Eulerian partition of $D^{(2)}$ (in blue). 5.4d shows how we update our flow based on the near Eulerian partition. 5.4e gives the result of this flow update; notice that some vertices not in S and T have non-zero deficit. 5.4f gives the S-T subflow we return where only vertices in S and T have non-zero deficit. 110
5.5 An illustration of how we round weights according to ϵ, λ and h. Here $h = 5$, $\lambda = 6$ and $\epsilon = .5$ and so we round to multiples of $\frac{\epsilon}{h}\lambda = \frac{3}{5}$. 5.5a gives our input DAG where each arc is labeled with its weight, then length then capacity and 5.5b gives the weights after we round them where we color each lightest 5-length path from s to t. 113
5.6 An illustration of $D^{(h,\lambda)}$ where D and the parameters we use are given by Figure 5.5, $\kappa = 100$, $S = \{s\}$ and $T = \{t\}$. Copy $v(x, h')$ of vertex v is in the (x, h')th grid cell and each arc is labelled with its capacity. We only illustrate the subgraph between $s(0, 0)$ and $t(\frac{33}{5}, 5)$. Each path is colored according to the path in Figure 5.5b of which it is a copy. Notice that the graph induced by all 5-length lightest paths in Figure 5.5b is not a DAG but $D^{(h,\lambda)}$ is. 114
5.7 Illustration of our reduction on a single edge or arc between u and v for reducing maximal or maximum vertex-disjoint paths, edge-disjoint paths or vertex-disjoint directed paths to arc-disjoint directed paths. 124

6.1 A summary of the SPR distortion for (connected) K_h-minor-free graphs achieved in prior work and our own. Graph classes illustrated according to containment. We also give the forbidden minors for each graph family. 144
6.2 In (a) we illustrate a clawed cycle where the cycle C is given in solid black and each path is given in dotted black. In (b) we illustrate a scattering partition with $\tau = 3$ and how one path P of length at most Δ is incident to at most three parts where we color the subpaths of P according to the incident part. 147
6.3 Two levels of Δ-chops on the grid graph for $\Delta = 3$. We give the edges of the BFS trees we use in pink; roots of these trees are given as squares. Background colors give the annuli of nodes. 150

6.4 An example (of an outerplanar graph) where a Δ-chop does not produce a scattering partition but how perturbing said chop does. Here, we imagine that the root is at the top of the graph and each edge incident to the root has length $\Delta - 3$. We highlight the path P that either ends up in many or one connected component depending on whether we perturb our Δ-chop in yellow. 150
6.5 A c-fuzzy Δ-chop that is 1-scattering. We draw each fuzzy annulus in a distinct color. In (b) we visualize some shortest paths of length at most Δ and highlight cut edges in red. 156
6.6 An illustration of a hammock and a hammock-fundamental cycle for $e_1, e_2 \in E_c$. Edges of T_{BFS} in pink, cross edges in black, hammock roots are a black square and diamond and the hammock-fundamental cycle is in yellow. 158
6.7 An illustration of one tree of hammocks in a hammock decomposition $\{H_i\}_i$ and a hammock decomposition consisting of two trees of hammocks. Each r_i and r'_i given as a square and diamond colored according to corresponding H_i. Edges in T_{BFS} in pink and highlighted according to the H_i which contains them. Edges of E_c colored according to the H_i which contains them. T_0 in the hammock decomposition given in dark red. 159
6.8 An illustration of the LCA-equivalence classes of a series-parallel graph G. Edges of T_{BFS} in pink. Edges of E_c in black in Figure 6.8a and colored according to their LCA-equivalence class in Figure 6.8b. Notice that edges with the same LCA can belong to distinct equivalence classes. 163
6.9 The three cases of the proof of Theorem 163. Edges in C solid, edges outside of C transparent. T_{BFS} in pink and E_c in black. In (a) we give the path between v_l and v_r contained in $T_{\text{BFS}}(x) \setminus \{x\}$ with a dotted yellow path. In (b) we illustrate give $l(e_1) = l(e_2)$ at the top. In (c) we highlight the cycle and the paths of the clawed cycle in solid and dotted yellow respectively. 167
6.10 An illustration of our initial hammocks $\hat{\mathcal{H}} = \{\hat{H}_i\}_i$ and how we extend them to our final hammocks $\bar{\mathcal{H}}$. Roots and edges of initial hammocks colored according to i. Notice that one vertex is the root of two hammocks (and so colored with two colors). 168
6.11 The contradiction in the proof of Theorem 167. In (a) we illustrate $\bar{H}_i$ in light blue and H_j in dark blue. In (b) we highlight F_j in solid yellow and P_1, P_2 and P_3 in dotted yellow. 169
6.12 An illustration of how in the proof of Theorem 173 we may assume that $|C \cap C_i| \leq 1$ by shortcutting C in $\bar{H}_i$. We highlight C in yellow both before and after shortcutting. 172
6.13 Each of the three contradictions we arrive at in the proof of Theorem 182 based on the value of γ. In (c) we highlight the cycle of our clawed cycle in solid yellow and each of its paths in dotted yellow. 178
6.14 An illustration of the paths of our hammock decomposition. Each such path dotted and given in a color corresponding to its constituent hammock. On the left we give the paths and on the right we give the result of adding these paths to $\bar{\mathcal{H}}$, resulting in $\tilde{\mathcal{H}}$. Notice that each root hammock has two paths in $\mathcal{P}$. 179
6.15 The result of adding the dangling trees to each of our hammocks and the final hammock decomposition. On the right we give T_0 in dark red. 181

6.16 Construction of a c-fuzzy, $O(1)$-scattering, Δ-chop. There are two hammocks in the picture, H_j and H_k. 183
6.17 Setting for the proof of Theorem 189. The path P is the red, dotted path in the left. The green cycle and the blue claw on the right together form a clawed cycle. 184
6.18 An illustration of the construction of our hammock decomposition on a series-parallel graph. 189

List of Tables

4.1 Our bicriteria approximation results. All results are for poly-time algorithms that succeed with high probability (at least $1 - \frac{1}{\mathsf{poly}(n)}$). For some of the problems we assume certain parameters are $\mathsf{poly}(n)$ to simplify presentation; see the relevant sections for more details. All results are new except for the k-Steiner tree result which is implied by [126]. 44

Chapter 1

Introduction

Graphs and metrics are two of the most versatile tools in modern computing. Both are general enough to find wide-reaching applications but sufficiently structured enough to form the foundation of many algorithms. Indeed the rich and beautiful structure of graphs and metrics has driven algorithmic advances in areas as varied as computational biology, distributed computing, machine learning and chip design. In large part these applications are made possible by the ability of graphs and metrics to model costs, congestion and distances in a variety of networks—including protein, transportation, manufacturing, computer, social and epidemiological networks. More generally, their ability to mathematically formalize a notion of "connectedness" and "closeness" lays the foundation for many applications such as planning and knowledge representation in AI.

While graphs and metrics form the foundation of much of modern computing, a proliferation of data has led to modern graphs and metrics that dwarf their predecessors. For instance, the social and knowledge graphs of companies like Google and Facebook have billions of vertices and hundreds of billions of edges [63]. Similarly, a flurry of work in molecular biology has led to the discovery of genomic and protein networks of interest whose interactions are described by singularly massive graphs [53]. Thus, modern algorithms for graphs and metrics are faced with the daunting task of computing over huge and complex inputs.

One of the most powerful and flexible tools for meeting the algorithmic challenges posed by massive graphs and metrics is sparsification and compression. Here, otherwise intractably large graphs and metrics are sparsified or otherwise made concise while preserving salient properties of the input. As exact compressions of a graph or metric is often impossible, these techniques often only approximately preserve certain properties of the graph. Applying these strategies can make otherwise intractably large problems significantly smaller, thereby opening the door to efficient algorithms.

> *In this thesis we give new results in graph sparsification and compression. Specifically, we give two new types of so-called tree embeddings (copy tree embeddings and hop-constrained tree embeddings), new algorithms for hop-constrained flows as well as constant distortion solutions for Steiner point removal in series-parallel graphs. We will use tools and perspectives from combinatorial optimization, metric embeddings, approximation algorithms and online algorithms.*

1.1 Overview

We provide a brief overview of the content of this thesis below. In the first part of this thesis we provide new ways to embed a graph into a tree with applications in online and approximation algorithms. In the second part we provide new algorithms for efficiently computing "hop-constrained flows", use these algorithms to efficiently decompose a graph into parts which admit efficient communication. Lastly, in the third part we provide new vertex-sparsification results by way of new results for the Steiner point removal problem.

1.1.1 New Tree Embeddings (Part I)

Trees are among the simplest graphs. Indeed, by way of classical ideas—such as dynamic programming—their simple structure makes many otherwise intractable problems efficiently solvable. For this reason a great deal of work has focused on how to approximate arbitrary graphs and their metrics by trees by way of so-called "tree embeddings." In the first part of this thesis we study new tree embedding paradigms as well as tree embeddings that incorporate the "hop structure" of a graph.

Copy Tree Embeddings (Chapter 3)

As an arbitrary metric is provably inapproximable by a single tree [7], conventional tree embeddings have focused on how to approximate a graph by a distribution over trees [23]. However, the probabilistic nature of these embeddings makes these embeddings unsuitable for many well-studied settings—such as online settings with adaptive adversaries.

In the first chapter of this thesis we provide new tree embeddings of metrics which *deterministically* approximate an arbitrary metric by a tree. Our key insight is that a metric can be approximated by a single tree if we allow our tree to contain (boundedly-many) copies of each vertex. We term these embeddings copy tree embeddings. We give constructions of copy tree embeddings that $O(\log^2 n)$ approximate the cost of subgraphs while embedding each vertex into only $O(\log n)$ copies. Specifically, subgraphs of the input graph will map to subgraphs in the tree at a multiplicative cost increase of $O(\log^2 n)$ while guaranteeing that if two vertices are connected in the original subgraph then two of their copies will be connected in the tree.

The key insight to this construction is the observation that if we are only interested in embedding a large subset of vertices into a tree then there do, in fact, exist trees which preserve distances between these subsets; by combining many such trees we are able to achieve the above result.

We demonstrate the algorithmic utility by using them to give the first non-trivial deterministic approximation algorithms for online group Steiner tree which is a generalization of the well-studied set cover and Steiner tree problems. Similarly, we use our copy tree embeddings to give the first deterministic (bicriteria) approximation algorithms for the covering Steiner problem, itself a generalization of the group Steiner problem. Many of these results appear in a joint work with Hauepler and Zuzic [112].

Hop-Constrained Tree Embeddings (Chapter 4)

In the next chapter, we proceed to study how the the interactions between graph structure and their metric affects tree embeddings.

Specifically, while every graph induces a metric, a graph can provide a metric with rich structure that a metric alone does not exhibit. While the metric we typically associate with a weighted graph is the shortest path metric—where the distance between two vertices is the minimum weight of a path connecting them—a weighted graph really induces two metrics. The first is the usual shortest path metric but the second is the hop-distance metric where the distance between two nodes is the minimum number of edges in a path connecting these nodes. These two metrics together define a notion of hop-constrained distances where the distance between two nodes is defined as the shortest path with at most some specified number of edges. A metric without the graph that induces it provides no such notion of hop-constrained distances.

Just as it is useful to embed a metric into simple structures such as trees, so too is it useful to embed hop-constrained distances into trees. Indeed, incorporating graph structure into a solution can be of great practical value. Consider, for example, the case of hop-constrained minimum spanning tree (MST). MST has been extensively used as a subroutine to compute low-cost (computer) networks in which every node is able to communicate with every other node. A hop-constrained MST naturally lead to networks with lower latencies as messages between nodes must travel fewer links in the network [158, 167]. Additionally, if we imagine that a transmission over an edge fails with some probability—as has been observed to occur in practice—then by minimizing the number of edges messages must travel we can reduce the probability that a communication fails and achieve more reliable networks [158, 167].

However, the complexity inherent in hop-constrained distances makes tree-like summarizations significantly more challenging than the metric case. Indeed, not only are hop-constrained distances not metric (as they fail to satisfy the triangle inequality) but, as we observe in this chapter, the path graph demonstrates that they are, in some sense, inapproximable by metrics. This fact not only rules out tree embeddings in the conventional sense, but it rules out embedding hop-constrained distances into any distribution of metrics.

Despite these apparent roadblocks, we demonstrate how to concisely represent hop-constrained distances by the metric of a weighted tree. The crucial insight we make is that the above impossibility results only hold if we our goal is to embed the hop-constrained distances between *all nodes* into a metric. By only embedding a (large) constant fraction of the nodes in the input graph we are able to show how to $O(\log^2 n)$ approximate the hop-constrained distances between all nodes with a bicriteria $O(\log^3 n)$ relaxation in the hop constraint. Likewise, we combine these embeddings with the above notion of copy tree embeddings to give "hop-constrained copy tree embeddings."

Additionally, we critically rely on what we call the "mixture metric" which is a notion of distance that encodes information about both hops and shortest path distances. In particular, it is a convex combination of hop-distances (where the distance between two nodes is defined as the minimum number of edges in a path that connects them) and the standard shortest path metric; as this notion of distance indeed induces a metric we can make use of tools from metric embeddings and, in particular, so-called low diameter decompositions to extract structure on this mixture metric. By using a series of different convex combinations we will be able to encode different parts of the hop-constrained shortest path distances, again, using metric embeddings to extract structure on each of these parts.

We proceed to study several hop-constrained network design problems. Specifically, in classic network design problems our goal is to find a minimum cost subgraph connecting certain nodes. In

hop-constrained network design problems our goal is to find a minimum cost subgraph connecting certain nodes *along paths that consist of boundedly-many edges*. This additional constraint is desirable for the above-stated reasons regarding MSTs.

Unfortunately, hop-constraints make network design problems significantly more challenging. For example, while Steiner forest admits a constant approximation, Steiner forest with hop-constraints is known to admit no $o(2^{\log^{1-\epsilon} n})$-approximation for any $\epsilon > 0$ [68].

Nevertheless, using our embeddings of hop-constrained distances into trees and by relaxing our algorithms to be bicriteria, we overcome these impossibility results and give the first poly-time (poly-log, poly-log) bicriteria approximations for the hop-constrained versions of many classic network design problems such as Steiner forest, group Steiner tree and group Steiner forest as well as the online and oblivious versions of these problems. Much of this chapter is based on a joint work with Haeupler and Zuzic [113].

1.1.2 New Primitives for Graph Decompositions (Part II)

Computing routing schemes that support both high throughput and low latency is one of the core challenges of network optimization. Such routes can be formalized as h-length flows which are defined as flows whose flow paths are restricted to have length at most h. Many well-studied algorithmic primitives—such as maximal and maximum length-constrained disjoint paths—are special cases of h-length flows.

These algorithmic primitives form the backbone of many recent algorithmic advances in so-called expander decompositions of graphs which, informally speaking, partition a graph into its highly expanding parts while only cutting a small fraction of edges. Likewise the optimal h-length flow is a fundamental quantity in network optimization, characterizing, up to poly-log factors, how quickly a network can accomplish numerous distributed primitives [109].

Fast Algorithms for Length-Constrained Flows (Chapter 5)

In this chapter, we give the first efficient algorithms for computing $(1-\epsilon)$-approximate h-length flows in several models of computation. We give deterministic algorithms that take $\tilde{O}(\text{poly}(h, \frac{1}{\epsilon}))$ parallel time and $\tilde{O}(\text{poly}(h, \frac{1}{\epsilon}) \cdot 2^{O(\sqrt{\log n})})$ distributed CONGEST time. We also give a CONGEST algorithm that succeeds with high probability and only takes $\tilde{O}(\text{poly}(h, \frac{1}{\epsilon}))$ time.

The basic approach we take will be an instantiation of the multiplicative weights framework. In particular, we will gradually add to our flow by finding a large batch of edge-disjoint h-length paths along which to send a small amount of new flow. We will choose these paths by maintaining a dual solution which we also gradually update. This dual solution corresponds to an edge weighting and the paths that we send flow along will correspond to a collection of (near)-lightest h-length paths which together "cover" all (near)-lightest h-length paths. Each time we send flow along such a batch of paths we update our dual solution by a multiplicative $(1+\epsilon)$ on all edges incident to one of these paths; thus, each iteration of this process is guaranteed to substantially increase the weight of the lightest h-length path between our sources and sinks and so overall we will only require $\tilde{O}(1)$-many such iterations.

The main challenge in instantiating this idea then becomes efficiently finding such a collection of paths and for this we develop some new approximate representations of (near)-lightest h-length

paths in arbitrary digraphs by DAGs as well as new deterministic flow rounding techniques in a distributed setting by making use of so-called "cycle covers."

Using our h-length flow algorithms, we give the first efficient deterministic CONGEST algorithms for the maximal length-constrained disjoint paths problem—settling an open question of Chang and Saranurak [45]—as well as essentially-optimal parallel and distributed approximation algorithms for maximum length-constrained disjoint paths. The former greatly simplifies deterministic CONGEST algorithms for computing expander decompositions. We also use our techniques to give the first efficient $(1-\epsilon)$-approximation algorithms for bipartite b-matching in CONGEST. Lastly, using our flow algorithms, we give the first algorithms to efficiently compute h-length cut-matches, an object at the heart of recent advances in length-constrained expander decompositions. This chapter is based on a joint work with Saranurak and Haeupler [110].

1.1.3 Steiner Point Removal (Part III / Chapter 6)

Just as a graph imbues its metric with additional structure by way of its hop structure, so too can a graph give additional structure to its metric by belonging to a structured family of graphs. In particular, the set of all metrics induced by a minor-closed family of graphs is a strict subset of all metrics: for example, not every metric can be induced by a planar graph.

In the last chapter of this thesis, we investigate how the structure of a graph can enable improved compact representations of metrics. Specifically, we focus on the Steiner point removal (SPR) problem wherein we are given a weighted graph and a collection of terminals and must compute a weighted minor just on the terminals which approximates the input metric. It is conjectured that if the input graph belongs to a (non-trivial) minor-closed family then such an embedding is possible with multiplicative distortion $O(1)$.

We show that series-parallel graphs—i.e. all K_4-minor-free graphs admit $O(1)$ distortion SPR solutions, extending the frontier of what minor-closed families are known to admit $O(1)$ distortion SPR solutions. In particular, we give a general reduction of $O(1)$ distortion SPR solutions in K_r-minor-free graphs to what we call $O(1)$-scattering chops, which are, roughly, a separation of the graph into annuli that respect the shortest path structure of the graph. Next, we give new metric decompositions for series-parallel graphs which we call hammock decompositions. Roughly, hammock decompositions decompose a series-parallel graph into a tree-like subgraph which respect the shortest path structure of the input graph. We will argue that such a structure exists because when it fails to exist we can find a K_4-minor in our input graph. Lastly, we use our hammock decompositions to show that series-parallel graphs admit $O(1)$-scattering chops. These results appear in a joint work with Li [117].

1.2 Notation and Conventions

General: We let $[k] := \{1, 2, \ldots, k\}$ for any non-negative integer k. We let $A \sqcup B$ denote 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.

Graphs: Given a graph $G = (V, E)$ we denote its vertex set by $V(G)$ and $E(G)$, or simply V and E if G is clear from context. We let $n := |V|$. Most commonly, we consider undirected weighted graphs $G = (V, E, w)$ with weights $w : E \to \{1, 2, \ldots, L\}$. The value L is called the aspect ratio

and throughout this thesis we assume $L = \mathsf{poly}(n)$. We will let w_G be G's weight function if G is not clear from context.

Subgraphs: Given a weighted graph $G = (V(G), E(G), w_G)$, we will often consider a subgraph $H = (V(H), E(H))$ where $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We will often identify a subset of edges $E' \subseteq E(G)$ of a graph G with the subgraph induced by these edges, i.e., the subgraph H with $E(H) = E'$ and $V(H) = \bigcup_{e \in E(H)} e$. We define the weight of a subgraph $w_G(H) := \sum_{e \in E(H)} w_G(e)$ as the sum of weights of its edges. Given graphs G and H, we will use the notation $H \subseteq G$ to indicate that H is a subgraph of G. The weak diameter of a subgraph H is $\max_{u,v \in V(H)} d_G(u, v)$.

Induced Graphs and Edges: Given an edge set E and disjoint vertex sets V_1 and V_2, we let $E(V_1, V_2) := \{e = \{v_1, v_2\} \in E : v_1 \in V_1, v_2 \in V_2\}$ be all edges between V_1 and V_2. Given graph $G = (V, E)$ and a vertex set $U \subseteq V$, we let $G[U] = (U, E_U)$ be the "induced subgraph" of G where $\{u', v'\} \in E_U$ iff $\{u', v'\} \in E$. Given a collection of subgraphs $\mathcal{H} = \{H_i\}_i$ of a graph we call $G[\mathcal{H}] := (\bigcup_i V(H_i), \bigcup_i E(H_i))$ the induced subgraph of $\mathcal{H}$. Similarly, we will let $E(\mathcal{H}) := \bigcup_i E(H_i)$ give the edges of $\mathcal{H}$. We emphasize that it is not necessarily the case that $G[\mathcal{H}] = G[V(\mathcal{H})]$.

Well-Separated Trees: We will often work with well-separated rooted trees. We say that a weighted rooted tree $T = (V, E, w)$ with root $r \in V$ is well-separated if every root-to-leaf path has weights that are decreasing powers of 2. That is, if e' is a child edge of e in T then $w(e') = \frac{1}{2} w(e)$.

Distances and Metrics: For a set V we call any positive real function $d : V \times V \to R_{\geq 0}$ which is symmetric, i.e., satisfies $d(u, v) = d(v, u)$ for all $u, v \in V$, and satisfies the identity of indiscernibles, i.e., $d(u, v) = 0 \Leftrightarrow u = v$, a distance function (such a function is also often called a semimetric). If d also satisfies the triangle inequality $d(u, w) \leq d(u, v) + d(v, u)$ for all $u, v, w \in V$ then d is called a metric. We also extend the definition of d to sets in the standard way: $d(U, U') := \min_{u \in U, u' \in U'} d(u, u')$. We will talk about the diameter of a metric (V, d) which is $\max_{u,v \in V} d(u, v)$. We use $B(v, x) := \{u \in V : d(v, u) \leq x\}$ to stand for the closed ball of v of radius x in metric (V, d) and $B_G(v, x)$ if (V, d) is the shortest path metric of G and we need to disambiguate which graph we are taking balls with respect to. We will sometimes identify a graph with the metric which it induces.

Paths, Path Length, and Hop Length: A sequence $P = (v_0, v_1, \ldots, v_\ell)$ of nodes in a graph G is called a path if for all $i \in [\ell]$ we have $\{v_{i-1}, v_i\} \in E(G)$ and we say $E(P) := \bigcup_i \{\{p_{i-1}, p_i\}\} \subseteq E(G)$ is the edge set of P. Given a path $P = (v_0, v_1, \ldots, v_k, v_{k+1})$ we will use $\text{internal}(P) := \{v_1, \ldots, v_k\}$ to refer to the internal vertices of P. We will say that a path P is between two vertex sets U and W if its first and last vertices are in U and W respectively and $\text{internal}(P) \cap U = \emptyset$ and $\text{internal}(P) \cap W = \emptyset$. We will sometimes abuse notation and use P and $E(P)$ interchangeably. We will also sometimes say such a path is "from" U to W interchangeably with a path is "between" U and W. We will use $P \oplus P'$ to refer to the concatenation of two paths which share an endpoint throughout this thesis. If the nodes in P are distinct we say that P is simple. In this thesis paths are not assumed to be simple. We denote the number of hops in P with $\text{hop}(P) := \ell$ and call $\text{hop}(P)$ the hop length of P. If $G = (V, E, w)$ is weighted, we define the weight of a path P in G to be the sum of weights of its edges: $w(P) := \sum_{e \in E(P)} w(e)$.

Hop Distance and Hop Diameter: For a (non-complete) subgraph $H = (V(H), E(H))$ of a (complete) graph G we let $\text{hop}_H(u, v)$ be the minimum number of edges of a path between

u and v in H (i.e., using only the edges $E(H)$). We also define the hop diameter of H as $\mathrm{hop}(H) := \max_{u,v \in V(H)} \mathrm{hop}_H(u, v)$.

Shortest-Path Metric and Tree Metric: For a weighted graph G the distance between any two nodes $u, v \in V$ is defined as $d_G(u, v) := \min\{w(P) \mid \text{path } P \text{ between } u, v\}$. It is easy to verify that d_G is a metric on V and for this reason d_G is called the shortest path metric of G. Any metric d on a set V which is identical to a shortest path metric of a weighted tree $T = (V, E, w)$ is called a tree metric; for this reason we will sometimes conflate a tree metric with its corresponding tree.

Part I

New Tree Embeddings

Chapter 2

Tree Embedding Background

Embeddings of general metrics into trees is one of the most versatile tools in combinatorial and network optimization. The following definition gives the standard formalism for these embeddings.

Definition 1 (Probabilistic Tree Embedding)**.** *Given graph $G = (V, E)$ with edge weights $w_G : E \to \mathbb{R}_{\geq 0}$, a probabilistic tree embedding with distortion α is a distribution over edge-weighted trees $\mathcal{T}$ where each $T \in \mathcal{T}$ is on V, has its own weight function w_T, edge set E_T and furthermore:*

1. ***Non-Contracting:*** *$d_G(u, v) \leq d_T(u, v)$ for any $u, v \in V$ and any $T \in \mathcal{T}$*
2. ***Low Expected Distance Increase:*** *$\mathsf{E}_{T \sim \mathcal{T}}[d_T(u, v)] \leq \alpha \cdot d_G(u, v)$ for any $u, v \in V$.*

Observe that a given edge $e = \{u, v\}$ in a given tree $T \in \mathcal{T}$ corresponds to a path P_e of only less weight in G by the above non-contracting property. Thus, any subgraph H_T of $T \in \mathcal{T}$ can be "projected" to a subgraph $H_G := \bigcup_{e \in H_T} P_e$ of G where $w_G(H_G) \leq w_H(H_T)$ and where if u and v are connected in H_T then they will also be connected in H_G.

The beauty and utility of these tree embeddings comes from the fact that their application is often simple, yet extremely powerful. Indeed, when modeling a network with length, costs, or capacities as a weighted graph, these embeddings often allow one to pretend that the graph is a tree. A common template for countless network design algorithms is to (1) embed the input weighted graph G into a tree that approximately preserves the weight structure of G by sampling according to the above distribution; (2) solve the input problem on T to get subgraph H_T and; (3) return as the solution for the input problem the projection of H_T back into G as described above.

By appealing to the low expexted distance increase, a simple proof shows that this template provides a generic way of turning a β-approximation for many network design problems on a tree into an $\alpha \cdot \beta$-approximation on general graphs where α is the distortion of the probabilistic tree embedding. Thus, the natural question then becomes what sort of distortions are possible for probabilistic tree embeddings.

A long and celebrated line of work [7, 23, 79, 122] culminated in the embedding of Fakcharoenphol, Rao and Talwar [79]—henceforth the "FRT embedding"—which showed that how to achieve $O(\log n)$ distortion probabilistic tree embeddings. Together with the above template this reduces many graph problems to much easier problems on trees at the cost of an $O(\log n)$ approximation

factor. This has lead to a myriad of approximation, online, and dynamic algorithms with poly-logarithmic approximations and competitive ratios for NP-hard problems such as for k-server [20], metrical task systems [26], group Steiner tree and group Steiner forest [8, 92, 149], buy-at-bulk network design [15] and (oblivious) routing [153]. For many of these problems tree embeddings are the only known way of obtaining such algorithms on general graphs.

There has also been considerable work on extending the power of tree embeddings to a variety of settings including tree embeddings for planar graphs [131], online tree embeddings [27], dynamic tree embeddings [48, 89], distributed tree embeddings [125] and tree embeddings where the resulting tree is a subgraph of the input graph [1, 2, 7, 73, 134]. Lastly, the notion of Ramsey trees and Ramsey tree covers is similar to several of the ideas we will explore in the subsequent chapters. Specifically, it is known that for every metric (V, d) and k there is some subset $S \subseteq V$ of size at least $n^{1-1/k}$ which embeds into a tree—a so-called Ramsey tree—with distortion $O(k)$ [3, 32, 143, 147]. Iterating (a slight strengthening of) this fact shows that there exist collections of Ramsey trees—so-called Ramsey tree covers—where each vertex v has some "home tree" in which the distances to v are preserved.

2.0.1 Network Design and Group Steiner Problems

The field of network design studies how to efficiently construct and use large networks. Over the past several decades researchers have paid particular attention to the construction of low-cost computer and transportation networks that enable specified communication and delivery demands.

Formally, these problems require computation of low-cost structures in graphs, such as paths, trees or subgraphs, that satisfy specified connectivity requirements.

Tree embeddings have been particularly successful in giving new approximation and online algorithms for network design problems. Two problems—which we explore in the subsequent two chapters—that have seen particular success with tree embeddings are the group Steiner tree and group Steiner forest problems.

(Offline and Online) Group Steiner Tree

Offline Group Steiner Tree: In the (offline) group Steiner Tree problem we are given a weighted graph $G = (V, E, w)$ as well as pairwise disjoint groups $g_1, g_2, \ldots, g_k \subseteq V$ and root $r \in V$. We let $N := \max_i |g_i|$. Our goal is to find a tree T rooted at r which is a subgraph of G and satisfies $T \cap g_i \neq \emptyset$ for every i. We wish to minimize our cost, $w(T) := \sum_{e \in E(T)} w(e)$.

The group Steiner tree problem was introduced by Reich and Widmayer [156] as an important problem in VLSI design. [92] gave the first randomized poly-log approximation for offline group Steiner tree using linear program rounding. Charikar et al. [46] derandomized this result and Chekuri et al. [49] showed that a greedy algorithm achieves similar results. Demaine et al. [62] gave improved algorithms for group Steiner tree on planar graphs.

Online Group Steiner Tree: Online group Steiner tree is the same as group Steiner tree as defined in Section 4.5.2 but where our solution need not be a tree and groups are revealed in time steps $t = 1, 2, \ldots$. That is, in time step t an adversary reveals a new group g_t and the algorithm must maintain a solution T_t where: (1) $T_{t-1} \subseteq T_t$; (2) T_t is feasible for the group Steiner tree problem on groups $g_1, \ldots g_t$ and; (3) T_t is competitive with the optimal offline solution for this problem where the competitive ratio of our algorithm is $\max_t w(T_t)/\mathrm{OPT}_t$ where OPT_t is the

cost of the optimal offline group Steiner tree solution on the first t groups. We assume that the possible groups revealed by the adversary are known ahead of time as otherwise this problem is known to admit no sub-polynomial approximations [8] and let k be the number of possible groups revealed by the adversary.

Alon et al. [8] gave the first poly-logarithmic online algorithm for group Steiner tree. Alon et al. [8] posed the existence of a deterministic poly-log approximation for online group Steiner tree as an open question which has since been restated several times [31, 40]. Very recently Bienkowski et al. [31] made exciting progress towards this open question by giving a poly-log deterministic approximation for online non-metric facility location—which is equivalent to the online group Steiner tree on trees with depth 2. Bartal et al. [27] also recently gave the first online algorithm for this problem which does not have n in its approximation ratio.

(Offline and Online) Group Steiner Forest

Offline Group Steiner Forest: Group Steiner forest generalizes group Steiner tree. In the (offline) group Steiner forest problem we are given a weighted graph $G = (V, E, w)$ as well as pairs of subsets of nodes $(S_1, T_1), (S_2, T_2), \ldots, (S_k, T_k)$ where $S_i, T_i \subseteq V$. Our goal is to find a forest F which is a subgraph of G and in which for each i there is an $s_i \in S_i$ and $t_i \in T_i$ such that s_i and t_i are connected in F. We wish to minimize our cost, $w(F) := \sum_{e \in E(F)} w(e)$.

[8] introduced the group Steiner forest problem to study online network formation. [52] gave the first poly-log approximation algorithm for group Steiner forest. [149] gave a worse poly-log approximation for group Steiner forest but one which was based on tree embeddings which will be useful for our purposes.

Online Group Steiner Forest: Online group Steiner forest is the same as group Steiner forest as defined in Section 4.7.3 but each pair (S_t, T_t) is revealed at time step $t = 1, 2, \ldots$ by an adversary and in each time step t we must maintain a forest F_t which is feasible for pairs $(S_1, T_1), \ldots (S_t, T_t)$ so that $F_{t-1} \subseteq F_t$. The competitive ratio of an online algorithm with solution $\{F_t\}_t$ is $\max_t w(F_t)/\mathrm{OPT}_t$ where OPT_t is the optimal offline solution for the group Steiner forest problem we must solve in time step t. We assume that the possible pairs revealed by the adversary are known ahead of time as otherwise this problem is known to admit no sub-polynomial approximations [8] and let k be the number of possible pairs.

The existence of a poly-log-competitive online algorithm for group Steiner forest was first posed as an open question by [52]. [149] answered this question in the affirmative by showing that such an algorithm exists.

Note that group Steiner forest directly generalizes group Steiner tree since a tree instance on a weighted graph G with root $r \in V(G)$ can be reduced to an equivalent forest instance on the same graph G by mapping each group g to the pair $(\{r\}, g)$. This reductions is valid in both the offline and online setting.

Chapter 3

Copy Tree Embeddings

3.1 Introduction

Probabilistic tree embeddings have one drawback: Algorithms based on them naturally require randomization and their approximation guarantees only hold in expectation. For approximation algorithms—i.e., in the offline setting—there are derandomization tools, such as the FRT derandomizations given in [24, 47, 79], to overcome these issues. These derandomization results are so general that essentially any offline algorithm based on tree embeddings can be transformed into a deterministic algorithm with matching approximation guarantees (with only a moderate increase in running time). Unfortunately, these strategies are not applicable to online or dynamic settings where an adversary progressively reveals the input. Indeed, most online and dynamic algorithms that use FRT are randomized (e.g. [8, 26, 75, 76, 80, 100, 106, 149]).

This overwhelming evidence in the literature is driven by a well-known and fundamental barrier to the use of probabilistic tree embeddings in deterministic online and dynamic algorithms. More specifically and even worse, this is a barrier which prevents these algorithms from working against all but the weakest type of adversary. In particular, designing an online or dynamic algorithm which is robust to an oblivious adversary (which fixes all requests in advance, independently of the algorithm's randomness) is often much easier than designing an algorithm which is robust to an adaptive adversary (which chooses the next request based on the algorithm's current solution). As the actions of a deterministic algorithm can be fully predicted this distinction only holds for randomized algorithms—any deterministic algorithm has to always work against an adaptive adversary. For these reasons, many online and dynamic algorithms have exponentially worse competitive ratios in the deterministic or adaptive adversary setting than in the oblivious adversary setting. This is independent of computational complexity considerations.

The above barrier results from a repeatedly recognized and seemingly unavoidable phenomenon which prevents online algorithms built on FRT from working against adaptive adversaries. Specifically, there are graphs where every tree embedding must have many node pairs with polynomially-stretched distances [23]. There is nothing that prevents an adversary then from learning through the online algorithm's responses which tree was sampled and then tailoring the remainder of the online instance to pairs of nodes that have highly stretched distances. The exact same phenomenon occurs in the dynamic setting; see, for example, Guo et al. [100] and Gupta et al. [106] for dynamic algorithms with expected cost guarantees that only hold against oblivious adversaries because

they are based on FRT. In summary, online and dynamic algorithms that use probabilistic tree embeddings seem inherently randomized and seem to necessarily only work against adversaries oblivious to this randomness.

Overall it seems fair to say that prior to this work tree embeddings seemed fundamentally incapable of enabling adaptive-adversary-robust and deterministic algorithms in several well-studied settings.

3.1.1 Our Contributions

We provide a new type of metric embedding—the copy tree embedding— which is deterministic and therefore also adaptive-adversary-robust. Specifically, we show that any weighted graph G can be deterministically embedded into a single weighted tree with a small number of copies for each vertex. Any subgraph of G will project onto this tree in a connectivity and approximate-cost preserving way.

To precisely define our embeddings we define a copy mapping ϕ which maps a vertex v to its copies.

Definition 2 (Copy Mapping)**.** *Given vertex sets V and V' we say $\phi : V \to 2^{V'}$ is a copy mapping if every node has at least one copy (i.e. $|\phi(v)| \geq 1$ for all $v \in V$), copies are disjoint (i.e. $\phi(v) \cap \phi(u) = \emptyset$ for $u \neq v$) and every node in V' is a copy of some node (i.e. for every $v' \in V'$ there is some $v \in V$ where $v' \in \phi(v)$). For $v' \in V'$, we use the shorthand $\phi^{-1}(v')$ to stand for the unique $v \in V$ such that $v' \in \phi(v)$.*

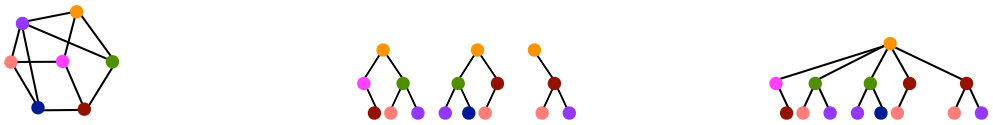
A copy tree embedding for a weighted graph G now simply consists of a tree T on copies of vertices of G with one distinguished root and two mappings $\pi_{G\to T}$ and $\pi_{T\to G}$ which map subsets of edges from G to T and from T to G in a way that preserves connectivity and approximately preserves costs. We say that *two vertex subsets U, W are connected* in a graph if there is a $u \in U$ and $w \in W$ such that u and w are connected. We also say that a mapping $\pi : 2^E \to 2^{E'}$ is *monotone* if for every $A \subseteq B$ we have that $\pi(A) \subseteq \pi(B)$.

Definition 3 (α-Approximate Copy Tree Embedding with Copy Number χ)**.** *Let $G = (V, E, w)$ be a weighted graph with some distinguished vertex $r \in V$ called the root. An α-approximate copy tree embedding with copy number χ consists of a weighted rooted tree $T = (V', E', w')$, a copy mapping $\phi : V \to 2^{V'}$ and edge mapping functions $\pi_{G\to T} : 2^E \to 2^{E'}$ and $\pi_{T\to G} : 2^{E'} \to 2^E$ where $\pi_{T\to G}$ is monotone and:*

1. ***Connectivity Preservation:*** *For all $F \subseteq E$ and $u, v \in V$ if u, v are connected by F, then $\phi(u), \phi(v) \subseteq V'$ are connected by $\pi_{G\to T}(F)$. Symmetrically, for all $F' \subseteq E'$ and $u', v' \in V'$ if u' and v' are connected by F' then $\phi^{-1}(u')$ and $\phi^{-1}(v')$ are connected by $\pi_{T\to G}(F')$.*
2. ***α-Cost Preservation****: For any $F \subseteq E$ we have $w(F) \leq \alpha \cdot w'(\pi_{G\to T}(F))$ and for any $F' \subseteq E'$ we have $w'(F') \leq w(\pi_{T\to G}(F'))$.*
3. ***Copy Number:*** *$|\phi(v)| \leq \chi$ for all $v \in V$ and $\phi(r) = \{r'\}$ where r' is the root of T.*

A copy tree embedding is efficient if T, ϕ, and $\pi_{T\to G}$ are deterministically poly-time computable and well-separated if T is well-separated.

We emphasize that, whereas standard tree embeddings guarantee costs are preserved in expectation, our copy tree embeddings preserve costs deterministically. Also notice that for efficient copy



(a) Graph G. **(b)** Compute partial tree embeddings. **(c)** Merge trees.

Figure 3.1: Illustration of our first construction where we merge $O(\log n)$ partial tree embeddings.

(a) Graph G. **(b)** Enumerate FRT support. **(c)** Merge trees.

Figure 3.2: Illustration of our second construction where we merge the $O(n \log n)$ trees in the FRT support.

tree embeddings we do not require that $\pi_{G \to T}$ is efficiently computable; this is because $\pi_{G \to T}$ will be used in our analyses but not in any of our algorithms. The idea of embeddings which map vertices to several copies has previously been explored by Bartal and Mendel [25] and was recently explored in a concurrent work of Filtser [84]. The key difference between these works and our own is that the number of copies that each vertex is mapped to is unboundedly large (in the case of Bartal and Mendel [25]) or only small in expectation (in the case of Filtser [84]). On the other hand, the analogue of α-cost preservation in Bartal and Mendel [25] ("path preservation") is stronger than our α-cost preservation.

We first give two copy tree embedding constructions which trade off between the number of copies and cost preservation. Both constructions are based on the idea of merging appropriately chosen tree embeddings as pictured in Figure 3.1 and Figure 3.2 where we color nodes according to the node whose copy they are.

Construction 1: Merging Partial Tree Embeddings (Section 3.2). The cornerstone of our first construction is the idea of merging embeddings which give good *deterministic* distance preservation. If our goal is to embed the entire input metric into a tree this is impossible. However, it is possible to embed a random constant fraction of nodes in an input metric into a tree in a way that deterministically preserves distances of the embedded nodes; an embedding which we call a "partial tree embedding" (see also Gupta et al. [105], Haeupler et al. [111]). We then use the method of conditional expectation to derandomize a node-weighted version of this random process and apply this derandomization $O(\log n)$ times, down-weighting nodes as they are embedded. The result of this process is $O(\log n)$ partial tree embeddings where a multiplicative-weights-type argument shows that each node appears in a constant fraction of these embeddings. Merging these $O(\log n)$ embeddings gives our copy tree while an Euler-tour-type proof shows that subgraphs of the input graph can be mapped to our copy tree in a cost and connectivity-preserving fashion. The following theorem summarizes our first construction.

Theorem 4. *There is a poly-time deterministic algorithm which given any weighted graph $G = (V, E, w)$ and root $r \in V$ computes an efficient and well-separated $O(\log^2 n)$-approximate copy tree embedding with copy number $O(\log n)$.*

Construction 2: Merging FRT Support (Section 3.2.3). Our second construction follows from a known fact that the size of the support of the FRT distribution can be made $O(n \log n)$ and this

support can be computed deterministically in poly-time [47]. Merging each tree in this support at the root and some simple probabilistic method arguments give a copy tree embedding that is $O(\log n)$-cost preserving but with an $O(n \log n)$ copy number. Equivalently, it can be inferred from Bartal and Mendel [25]. The next theorem summarizes this construction.

Theorem 5. *There is a poly-time deterministic algorithm which given any weighted graph $G = (V, E, w)$ and root $r \in V$ computes an efficient and well-separated $O(\log n)$-approximate copy tree embedding with copy number $O(n \log n)$.*

While our second construction achieves a slightly better cost bound than our first construction, it has the significant downside of a linear copy number. Notably, this linear copy number makes our second construction unsuitable for some applications, including, for example, our second application as described below. Moreover, our first construction also has several desirable properties which our second does not which we expect might be useful for future applications. These include: (1) $\pi_{G \to T}$ is monotone (in addition to $\pi_{T \to G}$ being monotone as stipulated by Theorem 3); (2) if u and v are connected by $F \subseteq E$ then $\Omega(\log n)$ vertices of $\phi(u)$ are connected to $\Omega(\log n)$ vertices of $\phi(v)$ in $\pi_{G \to T}(F)$ (as opposed to just one vertex of $\phi(u)$ and one vertex of $\phi(v)$ as in Theorem 3) and; (3) if u is connected to r by $F \subseteq E$ then every vertex in $\phi(u)$ is connected to $\phi(r)$ in $\pi_{G \to T}(F)$ (as opposed to just one vertex of $\phi(u)$ as in Theorem 3).

The reason our embeddings are well-suited to group Steiner problems and its generalizations is that mapping it onto a copy tree embedding simply results in another instance of the group Steiner tree problem, this time on a tree. Indeed, our embeddings almost immediately reduce the open question of Alon et al. [8]—solving online group Steiner tree and forest deterministically on a general graph—to its tree case (see Section 3.4 for details). Equivalently, this reduction be inferred from the embeddings Bartal and Mendel [25], though Alon et al. [8] seems to have overlooked this connection.

Application: Deterministic Online Covering Steiner (Section 3.3). As an application of our embedding we make progress on the aforementioned open question of Alon et al. [8] by showing that the online covering Steiner problem admits a *bicriteria* deterministic poly-log approximation. Specifically, note that the covering Steiner problem generalizes group Steiner tree but unlike group Steiner tree it admits a natural bicriteria relaxation: instead of connecting, for example, $\frac{1}{2}$ of the nodes in each group we could require that our algorithm only connects, say, $\frac{(1-\epsilon)}{2}$ of all nodes in each group for some $\epsilon > 0$. Thus, our result can be seen as showing that there is indeed a deterministic poly-log competitive algorithm for online group Steiner tree—as posed in the above open question of Alon et al. [8]—*provided the algorithm can be bicriteria* in the relevant sense. We use our embeddings for this application. Those of Bartal and Mendel [25] or Filtser [84] are not suitable for our first application since this application requires a bound on the number of copies of each vertex. More formally, we obtain a deterministic poly-log bicriteria approximation for this problem which connects at least $\frac{1-\epsilon}{2}$ of the nodes in each group (notated "$(1-\epsilon)$-connection competitive" below) by using our copy tree embeddings and a "water-filling" algorithm to solve the tree case.

Theorem 6. *There is a deterministic poly-time algorithm for online covering Steiner (on general graphs) which is $O(\frac{\log^3 n}{\epsilon} \cdot \max_i \frac{|g_i|}{r_i})$-cost-competitive and $(1-\epsilon)$-connection-competitive.*

As we later observe, providing a deterministic poly-log-competitive algorithm for the online covering Steiner problem with any constant bicriteria relaxation is strictly harder than providing a

deterministic poly-log-competitive algorithm for online (non-group) Steiner tree. Thus, this result also generalizes the fact that a deterministic poly-log approximation is known for online (non-group) Steiner tree [119]. Additionally, as a corollary we obtain the first non-trivial deterministic approximation algorithm for online group Steiner tree—albeit one with a linear dependence on the maximum group size.[1]

Corollary 7. *There is an $O(N \log^3 n)$-competitive deterministic algorithm for online group Steiner tree where $N := \max_i |g_i|$ is the maximum group size.*

3.2 Copy Tree Embedding Constructions

In this section we give our two constructions of copy tree embeddings. We begin by giving our first copy tree embedding construction based on merging partial tree embeddings.

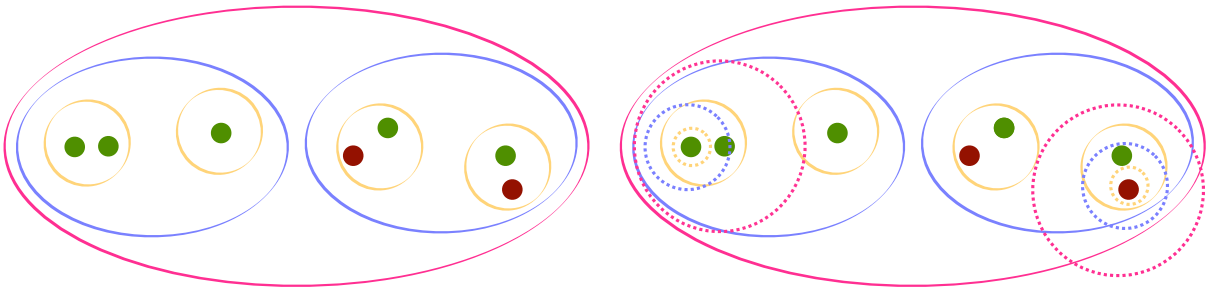
Theorem 4. *There is a poly-time deterministic algorithm which given any weighted graph $G = (V, E, w)$ and root $r \in V$ computes an efficient and well-separated $O(\log^2 n)$-approximate copy tree embedding with copy number $O(\log n)$.*

If it were possible to give a single tree embedding which simultaneously preserved all distances between all nodes then we could simply take such a tree embedding as our copy tree embedding. However, such a tree embedding is, in general, impossible. The key insight we use to overcome this issue is that one can approximately preserve distances in a *deterministic* way if one only embeds a constant fraction of all nodes in the input metric; we call such an embedding a partial tree embedding. Combining $O(\log n)$ such partial tree embeddings will give our construction.

In more detail, in Section 3.2.1 we show that an appropriate $O(\log n)$ "padded hierarchical decompositions" gives $O(\log n)$ partial tree embeddings where every node is embedded a constant number of times. Next, we show that such a collection of partial tree embeddings indeed gives us a copy tree embedding as in Theorem 4; the main observation that this reduction relies on is the constant congestion induced by Euler tours which will allow us to project from our input graph to our partial tree embeddings in a cost and connectivity-preserving fashion. Many of the ideas in this section appear either implicitly or explicitly in other works, including those of Mendel and Naor [143] and Gupta et al. [105]. Thus, our goal after this point is to compute an appropriate collection of padded hierarchical decompositions.

In Section 3.2.2 we proceed to show how to compute the required collection of padded hierarchical decompositions. Our construction of hierarchical decompositions will make use of the FRT cutting scheme and paddedness properties of it previously observed by Gupta et al. [105]. To this end, we provide a novel derandomization of a node-weighted version of the FRT cutting scheme by combining the powerful multiplicative weights methodology [13] together with the classic method of conditional expectation and pessimistic estimators. The recent concurrent work of Filtser [84] achieves similar results but by quite different means, namely though a deterministic metric Ramsey-type embedding which is inherently node-weighted.

[1]We explicitly note here that this bicriteria guarantee does not yield a solution to the open problem of Alon et al. [8] of finding a poly-log deterministic approximation to the online group Steiner tree problem.



(a) Hierarchical decomposition.

(b) Why node on left is padded and node on right is not.

Figure 3.3: Illustration of a hierarchical decomposition $\mathcal{H}$ with $h = 4$ with $n = 7$. Each part in each $\mathcal{P}_i \in \mathcal{H}$ is colored according to i; singleton parts not pictured. We give α-padded nodes in green and all other nodes in red where we illustrate why the node on the far left is α-padded and the node on the far right is not by drawing $B(v, \alpha \cdot 2^i)$ for $i \geq 1$ in colors according to i for these two nodes.

3.2.1 From Padded Hierarchical Decompositions to Copy Tree Embeddings

Gupta et al. [105] introduced the idea of padded hierarchical decompositions which we illustrate in Figure 3.3.

Definition 8. *A hierarchical decomposition $\mathcal{H}$ of a metric (V, d) of diameter D is a sequence of partitions $\mathcal{P}_0, \ldots, \mathcal{P}_h$ of V where $h = \Theta(\log D)$ and:*

1. *The partition $\mathcal{P}_h$ is one part containing all of V;*
2. *Each part in $\mathcal{P}_i$ has diameter at most 2^i;*
3. *$\mathcal{P}_i$ is a refinement of $\mathcal{P}_{i+1}$; that is, every part in $\mathcal{P}_i$ is contained in some part of $\mathcal{P}_{i+1}$.*

Notice that each part of $\mathcal{P}_0$ is a singleton node by our assumption that edge weights are at least 1 (we assume that the constant in the theta notation of $h = \Theta(\log D)$ is sufficiently large).

Definition 9 (α-Padded Node)**.** *For some $\alpha \leq 1$, a node v is α-padded in hierarchical decomposition $\mathcal{P}_0, \ldots, \mathcal{P}_h$ if for all $i \in [0, h]$ the ball $B(v, \alpha \cdot 2^i)$ is contained in some part of $\mathcal{P}_i$.*

The main result we show in this section is how to use a collection of padded hierarchical decompositions to construct a copy tree embedding.

Lemma 10. *Let $\{\mathcal{H}_i\}_{i=1}^k$ be a collection of hierarchical decompositions of weighted graph $G = (V, E, w)$ such that every v is α-padded in at least $.9k$ decompositions. Then, there is a poly-time deterministic algorithm which, given $\{\mathcal{H}_i\}_{i=1}^k$ and a root $r \in V$, returns an efficient and well-separated $O(\frac{k}{\alpha})$-approximate copy tree embedding with copy number k.*

From Padded Hierarchical Decompositions to Partial Tree Embeddings

We now formalize the notion of a partial tree embedding.

Definition 11 (Partial Tree Embedding)**.** *A γ-partial tree embedding of metric (V, d) is a well-separated weighted tree $T = (V', E', w)$ where:*

1. ***Partial Embedding:** $V' \subseteq V$;*

2. ***Worst-Case Distance Preservation*** *For any $u, v \in V'$ we have $d(u,v) \leq d_T(u,v) \leq \gamma \cdot d(u,v)$.*

In the remainder of this section we show how good padded hierarchical decompositions deterministically give good partial tree embeddings.

The reason padded decompositions will be useful for us is that—as we prove in the following lemma—all distances between padded nodes are well-preserved.[2] Given a hierarchical decomposition $\mathcal{H}$ we let $T_{\mathcal{H}}$ be the natural well-separated tree corresponding to $\mathcal{H}$. In particular, a hierarchical decomposition $\mathcal{H}$ naturally corresponds to a well-separated tree which has a node for each part and an edge of weight 2^i between a part in $\mathcal{P}_i$ and a part in $\mathcal{P}_{i+1}$ if the latter contains the former. In Figure 3.4a we illustrate the well-separated tree corresponding to the hierarchical decomposition in Figure 3.3a. We will slightly abuse notation and identify each singleton set in such a tree with its one constituent vertex.

Lemma 12. *If nodes u, v are α-padded in a hierarchical decomposition $\mathcal{H}$ then $d(u,v) \leq d_{T_{\mathcal{H}}}(u,v) \leq O\left(\frac{1}{\alpha} \cdot d(u,v)\right)$.*

Proof. Let $T_{\mathcal{H}}$ be the well-separated tree corresponding to $\mathcal{H}$. Let w be the least common ancestor of u and v in $T_{\mathcal{H}}$ and let l be the height of w in $T_{\mathcal{H}}$. By the definition of $T_{\mathcal{H}}$, the distance between u and v in $T_{\mathcal{H}}$ is $d_{T_{\mathcal{H}}}(u,v) = 2 \cdot \sum_{i=0}^{l} 2^i$ and so we have

$$2^{l+1} \leq d_{T_{\mathcal{H}}}(u,v) \leq 2^{l+2}. \tag{3.2.1}$$

We next prove that $d_{T_{\mathcal{H}}}(u,v) \leq O(\frac{1}{\alpha} \cdot d(u,v))$. Notice that for $j = \lceil \log(d(u,v)/\alpha) \rceil$ we know that $B(v, \alpha \cdot 2^j)$ contains u since for this j it holds that $\alpha \cdot 2^j \geq d(u,v)$. Since $\mathcal{H}$ is α-padded it follows that $B(v, \alpha \cdot 2^j)$ is contained in some part of $\mathcal{P}_j$; but it then follows that the least common ancestor of u and v is at height at most j and so $l \leq \lceil \log(d(u,v)/\alpha) \rceil$. Combining this with the upper bound in Equation (3.2.1) we have

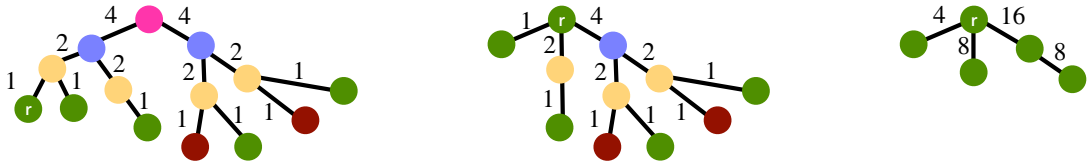
$$\begin{aligned} d_{T_{\mathcal{H}}}(u,v) &\leq 2^{l+2} \\ &\leq 2^{\lceil \log(d(u,v)/\alpha) \rceil + 2} \\ &\leq O\left(\frac{1}{\alpha} \cdot d(u,v)\right) \end{aligned}$$

We now prove that $d(u,v) \leq d_{T_{\mathcal{H}}}(u,v)$. Since the diameter of each part in $\mathcal{P}_i$ is at most 2^i we know that the least common ancestor of u and v in T corresponds to a part with diameter at most 2^l. However, since the least common ancestor of u and v corresponds to a part which contains both u and v, we must have $d(u,v) \leq 2^l \leq 2^{l+1}$. Combining this with the lower bound in Equation (3.2.1) we have $d(u,v) \leq d_{T_{\mathcal{H}}}(u,v)$ as desired. □

We show how to turn a hierarchical decomposition into a partial tree embedding in the next lemma which we illustrate in Figure 3.4.

Lemma 13. *Given a hierarchical decomposition $\mathcal{H}$ on metric (V,d) and root $r \in V$ which is α-padded in $\mathcal{H}$, one can compute in deterministic poly-time a $O(\frac{1}{\alpha})$-partial tree embedding*

[2] This fact seems to be implicit in Gupta et al. [105] among other works but there does not seem to be a readily citable version of it.



(a) Tree corresponding to Figure 3.3a hierarchical decomposition.

(b) Contract to ensure r is root of resulting tree.

(c) Multiply weights by 4 and contract non-α-padded vertices.

Figure 3.4: How to turn a hierarchical decomposition into a partial tree embedding. We color nodes from the input metric in green if they are padded and red otherwise. Remaining nodes colored according to their corresponding hierarchical decomposition part. r is the node on the far left of the tree.

$T = (V', E')$ *with root* r *where* $V' := \{v \in V : v \text{ is } \alpha \text{ padded}\}$,

Proof. Let $T_{\mathcal{H}}$ be the well-separated tree which corresponds to $\mathcal{H}$ as described above.

We construct T from $T_{\mathcal{H}}$ using Theorem 12 and a trick of Konjevod et al. [131]. Let V' be all leaves of $T_{\mathcal{H}}$ whose corresponding nodes are α-padded in $\mathcal{H}$. Next, contract the path from r to the root of $T_{\mathcal{H}}$ and identify the resulting node with r. Then, delete from $T_{\mathcal{H}}$ all sub-trees which do not contain a node in V'; in the resulting tree every node is either in V' or the ancestor of a node in V'. Next, while there exists a node v such that its parent u is not in V' we contract $\{v, u\}$ into one node and identify the resulting node with v. Lastly, we multiply the weight of every edge by 4 and return the result as $T = (V', E', w)$ where, again, w is the weight function of $T_{\mathcal{H}}$ times 4.

Clearly, the vertex set of T will be V'. Moreover, T is well-separated since $T_{\mathcal{H}}$ was well-separated and r will be the root of T by construction.

We now use an analysis of Konjevod et al. [131] to show that for any pair of vertices $u, v \in V'$ we have

$$d_{T_{\mathcal{H}}}(u, v) \leq d_T(u, v) \leq 4 \cdot d_{T_{\mathcal{H}}}(u, v) \tag{3.2.2}$$

The upper bound is immediate from the fact that we only contract edges and then multiply all edge weights by 4. To see the lower bound—$d_{T_{\mathcal{H}}}(u, v) \leq d_T(u, v)$—notice that if u and v have a least common ancestor a at height l in $T_{\mathcal{H}}$, then $d_{T_{\mathcal{H}}}(u, v) = 2^{l+2} - 4$. However, the closest u and v can be in T is if (without loss of generality) u is identified with a and (without loss of generality) v is a child of u in T; the length of this edge is the length of a child edge of a in $T_{\mathcal{H}}$ times four which is 2^{l+2}. Thus $d_{T_{\mathcal{H}}}(u, v) = 2^{l+2} - 4 \leq 2^{l+2} = d_T(u, v)$.

Finally, we conclude by applying Theorem 12. In particular, it remains to show $d(u, v) \leq d_T(u, v) \leq O(\frac{1}{\alpha} \cdot d(u, v))$ but this is immediate by combining Theorem 12 and Equation (3.2.2). □

From Partial Tree Embeddings to Copy Tree Embeddings

We now describe how partial tree embeddings satisfy useful connectivity properties and then use these properties to construct a copy tree embedding from a collection of good partial tree embeddings.

The following two lemmas demonstrate how to map to and from partial tree embeddings in a way that preserves cost and connectivity.

Lemma 14 (Graph $\to$ Partial Tree Projection)**.** *Let $G = (V, E, w_G)$ be a weighted graph and let $T = (V', E', w_T)$ be a γ-partial tree embedding of (the metric induced by) G. There exists a deterministic, poly-time computable function $\pi : 2^E \to 2^{E'}$ such that for all sets of edges $F \subseteq E$ the following holds:*

1. ***Connectivity Preservation:*** *If $u, v \in V'$ are connected by F in G, then they are connected in $\pi(F)$ in T;*
2. ***Cost Preservation:*** $w_T(\pi(F)) \leq O(\gamma) \cdot w_G(F)$.

Proof. We first simplify F by noticing it is sufficient to prove the claim on every connected component in isolation. Furthermore, we can assume without loss of generality that F is a tree since taking a spanning tree of F can only decrease $w_G(F)$ and appropriately maintains connectivity. Finally, we delete every leaf that is not in V', which decreases $w_G(F)$ and maintains connectivities in V'.

We define $\pi(F)$ to be the unique minimal subtree of T which contains all nodes of V' that are incident to an edge in F. By transitivity of connectedness, we know that if $u, v \in V'$ are connected in F then they must also be connected in $\pi(F)$. Also, note that π is trivially deterministic poly-time computable.

It remains to argue the γ-cost preservation property. Double the edges of F; we call this multigraph $2F$. Since the degree of every vertex in $2F$ is even, we know that $2F$ has an Euler tour. Using this tour we can partition $2F$ into a set $\mathcal{P}$ of paths where each path connects two nodes in V' and the paths in $\mathcal{P}$ are multiedge-disjoint. Therefore, we have that $2w_G(F) = \sum_{P \in \mathcal{P}} w_G(P)$.

For each path $P \in \mathcal{P}$ in the tour between nodes $u, v \in V'$, we say that P **covers** all edges in T between u and v and let P' be the path in T between u and v. We note that every edge in $\pi(F)$ is covered by at least one path, hence $w_T(\pi(F)) \leq \sum_{P \in \mathcal{P}} w_T(P')$.

For every path in G connecting two nodes $u, v \in V'$ the distance-preservation properties of γ-partial tree embeddings implies that $w_T(P') \leq O(\gamma) \cdot w_G(P)$. Hence we have that $w_T(\pi(F)) \leq \sum_{P \in \mathcal{P}} w_T(P') \leq O(\gamma) \cdot \sum_{P \in \mathcal{P}} w_G(P) \leq O(\gamma) \cdot w_G(F)$ as required. □

We now show how to project in the reverse direction.

Lemma 15 (Partial Tree $\to$ Graph Projection)**.** *Let $G = (V, E, w_G)$ be a weighted graph and let $T = (V', E', w_T)$ be a γ-partial tree embedding of (the metric induced by) G. There exists a deterministic, poly-time computable function $\imath : 2^{E'} \to 2^E$ such that for all sets of edges $F' \subseteq E'$ the following holds:*

1. ***Connectivity Preservation:*** *If $u, v \in V'$ are connected by F' in T, then they are connected by $\imath(F)$ in G;*
2. ***Cost Preservation:*** $w_G(\imath(F')) \leq w_T(F')$.

Proof. For an edge $e' \in E'$, connecting $u, v \in V'$, we define $\imath(\{e'\})$ as some shortest path between u and v in G. Note that this implies that $w_G(\imath(\{e'\})) \leq w_T(e')$ by the properties of a partial tree embedding. We extend $\imath$ to $F' \subseteq E'$ by defining $\imath(F') := \bigcup_{e' \in F'} \imath(\{e'\})$. Notice that $\imath$ is indeed deterministic, poly-time computable and is connectivity preserving by the transitivity of connectivity.

We now verify the cost preservation of $\imath$: we have that $w_G(\imath(F')) = w_G(\bigcup_{e' \in F'} \imath(\{e'\})) \leq \sum_{e' \in F'} w_G(\imath(\{e'\})) \leq \sum_{e' \in F'} w_T(e') = w_T(F')$. □

Using these two properties we can conclude our proof of Theorem 10, which we restate here.

Lemma 16. *Let $\{\mathcal{H}_i\}_{i=1}^k$ be a collection of hierarchical decompositions of weighted graph $G = (V, E, w)$ such that every v is α-padded in at least $.9k$ decompositions. Then, there is a poly-time deterministic algorithm which, given $\{\mathcal{H}_i\}_{i=1}^k$ and a root $r \in V$, returns an efficient and well-separated $O(\frac{k}{\alpha})$-approximate copy tree embedding with copy number k.*

Proof. Our embedding is gotten by combining the above lemmas in the natural way.

Specifically, we first apply Theorem 13 to all decompositions in $\{\mathcal{H}_i\}_{i=1}^k$ in which r is α-padded to get back $O(\frac{1}{\alpha})$-partial tree embeddings $\{T_i\}_i$ where $V(T_i) = \{v : v \text{ is } \alpha\text{-padded in } \mathcal{H}_i\}$. Next we apply Theorem 14 and Theorem 15 to each T_i to get back mapping functions π_i and $\imath_i$ respectively.

We now describe our $O(\frac{k}{\alpha})$-approximate copy tree embedding $(T, \phi, \pi_{G\to T}, \pi_{T\to G})$. We let T be the tree resulting from taking all trees in $\{T_i\}_i$ and then identifying all copies of r as the same vertex. Similarly, we let $\phi(v)$ be the set of all copies of v in T in the natural way. Next we let $\pi_{G\to T}(F)$ be $\bigcup_i \pi_i(F)$ where π_i is projected onto T in the natural way. We let $\pi_{T\to G}(F') := \bigcup_i \imath_i(F')$ be defined analogously.

Since each vertex appears in at least a $.9$ fraction of all T_i, by the pigeonhole principle we know that any pair connected by F in G must occur in some $\mathcal{H}_i$ together with r and so must be connected in $\pi_i(F)$ for some i where $T_i \in \{T_i\}_i$ and so some pair of corresponding copies are connected by $\pi_{G\to T}$; an analogous result holds for $\pi_{T\to G}$. The remaining properties of our embedding are immediate from the above cited lemmas. □

3.2.2 Deterministically Constructing Padded Hierarchical Decompositions

In the previous section we reduced computing good copy tree embeddings to computing good hierarchical decompositions. The existence of good hierarchical decompositions is immediate from prior work of Gupta et al. [105] and FRT.

Lemma 17 (Gupta et al. [105])**.** *Let $\mathcal{H}$ be the hierarchical decompositions resulting from a tree drawn from the Fakcharoenphol et al. [79] cutting scheme. Then, every vertex is $\Omega(\frac{1}{\log n})$-padded with constant probability in $\mathcal{H}$.*

A simple Chernoff and union bound proof then gives that $O(\log n)$ draws gives a collection of hierarchical decompositions in which every vertex is $\Omega(\frac{1}{\log n})$-padded in a constant fraction of the decompositions *with high probability*, i.e. at least $1 - \frac{1}{\mathsf{poly}(n)}$.

However, we are ultimately interested in a deterministic algorithm which is robust to adaptive adversaries and so we must derandomize the above with high probability result. We proceed to do so in this section.

To our knowledge, prior derandomizations of this cutting scheme—see, e.g. Chekuri et al. [51] or Fakcharoenphol et al. [79]—do not provide sufficiently strong guarantees for our purposes. We also note that the authors of Gupta et al. [105] claim to give a deterministic algorithm for computing hierarchical decompositions in a forthcoming journal version of their paper but said journal version never seems to have been published.

Derandomization Intuition

The intuition behind our derandomization is as follows. A single draw from the FRT cutting scheme guarantees that each node is $\Omega(1/\log n)$-padded with constant probability. If we could derandomize this result then we could produce one hierarchical decomposition such that at least a .99 fraction of all nodes are $\Omega(1/\log n)$-padded. Indeed, as we will see, standard derandomization techniques—the method of pessimistic estimators and conditional expectation—will allow us to do exactly this. However, since we must produce a collection of hierarchical decompositions in which every node is in a large percentage in all decompositions it is not clear how, then, to handle the remaining .01 fraction of nodes. One might simply rerun the aforementioned derandomization result on the remaining .01 nodes, then on the remaining .001 nodes and so on logarithmically-many times; however, it is easy to see that in the resulting collection of decompositions, while every node is padded in some decomposition, no node is necessarily padded in a large fraction of all the decompositions.

Rather, we would like to repeatedely run our derandomization on all nodes but in a way that takes into account which nodes are already padded in a large fraction of the decompositions we have already produced. In particular, if a node was already padded in most of the decompositions we have so far produced, we need not worry about producing decompositions in which this node is padded. Thus, we would like to derandomize in a way that would make such a node less likely to be padded in the remaining decompositions we produce while making nodes which have not so far been padded in many decompositions we produced more likely to be padded.

To accomplish this, we will formulate and then derandomize a *node-weighted* version of Theorem 17; this, in turn, will allow us to down-weight nodes which are padded in a large fraction of the decompositions we have so far produced when we run our derandomization; a multiplicative-weights-type analysis will then allow us to conclude our deterministic construction.

The FRT Cutting Scheme

In order to give our deterministic construction we must unpack the black box of the FRT cutting scheme.

The Fakcharoenphol et al. [79] cutting scheme (due to Calinescu et al. [41]) given metric (V, d) where $d(u, v) \geq 1$ for all $u, v \in V$ produces a hierarchical decomposition $\mathcal{H} = \{\mathcal{P}_0, \ldots, \mathcal{P}_h\}$ and is as follows. We first pick a uniformly random permutation π on V and a uniformly random value $\beta \in [\frac{1}{2}, 1)$. We let the radius for level i be $r_i := 2^{i-1} \cdot \beta$.

We let $\mathcal{P}_h$ be the trivial partition containing all vertices of V with $h = O(\log D)$. Next, we construct $\mathcal{P}_i$ by refining $\mathcal{P}_{i+1}$; in particular we divide each part $P_{i+1} \in \mathcal{P}_{i+1}$ into additional parts as follows. Each $v \in P_{i+1}$ is assigned to the first vertex u in π for which $v \in B(u, r_i)$. Notice that u need not be in P_{i+1}. Let C_u be all vertices in P_{i+1} which are assigned to u and add to $\mathcal{P}_i$ all C_u which are non-empty. Notice that here C_u really depends on i; we suppress this dependence in our notation for cleanliness of presentation.

One can easily verify that the resulting partitions indeed form a hierarchical decomposition.

Derandomizing via Multiplicative Weights and Pessimistic Estimators

As discussed above, our goal is to derandomize Theorem 17 while taking node weights into account. Suppose we have a distribution $\{p_v\}_v$ over vertices in v; intuitively this distribution how important each vertex is in regards to being α-padded. Then by Theorem 17 and linearity of expectation we have

$$\mathsf{E}_{\pi,\beta}\left[\sum_v p_v \cdot \mathbb{I}\left(v \text{ is } \Omega\left(\frac{1}{\log n}\right)\text{-padded in } \mathcal{H}\right)\right] = \sum_v p_v \cdot \Pr_{\pi,\beta}\left(v \text{ is } \Omega\left(\frac{1}{\log n}\right)\text{-padded in } \mathcal{H}\right) \geq .99.$$

where $\mathbb{I}$ is the indicator function.

Thus, our goal will be to gradually fix the randomness of π and β until we have found a way to deterministically set β and π so that at least a .95 fraction of nodes (weighted by p_vs) are $\Omega(\frac{1}{\log n})$-padded. That is, we aim to use the method of conditional expectation. We will treat a permutation π as an ordering of the elements of $[V]$. E.g. (v_2, v_1, v_3) is a permutation of $V = \{v_1, v_2, v_3\}$. Now, suppose we have fixed a prefix π_P of π which orders nodes $P \subseteq V$ and among the remaining $\bar{P} := V \setminus P$ we uniformly at randomly choose the remaining suffix $\pi_{\bar{P}}$. That is, $\pi = \pi_P \odot \pi_{\bar{P}}$ where π_P is fixed and $\pi_{\bar{P}}$ is a uniformly random permutation over $\bar{P}$ and $\odot$ is concatenation. Notice that it follows that every vertex of P will precede every vertex of $\bar{P}$ in π.

Let $\mathcal{H}(\pi_P, \beta)$ be the hierarchical decomposition returned when we run the FRT cutting scheme as above with the input value of β and with π chosen as $\pi = \pi_P \odot \pi_{\bar{P}}$. Notice that provided $P \neq V$ we have that $\mathcal{H}$ is a randomly generated. Let $f(\pi_P, \beta) := \sum_v p_v \cdot \Pr_{\pi_{\bar{P}}}\left(v \text{ is } \Omega\left(\frac{1}{\log n}\right)\text{-padded in } \mathcal{H}(\pi_P, \beta)\right)$ be the fraction of $\Omega(\frac{1}{\log n})$-padded nodes by weight in expectation in $\mathcal{H}(\pi_P, \beta)$. We now show that there is a so called "pessimistic estimator" $\hat{f}$ of f.

Lemma 18. *There is a function $\hat{f}$ such that*

1. ***Good start:** There is some deterministically poly-time computable set $R \subseteq \mathbb{R}$ such that for some $\beta \in R$ we have $\hat{f}(\pi_\emptyset, \beta) \geq .95$.*

and for any $P \subseteq V$, π_P and β

2. ***Computable:** $\hat{f}(\pi_P, \beta)$ is computable in deterministic poly-time;*
3. ***Monotone:** $\hat{f}(\pi_P, \beta) \leq \hat{f}(\pi_{P\cup\{v\}}, \beta)$ for some $v \in \bar{P}$;*
4. ***Pessimistic:** $\hat{f}(\pi_P, \beta) \leq f(\pi_P, \beta)$ for all π_P and β.*

Proof. We will use an analysis similar to Gupta et al. [105] but which accounts for the fixed prefix π_P of our permutation, demonstrates the above properties of our pessimistic estimator and which guarantees that R is computable in deterministic, poly-time.

We begin by defining $\hat{f}$. Fix a π_P and β and let $\alpha = \Omega(\frac{1}{\log n})$.

For node v, let $B_{i,v} := B(v, \alpha 2^i)$. Say that node u *protects* $B_{i,v}$ if its ball at level i contains $B_{i,v}$, i.e. if $r_i \geq d(u, v) + 2^i\alpha$. Say that u *threatens* $B_{i,v}$ if its ball at level i intersects $B_{i,v}$ but does not contain it, i.e. $d(u, v) - \alpha 2^i < r_i < d(u, v) + 2^i\alpha$. Finally, say that u *cuts* $B_{i,v}$ if it threatens $B_{i,v}$ and is the first node in π to threaten or protect $B_{i,v}$. Notice that if $B_{i,v}$ is not cut by any node for all i then v will be α-padded.

In order for $B_{i,v}$ to be cut by u it must be the case that u threatens $B_{i,v}$ and no node before u in π threatens or protects $B_{i,v}$. By how we choose r_i, u threatens $B_{i,v}$ if

$$d(u,v) - 2^i\alpha < \beta \cdot 2^{i-1} < d(u,v) + 2^i\alpha \tag{3.2.3}$$

In order for u to be the first node to threaten or protect $B_{i,v}$, it certainly must be the case that every node which is closer to v than u appears after u in π (since every such node either threatens or protects $B_{i,v}$). Thus, we let $N_v(u) := \{w : d(w,v) \leq d(u,v)\}$ be all nodes which are nearer to v than u.

Lastly, a node which is too far or too close to v cannot cut $B_{i,v}$. In particular, a node u can only cut $B_{i,v}$ if

$$2^{i-2} - 2^i\alpha \leq d(u,v) \leq 2^{i-1} + 2^i\alpha \tag{3.2.4}$$

We let $C_{i,v} := \{u : 2^{i-2} - 2^i\alpha \leq d(u,v) \leq 2^{i-1} + 2^i\alpha\}$ be all such nodes which might cut $B_{i,v}$. It follows that we have that $B_{i,v}$ is cut only if there exists some u in $C_{i,v}$ which both threatens v and precedes all $w \in N_v(u) \setminus \{u\}$ in π. Thus, we define $\hat{f}$ as follows

$$\hat{f}(\pi_P, \beta) := 1 - \sum_{v,i} p_v \sum_{u \in C_{i,v}} \Pr_{\pi_{\bar{P}}}(u \text{ precedes all } w \in N_v(u) \setminus \{u\} \text{ in } \pi) \cdot \mathbb{I}(u \text{ threatens } B_{i,v}).$$

where, again, $\mathbb{I}$ is the indicator function. We now verify properties (2)-(4).

2. Computable: Clearly $C_{i,v}$ is deterministically computable in poly-time since we need only check if Equation (3.2.4) holds for each vertex. Similarly $\mathbb{I}(u \text{ threatens } B_{i,v})$ for each $u \in C_{i,v}$ can be computed by checking if Equation (3.2.3) holds. We can deterministically compute $\Pr_{\pi_{\bar{P}}}(u \text{ precedes all } w \in N_v(u) \setminus \{u\} \text{ in } \pi)$ for each $u \in C_{i,v}$ as follows: if u precedes all $w \in N_v(u) \cap \pi_P$ then this probability is 1; if u is preceded in π_P by some $w \in N_v(u)$ then this probability is 0; otherwise $\pi_P \cap N_v(u) = \emptyset$, meaning all nodes in $N_v(u)$'s order in π are set by $\pi_{\bar{P}}$; in this case u precedes all nodes in $N_v(u) \setminus \{u\}$ with probability exactly $\frac{1}{|N_v(u)|}$.

3. Monotonicity is immediate by an averaging argument: in particular, $\hat{f}(\pi_P, \beta)$ is just an expectation taken over the randomness of $\pi_{\bar{P}}$ and so there must be some way to fix an element of P to achieve the expectation.

4. Pessimism is immediate from the above discussion; in particular, as discussed above a ball $B_{i,v}$ is cut only if there is some $u \in C_{i,v}$ which threatens $B_{i,v}$ and which precedes all w in $N_v(u) \setminus \{u\}$ in π; it follows by a union bound that v fails to be α-padded with probability at most

$$\sum_i \sum_{u \in C_{i,v}} \Pr_{\pi_{\bar{P}}}(u \text{ precedes all } w \in N_v(u) \setminus \{u\} \text{ in } \pi) \cdot \mathbb{I}(u \text{ threatens } B_{i,v}).$$

Finally, we conclude property (1): that there is some $\beta \in R$ where R is computable in deterministic poly-time and $\hat{f}(\pi_\emptyset, \beta) \geq .95$. Consider drawing a $\beta \in [\frac{1}{2}, 1]$ as in the FRT cutting scheme; we will argue that $\mathsf{E}_\beta\left[\hat{f}(\pi_\emptyset, \beta)\right] \geq .95$ and so there must be some β for which $\hat{f}(\pi_\emptyset, \beta) \geq .95$.

Letting π be a uniformly random permutation, we have

$$\mathsf{E}_\beta\left[\hat{f}(\pi_\emptyset,\beta)\right] = 1 - \sum_{v,i} p_v \sum_{u\in C_{i,v}} \Pr_\pi(u \text{ precedes all } w \in N_v(u)\setminus\{u\} \text{ in } \pi)\cdot \Pr_\beta(u \text{ threatens } B_{i,v}).$$

If u is the sth closest node to v then we have that $\Pr_\pi(u \text{ precedes all } w \in N_v(u)\setminus\{u\} \text{ in } \pi) = \frac{1}{s}$. Moreover, u threatens $B_{i,v}$ only if Equation (3.2.3) holds and since $\beta\cdot 2^{i-1}$ is distributed uniformly in $[2^{i-2},2^{i-1})$, this happens with probability $2^{i+1}\alpha/2^{i-2} = 8\alpha$. Next, we claim that for a fixed v, each u occurs in at most 3 of the $C_{i,v}$. In particular, notice that if u is in $C_{i,v}$ and $C_{i',v}$ then we know that $2^{i-2} - 2^i\alpha \le d(u,v) \le 2^{i'-1} + 2^{i'}\alpha$ which for $\alpha \le \frac{1}{8}$ (which we may assume since $\alpha = \Omega(\frac{1}{\log n})$) implies $i < i'+3$. Combining these facts with the fact that $H_n := \sum_{i=1}^n \frac{1}{i} \le O(\log n)$ we get

$$\mathsf{E}_\beta\left[\hat{f}(\pi_\emptyset,\beta)\right] \ge 1 - O(\alpha\log n).$$

and since $\alpha = \Omega(\frac{1}{\log n})$, by fixing the constant in the $\Omega(\frac{1}{\log n})$ to be sufficiently small we have $\mathsf{E}_\beta\left[\hat{f}(\pi_\emptyset,\beta)\right] \ge .95$ as desired

Lastly, we define R and argue that there must be some $\beta \in R$ such that $\hat{f}(\pi_\emptyset,\beta) \ge .95$. In particular, notice that since $\mathsf{E}_\beta\left[\hat{f}(\pi_\emptyset,\beta)\right] \ge .95$, it suffices to argue that there are polynomially-many efficiently computable intervals which partition $[\frac{1}{2},1)$ such that any β_1 and β_2 in the same interval satisfy $\hat{f}(\pi_\emptyset,\beta_1) = \hat{f}(\pi_\emptyset,\beta_2)$; letting R take an arbitrary element from each such interval will give the desired result.

Notice that $\hat{f}(\pi_\emptyset,\beta_1) \ne \hat{f}(\pi_\emptyset,\beta_2)$ only if there is some i,v and u such that u threatens $B_{i,v}$ with β set to β_1 but does not threaten $B_{i,v}$ with β set to β_2. By definition of what it means to threaten, we have

$$d(u,v) - 2^i\alpha < \beta_1\cdot 2^{i-1} < d(u,v) + 2^i\alpha$$

but either $d(u,v) - 2^i\alpha \ge \beta_2\cdot 2^{i-1}$ or $\beta_2\cdot 2^{i-1} \ge d(u,v) + 2^i\alpha$. We then have either

$$\beta_2 \le d(u,v)\cdot 2^{1-i} - 2\alpha < \beta_1 \tag{3.2.5}$$

or

$$\beta_1 < d(u,v)\cdot 2^{1-i} + 2\alpha \le \beta_2. \tag{3.2.6}$$

With Equations 3.2.5 and 3.2.6 in mind, we define $R_l := \{d(u,v)\cdot 2^{1-i} + 2\alpha : u,v\in V, i\in[h]\}$ to be all the lower thresholds of when a change in β affects $\hat{f}$ and define $R_u := \{d(u,v)\cdot 2^{1-i} - 2\alpha : u,v\in V, i\in[h]\}$ to be all such upper thresholds. Let $t^{(l)}$ be the lth largest element of $(R_l\cup R_r)\cap[\frac{1}{2},1)$ and let R consist of one arbitrary element from the interval between $t^{(l)}$ and $t^{(l+1)}$ for $l\ge 0$ where the interval includes $t^{(l)}$ only if $t^{(l)}\in R_l$ and $t^{(l+1)}$ only if $t^{(l+1)}\in R_u$; $t^{(0)} = \frac{1}{2}$ is always included and $t^{(|R|)} = 1$ is never included. By the above discussion every β_1 and β_2 which are in the same interval satisfy $\hat{f}(\pi_\emptyset,\beta_1) = \hat{f}(\pi_\emptyset,\beta_1)$; moreover, these intervals partition $[\frac{1}{2},1]$ by construction.

We know $|R| = \text{poly}(n)$ since $h \leq O(\log n)$ by our assumption that $\max_{u,v} d(u,v)$ is $\text{poly}(n)$ and there are n^2 pairs u, v. Clearly R is computable in deterministic poly-time. Thus, by the above discussion R must contain some β such that $\hat{f}(\pi_\emptyset, \beta) \geq .95$. □

We now formalize our node-weighted derandomization.

Lemma 19. *There is a deterministic algorithm which given metric (V, d) and a distribution $\{p_v\}_v$ over nodes returns a hierarchical decomposition $\mathcal{H}$ in which at least a .95 fraction of nodes are $\Omega(\frac{1}{\log n})$-padded by weight; i.e.*

$$\sum_v p_v \cdot \mathbb{I}\left(v \text{ is } \Omega\left(\frac{1}{\log n}\right)\text{-padded in } \mathcal{H}\right) \geq .95.$$

Proof. Our derandomization algorithm is as follows. First, choose the $\beta \in R$ which maximizes $\hat{f}(\pi_\emptyset, \beta)$. Call this β^*. Next, initially let $P = \emptyset$ and repeat the following until $P = V$: for $v \in \bar{P}$ we compute $f(\pi_{P\cup\{v\}}, \beta^*)$; we add to P whichever v maximizes $f(\pi_{P\cup\{v\}}, \beta^*)$. Lastly, we return $\mathcal{H}(\pi_V, \beta^*)$.

By Theorem 18 we know that β^* will satisfy $\hat{f}(\pi_\emptyset, \beta^*) \geq .95$. Moreover, since $\hat{f}$ is monotone by Theorem 18 we know that the π_V we choose will satisfy $\hat{f}(\pi_V, \beta^*) \geq .95$. Lastly, since $\hat{f}$ is pessimistic, it follows that $f(\pi_V, \beta^*) \geq f(\pi_V, \beta^*) \geq .95$ and so $\mathcal{H}(\pi_V, \beta^*)$ is padded on a .95 fraction of nodes by weight as desired.

The deterministic polynomial runtime of our algorithm is immediate from the deterministic poly-time computability of $\hat{f}$ and the fact that R is computable in deterministic poly-time. □

Using the above node-weighted derandomization lemma gives our deterministic copy tree embedding construction. In particular, we run the following multiplicative-weights-type algorithm with $\epsilon = .01$ and set the number of iterations as $\tau := 4 \ln n/\epsilon^2$. In the following we let $p_v^{(t)} := w_v^{(t)} / \sum_v w_v^{(t)}$ be the proportional share of v's weight in iteration t.

1. Uniformly set the initial weights: $w_v^{(1)} = 1$ for all $v \in V$.
2. For $t \in [\tau]$:
   (a) Run the algorithm given in Theorem 19 using distribution $p^{(t)}$ and let $\mathcal{H}_t$ be the resulting hierarchical decomposition.
   (b) **Set mistakes:** For each vertex v which is $\Omega(\frac{1}{\log n})$-padded in $\mathcal{H}_t$ let $m_v^{(t)} = 1$. Let $m_v^{(t)} = 0$ for all other v.
   (c) **Update weights:** for all $v \in V$, let $w_v^{(t+1)} \leftarrow \exp(-\epsilon m_v^{(t)}) \cdot w_v^{(t)}$.
3. Return $(\mathcal{H}_t)_{t=1}^{\tau}$.

We state a well-known fact regarding multiplicative weights in our notation. Readers familiar with multiplicative weights may recognize this as the fact that the expected performance of multiplicative weights over logarithmically-many rounds is competitive with every expert.

Lemma 20 ([13, 90])**.** *The above algorithm guarantees that for any $v \in V$ we have*

$$\frac{1}{T}\sum_{t\leq\tau} p^{(t)} \cdot m^{(t)} \leq \epsilon + \frac{1}{T}\sum_{t\leq\tau} m_v^{(t)}$$

where $p^{(t)} \cdot m^{(t)} := \sum_v p_v^{(t)} m_v^{(t)}$ *is the usual inner product.*

Using this fact we conclude that we are able to produce a good set of hierarchical decompositions.

Lemma 21. *The above algorithm returns a collection of hierarchical decompositions* $\{\mathcal{H}_t\}_{t=1}^{\tau}$ *where* $\tau = \Theta(\log n)$ *and every vertex is* $\Omega(1/\log n)$*-padded in at least* $.9\tau$ *of the decompositions.*

Proof. Since $\tau := 4 \ln n / \epsilon^2$ we know that $\tau = \Theta(\log n)$.

We need only argue, then, that each node is padded in at least a $.9$ fraction of the τ total $\mathcal{H}_t$. Let

$$f_v := \frac{1}{\tau} \sum_{t \leq \tau} \mathbb{I}\left(v \text{ is } \Omega\left(\frac{1}{\log n}\right)\text{-padded in } \mathcal{H}_t\right)$$

be the fraction of the decompositions in which v is padded. Consider a fixed node v. By Theorem 20 we know that

$$\frac{1}{\tau} \sum_{t \leq \tau} p^{(t)} \cdot m^{(t)} \leq \epsilon + \frac{1}{\tau} \sum_{t \leq \tau} m_v^{(t)} \tag{3.2.7}$$

By definition of $m_v^{(t)}$ we have that the right hand side of Equation (3.2.7) is $\epsilon + f_v$. On the other hand, by how we set $m^{(t)}$, the left hand side of Equation (3.2.7) is $\frac{1}{\tau} \sum_t \sum_v p_v^{(t)} \cdot \mathbb{I}(v \text{ is } \Omega(\frac{1}{\log n})\text{-padded in } \mathcal{H})$ which by Theorem 19 is at least $.95$. Combining these facts we have $.95 \leq \epsilon + f_v$ and so by our choice of ϵ we know $.9 \leq f_v$ as desired. □

Combining Theorem 21 with Theorem 10 gives Theorem 4.

3.2.3 Construction 2: Merging FRT Support

In this section we observe that the support of the FRT distribution can be merged to produce copy tree embeddings with cost stretch $O(\log n)$ and copy number $O(n \log n)$. In particular, we rely on the known fact that one can make the size of the support of the FRT distribution $O(n \log n)$ and compute said support in deterministic poly-time, as summarized in the following theorem.

Theorem 22 ([47, 79, 131])**.** *Given a weighted graph* $G = (V, E, w)$ *and root* $r \in V$*, there exists a distribution* $\mathcal{D}$ *being supported over* $O(n \log n)$ *well-separated weighted trees on* V *rooted at* r *where for any* $u, v \in V$ *we have* $\mathsf{E}_{T \sim \mathcal{D}}[d_T(u, v)] \leq O(\log n \cdot d_G(u, v))$ *and for every* T *in the support of* $\mathcal{D}$ *we have* $d_G(u, v) \leq d_T(u, v)$*. Also, (the support and probabilities of)* $\mathcal{D}$ *can be computed in deterministic poly-time.*

Merging the trees of this distribution and some simple probabilistic method arguments give a copy tree embedding with the desired properties.

Theorem 5. *There is a poly-time deterministic algorithm which given any weighted graph* $G = (V, E, w)$ *and root* $r \in V$ *computes an efficient and well-separated* $O(\log n)$*-approximate copy tree embedding with copy number* $O(n \log n)$*.*

Proof. Let $T_1, \ldots, T_k$ with $k = O(n \log n)$ be the trees in the support of the distribution $\mathcal{D}$ as guaranteed by Theorem 22. Then, we let T be the result of identifying each copy of r as the same vertex in each T_i (but not identifying copies of other vertices in V as the same vertex); that is, $|V(T)| = k \cdot n - (k - 1)$. T's weight function is inherited from each T_i in the natural

way. Similarly, we let $\phi(v)$ be the set containing each copy of v in each of the T_i. It is easy to verify that ϕ is indeed a copy mapping. Also, note that $\phi(v)$ is computable in deterministic poly-time, our copy number is $O(n \log n)$ by construction and that T is well-separated since each T_i is well-separated.

We next specify $\pi_{G \to T}(F)$ for a fixed F. For tree T_i, let $T_i' \subseteq T_i$ be the subgraph of T_i which contains the unique tree path between u and v iff $\{u, v\} \in F$. By Theorem 22 we know that $\mathsf{E}_{T_i \sim D}[w_{T_i}(T_i')] \leq O(\log n \cdot w_G(H))$ and so there must be some j such that $w_{T_j}(T_j') \leq O(\log n \cdot w_G(F))$. Thus, we let $\pi_{G \to T}(F) := T_j'$. We argue that $\pi_{G \to T}$ requires the stated connectivity properties. In particular, notice that by construction we have that if u and v are connected in F then they will have some copy connected in $\pi_{G \to T}(F)$: if u and v are connected in F by path $(v_1, v_2, \ldots)$ then the path in T_j which connects the copy of v_l and the copy of v_{l+1} is contained in $\pi_{G \to T}(F)$ and the concatenation of these paths for all l connects the copies of u and v contained in T_j. Moreover, notice that $\pi_{G \to T}(F)$ satisfies the required cost preservation properties since $w_T(\pi_{G \to T}(F)) = w_{T_j}(T_j') \leq O(\log n \cdot w_G(F))$ by construction.

Lastly, we specify $\pi_{T \to G}(F')$. We let $\pi_{T \to G}(F')$ be the graph induced by $\{P_{uv} : \{u', v'\} \in F'\}$ where P_{uv} is an arbitrary shortest path in G between u and v and u' and v' are copies of u and v. We first verify the required connectivity preservation properties: if u' and v' are connected in F' by path $(v_1', v_2' \ldots)$ then we know that v_l and v_{l+1} will be connected in $\pi_{T \to G}(F')$ for every l by $P_{v_l v_{l+1}}$ where v_i' is some copy of v_i. Thus, u and v will be connected in $\pi_{T \to G}(F')$. We next verify the required cost-preservation properties. By Theorem 22 we have for every i that $w_{T_i}(e') \geq w_G(P_{uv})$ for each $e' = \{u', v'\} \in T_i$. Thus, $w_T(F') = \sum_{e' \in F'} w_T(e') \geq \sum_{\{u', v'\} \in F'} w_G(P_{uv}) \geq w_G(\pi_{T \to G}(F'))$ where we have again used u and v to stand for the $\phi^{-1}(u')$ and $\phi^{-1}(v')$ respectively. Lastly, we note that $\pi_{T \to G}(F')$ is trivially computable in deterministic poly-time. □

3.3 Online Covering Steiner

In this section we give a deterministic bicriteria algorithm for the online covering Steiner problem which is the same as online group Steiner tree but where we must connect at least r_i vertices from each group g_i to the root. The algorithm is bicriteria in the sense that it relaxes both the r_i-connectivity guarantee and the cost.

As mentioned in the introduction, this problem generalizes group Steiner tree. Moreover, it is also easy to see that any deterministic bicriteria algorithm for online covering Steiner also gives a poly-log-competitive deterministic (unicriteria) algorithm for online (non-group) Steiner tree. In particular, given an instance of Steiner tree on weighted graph $G = (V, E, w)$ with root r where we must connect terminals $A \subseteq V$ to r, it suffices to solve the covering Steiner problem where each vertex in A is in a singleton group with any constant bicriteria relaxation. This is because connecting any $c > 0$ fraction of each group to r will connect at least one vertex to r by the integrality of the number of connected vertices. Thus, our result generalizes the fact that deterministic poly-log approximations are known for online (non-group) Steiner tree [119]. However, we do note that our (deterministic) poly-log-approximate bicriteria online covering Steiner problem algorithm does not imply there is a (deterministic) poly-log-approximate online (non-partial) group Steiner tree algorithm (due to the nature of the bicriteria guarantee).

Offline Covering Steiner Problem: In the covering Steiner problem we are given a weighted

graph $G = (V, E, w)$ as well as pairwise disjoint groups $g_1, g_2, \ldots, g_k \subseteq V$, desired connected vertices $1 \leq r_i \leq |g_i|$ for each group g_i and root $r \in V$. Our goal is to find a tree T rooted at r which is a subgraph of G and satisfies $|T \cap g_i| \geq r_i$ for every i. We wish to minimize our cost, $w(T) := \sum_{e \in E(T)} w(e)$.[3]

Online Covering Steiner Problem: The online covering Steiner problem is the same as offline covering Steiner problem but where our solution need not be a tree and groups are revealed in time steps $t = 1, 2, \ldots$. That is, in time step t an adversary reveals a new group g_t and the algorithm must maintain a solution T_t where: (1) $T_{t-1} \subseteq T_t$; (2) T_t is feasible for the (offline) covering Steiner tree problem on groups $g_1, \ldots g_t$ and; (3) T_t is cost-competitive with the optimal offline solution for this problem where the cost-competitive ratio of our algorithm is $\max_t w(T_t)/\mathrm{OPT}_t$ where OPT_t is the cost of the optimal offline covering Steiner problem solution on the first t groups. We will give a bicriteria approximation for online covering Steiner; thus we say that an online solution is ρ-connection-competitive if for each t we have $|T_t \cap g_i| \geq r_i \cdot \rho$ for every $i \leq t$.

3.3.1 Online Covering Steiner on a Tree

We begin by giving a bicriteria deterministic online algorithm for covering Steiner on trees based on a "water-filling" approach. Informally, in iteration t each unconnected vertex in each group will grow the solution towards the root at an equal rate until at least $r_i \cdot (1 - \epsilon)$ vertices in g_t are connected to r.

Problem

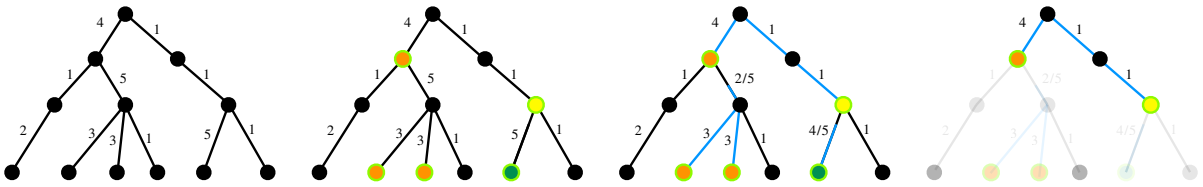
More formally we will solve a problem which is a slight generalization of covering Steiner on trees. We solve this problem on a tree rather than just covering Steiner on a tree because, unlike group Steiner tree, the "groupified" version of covering Steiner is not necessarily another instance of covering Steiner. Roughly, instead of groups we now have groups of groups, hence we call this problem 2-level covering Steiner.

Offline 2-Level Covering Steiner Problem: In the 2-level covering Steiner problem we are given a weighted graph $G = (V, E, w)$, root $r \in V$ and groups of groups $\mathcal{G}_1, \ldots \mathcal{G}_k$ where $\mathcal{G}_i$ consists of groups $\{g_1^{(i)}, \ldots g_{k_i}^{(i)}\}$ where each $g_j^{(i)} \subseteq V$. We are also given connectivity requirements $r_1, \ldots, r_k$. Our goal is to compute a minimum-weight tree T containing r where for each $i \leq k$ we have $|\{g_j^{(i)} : g_j^{(i)} \cap T \neq \emptyset\}| \geq r_i$. We let $n_i := |\{v : \exists j \text{ s.t. } v \in g_j^{(i)}\}|$. Notice that covering Steiner is just 2-level covering Steiner where each $g_i^{(j)}$ is a singleton set.

Online 2-Level Covering Steiner Problem: Online 2-level covering Steiner is the same as the offline problem but where $\mathcal{G}_t$ is revealed in time step t by an adversary. In particular, for each time step t we must maintain a solution T_t where: (1) $T_{t-1} \subseteq T_t$ for all t; (2) T_t is feasible for the (offline) 2-level covering Steiner problem on $\mathcal{G}_1, \ldots, \mathcal{G}_t$ with connectivity requirements $r_1, \ldots, r_t$ and; (3) T_t is cost-competitive with the optimal offline solution for this problem where the cost-competitive ratio of our algorithm is $\max_t w(T_t)/\mathrm{OPT}_t$ where OPT_t is the cost of the optimal offline 2-level covering Steiner problem solution on the first t groups of groups.

We will give a bicriteria approximation for online 2-level covering Steiner problem on trees; thus

[3] As with group Steiner tree the assumption that the tree is rooted and that the groups are pairwise disjoint is without loss of generality.



(a) Graph T. **(b)** $\mathcal{G}_1$ arrives. **(c)** "Fill water." **(d)** Choose solution.

Figure 3.5: Solution our algorithm gives after one group of groups, $\mathcal{G}_1$, is revealed where $r_1 = 2$. Nodes in groups in $\mathcal{G}_1$ outlined in green and nodes colored according to the group of $\mathcal{G}_1$ which contains them. Saturated edges given in blue and edges with $0 < x_e < w_e$ annotated with "x_e/w_e". All other edges labeled by w_e.

we say that an online solution is ρ-connection-competitive if for each t we have $|\{g_j^{(i)} : g_j^{(i)} \cap T \neq \emptyset\}| \geq \rho \cdot r_i$ for every $i \leq t$.

Algorithm

We now formally describe our algorithm for the 2-level covering Steiner problem on weighted tree $T = (V, E, w)$ given an $\epsilon > 0$. We will maintain a fractional variable $0 \leq x_e \leq w_e$ for each edge indicating the extent to which we buy e where our x_es will be monotonically increasing as our algorithm runs. Say that an edge e is saturated if $x_e = w_e$.

Let us describe how we update our solution in the tth time step. Let T_t be the connected component of all saturated edges containing r. Then, we repeat the following until $|\{g_j^{(t)} : g_j^{(t)} \cap T_t \neq \emptyset\}| \geq r_t \cdot (1 - \epsilon)$. Let $\mathcal{G}'_t := \{g_j^{(t)} \in \mathcal{G}_t : g_j^{(t)} \cap T_t = \emptyset\}$ be all groups in $\mathcal{G}_t$ not yet connected and let $g'_t := \bigcup_{S \in \mathcal{G}'_t} S$ be all vertices in a group which have not yet been connected to r. We say that e is on the frontier for $v \in g'_t$ if it is the first edge on the path from v to r which is not saturated. Similarly, let r_e be the number of vertices in g'_t for which e is on the frontier for v. Then, for each edge e we increase x_e by $r_e \cdot \delta$ where $\delta = \min_e (w_e - x_e)/r_e$. Our solution in the tth time step is T_t once $|\{g_j^{(t)} : g_j^{(t)} \cap T_t \neq \emptyset\}| \geq (1 - \epsilon) \cdot r_t$.

We illustrate one iteration of this algorithm in Figure 3.5

Analysis

We proceed to analyze the above algorithm and give its properties.

Theorem 23. *There is a deterministic poly-time algorithm for online 2-level covering Steiner on trees which is* $\frac{1}{\epsilon} \cdot (\max_i \frac{n_i}{r_i})$*-cost-competitive and* $(1 - \epsilon)$*-connection-competitive.*

Proof. We begin by verifying that our algorithm returns a monotonically increasing and $(1 - \epsilon)$-connection-competitive solution. First, notice that our solution is monotonically increasing since our x_es are monotonically increasing and our solution only includes saturated edges. To see that our solution is $(1 - \epsilon)$-connection-competitive notice that at least one new edge becomes saturated from each update to the x_es (namely $\arg\min_e (w_e - x_e)/r_e$) and since if all edges are saturated then $T_t = T$ which clearly satisfies $|\{g_j^{(t)} : g_j^{(t)} \cap T_t \neq \emptyset\}| \geq (1 - \epsilon) \cdot r_t$, this process will eventually halt with a $(1 - \epsilon)$-connection-competitive solution in the tth iteration. For the same reason our algorithm is deterministic poly-time.

It remains to argue that our solution is $\frac{1}{\epsilon} \cdot (\max_i \frac{n_i}{r_i})$-cost-competitive. We will argue that we can

uniquely charge each unit of increase of our x_es to an appropriate cost portion of the optimal solution. Fix an iteration t. Next, let $\delta^{(i,j)}$ for $i \leq t$ be the value of δ in the ith iteration the jth time we increase the value of our x_es. Similarly, let $\delta_x^{(i,j)}$ be the increase in $\sum_e x_e$ when we do so and let $\delta_y^{(i,j)}$ be the increase in $\sum_{e \in T_t^*} x_e$ where T_t^* is the optimal offline solution to the 2-level covering Steiner problem we must solve in the tth iteration. Lastly, let $y := \sum_{i \leq t} \sum_j \delta_y^{(i,j)}$ be the value of $\sum_{e \in T_t^*} x_e$ at the end of the tth iteration; clearly we have $y \leq \text{OPT}_t$. We claim that it suffices to show that for each $i \leq t$ and each j that $\delta_x^{(i,j)} \leq \frac{1}{\epsilon} \delta_y^{(i,j)} \frac{n_i}{r_i}$ since it would follow that at the end of iteration t we have that

$$w(T_t) \leq \sum_e x_e = \sum_{i \leq t} \sum_j \delta_x^{(i,j)} \leq \frac{1}{\epsilon} \sum_{i \leq t} \sum_j \frac{n_i}{r_i} \delta_y^{(i,j)} \leq \frac{1}{\epsilon} \left(\max_i \frac{n_i}{r_i} \right) y \leq \frac{1}{\epsilon} \left(\max_i \frac{n_i}{r_i} \right) \text{OPT}_t.$$

We proceed to show that $\delta_x^{(i,j)} \leq \frac{1}{\epsilon} \delta_y^{(i,j)} \frac{n_i}{r_i}$ for each $i \leq t$ and j. We fix an i and j and for cleanliness of notation we will drop the dependence on i and j in our δs henceforth.

First, notice that we have that

$$\delta_x \leq n_i \cdot \delta \tag{3.3.1}$$

since each vertex $v \in g_i$ is uniquely responsible for up to a δ increase on x_e where e is the edge on v's frontier.

On the other hand, notice that if a group in $\mathcal{G}_i$ is connected to r by T_t^* but is not yet connected by T_i then such a group uniquely contributes at least δ to δ_y. Since T_t^* connects at least r_i groups in $\mathcal{G}_i$ to r but at the moment of our increase T_i connects at most $(1-\epsilon) \cdot r_i$, there are at least $\epsilon \cdot r_i$ such groups in $\mathcal{G}_i$ which are connected to r by T_t^* but not by T_i. Thus, we have that

$$\delta_y \geq \epsilon \cdot r_i \cdot \delta \tag{3.3.2}$$

Combining Equations 3.3.1 and 3.3.2 shows $\delta_x \leq \frac{1}{\epsilon} \delta_y \frac{n_i}{r_i}$ as required. □

3.3.2 Online Covering Steiner on General Graphs

Next, we apply our first construction to give an algorithm for covering Steiner on general graphs. Crucially, the following result relies on a single copy tree embedding with poly-logarithmic copy number, making our second construction unsuitable for this problem.

Theorem 6. *There is a deterministic poly-time algorithm for online covering Steiner (on general graphs) which is* $O(\frac{\log^3 n}{\epsilon} \cdot \max_i \frac{|g_i|}{r_i})$*-cost-competitive and* $(1-\epsilon)$*-connection-competitive.*

Proof. We will use our copy tree embedding to produce a single tree on which we must deterministically solve online 2-level covering Steiner. We will then apply the algorithm from Theorem 23 to solve online 2-level covering Steiner on this tree.

More formally, consider an instance of online covering Steiner on weighted graph $G = (V, E, w)$ with root r. Then, we first compute a copy tree embedding $(T, \phi, \pi_{G \to T}, \pi_{T \to G})$ deterministically with respect to G and r as in Theorem 4 with cost approximation $O(\log^2 n)$ and copy number $O(\log n)$. Next, given our instance I_t of covering Steiner on G with groups $g_1, \ldots g_t$ and con-

nection requirements $r_1, \ldots, r_t$ we let I'_t be the instance of 2-level covering Steiner on T with groups of groups $\mathcal{G}_1, \ldots \mathcal{G}_t$ where $\mathcal{G}_i = \{\phi(v) : v \in g_i\}$, connection requirements $r_1, \ldots, r_t$ and root $\phi(r)$. Then if the adversary has required that we solve instance I_t in time step t, then we require that the algorithm in Theorem 23 solves I'_t in time step t and we let H'_t be the solution returned by our algorithm for I'_t. Lastly, we return as our solution for I_t in time step t the set $H_t := \pi_{T\to G}(H'_t)$.

Let us verify that the resulting algorithm is indeed feasible (i.e. monotone and $(1-\epsilon)$-connection-competitive) and of the appropriate cost.

First, we have that $H_t \subseteq H_{t+1}$ for every t since $H'_t \subseteq H'_{t+1}$ because our algorithm for trees returns a feasible solution for its online problem and $\pi_{T\to G}$ is monotone by definition of a copy tree embedding. Moreover, we claim that H_t connects at least $(1-\epsilon)\cdot r_i$ vertices from g_i to r for $i \le t$ and every t. To see this, notice that there at least $(1-\epsilon)\cdot r_i$ groups from $\mathcal{G}_i$ containing a vertex connected to r by H'_t. Since each such group consists of the copies of a distinct vertex, by the connectivity preservation properties of a copy tree it follows that H_t connects at least $(1-\epsilon)\cdot r_i$ vertices from g_i to r.

Next, we verify the cost of our solution. Let OPT'_t be the cost of the optimal solution to I'_t. Notice that since our copy number is $O(\log n)$, it follows that $n_i \le O(\log n \cdot |g_i|)$. Thus, by the guarantees of Theorem 23 we have

$$w_T(H'_t) \le \frac{1}{\epsilon}\cdot\left(\max_i \frac{n_i}{r_i}\right)\mathrm{OPT}'_t \le O\left(\frac{\log n}{\epsilon}\right)\cdot\left(\max_i \frac{|g_i|}{r_i}\right)\mathrm{OPT}'_t. \tag{3.3.3}$$

Next, we bound OPT'_t. Let H^*_t be the optimal solution to I_t. We claim that $\pi_{G\to T}(H^*_t)$ is feasible for I'_t. This follows because H^*_t connects at least r_i vertices from g_i to r for $i \le t$ and so by the connectivity preservation property of copy tree embeddings we know that there are at least r_i groups in $\mathcal{G}_i$ with a vertex connected to r by $\pi_{G\to T}(H^*_t)$. Thus, combining this with the $O(\log^2 n)$ cost preservation of our copy tree embedding we have

$$\mathrm{OPT}'_t \le w_T(\pi_{G\to T}(H^*_t)) \le O(\log^2 n)\cdot w_G(H^*_t). \tag{3.3.4}$$

Lastly, by the cost preservation property of our copy tree embedding we have that $w_G(H_t) \le w_T(H'_t)$ which when combined with Equations 3.3.3 and 3.3.4 gives

$$w_G(H_t) \le O\left(\frac{\log^3 n}{\epsilon}\cdot\max_i \frac{|g_i|}{r_i}\right)\cdot w_G(H^*_t).$$

thereby showing that our solution is within the required cost bound. □

Since group Steiner tree is exactly covering Steiner where $r_i = 1$ in which case $\max_i \frac{|g_i|}{r_i} \le N$ where again N is the maximum size of a group. Moreover, since any solution can only connect an integral number of vertices from each group, it follows that a $\frac{1}{2}$-connection-competitive solution for covering Steiner where $r_i = 1$ (i.e. for group Steiner tree) connects at least one vertex from each group. Thus, as a corollary of the above result we have the following deterministic algorithm for online group Steiner tree.[4]

[4]We note that one can use an aforementioned property of our first construction—that if u is connected to r by $F \subseteq E$ then every vertex in $\phi(u)$ is connected to $\phi(r)$ in $\pi_{G\to T}(F)$—to reduce the $O(\log^3 n)$s in this section to

Corollary 24. *There is an $O(N \log^3 n)$-competitive deterministic algorithm for online group Steiner tree where $N := \max_i |g_i|$ is the maximum group size.*

3.4 Deterministic Online Group Steiner Reductions

In this section we prove that the guarantees of our copy tree embeddings are sufficient to generalize any deterministic algorithm for online group Steiner tree on trees to general graphs, thereby reducing an open question posed by Alon et al. [8] to its tree case. We show that a similar result holds for the online group Steiner forest problem which generalizes online group Steiner tree; recall that these problems are defined in Section 2.0.1.

Theorem 25. *If there exists an α-competitive poly-time deterministic algorithm for group Steiner tree (resp. group Steiner forest) on well-separated trees then there exists an $O(\log n \cdot \alpha)$-competitive poly-time deterministic algorithm for group Steiner tree (resp. group Steiner forest) on general graphs.*

In general, mapping an instance of a problem P onto an equivalent instance I' on the copy tree embedding often results that I' is not an instance of the same problem P. However, group Steiner tree (resp., forest) problems have the notable property that mapping them onto a copy tree embedding simply results in another instance of the group Steiner tree (resp., forest) problem, this time on a tree. This property, albeit somewhat hidden in the proof, is the main reason why copy tree embeddings are well suited for these two problems.

Because past work on group Steiner and group Steiner forest have stated runtimes and approximation guarantees as functions of the maximum group size and number of groups rather than just n—see e.g. [27, 92]—we will give our results in the same generality with respect to these parameters.

3.4.1 Deterministic Online Group Steiner Tree

We begin with our results for online group Steiner tree.

Theorem 26. *If there exists:*

1. *A poly-time deterministic algorithm to compute an efficient, well-separated α-approximate copy tree embedding with copy number χ and;*
2. *A poly-time $f(n, N, k)$-competitive deterministic algorithm for online group Steiner tree on well-separated trees*

then there exists an $(\alpha \cdot f(\chi n, \chi N, k))$-competitive deterministic algorithm for group Steiner tree (on general graphs).

Proof. We will use our copy tree embedding to produce a single tree on which we must solve deterministic online group Steiner tree.

$O(\log^2 n)$s. In particular, if one were to use this property then when we map the solution to our covering Steiner problem on G to our copy tree embedding, the resulting solution will connect at least r_i groups in $\mathcal{G}_i$ at least $\Theta(\log n)$ times. It follows that when we run our water filling algorithm each time it increases $\sum_e x_e$ by 1 we know that it cover at least $\Omega(\log n)$ units of the optimal solution by weight rather than 1 unit of the optimal solution as in the current analysis.

In particular, consider an instance of online group Steiner tree on weighted graph $G = (V, E, w)$ with root r. Then, we first compute a copy tree embedding $(T, \phi, \pi_{G\to T}, \pi_{T\to G})$ deterministically with respect to G and r as we assumed is possible by assumption. Next, given an instance I_t of group Steiner tree on G with groups $g_1, \dots g_t$, we let I'_t be the instance of group Steiner tree on T with groups $\phi(g_1), \dots \phi(g_t)$ and root $r' := \phi(r)$ where we have used the notation $\phi(g_i) := \bigcup_{v\in g_i} \phi(v)$. Then, if the adversary has required that we solve instance I_t in time step t, then we require that our deterministic algorithm for online group Steiner tree on trees solves I'_t in time step t and we let H'_t be the solution returned by our algorithm for I'_t. Lastly, we return as our solution for I_t in time step t the set $H_t := \pi_{T\to G}(H'_t)$.

Let us verify that the resulting algorithm is indeed feasible and of the appropriate cost.

First, we have that $H_t \subseteq H_{t+1}$ for every t since $H'_t \subseteq H'_{t+1}$ because our algorithm for trees returns a feasible solution for its online problem and $\pi_{T\to G}$ is monotone by definition of a copy tree embedding. Moreover, we claim that H_t connects at least one vertex from each g_i to r for $i \le t$ and every t. To see this, notice that H'_t connects at least one vertex from $\phi(g_t)$ to $r' = \phi(r)$ in t since it is a feasible solution for I'_t and so at least one copy of a vertex in g_t; by the connectivity preservation properties of a copy tree it follows that at least one vertex from g_t is connected to r. Thus, our solution is indeed feasible in each time step.

Next, we verify the cost of our solution. Let OPT'_t be the cost of the optimal solution to I'_t and let n' and N' be the number of vertices and maximum size of a group in I'_t for any t. By our assumption on the cost of the algorithm we run on T and since $n' \le \chi n$ and $N' \le \chi N$ by definition of copy number, we know that

$$w_T(H'_t) \le \mathrm{OPT}'_t \cdot f(n', N', k) = \mathrm{OPT}'_t \cdot f(\chi n, \chi N, k).$$

Next, let H^*_t be the optimal solution to I_t. We claim that $\pi_{G\to T}(H^*_t)$ is feasible for I'_t. This follows because H^*_t connects a vertex from $g_1, \dots, g_t$ to r and so by the connectivity preservation property of copy tree embeddings we know that some vertex from each of $\phi(g_1), \dots, \phi(g_t)$ is connected to $r' = \phi(r)$. Applying this feasibility of $\pi_{G\to T}(H^*_t)$ and the cost preservation property of our copy tree embedding, it follows that $\mathrm{OPT}'_t \le w_T(\pi_{G\to T}(H^*_t)) \le \alpha \cdot w_G(H^*_t) = \alpha \cdot \mathrm{OPT}_t$.

Similarly, we know by the cost preservation property of our copy tree embedding that $w_G(\pi_{T\to G}(H'_t)) \le w_T(H'_t)$. Combining these observations we have

$$w_G(\pi_{T\to G}(H'_t)) \le w_T(H'_t) \le \mathrm{OPT}'_t \cdot f(\chi n, \chi N, k) \le \mathrm{OPT}_t \cdot \alpha \cdot f(\chi n, \chi N, k),$$

thereby showing that our solution is within the required cost bound. □

Plugging in our first construction (Theorem 4) or our second construction (Theorem 5) of a copy tree embedding immediately gives the follow corollary.

Corollary 27. *If there is an $f(n, N, k)$-competitive deterministic algorithm for online group Steiner tree on well-separated trees then there are $O(\log n \cdot f(O(n^2 \log n), O(nN), k))$ and $O(\log^2 n \cdot f(O(n \log n), O(N \log n), k))$-competitive deterministic algorithms for online group Steiner tree (on general graphs).*

3.4.2 Deterministic Online Group Steiner Forest

In this section we show a black-box reduction from the poly-log-approximate online deterministic group Steiner forest in a general graph G to poly-log-approximate online deterministic group Steiner forest when the underlying graph is a tree. A formal definition of the problem follows.

We now show that a deterministic algorithm for online group Steiner forest on trees gives a deterministic algorithm for online group Steiner forest on general graphs up to small losses. These results and the corresponding proofs will be quite similar to those of the previous section so we defer a full proof to the appendix.

Theorem 28. *If there exists:*

1. *A poly-time deterministic algorithm to compute an efficient, well-separated α-approximate copy tree embedding with copy number χ and;*
2. *A poly-time $f(n, N, k)$-competitive deterministic algorithm for online group Steiner forest on well-separated trees*

then there exists an $(\alpha \cdot f(\chi n, \chi N, k))$-competitive deterministic algorithm for group Steiner forest (on general graphs).

Proof. We will use our copy tree embedding to produce a single tree on which we must solve deterministic online group Steiner forest.

In particular, consider an instance of online group Steiner forest on weighted weighted $G = (V, E, w)$. Then, we first compute a copy tree embedding $(T, \phi, \pi_{G\to T}, \pi_{T\to G})$ deterministically with respect to G and an arbitrary root $r \in V$ as we assumed is possible by assumption. Next, given an instance I_t of group Steiner forest on G with pairs $(S_1, T_1), \dots (S_t, T_t)$, we let I'_t be the instance of group Steiner forest on T with pairs $(\phi(S_1), \phi(T_1)), \dots (\phi(S_t), \phi(T_t))$ where we have used the notation $\phi(W) := \bigcup_{v\in W} \phi(v)$ for $W \subseteq V$. Then if the adversary has required that we solve instance I_t in time step t, then we require that our deterministic algorithm for online group Steiner forest on trees solves I'_t in time step and we let H'_t be the solution returned by our algorithm for I'_t. Lastly, we return as our solution for I_t in time step t the set $H_t := \pi_{T\to G}(H'_t)$.

Let us verify that the resulting algorithm is indeed feasible and of the appropriate cost.

First, we have that $H_t \subseteq H_{t+1}$ for every t since $H'_t \subseteq H'_{t+1}$ because our algorithm for trees returns a feasible solution for its online problem and $\pi_{T\to G}$ is monotone by definition of a copy tree embedding. Moreover, we claim that H_t connects at least one vertex from S_i to at least one vertex from T_i for $i \le t$ and every t. To see this, notice that H'_t connects at least one vertex from $\phi(S_i)$ to some vertex in $\phi(T_i)$ since it is a feasible solution for I'_t and so at least one copy of a vertex in $\phi(S_i)$ is connected to at least one copy of a vertex in $\phi(T_i)$; by the connectivity preservation properties of a copy tree it follows that at least one vertex from S_i is connected to at least one vertex from T_i. Thus, our solution is indeed feasible in each time step.

Next, we verify the cost of our solution. Let OPT'_t be the cost of the optimal solution to I'_t, let n' be the number of vertices in T and let N' be the maximum size of a set in a pair in I'_t for any t. By our assumption on the cost of the algorithm we run on T and since $n' \le \chi n$ and $N' \le \chi N$ by definition of copy number, we know that

$$w_T(H'_t) \le \mathrm{OPT}'_t \cdot f(n', N', k) = \mathrm{OPT}'_t \cdot f(\chi n, \chi N, k).$$

Next, let H_t^* be the optimal solution to I_t. We claim that $\pi_{G\to T}(H_t^*)$ is feasible for I'_t. This follows because H_t^* connects a vertex from S_i to T_i for every $i \le t$ and so by the connectivity preservation property of copy tree embeddings we know that some vertex from $\phi(S_i)$ is connected to some vertex of $\phi(T_i)$ for every $i \le t$ in $\pi_{G\to T}(H_t^*)$. Applying this feasibility of $\pi_{G\to T}(H_t^*)$ and the cost preservation property of our copy tree embedding, it follows that $\mathrm{OPT}'_t \le w_T(\pi_{G\to T}(H_t^*)) \le \alpha \cdot w_G(H_t^*) = \alpha \cdot \mathrm{OPT}_t$.

Similarly, we know by the cost preservation property of our copy tree embedding that $w_G(\pi_{T\to G}(H'_t)) \le w_T(H'_t)$. Combining these observations we have

$$w_G(\pi_{T\to G}(H'_t)) \le w_T(H'_t) \le \mathrm{OPT}'_t \cdot f(\chi n, \chi N, k) \le \mathrm{OPT}_t \cdot \alpha \cdot f(\chi n, \chi N, k),$$

thereby showing that our solution is within the required cost bound. □

Plugging in our first construction (Theorem 4) or our second construction (Theorem 5) of a copy tree embedding immediately gives the follow corollary.

Corollary 29. *If there is an $f(n, N, k)$-competitive deterministic algorithm for online group Steiner forest on well-separated trees then there are $O(\log n \cdot f(O(n^2 \log n), O(nN), k))$ and $O(\log^2 n \cdot f(O(n \log n), O(N \log n), k))$-competitive deterministic algorithms for online group Steiner forest (on general graphs).*

Lastly, we note that Theorem 25 follows immediately from Theorem 27 and Theorem 29.

3.5 Conclusion and Future Work

Online and dynamic algorithms built on probabilistic tree embeddings seem inherently randomized and necessarily not robust to adaptive adversaries. In this chapter we gave an alternative to probabilistic tree embeddings—the copy tree embedding—which is better suited to deterministic and adaptive-adversary-robust algorithms. We illustrated this by giving new results in online algorithms, including a reduction of deterministic online group Steiner tree and group Steiner forest to their tree cases and a bicriteria deterministic algorithm for online covering Steiner.

We conclude by providing some directions for such future works.

As mentioned earlier, Bienkowski et al. [31] recently gave a deterministic algorithm for online non-metric facility location—which is equivalent to online group Steiner tree on trees of depth 2—with a poly-log-competitive ratio and stated that they expect their techniques will extend to online group Steiner tree on trees. A very exciting direction for future work would thus be to extend these techniques to general depth trees which, when combined with our reduction to the tree case, would prove the existence of a deterministic poly-log-competitive algorithm for online group Steiner tree, settling the open question of Alon et al. [8].

While our focus has been on two specific constructions, it would be interesting to prove lower bounds on copy tree embedding parameters, such as, more rigorously characterizing the tradeoffs between the number of copies and the cost approximation factor. One should also consider the possibility of improved constructions. For example: Is it possible to get a logarithmic approximation with few copies, maybe even a constant number of copies? It is easy to see that with an exponential number of copies—one for each possible subgraph—a perfect cost approximation factor of one is possible. Can one show that a sub-logarithmic distortion is

impossible with a polynomial number of copies? We currently do not even have a proof that excludes a constant cost approximation factor with a constant copy number.

Chapter 4

Hop-Constrained Tree Embeddings

4.1 Introduction

When doing network design, connectivity alone is often not sufficient for fast and reliable networks. Indeed, we often also desire that our networks be hop-constrained; namely we desire that demands are not just appropriately connected but connected with a path consisting of a low number of edges (a.k.a. hops). By reducing the number of traversed edges, hop constraints facilitate fast communication [6, 61]. Furthermore, low-hop networks tend to also be more reliable: if a transmission over an edge fails with some probability, the greater the number of hops between the source and destination, the greater the probability that this transmission fails [158, 167].

Unfortunately, adding hop constraints to network design problems makes them significantly harder. MST is solvable in polynomial time but MST with hop constraints is known to admit no $o(\log n)$ poly-time approximation algorithm [21]. Similarly, Steiner forest has a constant approximation [5] but hop-constrained Steiner forest has no poly-time $o(2^{\log^{1-\epsilon} n})$-approximation for any constant $\epsilon > 0$ [68].[1] Indeed, although there has been extensive work on approximation algorithms for simple connectivity problems like spanning tree and Steiner tree with hop constraints [9, 116, 121, 126, 130, 132, 142, 155], nothing is known regarding algorithms for many well-studied generalizations of these problems with hop constraints. For instance, no non-trivial approximation algorithms are known for Steiner forest, group Steiner tree, group Steiner forest or online Steiner tree with hop constraints.

By allowing an algorithm to "pretend" that the input graph is a tree, probabilistic tree embeddings have had enormous success as the foundation of many poly-log approximation algorithms for network design; thus, we might naturally expect them to be useful for hop-constrained network design. In the h-hop-constrained setting for some $h \geq 1$, the natural notion of distance to consider between vertices u and v is the h-hop-constrained distance—the length of the shortest path between u and v according to w with at most h hops. Thus, to use tree embeddings for hop-constrained network design we must first understand how to approximate these distances with trees.

[1] Under standard complexity assumptions.

4.1.1 Our Contributions

In this chapter we initiate the study of metric approximations for hop-constrained distances and their use in algorithms for hop-constrained network design. Broadly, our results fall into four categories.

Impossibility of Approximating Hop-Constrained Distances with Metrics

We begin by observing that hop-constrained distances are inapproximable by metrics (Section 4.3.1).

Results: Not only are hop-constrained distances not a metric (since they do not satisfy the triangle inequality) but given a hop constraint there are weighted graphs where any metric that approximates hop-constrained distances does so with an $\Omega(L)$ multiplicative error where L is the aspect ratio (Theorem 32). This lower bound is matched by a trivial upper bound (Theorem 33).

Discussion: Since the expected distance between two nodes in a distribution over metrics is itself a metric, our impossibility result also rules out approximating hop-constrained distances with distributions over metrics as in FRT.

Techniques: This observation is proved by careful analysis of a simple example: a path graph.

Approximating Hop-Constrained Distances with Partial Tree Metrics

Despite these apparent roadblocks, we show that—somewhat surprisingly—it is indeed possible to to approximate hop-constrained distances with trees (Sections 4.3.2, 4.3.3).

Results: We show that a distribution over "partial tree metrics" can approximate hop-constrained distances with an expected distance stretch of $O(\log n \log\log n)$ and a worst-case distance stretch of $O(\log^2 n)$ with an $O(\log^2 n)$ relaxation in the hop constraint (Theorem 37).

Discussion: This result differs from FRT in two notable ways: (1) our partial tree metrics are partial in the sense that they contain only a constant fraction of nodes from the input graph—indeed, this is what allows us to overcome the impossibility of approximating hop-constrained distances with metrics; (2) our result provides a worst-case guarantee, unlike FRT which only gives a guarantee in expectation.

Techniques: We show this result by first proving a decomposition lemma (Theorem 40), which applies padded decompositions to a "mixture metric" that combines hops and (unconstrained) distances. We then recursively apply this decomposition, using different combinations of hops and distances in our recursive calls.

h-Hop Partial Tree Embeddings

We next build embeddings for hop-constrained network design from our metric approximations (Sections 4.4, 4.6).

Results: Specifically, we show that one can construct a distribution over "h-hop partial tree embeddings" of hop-constrained distances with expected distance stretch $O(\log n \log\log n)$ and a worst-case distance stretch $O(\log^2 n)$ with an $O(\log^3 n)$ relaxation in the hop constraint (Theorem 46). Further, we show that these embeddings can be used for hop-constrained network design as in the above template for network design that uses FRT. Notably, our embeddings reduce many

hop-constrained network design problems to their *non-hop-constrained* versions on trees. Since our embeddings, like our partial tree metrics, are also partial, we build on these embeddings by constructing "h-hop copy tree embeddings," which represent many draws from our distribution over partial tree embeddings as a single tree.[2]

Discussion: Like our tree metrics and unlike FRT, our tree embeddings are partial and give worst-case guarantees. Moreover, our embeddings follow almost immediately from our metric approximations. However, a notable difference between our embeddings and those of FRT is that demonstrating that they can be used for hop-constrained network design requires a non-trivial amount of work. In particular, while appropriately projecting from an input graph to a tree embedding is trivial in the FRT case, the partialness of our embeddings makes this projection significantly more troublesome. Thus, we develop a projection theorem (Theorem 48), which informally shows that a natural projection from G to one of our tree embeddings appropriately preserves cost and connectivity.

Techniques: We prove our projection theorem using "h-hop-connectors" which are, informally, a hop-constrained version of Euler tours. We emphasize that this projection theorem is only used in the analysis of our algorithms.

Applications to Hop-Constrained Network Design

Lastly, we use our embeddings to develop the first non-trivial approximation algorithms for the hop-constrained versions of many classic network design problems (Sections 4.5, 4.7).

Results: As detailed in Table 4.1, we give numerous (poly-log, poly-log) bicriteria algorithms for hop-constrained network design problems that relax both the cost and hop constraint of the solution.

Discussion: As noted above, bicriterianess is necessary for any poly-log approximation for Steiner forest and its generalizations. Furthermore, while the results in Table 4.1 are stated in utmost generality, many special cases of our results were to our knowledge not previously known. For example, our algorithm for hop-constrained oblivious Steiner forest immediately gives new algorithms for hop-constrained Steiner forest, hop-constrained online Steiner tree and hop-constrained online Steiner forest, as well as min-cost h-spanner (see Section 4.5.1 for details). Similarly, our algorithm for oblivious network design immediately gives new algorithms for the hop-constrained version of the well-studied buy-at-bulk network design problem [15].

Techniques: All of our algorithms for these problems use the above mentioned tree embedding template with either our h-hop partial tree embeddings or our h-hop copy tree embeddings.

Before proceeding we note that generally, weighted graphs in this chapter are assumed to be complete, i.e., $E = \binom{V}{2}$. In the context of this chapter this is without loss of generality. In particular, one can transform any non-complete weighted graph $G = (V, E, w)$ with aspect ratio L into an equivalent complete graph G' with aspect ratio $L' = n^2 \cdot L$ which gives a weight of L' to any edge not in E without affecting any of the results in this chapter.

[2] In a previous version of this work "h-hop copy tree embeddings" were called "h-hop repetition tree embeddings."

Hop-Constrained Problem	Cost Apx.	Hop Apx.	Cost In E	Section
Offline Problems				
Relaxed k-Steiner Tree	$O(\log^2 n)$	$O(\log^3 n)$		4.5.3
k-Steiner Tree	$O(\log^3 n)$	$O(\log^3 n)$		4.5.3
Group Steiner Tree	$O(\log^5 n)$	$O(\log^3 n)$		4.5.2, 4.7.1
Group Steiner Forest	$O(\log^7 n)$	$O(\log^3 n)$		4.7.3
Online Problems				
Group Steiner Tree	$O(\log^6 n)$	$O(\log^3 n)$	✓	4.7.2
Group Steiner Forest	$O(\log^8 n)$	$O(\log^3 n)$	✓	4.7.4
Oblivious Problems				
Steiner Forest	$O(\log^3 n)$	$O(\log^3 n)$		4.5.1
Network Design	$O(\log^4 n)$	$O(\log^3 n)$		4.5.4

Table 4.1: Our bicriteria approximation results. All results are for poly-time algorithms that succeed with high probability (at least $1 - \frac{1}{\mathsf{poly}(n)}$). For some of the problems we assume certain parameters are $\mathsf{poly}(n)$ to simplify presentation; see the relevant sections for more details. All results are new except for the k-Steiner tree result which is implied by [126].

4.2 Hop-Constrained Network Design Related Work

Before proceeding we give a brief overview of additional work on approximation algorithms for hop-constrained network design.

For some simple hop-constrained network design problems non-trivial (unicriteria) approximation algorithms are known. [9] gave an $O(\log n)$ approximation for minimum depth spanning tree on metrics. [130] gave a $O(\sqrt{\log n})$ for the degree-bounded minimum diameter spanning tree problem. [132] gave a $O(d \log n)$ approximation for computing a minimum cost Steiner tree with depth at most d. [121] gave a constant approximation for the minimum depth Steiner tree problem on a metric.

However, hop constraints often make otherwise easy problems so challenging that the only non-trivial approximation algorithms known or possible are bicriteria. The apparent necessity of bicriterianess in hop-constrained optimization is highlighted by the existence of many bicriteria algorithms. For example, [155] and [142] gave an $(O(\log n), O(\log n))$ bicriteria approximation algorithms for MST and Steiner tree with hop constraints.[3] Similarly, [116] gave a $(O(\log^4 n), O(\log^2 n))$-bicriteria algorithm for k-Steiner tree with hop constraints which was later improved to $(O(\log^2 n), O(\log n))$ by [126]; here the first term is the approximation in the cost while the second term is the approximation in the hop constraint.

Lastly, hop-constrained network design has also received considerable attention from the operations research community; see, for example, [6, 33, 35, 36, 61, 64, 65, 96, 97, 98, 99, 138, 158, 163, 165, 167] among many other papers.

[3] A later paper of [148] claimed to improve this result to a $(O(\log n), 2)$-approximation but it is our understanding that this paper was retracted due to a bug.

4.3 Approximating Hop-Constrained Distances

In this section we show that even though hop-constrained distances are not well-approximated by any metric, they are approximated by a distribution over what we call partial tree metrics. More specifically, we consider hop-constrained distances defined as follows.

Definition 30 (Hop-Constrained Distances)**.** *For a (complete) weighted graph $G = (V, E, w)$ and a hop constraint $h \geq 1$ we define the h-hop distance between any two nodes $u, v \in V$ as*

$$d_G^{(h)}(u, v) := \min\{w(P) \mid \textit{path } P \textit{ in } G \textit{ between } u, v \textit{ with } \mathrm{hop}(P) \leq h\}.$$

As we have assumed that our graph G is complete without loss of generality, the above is always well-defined for any $u, v \in V$. We note that all omitted proofs in this section appear in Section 4.9.

4.3.1 Hop-Constrained Distances Are Inapproximable by Metrics

We begin by observing that, not only is $d_G^{(h)}$ not a metric, but it is, in general, innaproximable by any metric.

It is easy to verify that $d_G^{(h)}$ is a valid distance function on $V(G)$. Indeed $d_G^{(h)}$ is clearly symmetric, i.e., $d^{(h)}(u, v) = d^{(h)}(v, u)$, and satisfies the identity of indiscernibles, i.e., $d^{(h)}(u, v) = 0 \Leftrightarrow u = v$. However, it is also simple to see that hop-constrained distances are not necessarily metrics since they do not obey the triangle inequality. Indeed, the existence of a short h-hop path from u to v and a short h-hop path from v to w does not imply that the existence of a short h-hop path between u and w. More formally it is possible that $d^{(h)}(u, w) \gg d^{(h)}(u, v) + d^{(h)}(v, w)$. See [11] for a similar observation.

Of course with a factor 2 relaxation in the hop constraint the relaxed triangle inequality $d^{(2h)}(u, w) \leq d^{(h)}(u, v) + d^{(h)}(v, w)$ holds for any graph G and any $u, v, w \in V(G)$. This suggests—albeit incorrectly—that one might be able to approximate hop-constrained distance by allowing constant slack in the hop constraint and length approximation as in the following definition.[4]

Definition 31. *A distance function $\tilde{d}$ approximates the h-hop constrained distances $d_G^{(h)}$ for a weighted graph $G = (V, E, w)$ where $h \geq 1$ with **distance stretch** $\alpha \geq 1$ and **hop stretch** $\beta \geq 1$ if for all $u, v \in V$ we have*

$$d_G^{(\beta h)}(u, v) \leq \tilde{d}(u, v) \leq \alpha \cdot d_G^{(h)}(u, v).$$

As we next observe, no metric provides such an approximation without a very large hop or distance stretch.

[4] In addition to naturally arising when trying to get hop-constrained distances to satisfy the triangle inequality, relaxing hop distances is further motivated by the following. As we later show, a relaxation in the hop constraint in a hop-constrained tree embedding propagates through to an approximation on the hop diameter of the solutions we find for our network design problems. Since, as mentioned above, some amount of approximation on the hop diameter is necessary for many of our problems to admit a poly-log approximation in the cost, relaxing our hop constraints in our notion of approximating hop-constrained distances is a natural way we can set ourselves up for success when aiming for poly-log cost approximations for our problems.

Lemma 32. *For any hop constraint $h \geq 1$, distance stretch α, hop stretch β and any $L > 1$, there exists a graph $G = (V, E, w)$ with aspect ratio L such that if a metric $\tilde{d}$ approximates $d_G^{(h)}$ with distance stretch α and hop stretch β then $\alpha(\beta h + 1) \geq L$.*

Indeed, an approximation with the above large stretch is always trivially attainable. In particular, no metric can approximate $d^{(h)}$ any better than the trivial approximation by the scaled shortest-path metric $\alpha \cdot d_G$ which gives value $\alpha \cdot d_G(u, v)$ to each $u, v \in V$, as shown by the following.

Lemma 33. *Given any graph $G = (V, E, w)$ with aspect ratio L and a distance stretch α and hop stretch β satisfying $\alpha(\beta h + 1) \geq L$, we have that $\alpha \cdot d_G$ approximates $d_G^{(h)}$ with distance stretch α and hop stretch β.*

Thus, hop-constrained distances can be maximally far from *any* metric in the sense that the only way to approximate them by a metric requires so much slack in the hop and distance stretch that the approximation becomes trivial. Moreover, since the expected distance between two nodes in a distribution over metrics is itself a metric, the above result also rules out approximating $d^{(h)}$ in a non-trivial way with distributions over metrics as in FRT. This impossibility remains even when one allows for relaxations of the hop constraint.

4.3.2 Distances Induced by Distributions Over Partial Metrics

While Theorem 32 shows that no metric can approximate $d_G^{(h)}$ on all vertices, it does not rule out the possibility that some metric approximates $d_G^{(h)}$ on a large subset of V. Thus, we introduce the following concept of partial metrics.

Definition 34 (Partial Metric)**.** *Any metric d defined on a set V_d is called a **partial metric** on V if $V_d \subseteq V$.*

We will often talk about how partial metric d approximates $d_G^{(h)}$ on V_d with hop and distance stretches α and β by which we mean that the inequality of Theorem 31—$d_G^{(\beta h)}(u, v) \leq d(u, v) \leq \alpha \cdot d_G^{(h)}(u, v)$—holds for every $u, v \in V_d$. Of course, a partial metric on the empty set trivially approximates $d_G^{(h)}$ and we are ultimately interested in estimating $d^{(h)}$ on all pairs of nodes. For this reason, we give the following notions of exclusion probability and how a distribution over partial metrics can induce a distance function between all nodes.

Definition 35 (Distances of Partial Metric Distributions)**.** *Let $\mathcal{D}$ be a distribution of partial metrics of V for weighted graph $G = (V, e, w)$. We say $\mathcal{D}$ has **exclusion probability** ϵ if for all $v \in V$ we have $\Pr_{d \sim \mathcal{D}}[v \in V_d] \geq 1 - \epsilon$. If $\epsilon \leq \frac{1}{3}$ then we say that $\mathcal{D}$ induces the distance function $d_{\mathcal{D}}$ on V, defined as*

$$d_{\mathcal{D}}(u, v) := \mathsf{E}_{d \sim \mathcal{D}} \left[d(u, v) \cdot \mathbb{I}[u, v \in V_d] \right].$$

It is easy to verify that $d_{\mathcal{D}}$ is indeed a distance function. In particular, we trivially have that $d_{\mathcal{D}}(v, v) = 0$ since $d(v, v) = 0$ for all d in the support of $\mathcal{D}$. An exclusion probability bounded above by $\frac{1}{2}$ guarantees that $\Pr_{d \sim \mathcal{D}}[u, v \in V_d] > 0$ for any $u, v \in V$. This guarantees that $d_{\mathcal{D}}(u, v) > 0$ for $u \neq v$ which makes $d_{\mathcal{D}}$ a valid distance function.

Since we are treating the distance between u and v as 0 in trees which only contain one of u or v it may happen that $d_{\mathcal{D}}(u, v) < d(u, v)$ which may seem strange. However, provided ϵ is at most some fixed constant, the above notion of distance (up to constants) is equal to the arguably more

natural notion of distance $\mathsf{E}_{d\sim\mathcal{D}_{uv}}[d(u,v)]$ where $\mathcal{D}_{uv}$ is $\mathcal{D}$ conditioned on both u and v being in the drawn partial metric; for any $u, v \in V$ this distance is always at least $d(u,v)$. Thus, at the loss of constants the reader may think of $d_{\mathcal{D}}(u,v)$ as a conditional expected distance where we condition on u and v both being in the metric drawn from $\mathcal{D}$. We choose the above notion of distance as opposed to the conditional expectation version as it simplifies our exposition but we emphasize that since these two notions only differ by constants this choice does not impact any of our results.

With these definitions in place we can define what it means for a distribution of partial metrics to approximate hop-constrained distances.

Definition 36 (Stretch of Partial Metric Distribution)**.** *A distribution $\mathcal{D}$ of partial metrics on V with exclusion probability at most $\frac{1}{3}$ approximates $d^{(h)}$ on weighted graph $G = (V, E, w)$ for hop constraint $h \geq 1$ with **worst-case distance stretch** $\alpha_{WC} \geq 1$ and **hop stretch** $\beta \geq 1$ if each d in the support of $\mathcal{D}$ approximates $d_G^{(h)}$ on V_d with distance stretch α_{WC} and hop stretch β, i.e. for each d in the support of $\mathcal{D}$ and all $u, v \in V_d$ we have*

$$d_G^{(\beta h)}(u,v) \leq d(u,v) \leq \alpha \cdot d_G^{(h)}(u,v).$$

*Furthermore, $\mathcal{D}$ has **expected distance stretch** α_{E} if for all $u, v \in V$ we have*

$$d_{\mathcal{D}}(u,v) \leq \alpha_{\mathsf{E}} \cdot d_G^{(h)}(u,v).$$

4.3.3 Approximating Hop-Constrained Distances with Partial Tree Metrics

Even though h-hop distances are generally inapproximable by distributions over metrics, we now show that they are well-approximated by distributions over very simple partial metrics, namely well-separated partial tree metrics.

Theorem 37. *For any (complete) weighted graph G, any hop-constraint $h \geq 1$, and any $0 < \epsilon < \frac{1}{3}$ there is a distribution $\mathcal{D}$ over well-separated tree metrics each of which is a partial metric on $V(G)$ such that $\mathcal{D}$ has exclusion probability at most ϵ and approximates $d_G^{(h)}$ with expected distance stretch $\alpha_{\mathsf{E}} = O(\log n \cdot \log \frac{\log n}{\epsilon})$, worst-case distance stretch $\alpha_{\mathrm{WC}} = O(\frac{\log^2 n}{\epsilon})$ and hop stretch $\beta = O(\frac{\log^2 n}{\epsilon})$.*

The rest of Section 4.3.3 is dedicated to the proof of Theorem 37. In Section 4.3.3 we define simple "mixture metrics" and show how combining these metrics with padded decompositions leads to random decompositions with desirable properties regarding hop-constrained distances. In Section 4.3.3 we show how recursively refining these partitions gives a random partial tree metric which proves Theorem 37.

Mixture Metrics and Padded Decompositions for Hop-Constrained Distances

To better understand the structure of hop-constrained distances, we develop a decomposition lemma which gives structure both in terms of weights and hops. In particular, we call a collection of disjoint vertex sets $C_1 \sqcup C_2 \sqcup \ldots \sqcup C_k$ a **partial vertex partition**; $C_1 \sqcup C_2 \sqcup \ldots \sqcup C_k$ is a **complete vertex partition** if $\bigcup_i C_i = V$. In a nutshell, we decompose the vertices of a weighted graph G into a partial vertex partition where (1) both the hop diameter and weight diameter of all

C_i's is small, (2) C_i and C_j for $i \neq j$ are well-separated both in terms of hops and weight and (3) almost every vertex is in the partial vertex partition. Our decomposition combines two simple ingredient.

Our first ingredient is what we call the mixture metric which is obtained by mixing together hop lengths and weights in the following way.

Definition 38 (Mixture Metric)**.** *Given a weighted graph $G = (V, E, w)$, a hop scale $h > 0$, and a weight scale $b > 0$, we define a **mixture weight** $w' : E \to \mathbb{R}_{\geq 0}$ of an edge $e \in E$ as $w'(e) := 1/h + w(e)/b$. The shortest path metric induced by w' is called the **mixture metric** $d' : V \times V \to \mathbb{R}_{\geq 0}$.*

The utility of the mixture metric is given by three easy to verify facts: It is a metric and so is amenable to standard metric decomposition theorems; if $d'(u, v) \leq \alpha$ in the mixture metric with hop scale h and weight scale b, then $d^{(\alpha \cdot h)}(u, v) \leq \alpha \cdot b$; if $d'(u, v) > \alpha$, then $d^{(\alpha \cdot h/2)}(u, v) > \alpha \cdot b/2$.

Our second ingredient is the well-studied padded decomposition [4, 103]. Given a metric space (V, d) we denote the ball of radius $r \geq 0$ around $x \in V$ with $B_d(x, r) := \{y \in V \mid d(x, y) \leq r\}$. Next, let $C = C_1 \sqcup \ldots \sqcup C_k$ be a (partial or complete) vertex partition. Then, for a subset $U \subseteq V$, we say that U **is broken in** C if $|\{i \mid U \cap C_i \neq \emptyset\}| > 1$. We also denote this event by $U \not\subseteq C$ and its logical negation by $U \subseteq C$. With this notation, we define padded decompositions:

Definition 39 (Padded Decompositions)**.** *Let (V, d) be a metric space and let $\mathcal{C}$ be a distribution over complete vertex partitions. $\mathcal{C}$ is a $(\rho_{\text{pad}}, \Delta)$-padded decomposition if:*

1. ***Diameter:*** $\max_{u,v \in C_i} d(u, v) \leq \Delta$ *for each $C = C_1 \sqcup C_2 \sqcup \ldots \sqcup C_k$ in the support of $\mathcal{C}$ and $i \in [k]$.*
2. ***Paddedness:*** $\Pr_{C \sim \mathcal{C}}[B_d(v, r) \not\subseteq C] < \frac{r \cdot \rho_{\text{pad}}}{\Delta}$ *for each $v \in V$ and every $r > 0$.*

In other words, each part of a partition in $\mathcal{C}$ has diameter at most Δ and the probability of a node being within r from a node in a different part is at most $\frac{r\rho_{\text{pad}}}{\Delta}$. The value ρ_{pad} is known as the **padding parameter**.[5] Combining padded decompositions with our mixture metric and its properties as observed above gives our decomposition lemma.

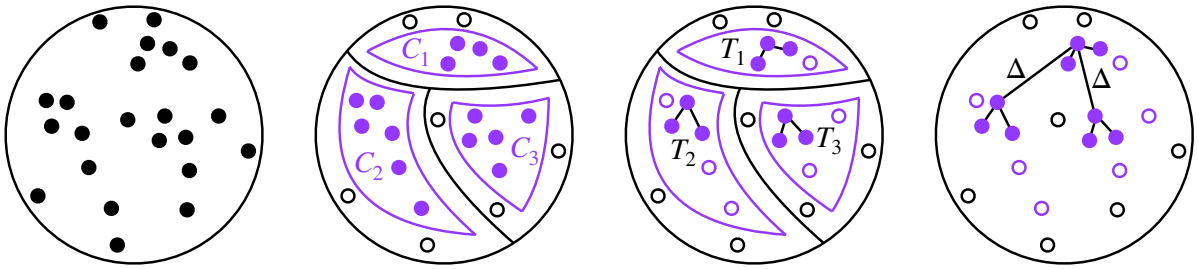
Lemma 40. *Let $G = (V, E, w)$ be a weighted graph with padding parameter ρ_{pad}. For any hop constraint $h > 0$, weight diameter $b > 0$, and exclusion probability $\gamma > 0$, there exists a distribution $\mathcal{C}$ over partial vertex partitions where for every $C = C_1 \sqcup \ldots \sqcup C_k$ in the support of $\mathcal{C}$:*

1. ***Hop-Constrained Diameter:*** $d_G^{(h)}(u, v) \leq b$ *for $i \in [k]$ and $u, v \in C_i$;*
2. ***Hop-Constrained Paddedness:*** $d_G^{(h\frac{\gamma}{2\rho_{\text{pad}}})}(u, v) \geq b \cdot \frac{\gamma}{2\rho_{\text{pad}}}$ *for every $u \in C_i$ and $v \in C_j$ where $i \neq j$.*

And:

3. ***Exclusion probability:*** $\Pr_{C \sim \mathcal{C}}[v \notin \bigcup_{i \in [k]} C_i] \leq \gamma$ *for each $v \in V$ where $C = C_1 \sqcup \ldots \sqcup C_k$;*
4. ***Path preservation:*** $\Pr_{C \sim \mathcal{C}}[V(P)$ *is broken in C*$] \leq (\text{hop}(P)/h + w(P)/b) \cdot \rho_{\text{pad}}$ *for each path P.*

[5] We note that our definition of ρ_{pad} slightly differs from that of other papers, albeit only by a constant factor.



(a) Graph G **(b)** Lem. 40 decomposition **(c)** Recursing **(d)** Merging recursions

Figure 4.1: An illustration of the top-level recursive call of the embedding of Theorem 37 on graph G (edges omitted from illustration). Vertices in the partial vertex partition of Theorem 40 given in purple. Vertices removed from the process given as empty circles and all other vertices given as filled-in circles.

Lastly, we note that it is known that every metric has padded decompositions with padding parameter $O(\log n)$ and so our decomposition lemma holds with $\rho_{\text{pad}} = O(\log n)$.

Lemma 41 ([4, 103])**.** *Every metric on n points admits a $(\rho_{\text{pad}}, \Delta)$-padded decomposition for $\rho_{\text{pad}} = O(\log n)$ and any $\Delta > 0$. Furthermore, such a decomposition can be computed in polynomial time.*

Constructing Tree Metrics for Hop-Constrained Distances and the Proof of Theorem 37

Next, we recursively apply the random partial vertex partitions of Theorem 40 to obtain a distribution over families of laminar subsets of nodes of G. This distribution will naturally correspond to a distribution over well-separated tree metrics which approximate h-hop constrained distances. In particular, a rough outline of our construction is as follows: we start with a large weight diameter $\Delta \leq \text{poly}(n)$ and hop constraint about h and compute the partial vertex partition $C_1 \sqcup \ldots \sqcup C_k \subseteq V(G)$ of Theorem 40. We remove from our process any vertices not in our partial vertex partition. We then recurse on each part C_i while keeping our hop constraint constant but shrinking Δ by a factor of 2. We combine the recursively constructed trees by hanging the roots of the returned trees off of the root of a fixed but arbitrary tree with edges of length Δ. The recursion stops when each C_i is a singleton. The resulting tree metric is partial since each application of Theorem 40 removes a small fraction of nodes. We illustrate our construction in Figure 4.1 and proceed to prove Theorem 37.

Proof of Theorem 37. We describe a recursive and randomized procedure that induces a distribution over well-separated rooted trees where each tree can be interpreted as a partial tree metric with the required properties. Given hop constraint h', weight diameter Δ and vertex set $V' \subseteq V$ where $d_G^{(h')}(u, v) \leq \Delta$, our procedure returns a rooted tree $(V(T), E(T), w_T)$ satisfying $V(T) \subseteq V'$. Let ρ_{pad} be the padding parameter of G; we will give our proofs in terms of ρ_{pad} and then conclude by applying Theorem 41. We fix $h' := h \cdot \kappa$ where we define $\kappa := O(\epsilon^{-1} \rho_{\text{pad}} \log n)$ throughout the procedure. We emphasize that h' will also be the same for all of our recursive calls. The construction procedure is initially invoked with the parameters $V' := V$ and weight scale Δ equal to the smallest power of 2 which is at least the aspect ratio $L \leq \text{poly}(n)$. That is, $\Delta \in [L, 2L) \geq \max_{u,v} d_G^{(h')}(u, v)$.

Construction procedure: We use the decomposition of Theorem 40 with hop constraint h', weight diameter $\Delta/2$, and exclusion probability $\gamma := \epsilon / O(\log n)$ (for a sufficiently large hidden

constant) to obtain a partial vertex partition $C_1 \sqcup C_2 \sqcup \ldots \sqcup C_k \subseteq V'$ where, plugging in our choice of parameters and the guarantees of Theorem 40, we have:

1. $\max_{u,v \in C_i} d_G^{(h')}(u,v) \leq \Delta/2$;
2. $d_G^{(h)}(C_i, C_j) = d_G^{(h'/\kappa)}(C_i, C_j) \geq \Delta/(2\kappa)$ for each $i, j \in [k]$ where $j \neq i$;
3. $\Pr[v \notin \bigcup_{i=1}^k C_i] \leq \frac{\epsilon}{O(\log n)}$ for all $v \in V$.

We recursively construct k rooted trees $T_1 = (V_1, E_1, w_1), \ldots, T_k = (V_k, E_k, w_k)$ by calling the same procedure with our distance scale set to $\Delta' \leftarrow \Delta/2$ on sets $C_1, \ldots, C_k$. We construct the tree $T = (V(T), E(T), w_T)$ returned by the procedure by connecting the roots of $T_2, \ldots, T_k$ to the root of T_1 via a tree edge of weight Δ. The procedure is stopped when the set of nodes V' is a singleton, at which point the trivial one-node tree is returned.

Exclusion probability analysis: Consider a recursive call with $v \in V'$ and suppose that the partial vertex partition in the call is $C_1 \sqcup \ldots \sqcup C_k$. By the properties of the partition, $\Pr[v \notin \bigcup_{i=1}^k C_i] \leq \epsilon/O(\log n)$ (for a sufficiently large constant). First, we note that $v \notin V(T)$ if and only if there is a recursive call where $v \in V' \setminus (\bigcup_i C_i)$, which happens with probability $\epsilon/O(\log n)$. Since v is in a unique recursive call on each level and there are $O(\log n)$ levels, we conclude via a union bound that this happens in at least one level with probability at most ϵ, proving that the exclusion probability of each node $v \in V$ is a most ϵ.

Worst-case distance stretch and hop stretch analysis: In the final tree T, for two nodes $u, v \in V(T)$ let $e_{u,v} := \arg\max\{w_T(e) \mid e \in T_{u,v}\}$ be the heaviest weight tree edge on the unique tree path between u and v. The weights w_T are strictly decreasing powers of 2 on any root-leaf path. Therefore, $w_T(e_{u,v}) \leq d_T(u,v) \leq O(w_T(e_{u,v}))$. Edge $e_{u,v}$ was created via a recursive call with the parameters V' and Δ where $V' \subseteq V$, $u, v \in V'$ and $d^{(h')}(u', v') \leq \Delta = w_T(e_{u,v})$ for all $u', v' \in V'$. Let $C_1 \sqcup \ldots \sqcup C_k$ be the partial vertex partition created by this recursive call where each C_i has weight diameter $\Delta/2$ and $d^{(h)}(C_i, C_j) \geq \Delta/(2\kappa)$ when $i \neq j$. Since $u, v \in V(T)$ we have that $u \in C_i$ and $v \in C_j$ for $i \neq j$ (since otherwise $e_{u,v}$ would not be created by this recursive call), hence $d_G^{(h)}(u,v) \geq \frac{\Delta}{2\kappa} = \frac{w_T(e_{u,v})}{2\kappa} = \Theta(\frac{d_T(u,v)}{\kappa})$. Consequently, $d_T(u,v) \leq O(\kappa \cdot d_G^{(h)}(u,v))$. Furthermore, since $u, v \in V'$ we have that $d^{(h')}(u,v) \leq \Delta \leq d_T(u,v)$, which can be rewritten as $d^{(\beta h)}(u,v) \leq d_T(u,v)$ for $\beta := O(\kappa)$. Combining the two bounds on d_T we have that both the worst-case distance stretch α_{WC} and hop stretch β are $O(\kappa) = O(\epsilon^{-1} \log n \cdot \rho_{\mathrm{pad}})$ which gives the desired bound when we plug in the $\rho_{\mathrm{pad}} = O(\log n)$ padded decomposition of Theorem 41.

Expected distance stretch analysis: Let Δ_l be the weight diameter of recursive calls at level $l \in [O(\log n)]$. In particular, $\Delta_1 \in (L, 2L]$ and $\Delta_{l+1} = \Delta_l/2$. Fix $u, v \in V$, let P be a path in G between u and v with at most h hops and weight $\delta := d_G^{(h)}(u,v)$ and let $e_{u,v}$ be defined—as in the worst-case distance stretch analysis—as the heaviest weight tree edge between u and v. As in the worst-case stretch analysis, it suffices to bound $w_T(e_{u,v})$. We now partition the $O(\log n)$ levels into three phases $H_1 \sqcup H_2 \sqcup H_3$ where $l \in H_1$ iff $\Delta_l > \delta \cdot (2\kappa)$, $l \in H_3$ iff $\Delta_l \leq \delta \cdot (2\rho_{\mathrm{pad}})$ and $l \in H_2$ in the remaining case where $\Delta_l \in (\delta \cdot 2\rho_{\mathrm{pad}}, \delta \cdot 2\kappa]$. We proceed to bound the probability that $e_{u,v}$ is created by a recursive call in H_1, H_2 and H_3 which, in turn, gives a bound on the expected distance between u and v.

We begin with calls at levels in H_1. In particular, we argue that a call at level $l \in H_1$ cannot create the edge $e_{u,v}$ (i.e., it cannot be that $\Delta_l = w_T(e_{u,v})$). This follows from the worst-case distance stretch analysis, which stipulates that $d_G^{(h)}(u,v) \geq \Delta_l/(2\kappa)$. However, this would yield

$d_G^{(h)}(u,v) > \delta$, which is a contradiction. Therefore, the contribution of edges corresponding to levels in H_1 to $w_T(e_{u,v})$ is 0:

$$\sum_{l \in H_1} \Pr[e_{u,v} \text{ created by level } l \text{ call}] \cdot \Delta_l \cdot \mathbb{I}[u,v \in V(T)] = 0$$

Next, suppose that $l \in H_2$ and suppose $e_{u,v}$ was created via a level l call with the vertex set V' and partial vertex partition $C_1 \sqcup \ldots \sqcup C_k \subseteq V'$. If this is the case, the path P between u and v is broken in $C_1 \sqcup \ldots \sqcup C_k$, which by Theorem 40 happens with probability at most

$$\rho_{\text{pad}} \left(\frac{\text{hop}(p)}{h'} + \frac{w_G(p)}{\Delta_l/2} \right) \leq \rho_{\text{pad}} \left(\frac{h}{h'} + \frac{\delta}{\Delta_l/2} \right) = \frac{\rho_{\text{pad}}}{\kappa} + \frac{\rho_{\text{pad}}\delta}{\Delta_l/2}.$$

Moreover, note that $|H_2| = O(\log(\kappa/\rho_{\text{pad}})) = O(\log(\epsilon^{-1}\log n))$ since $\Delta_{l+1} = \Delta_l/2$. Therefore:

$$\begin{aligned}
\sum_{l \in H_2} \Pr[e_{u,v} \text{ created by level } l \text{ call}] \cdot \Delta_l \cdot \mathbb{I}[u,v \in V(T)] &\leq \sum_{l \in H_2} \left(\frac{\rho_{\text{pad}}}{\kappa} + \frac{\rho_{\text{pad}}\delta}{\Delta_l/2} \right) \cdot \Delta_l \cdot 1 \\
&\leq \sum_{l \in H_2} \left(\Delta_l \cdot \frac{\rho_{\text{pad}}}{\kappa} + 2\rho_{\text{pad}}\delta \right) \\
&\leq \frac{\rho_{\text{pad}}}{\kappa} \cdot (\delta \cdot 2\kappa) + 2\rho_{\text{pad}}\delta|H_2|) \\
&\leq \delta \cdot O(\rho_{\text{pad}} \log(\epsilon^{-1}\log n)).
\end{aligned}$$

Lastly, for H_3 notice that we can coarsely upper bound $\sum_{l \in H_3} \Pr[e_{u,v} \text{ created by level } l \text{ call}] \cdot \Delta_l \cdot \mathbb{I}[u,v \in V(T)]$ as $\sum_{l \in H_3} \Delta_l \leq \delta \cdot 4\rho_{\text{pad}}$ by our choice of H_3 and the fact that our weight diameters are geometrically decreasing.

Combining our upper bounds on the probability that $e_{u,v}$ is created in each level gives an upper bound on the expectation of $w_T(e_{u,v})$, which in turn bounds the expected value of $d_T(u,v)$ since $d_T(u,v) = O(w_T(e_{u,v}))$. In the following we let $(\ldots)$ stand for $\Pr[e_{u,v} \text{ created by level } l \text{ call}] \cdot \Delta_l \cdot \mathbb{I}[u,v \in V(T)]$.

$$\begin{aligned}
\mathsf{E}[w_T(e_{u,v}) \cdot \mathbb{I}[u,v \in V(T)]] &\leq \sum_{l=1}^{O(\log n)} \Pr[e_{u,v} \text{ created by level } l \text{ call}] \cdot \Delta_l \cdot \mathbb{I}[u,v \in V(T)] \\
&\leq \sum_{l \in H_1} (\ldots) + \sum_{l \in H_2} (\ldots) + \sum_{l \in H_3} (\ldots) \\
&\leq 0 + \delta \cdot O(\rho_{\text{pad}} \log(\epsilon^{-1}\log n)) + \delta \cdot (4\rho_{\text{pad}}) \\
&= \delta \cdot O(\rho_{\text{pad}} \log(\epsilon^{-1}\log n))
\end{aligned}$$

Plugging in the padded decompositions of Theorem 41, we conclude that the expected distance stretch is $O(\rho_{\text{pad}} \log(\epsilon^{-1}\log n)) = O(\log n \log(\epsilon^{-1}\log n))$, as required. □

4.4 h-Hop Partial Tree Embeddings

In the preceding section we demonstrated that hop-constrained distances can be well-approximated by distributions over partial tree metrics. In this section we describe how this result gives embeddings which can be used for hop-constrained network design problems. In particular, in Section 4.4.1 we will define h-hop partial tree embeddings which are partial tree metrics along with a mapping of each edge in the tree metric to a path in G. As an (almost) immediate corollary of our results in the previous section, we have that one can produce such an embedding where h-hop distances are approximately preserved by T and each path to which we map an edge has a low number of hops and less weight than the corresponding edge in T.

However, ultimately we are interested in using these embeddings to instantiate the usual tree embedding template and the above properties alone are not sufficient to do so. In particular, recall that in the usual tree embedding template for network design we embed our input graph into a tree, solve our problem on the tree and then project our solution back onto the input graph. If the problem which we solve on the tree has a much greater cost than the optimal solution on our input graph then our solution has no hope of being competitive with the optimal solution. Thus, we require some way of projecting the optimal solution of G onto our embeddings in a way that produces low-cost, feasible solutions for our tree problems.

When tree embeddings are not partial—as in FRT—such a projection is trivial. However, the partial nature of our embeddings along with the fact that we must preserve "h-hop connectivity" makes arguing that such a low cost solution exists significantly more challenging than in the FRT case. Somewhat surprisingly, we show that a natural projection of the optimal solution onto T produces an appropriate subgraph of T, despite the fact that an FRT-like charging argument seems incapable of proving such a result. Our proofs will be based on what may be viewed as a hop-constrained version of Euler tours which we call h-hop connectors. We give further intuition and details in Section 4.4.2. Thus, while Section 4.4.1 is a straightforward extension of our results from the previous section, the primary technical contribution of this section is the projection result of Section 4.4.2 which shows that, indeed, these embeddings may be used for tree-embedding algorithms in the usual way.

4.4.1 Defining h-Hop-Partial Tree Embeddings

We begin by defining our partial tree embeddings and proceed to argue that we can map from the trees in these embeddings to our graphs in a weight and connectivity-preserving fashion.

We now define hop and distance stretch of partial tree embeddings analogously to how we defined these concepts for partial metrics and for the non-hop-constrained setting in the previous chapter (Theorem 11).

Definition 42 (h-Hop Partial Tree Embedding)**.** *An h-hop partial tree embedding consists of a rooted and weighted tree $T = (V(T), E(T), w_T)$ with $V(T) \subseteq V(G)$ and a path $T_e^G \subseteq G$ for every $e \in E(T)$ between e's endpoints satisfying $w_G(T_e^G) \leq w_T(e)$. We say this embedding has **distance stretch** $\alpha \geq 1$ and **hop stretch** $\beta \geq 1$ for graph $G = (V(G), E(G), w_G)$ if*

1. $d_G^{(\beta h)} \leq d_T(u, v) \leq \alpha \cdot d^{(h)}(u, v)$ *for all* $u, v \in V(T) \subseteq V(G)$;
2. $\text{hop}(T_{uv}^G) \leq \beta h$ *for all* $u, v \in V(T) \subseteq V(G)$.

In the above we extend the notation of T_e^G to nodes in T which are not adjacent: for any two vertices $u, v \in V(T)$, if e_i is the ith edge in T_{uv} (ordered, say, from u to v) then $T_{uv}^G := T_{e_1}^G \oplus T_{e_2}^G \oplus \dots$ where $\oplus$ is concatenation.

Notice that the above definitions show that one can map subgraphs of an h-hop partial tree embedding $(T, \{T_e^G\}_{e \in E(T)})$ for G to subgraphs of G in a cost and connectivity preserving way. In particular, given a $T' \subseteq T$ we have that $H := \bigcup_{e \in E(T')} T_e^G$ satisfies (1) $w_G(H) \leq w_T(T')$ and (2) if u and v are connecting in T' then $\mathrm{hop}_H(u, h) \leq \beta h$. In the next section we give a much more involved and interesting proof showing that one can also project from subgraphs of G to T in a cost and connectivity preserving way.

The next observation confirms that, up to an $O(\log n)$, hop stretch and distance stretch for h-hop partial tree embeddings and partial metrics are equivalent, provided the relevant trees are well-separated.

Lemma 43. *Let G be a weighted graph and let $h \geq 1$ be a hop constraint.*

- *If $(T, \{T_e^G\}_{e \in E(T)})$ is an h-hop partial tree embedding with distance stretch α and hop stretch β then T is a partial tree metric which approximates $d_G^{(h)}$ with distance stretch α and hop stretch β.*
- *Conversely, if T is a partial tree metric with hop diameter $D_T := \mathrm{hop}(T)$ which approximates $d_G^{(h)}$ with distance stretch α and hop stretch β then there is a collection of paths $\{T_e^G\}_{e \in E(T)}$ where $(T, \{T_e^G\}_{e \in E(T)})$ is a partial tree embedding with distance stretch α and hop stretch $D_T \cdot \beta$.*

Proof. Let $(T, \{T_e^G\}_{e \in E(T)})$ be a partial tree embedding. Then we immediately have that T is a partial tree embedding with distance stretch α and hop stretch β by definition of a partial tree embedding and partial tree metric.

On the other hand, let T be a partial tree metric which approximates the h-hop constrained distances $d_G^{(h)}$ of G on $V(T)$ with distance stretch α and hop stretch β. By definition, for every edge $e \in E(T)$ with $e = \{u, v\}$ we have $d_G^{(\beta h)}(u, v) \leq w_T(e)$. In particular, there exists a path between the endpoints of e with at most βh hops and length at most $w_T(e)$ in G. Defining T_e^G to be this path for every edge $e \in E(T)$ completes T into a partial tree embedding. The distance stretch of this partial tree embedding is trivial by definition. Similarly, for u and v not adjacent in T we have, by definition of T_{uv}^G that T_{uv}^G consists of at most $\beta D_T \cdot h$ hops as required. □

Analogously to our results for partial metrics, we will also talk about the exclusion probability of distributions over partial tree, the distances they induce and how well they approximate hop-constrained distances; in particular, the following definitions are analogous to Theorem 35 and Theorem 36 respectively. For the sake of presentation, here and later in the chapter we let $(T, \cdot)$ be shorthand for $(T, \{T_e^G\}_{e \in E(T)})$.

Definition 44 (Distances of Partial Tree Embedding Distributions)**.** *Let $\mathcal{D}$ be a distribution of partial tree embeddings on weighted graph $G = (V, E, w)$. We say $\mathcal{D}$ has **exclusion probability** ϵ if for all $v \in V$ we have $\Pr_{(T,\cdot) \sim \mathcal{D}}[v \in V(T)] \geq 1 - \epsilon$. If $\epsilon \leq \frac{1}{3}$ then we say that $\mathcal{D}$ induces the distance function $d_{\mathcal{D}}$ on V, defined as*

$$d_{\mathcal{D}}(u, v) := \mathsf{E}_{(T,\cdot) \sim \mathcal{D}} \left[d_T(u, v) \cdot \mathbb{I}[u, v \in V(T)] \right].$$

Definition 45 (Stretch of Partial Tree Embedding Distribution)**.** *A distribution $\mathcal{D}$ of h-hop partial tree embeddings on V with exclusion probability at most $\frac{1}{3}$ approximates $d^{(h)}$ on weighted graph $G = (V, E, w)$ for hop constraint $h \geq 1$ with **worst-case distance stretch** $\alpha_{WC} \geq 1$ and **hop stretch** $\beta \geq 1$ if each $(T, \cdot)$ in the support of $\mathcal{D}$ approximates $d_G^{(h)}$ on $V(T)$ with distance stretch α_{WC} and hop stretch β, i.e. for each $(T, \cdot)$ in the support of $\mathcal{D}$ and all $u, v \in V(T)$ we have*

$$d_G^{(\beta h)}(u, v) \leq d_T(u, v) \leq \alpha \cdot d_G^{(h)}(u, v).$$

*Furthermore, $\mathcal{D}$ has **expected distance stretch** α_{E} if for all $u, v \in V$ we have*

$$d_{\mathcal{D}}(u, v) \leq \alpha_{\mathsf{E}} \cdot d_G^{(h)}(u, v).$$

Concluding, we have that there exists an efficiently-computable distribution over partial tree embeddings with poly-logarithmic stretches.

Theorem 46. *Given weighted graph $G = (V, E, w)$, $0 < \epsilon < \frac{1}{3}$ and root $r \in V$, there is a poly-time algorithm which samples from a distribution over h-hop partial tree embeddings whose trees are well-separated and rooted at r with exclusion probability ϵ, expected distance stretch $\alpha_{\mathsf{E}} = O(\log n \cdot \log \frac{\log n}{\epsilon})$, worst-case distance stretch $\alpha_{WC} = O(\frac{\log^2 n}{\epsilon})$ and hop stretch $\beta = O(\frac{\log^3 n}{\epsilon})$.*

Proof. We begin by remarking that Theorem 37 can be adapted so that all trees are rooted at r in the following way. First, we can assume that $r \in V(T)$ by resampling trees until r is in $V(T)$. By a union bound, this increases the exclusion probability by a factor of at most 2, leaves the hop stretch and worst-case distance stretch unchanged, and increases the expected distance stretch by a factor of at most $\frac{1}{1-\epsilon} = O(1)$; these modifications to our sampling process leave the statement of our theorem unchanged.

Now, suppose that a sampled tree has $r \in V(T)$; we will observe that r can be assumed to be the root of T. In particular, recall that in the construction of T in Theorem 37 we recursively constructs trees $T_1, \ldots, T_k$ on the parts of a partial vertex partition and then outputs a tree by connecting the root of $T_2, \ldots, T_k$ to the root of T_1. We note that T_1 is chosen arbitrarily, and so we can choose T_1 to be the tree containing r. Since we may assume inductively that r is the root of T_1, the tree we return has r as its root. Choosing a root in this way does not change the guarantees of our partial tree metrics.

Our result then follows immediately from the fact that well-separated trees have hop diameter $O(\log n)$, Theorem 43, Theorem 37 and the observation that the construction procedures of Theorem 37 and Theorem 43 are poly-time. □

4.4.2 Projecting From The Graph to h-Hop Partial Tree Embeddings

In this section we show how to project the optimal solution for a hop-constrained problem onto a partial tree embedding to get a low-cost subgraph which will be feasible for the optimization problems on trees which we later solve. In particular, we show that it is possible to project *any* subgraph $H \subseteq G$ onto an h-hop partial tree embedding $(T, \cdot)$ with worst-case distance stretch α in a way that α-approximately preserves the cost of H and preserves "h-hop connectivity": that

is, the projection of H will have cost at most $O(\alpha \cdot w_G(H))$ and if u and v are within h hops in H then they will be connected by the projection of H onto our embedding.

In the (non-partial) tree embedding setting where we typically only care about the connectivity structure of nodes—as in FRT—such a projections is trivial. In particular, if T is a tree drawn from the FRT distribution then an edge $e \in E(G)$ can be projected onto the simple tree path $T_{uv} \subseteq T$ between u and v in T and the resulting path will have expected weight $O(\log n \cdot w_G(e))$. Thus, we can project a subgraph $H \subseteq G$ to $T(H) := \bigcup_{\{u,v\} \in E(H)} T_{uv}$. If u and v are connected in H then they are connected in $T(H)$ and so the connectivity of nodes is preserved. Moreover, we can upper bound the weight of $T(H)$ by summing up $w_T(T_{uv})$ over all $\{u, v\} \in E(H)$ to get that, in expectation, $w_T(T(H)) \leq O(\log n \cdot w_G(H))$ and so the cost of the projection is appropriately low.

We might naturally try to use the same projection as is used in the FRT case but only for the nodes embedded by T. Specifically, suppose that T is now the tree of a partial tree embedding with worst-case distance stretch α. Then, we could project H to $T(H) := \bigcup T_{uv}$ where the $\bigcup$ is taken over all u, v such that $\{u, v\} \in E(H)$ *and* $u, v \in V(T)$. Although we trivially have that $w_T(T(H)) \leq \alpha \cdot w_G(H)$ by summing up over edges in $E(H)$, such a projection has no hope of preserving h-hop-connectivity as required: if, for example, u and v are connected by exactly one path in H with h hops then if there is even a single node along this path which is not in $V(T)$ then u and v may not be connected in $T(H)$.

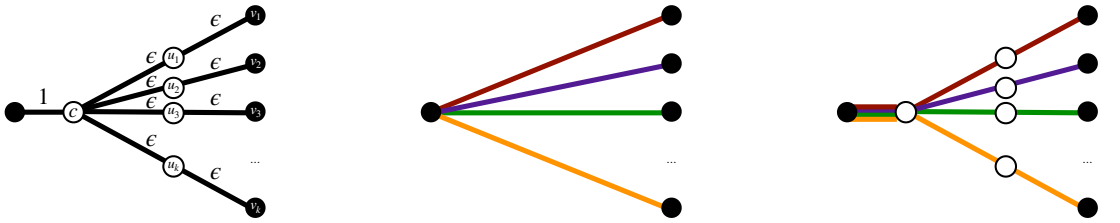
We could try to fix these connectivity issues by forcing all vertices in T which are within h hops in H to be connected in T as captured by the following definition.

Definition 47 ($T(H, h)$)**.** *Let $(T, \cdot)$ be a partial tree embedding. Then $T(H, h) := \bigcup T_{uv}$ where the $\bigcup$ is taken over u, v such that $u, v \in V(T)$ and $\text{hop}_H(u, v) \leq h$.*

$T(H, h)$ trivially preserve h-hop connectivity as needed: if u and v are connected by an h-hop path in H then they will be connected in $T(H, h)$. However, while $T(H, h)$ preserves h-hop connectivity, it seems to yield a subgraph of T of potentially unboundededly-bad cost. For example, let $h = 3$ and suppose H is a spider graph with $O(n)$ nodes in which one leg connects vertex r to center c with a cost 1 edge and the remaining ith leg connects c to u_i to v_i with a sufficiently small $\epsilon > 0$ cost edge. Further, suppose that $V(T)$ consists of r and all v_i. This example is illustrated in Figure 4.2a. $T(H, h)$ will buy T_{rv_i} for every i since there is an h-hop path from r to v_i (all such pairs illustrated in Figure 4.2b). Our worst-case distance guarantee ensures that $w_T(T_{rv_i})$ is at most $\alpha \cdot d_G(v_i, r) \approx \alpha$ and so we might hope to bound the cost of $T(H, h)$ as within $O(\alpha)$ times $w_G(H)$. However, if we try to apply the usual FRT-type proof and upper bound the cost of $T(H, h)$ in T as $\sum d_T(r, v_i)$ then our sum comes out to $O(\alpha \cdot n)$. On the other hand, $w_G(H) \approx 1$ and so $w_T(T(H, h))$ is a factor of $O(n \cdot \alpha)$ larger than $w_G(H)$ while we would like it to only be an $O(\alpha)$ factor larger. Thus, whereas FRT can charge each path in the projection of H to a unique edge of H, the partialness of our embedding means that we must charge paths in $T(H, h)$ to paths in H. These paths in H may induce large congestion—as illustrated in Figure 4.2c—which causes us to "overcharge" edges of H.

Surprisingly, in what follows we show that, while the above naive charging argument cannot succeed, a more nuanced proof shows that the above $T(H, h)$ is, in fact, competitive with the optimal solution up to small constants in the hop and distance stretch.

Theorem 48. *Fix $h \geq 1$, let H be a subgraph of weighted graph $G = (V, E, w_G)$ and let*



(a) Graph H **(b)** Pairs in Charged Sum **(c)** Congestion of Charged Paths

Figure 4.2: A counter-example to the naive charging argument for $T(H, h)$ where $\Theta(k) = \Theta(n)$. Edges labeled with their weights, vertices of $V(T)$ given as solid black circles and vertices of $V \setminus V(T)$ given as white-filled circles. Paths colored according to their corresponding pair.

$(T, \cdot)$ be an $8h$-hop partial tree embedding of G with worst-case distance stretch α. Then $w_T(T(H, h)) \leq 4\alpha \cdot w_G(H)$.

The basic idea of our proof will be to identify a collection of low congestion paths in H to which we can charge $T(H, h)$.

Warm-Up: Low Diameter Tree Case

To illustrate this idea we begin by showing how to prove Theorem 48 in the simple case where G is a tree with diameter at most h. In particular, on a tree of diameter at most h we can mitigate the congestion of charged paths by buying an Euler tour restricted to our embedded nodes; conveniently $T(H, h)$ will also be a subgraph of the projection of such an Euler tour onto T.

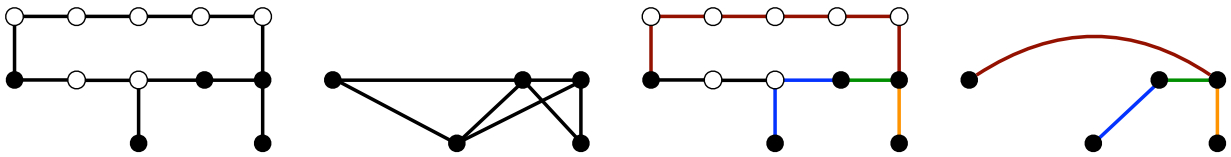
More specifically, suppose G is a tree with diameter at most h and let $(T, \cdot)$ be a partial tree embedding of G. Let G_2 be the multigraph of G where each edge is doubled. Let $t = (v_1, v_2, \ldots)$ be an Euler tour of G_2 and let $t' = (w_1, w_2, \ldots)$ be the vertices of $V(T)$ visited by this tour in the order in which they are visited. That is, t' is gotten from t be deleting from it all vertices not in $V(T)$ while leaving the ordering of the remaining vertices unchanged. Notice that vertices in $V(T)$ might occur multiple times in t'. We let P_ℓ be the path in G between w_ℓ and $w_{\ell+1}$ and let $\mathcal{P} := \{P_\ell\}_\ell$. Next, consider $T(\mathcal{P})$ which is the union of T_{uv} for every u, v where u and v form the endpoints of some path in $\mathcal{P}$.

First, notice that $T(H, h) \subseteq T(\mathcal{P})$. This follows since every $u, v \in V(T)$ which are within h hops (namely all $u, v \in V(T)$) are also visited by t' and so if T_{uv} is included in $T(H, h)$ then it will also be included in $T(\mathcal{P})$. Next, notice that $w_T(\mathcal{P}) \leq 2\alpha \cdot w_G(H)$ since our Euler tour when projected onto G visited each edge at most twice. This proves Theorem 48 for the h-diameter tree case.

h-Hop Connectors

The key observation of the above warm-up is that Euler tours allow us to mitigate the congestion induced in our charging arguments by providing a low-congestion collection of paths. We abstract such a collection of paths out in the form of what we call h-hop connectors.

For undirected and unweighted graph $G = (V, E)$ with $W \subseteq V$, we let $\mathcal{P}^{(h)}(W)$ be all simple paths between vertices in W with at most h hops. That is, each $P \in \mathcal{P}^{(h)}(W)$ has vertices in W as its first and last vertices and satisfies $|P \cap W| = 2$ and $\mathrm{hop}(P) \leq h$. Given a collection of paths $\mathcal{P}$ in G between vertices in W, we abuse notation and let $(W, \mathcal{P})$ be the graph with vertex set W



(a) Graph G **(b)** $(W, \mathcal{P}_h)$ **(c)** $\mathcal{P}$ in G **(d)** $(W, \mathcal{P})$

Figure 4.3: An illustration of an h-hop connector with congestion 1 and hop stretch 2 on a graph G for a vertex set $W \subseteq V(G)$ with $h = 3$. Vertices of W given as solid black circles; all other vertices of G given as white circles. Edges in $(W, \mathcal{P})$ and paths in $\mathcal{P}$ colored according to their correspondence.

and an edge $\{u, v\}$ iff there is a $P \in \mathcal{P}$ with endpoints $\{u, v\}$. We will refer to $(W, \mathcal{P}^{(h)}(W))$ as the h-hop connectivity graph of W. We let $c_e(\mathcal{P}) := |P \in \mathcal{P} : e \in P|$ be the congestion of e with respect to a collection of paths $\mathcal{P}$. With this notation in hand, we give our definition of h-hop connectors which we illustrate in Figure 4.3.

Definition 49 (h-Hop Connector)**.** *Let $G = (V, E)$ be an undirected and unweighted graph, let $h \geq 1$ and let $W \subseteq V$. An h-hop connector $\mathcal{P}$ of W with congestion C and hop stretch β is a collection of paths in G between vertices of W such that:*

1. ***Connecting:*** *all $u, v \subseteq W$ which are connected in $(W, \mathcal{P}^{(h)}(W))$ are connected in $(W, \mathcal{P})$;*
2. ***Edge Congestion:*** *For all $e \in E$ we have $c_e(\mathcal{P}) \leq C$;*
3. ***Hop Stretch:*** $\text{hop}(P) \leq \beta \cdot h$ *for all $P \in \mathcal{P}$.*

It is easy to observe that the existence of good h-hop connectors are sufficient to show Theorem 48.

Lemma 50. *Fix $h \geq 1$, let $H \subseteq G$ be a subgraph of weighted graph $G = (V, E, w_G)$ and let $(T, \cdot)$ be a (βh)-hop partial tree embedding of G with worst-case distance stretch α. If H has an h-hop connector on $V(T)$ with hop stretch β and congestion C then $w_T(T(H, h)) \leq C\alpha \cdot w_G(H)$.*

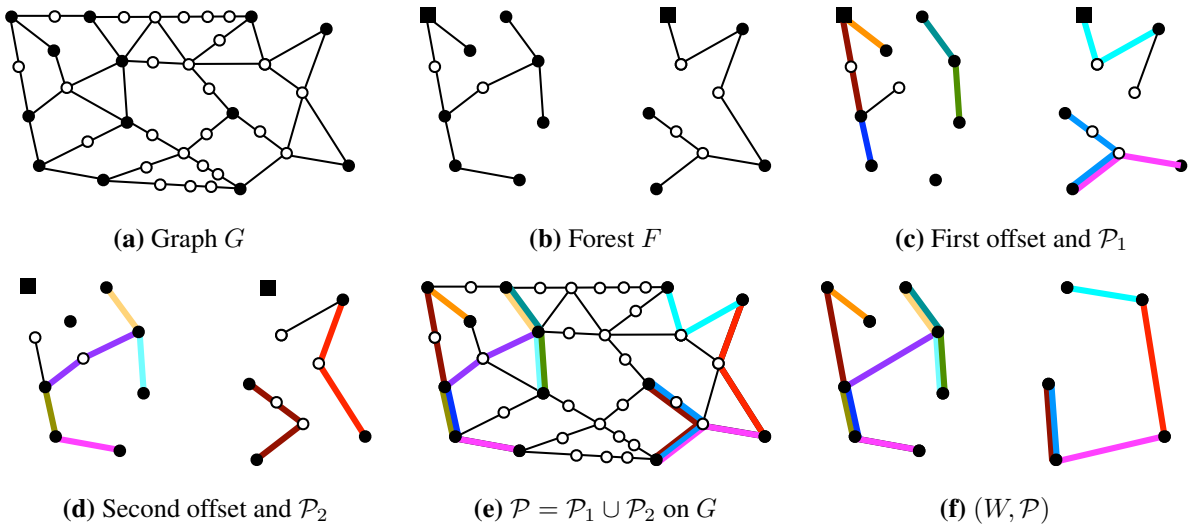
Proof. Let $\mathcal{P}$ be the stated h-hop connector, let $S := \{(u, v) : (u, \ldots, v) \in \mathcal{P}\}$ be the endpoints of its path and let $T(\mathcal{P}) := \bigcup_{(u,v)\in S} T_{uv}$ be the subgraph of T corresponding to $\mathcal{P}$. By the connecting property of our h-hop connector any u, v which are within h hops in H must also be connected in $T(\mathcal{P})$ and so $T(H, h) \subseteq T(\mathcal{P})$. Combining this with the edge congestion and hop stretch of our h-hop connector with the worst-case distance stretch of T we have $w_T(T(H, h)) \leq w_T(T(\mathcal{P})) \leq C\alpha \cdot w_G(H)$. □

Thus, we devote the remainder of this section to showing that every graph has an h-hop connector with hop stretch 8 and congestion 4.

A simple proof similar to the above warm-up shows that trees with low diameter have good h-hop connectors.

Lemma 51. *Let $G = (V, E)$ be a tree with diameter at most βh for $h \geq 1$. Then, G has an h-hop connector with congestion at most 2 and hop stretch at most β for every $W \subseteq V$.*

Proof. Suppose G is a tree. Let G_2 be the multigraph of G where each edge is doubled. Let $t = (v_1, v_2, \ldots)$ be an Euler tour of G_2 and let $t' = (w_1, w_2, \ldots)$ be the vertices of W visited by this tour in the order in which they are visited. That is, t' is gotten from t be deleting from it all vertices not in W while leaving the ordering of the remaining vertices unchanged. Notice that



(a) Graph G **(b)** Forest F **(c)** First offset and $\mathcal{P}_1$

(d) Second offset and $\mathcal{P}_2$ **(e)** $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ on G **(f)** $(W, \mathcal{P})$

Figure 4.4: An illustration of how to compute an h-hop connector on an arbitrary graph G for $W \subseteq V(G)$ with $h = 3$. Vertices of W given as solid black circles; roots of F given as black squares; all other vertices of G given as white circles. Paths in $\mathcal{P}_1$ and $\mathcal{P}_2$ colored to correspond to their edges in $(W, \mathcal{P})$.

vertices in W might occur multiple times in t'. We let P_ℓ be the path in G between w_ℓ and $w_{\ell+1}$ and let $\mathcal{P} := \{P_\ell\}_\ell$.

Since every vertex in W occurs at least once in t' we have that all vertices in W are connected in $(W, \mathcal{P})$. Since t used each edge of G_2 once, it follows that $c_e(\mathcal{P}) \leq 2$. Lastly, $\mathrm{hop}(P_\ell) \leq \beta h$ for all $P_\ell \in \mathcal{P}$ since each P_ℓ is a simple path in a tree with diameter at most βh. □

We proceed to show how to construct an h-hop connector with congestion 4 and hop stretch 8 on any graph. We first reduce the general graph case to the forest case: we show that, up to a factor of 2 in the hop stretch, every graph G has as a subgraph a forest F where an h-hop connector for F is an h-hop connector for G. We then reduce the forest case to the low diameter tree case by cutting each tree in F at $O(h)$-spaced annuli from an arbitrary root so that the resulting trees have low diameter. We apply Theorem 51 to the resulting low-diameter trees. More specifically, we perform these cuts and applications of Theorem 51 twice with two different offsets to get back paths $\mathcal{P}_1$ and $\mathcal{P}_2$; we then take our h-hop connector to be $\mathcal{P} := \mathcal{P}_1 \cup \mathcal{P}_2$. We illustrate this strategy in Figure 4.4.

We begin with a simple technical lemma which shows that the graphs induced by the connected components of the h-hop connectivity graph are disjoint. For a collection of paths $\mathcal{P}$ in G we let $G[\mathcal{P}] := G[\bigcup_{P \in \mathcal{P}} V(P)]$ be the graph induced by the union of all such paths.

Lemma 52. *Let $G = (V, E)$ be a graph, let $W \subseteq V$, and let U and U' be the vertices of two distinct connected components of $(W, \mathcal{P}^{(h)}(W))$. Then $G[\mathcal{P}^{(h)}(U)]$ and $G[\mathcal{P}^{(h)}(U')]$ are vertex-disjoint.*

Proof. It suffices to show that for any $P, P' \in \mathcal{P}^{(h)}(W)$ if $V(P) \cap V(P') \neq \emptyset$ then the edges corresponding to P and P' in $(W, \mathcal{P}^{(h)}(W))$ are in the same connected components in $(W, \mathcal{P}^{(h)}(W))$. Let $u, v \in W$ be the endpoints of P and let $u', v' \in W$ be the endpoints of P'. Suppose some $x \in V(P) \cap V(P')$. Let P_{ux} and P_{xv} be the subpaths of P from u and v to x respectively and define $P'_{u'x}$ and $P'_{xv'}$ symmetrically. Then, without loss of generality P_{ux} and $P'_{u'x}$ both have at

most $h/2$ edges meaning the concatenation of P_{ux} and $P'_{u'x}$ at x is in $\mathcal{P}^{(h)}(W)$. It follows that u and u' are in the same connected component in $(W, \mathcal{P}^{(h)}(W))$ and therefore, v and v' are also in this component. □

Applying the above lemma, we show that, up to a factor of 2 in the hop stretch, we may assume that our graph is a forest. We let $\mathcal{P}_G^{(h)}(W)$ be all paths with at most h hops between vertices in W *in graph* G.

Lemma 53. *Let $G = (V, E)$ be a graph, let $W \subseteq V$. Then there exists a subgraph $F \subseteq G$ which is a forest where $u, v \in W$ are connected in $(W, \mathcal{P}_G^{(h)}(W))$ iff u, v are connected in $(W, \mathcal{P}_F^{(2h)}(W))$.*

Proof. We will iteratively construct F. Specifically, for each connected component of $(W, \mathcal{P}_G^{(h)}(W))$ with vertex set U we will maintain a collection of paths $\mathcal{P}_U$ where these paths are all contained in $G[\mathcal{P}^{(h)}(U)]$ and F is the graph induced by the union of all these paths. It follows that by Theorem 52 if $G[\mathcal{P}_U]$ is a tree then the connected components of our final solution are indeed a forest. We will maintain the following invariants for our $\mathcal{P}_U$s where $\text{hop}_G(v, U) := \min_{u \in U} \text{hop}_G(v, u)$:

1. $U' := U \cap V(G[\mathcal{P}_U])$ is connected in $(U', \mathcal{P}_U)$;
2. $G[\mathcal{P}_U]$ is a tree;
3. $\text{hop}(P) \leq 2h$ for every $P \in \mathcal{P}_U$;
4. $\text{hop}_{G[\mathcal{P}_U]}(v, U) \leq h$ for every $v \in V(G[\mathcal{P}_U])$.

We initialize $\mathcal{P}_U$ to contain a path consisting of exactly one (arbitrary) vertex in U. Notice that our construction trivially satisfies these invariants initially.

Next, we repeat the following until $U' = U$. Let u be a vertex in $U \setminus U'$ where u has a path P of at most h hops to a vertex in U'; such a u and P must exist by the definition of U. Let x be the first vertex in $P \cap G[\mathcal{P}_U]$ where we imagine that P starts at u and let P_{ux} be the subpath of P from u to x. By invariant 4 we also know there is some path in $G[\mathcal{P}_U]$ from x to a $u' \in U'$ with at most h hops; call this path $P_{xu'}$ and let P' be the concatenation of P_{ux} and $P_{xu'}$; we add P' to $\mathcal{P}_U$. Notice that this adds u to U' and so this process will eventually terminate at which point $U' = U$.

Let us argue that our invariants hold. Our first invariant holds since before adding u to U', U' was connected and after adding u to U', u is connected to u' by P'. Our second invariant holds since x was the first vertex in $G[\mathcal{P}_U]$ incident to P. Our third invariant holds since P_{ux} and $P_{xu'}$ were each of at most h hops. Our fourth invariant holds since the only new vertices we add to $G[\mathcal{P}_U]$ are the vertices of P_{ux}, all of which are within h hops of u.

Lastly, notice that once $U' = U$ for every U, our claim follows from invariants 1,2 and 3. □

By turning our graph into a forest with Theorem 53 and then cutting the constituent trees at $O(h)$-spaced level sets with two different initial offsets, we can conclude that every graph has h-hop connectors with constant congestion and hop stretch.

Lemma 54. *Let $G = (V, E)$ be a graph. Then G has an h-hop connector with congestion 4 and hop stretch 8 for every $W \subseteq V$.*

Proof. By Theorem 53 we know that there is a forest F such that u, v are connected in $(W, \mathcal{P}_G^{(h)}(W)$ iff u, v are connected in $(W, \mathcal{P}_F^{(2h)}(W))$. Let T be a tree in this forest and notice that to get an h-hop connector on G with hop stretch 8 and congestion 4, it suffices to find a $2h$-hop connector on T with hop stretch 4 and congestion 4.

We do so as follows. Root T arbitrarily at root r and let $T_1, T_2, \ldots$ be the subtrees resulting from cutting T once every $4h$ levels and let $T'_1, T'_2, \ldots$ be the subtrees resulting from cutting T every $4h$ levels with an initial offset of $2h$. That is, $T_i = T[V(T_i)]$ and $v \in V(T_i)$ iff $4h(i-1) \leq d_T(v, r) < 4h \cdot i$ and $T'_i = T[V(T'_i)]$ where $v \in V(T'_i)$ iff $\max(4h(i-1) - 2h, 0) \leq d_T(v, r) < 4h \cdot i - 2h$. Notice that each T_i and T'_i has diameter at most $4(2h)$. Thus, by Theorem 51 we know that each T_i and T'_i have $2h$-hop connectors $\mathcal{P}_i$ and $\mathcal{P}'_i$ with congestion at most 2 and hop stretch at most 4. Thus, we let $\mathcal{P}_1 := \{\mathcal{P}_i\}_i$ and $\mathcal{P}_2 := \{\mathcal{P}'_i\}_i$ and we let our h-hop connector for T be $\mathcal{P} := \mathcal{P}_1 \cup \mathcal{P}_2$.

Let us argue that $\mathcal{P}$ is a $2h$-hop connector on T with hop stretch 8 and congestion 4. Our congestion bound is immediate from Theorem 51 and the fact that each edge occurs in at most 2 trees among all T_i and T'_i. To see why $\mathcal{P}$ is connecting notice that if u, v are within $2h$ hops of one another in T by some path P then this path must be fully contained in some T_i or T'_i; it follows that u and v will be connected in some $\mathcal{P}_i$ or $\mathcal{P}'_i$ and so connected in $\mathcal{P}$. Lastly, our hop bound is immediate by Theorem 51 since each T_i and T'_i has diameter at most $4(2h)$. □

Combining Theorem 54 with Theorem 50 immediately gives Theorem 48.

Before proceeding to our applications, we remark on a subtle issue regarding independence and expected distance stretch versus worst case distance stretch. Theorem 48 bounded the cost of projecting a subgraphs of G onto a partial tree embedding of G based on the tree embedding's worst-case distance stretch; one might naturally wonder if similar results are possible in terms of the expected distance stretch of a distribution over partial tree embeddings. Here, dependence issues and the partialness of our embeddings work against us. Specifically, one would have to argue that $T(H, h)$—and, in particular, the relevant h-hop connector for $T(H, h)$—has low cost in expectation where $(T, \cdot)$ is drawn from a distribution. However, while it is true that for a fixed H and T the relevant h-hop connector for H and T has low cost in expectation over the entire distribution of tree embeddings, it need not be the case that this h-hop connector has low cost when we condition on the fact that T is the tree we drew from our distribution. In short, Theorem 50 seems to fail to hold for the expectation case.

4.5 Applications of h-Hop Partial Tree Embeddings

In this section we apply our embeddings of $d^{(h)}$ to give approximation algorithms for hop-constrained versions of several well-studied network design problems; namely, oblivious hop-constrained Steiner forest, hop-constrained group Steiner tree, hop-constrained k-Steiner tree and hop-constrained oblivious network design. If unspecified, OPT will stand for the optimal value of the relevant hop-constrained problem throughout this section. We improve our results for hop-constrained group Steiner tree in a later section (Section 4.7.1).

4.5.1 Oblivious Hop-Constrained Steiner Forest

In this section we give our approximation algorithms for oblivious hop-constrained Steiner forest. While we give our results for oblivious hop-constrained Steiner forest, it is easy to see that an approximation algorithm for the oblivious version gives an approximation algorithm with the same approximation ratios for the online and offline versions of the problem; to our knowledge nothing was known for any of these variants prior to our work.

(Hop-Constrained Oblivious) Steiner Forest: In Steiner forest we are given a weighted graph $G = (V, E, w)$.

- **Offline:** In offline Steiner forest we are also given a collection of pairs of nodes $\{(s_i, t_i)\}_i$. Our goal is to find a subgraph $H \subseteq G$ so that every s_i is connected to every t_i in H.
- **Online:** In online Steiner forest in each time step $t = 1, 2, \ldots$ a new pair of vertices (u_t, v_t) is revealed and we must maintain a solution H_t for each t where $H_{t-1} \subseteq H_t$ which connects pairs in $\{(u_1, v_1), \ldots, (u_t, v_t)\}$.
- **Oblivious:** In oblivious Steiner forest we must specify a path P_{uv} for each pair of vertices $(u, v) \in V \times V$ before seeing any demands. The demands $\{(s_i, t_i)\}_i$ are then revealed, inducing our solution $H := \bigcup_i P_{s_i t_i}$.

In all three problems the cost of our solution H is $w(H) := \sum_{e \in E(H)} w(e)$. In the oblivious and offline versions, our approximation ratio is $w(H)/\mathrm{OPT}$ where OPT is the cost of the optimal offline solution for the given demand pairs. The competitive ratio of our solution in the online case is $\max_t w(H_t)/\mathrm{OPT}_t$ where OPT_t is the minimum cost subgraph of G connecting pairs in $\{(u_1, v_1), \ldots, (u_t, v_t)\}$.

In the hop-constrained versions of each of these problems we are additionally given a hop constraint $h \geq 1$ and if (s_i, t_i) is a demand pair then our solution H must satisfy $\mathrm{hop}_H(s_i, t_i) \leq h$ for all i. The optimal solution against which we measure our approximation ratio is similarly hop-constrained.

Notice that, unlike in the Steiner forest problem where we may assume without loss of generality that each connected component of H is a tree, in hop-constrained Steiner forest each connected component of H might not be a tree.

Related Work: We give some brief highlights from work in Steiner forest and hop-constrained Steiner forest: while NP-hard Agrawal et al. [5] gave the first constant approximation for offline Steiner forest; Berman and Coulston [30] gave an (optimal) $O(\log k)$ approximation for online Steiner forest and Gupta et al. [105] gave the first non-trivial approximation algorithm for oblivious Steiner forest, an $O(\log^2 n)$ approximation. There has also been quite a bit of work on approximation algorithms for h-spanners which can be seen as a special case of offline hop-constrained Steiner forest; see, for example, Dinitz and Zhang [66] and references therein. Notably for our purposes, Elkin and Peleg [71] and Dinitz et al. [67] show that unless NP $\not\subseteq$ BPTIME$(2^{\mathrm{poly}\log n})$ hop-constrained Steiner forest admits no $O(2^{\log^{1-\epsilon} n})$ approximation; this immediately rules out the possibility of a poly-log (unicriteria) approximation for hop-constrained Steiner forest. We also note that a recent work of Babay et al. [17] gave results for hop-constrained Steiner forest from a parameterized complexity perspective.

Algorithm: Roughly, our algorithm follows the usual tree-embedding template: we first apply our h-hop partial tree embeddings to reduce oblivious hop-constrained Steiner forest to oblivious Steiner forest on a tree; we then observe that oblivious Steiner forest is trivially solvable on trees and project our solution back to G. The only minor caveats are: (1) since our tree embeddings will only embed a constant fraction of nodes, we must repeat this process $O(\log n)$ times and (2) for each tree embedding we must use Theorem 48 to argue that there is a cheap, feasible solution for the relevant Steiner forest problem on each tree.

Formally, our algorithm to compute our solution H is as follows. We begin by applying Theorem 46 to sample $8h$-hop partial tree embeddings $T_1, T_2, \ldots, T_k$ where $k := O(\log n)$ for a

sufficiently large hidden constant, $\epsilon = .1$ and an arbitrary root. Given $u, v \in V$, assign the pair (u, v) to an arbitrary T_j such that $u, v \in V(T_j)$ (we will argue that such a T_j exists with high probability). Next, we let our path for u, v be $P_{uv} := (T_j)^G_{uv}$ the projection of the tree path between u and v onto G.

We now give the analysis of our algorithm.

Theorem 55. *There is a poly-time algorithm which given an instance of h-hop-constrained oblivious Steiner forest returns a collection of paths such that the induced solution H for any demand set satisfies $w(H) \leq O(\text{OPT} \cdot \log^3 n)$ and $\text{hop}_H(s_i, t_i) \leq O(h \cdot \log^3 n)$ with high probability.*

Proof. We use the above algorithm. We begin by arguing that H connects every s_i to t_i for every i with high probability with a path of at most $O(\log^3 n \cdot h)$ edges. Fix a vertex v. A standard Chernoff-and-union-bound-type argument shows that v is in at least $.8k$ of the T_j with high probability. Specifically, let X_j be the random variable which indicates if v is in $V(T_j)$, let $X := \sum_j X_j$ and apply a Chernoff bound to X.

Taking a union bound over all v we have that with high probability every v is in at least $.8k$ of the T_j. Since we have k total T_j, by the pigeonhole principle it follows that any pair of vertices (s_i, t_i) simultaneously occur in at least $.6k$ of the T_j, meaning that for each such pair there is a T_j where we buy $(T_j)^G_{s_i t_i}$ and so s_i will be connected to t_i in our solution. Since $\text{hop}((T_j)^G_{s_i t_i}) \leq O(h \cdot \log^3 n)$ by Theorem 46, it follows that $\text{hop}_H(s_i, t_i) \leq O(h \cdot \log^3 n)$.

We next argue that our solution satisfies the stated cost bound. Let H_{T_j} be the minimal subgraph of T_j connecting all pairs assigned to T_j and let $H_j := \bigcup_{e \in H_{T_j}} (T_j)^G_e$ be the projection of H_{T_j} onto G. Notice that it suffices to argue that $w_{T_j}(H_{T_j}) \leq O(\text{OPT} \cdot \log^2 n)$ for every j since if this held we would have by Theorem 46 that the cost of our solution is $w(H) \leq \sum_j w(H_j) \leq \sum_j \sum_{e \in H_{T_j}} w((T_j)^G_e) \leq \sum_j \sum_{e \in H_{T_j}} w_{T_j}(e) = \sum_j w_{T_j}(H_{T_j}) \leq O(\text{OPT} \cdot \log^3 n)$. However, applying Theorem 48 to the optimal solution H^* on G shows that $T(H^*, h)$ is a feasible solution for the Steiner forest problem on T_j which connects all pairs assigned to T_j with cost at most $O(\log^2 n \cdot \text{OPT})$. Since H_{T_j} is the optimal solution for such a Steiner forest problem, it follows that $w_{T_j}(H_{T_j}) \leq O(\text{OPT} \cdot \log^2 n)$ as required. □

Bicriteria Min-Cost Spanner Approximations

We end this section by remarking that our hop-constrained Steiner forest algorithm gives new bicriteria approximation algorithms for spanner problems. Notably, the aforementioned $\Omega(2^{\log^{1-\epsilon} n})$ hardness of approximation reductions break down for bicriteria approximation algorithms. For this reason, Chlamtáč and Dinitz [56] state the following regarding bicriteria approximation algorithms for spanner problems:

> *Obtaining good bicriteria approximations, or proving that they cannot exist, is an extremely interesting area for future research...*

As a corollary to our hop-constrained Steiner forest problem result, we give a new such bicriteria approximation algorithms for spanners. Specifically, in the minimum cost client-server h-spanner problem we are given a client graph $G_c = (V, E_c)$ and a weighted server graph $G_s = (V_s, E_s, w)$ and integer $h \geq 1$. We must find a subgraph H of G_s which minimizes $w(H)$ subject to the constraint that for each $\{u, v\} \in E_c$ we have $\text{hop}_H(u, v) \leq h$. [70] and [72] studied the unit cost

version of this problem for $h = 2$ and $h > 3$, giving bicriteria algorithms in the latter, but, to our knowledge no (poly-log, poly-log) bicriteria approximation algorithms are known for either the unit-cost version of this problem or the min-cost spanner problem (i.e. this problem when $E_s = E_c$).

By creating an offline hop-constrained Steiner forest problem which has a demand pair for each client edge, it is easy to see that Theorem 55 gives such a bicriteria approximation algorithm for min-cost client-server h-spanner.

Corollary 56. *There is a poly-time algorithm for min-cost client-server h-spanner which returns an $H \subseteq G_s$ where $w(H) \leq O(\mathrm{OPT} \cdot \log^3 n)$ and for each $\{u, v\} \in E_c$ we have $\mathrm{hop}_H(u, v) \leq O(h \cdot \log^3 n)$.*

4.5.2 Hop-Constrained Group Steiner Tree

As both set cover and Steiner tree are special cases of it, the group Steiner tree problem is one of the most general covering problems. In this section, we give $(O(\mathsf{poly}\log n), O(\mathsf{poly}\log n))$ bicriteria approximation algorithms for the hop-constrained variant of group Steiner tree. We later give an improved approximation based on "h-hop copy tree embeddings" which build on our h-hop partial tree embeddings. However, we include this result to highlight the fact that h-hop partial tree embeddings alone are sufficient for solving many hop-constrained problems.

Hop-Constrained Group Steiner Tree: See Section 2.0.1 for a definition of group Steiner Tree. Hop-constrained group Steiner tree is defined as group Steiner tree but we are additionally given a hop bound $h \geq 1$ and we must ensure that $\mathrm{hop}_T(g_i, r) \leq h$ where $\mathrm{hop}_T(g_i, r) := \min_{v \in g_i \cap V(T)} \mathrm{hop}_T(v, r)$.[6] Unlike hop-constrained Steiner forest, the optimal solution for hop-constrained Steiner tree is, in fact, a tree. In particular, if H is a feasible solution, then the shortest path tree on H rooted at r is also a feasible solution of cost at most the cost of H.

Algorithm: Our algorithm will reduce solving hop-constrained group Steiner tree to a series of group Steiner tree problems on trees. For this reason, we restate the following known result for group Steiner tree on trees.

Theorem 57 ([92])**.** *There exists a randomized algorithm which with high probability given an instance of group Steiner tree on a tree returns a solution of cost at most $O(\mathrm{OPT} \cdot \log N \log k)$.*

We might naively hope to realize the usual tree embedding template: sample $O(\log n)$ partial tree embeddings using Theorem 46, apply Theorem 57 to the resulting trees and then project the solutions back to the input graph. However, things are not so simple: since our partial tree embeddings only embed a subset of nodes, the optimal solution on the group Steiner tree problem on each of our partial tree embeddings has no guarantees of being close to optimal. For example,

[6]The assumption that the tree is rooted in group Steiner tree is without loss of generality as we may always brute-force search over a root. Similarly, the assumption that all groups are pairwise disjoint is without loss of generality since if v is in groups $\{g_1, g_2, \ldots\}$ then we can remove v from all groups and add vertices $v_1, v_2, \ldots$ to G which are connected only to v so that $v_i \in g_i$ and $w((v, v_i)) = 0$ for all i. For the unrooted hop-constrained group Steiner tree problem we might define the problem as unrooted group Steiner tree but with the additional constraint that T has diameter at most h; all of our results will hold for this unrooted version of the problem though its worth noting that in this case the optimal solution is no longer a tree without loss of generality. The aforementioned transformation also allows us to assume that groups are pairwise disjoint in hop-constrained group Steiner tree at a possible loss of an additive 1 in our hop stretch.

suppose $g_i = \{v_i, v_i'\}$ where $w((r, v_i)) = \epsilon$ for some small $\epsilon > 0$ and $w((r, v_i')) = 1$ for each group g_i. Then, if our partial tree embedding embeds v_i' but not v_i, connecting g_i to r on our tree will be arbitrarily more expensive than OPT.

We solve this issue by relaxing the instance of hop-constrained Steiner tree that we solve on each of our trees. Specifically, we will randomly merge groups so that there always exists a low cost group Steiner solution. For example, in the above example we could randomly partition groups into super-groups each consisting of $\Theta(\log n)$ groups. If we then solved group Steiner tree on our tree on these super-groups we would know that—by a standard Chernoff-union bound proof—every super-group has at least one constituent v_i embedded on the tree and so the optimal group Steiner tree problem restricted to embedded nodes on our tree still has low cost.

Formally, our algorithm for constructing our solution T is as follows. Initially every group is active; we let a be the number of active groups throughout our algorithm.

1. For phase $j \in [1000 \log n \log k]$ or until $a \leq 10 \log n$
   (a) For iteration $\ell \in [1000 \log n]$
      i. Apply Theorem 46 with hop bound $8h$, $\epsilon = .1$ and root r on G to get partial tree embedding $T_{j\ell}$
      ii. Let $(g'_{j\iota})_{\iota=1}^{k'}$ form a uniformly random partition of the vertices of all active groups where $k' = \lceil \frac{a}{10 \log n} \rceil$ and an active $g_i \subseteq g'_{j\iota}$ with probability $1/k'$
      iii. Apply Theorem 57 to the group Steiner tree instance on $T_{j\ell}$ with root r and groups $(g'_{j\iota})_{\iota=1}^{k'}$ to get back solution $H_{j\ell}$
   (b) Let $j^* := \arg\min_\ell w_{T_{j\iota}}(H_{j\ell})$
   (c) Add T_e^G to T for every $e \in E(H_{jj^*})$
   (d) Set all g_i which are now connected to r by T as inactive
2. For the up to $10 \log n$ remaining active groups we add to T the shortest path in G from r to each such group with at most h hops
3. Lastly, we set T to be a BFS tree on T rooted at r to ensure that T is a tree

We apply Theorem 48 and a standard Chernoff-union-bound-type argument to argue that each of our instances of group Steiner tree on a tree have a cheap solution.

Lemma 58. *Fix a phase j and let OPT_ℓ be the cost of the optimal group Steiner tree instance on $T_{j\ell}$ with root r and groups $(g'_{j\iota})_{i=1}^{k'}$. Then $\min_\ell \mathrm{OPT}_\ell \leq O(\log^2 n \cdot \mathrm{OPT})$ with probability at least $1 - \frac{1}{n^5}$.*

Proof. Since we have fixed a j, for ease of notation we let $T_\ell := T_{j\ell}$ and $g'_\iota = g'_{j\iota}$ for the remainder of the proof.

We will construct a solution T'_ℓ for every ℓ which has cost at most $O(\log^2 n \cdot \mathrm{OPT})$ and which is feasible for the aforementioned group Steiner instance with probability at least $\frac{1}{3}$. Our claim will then immediately follow from this and the fact that the feasibility of each T'_ℓ will be independent, meaning with probability at least $1 - (\frac{1}{3})^{1000 \log n} \geq 1 - \frac{1}{n^5}$ there is some feasible T'_ℓ with cost at most $O(\log^2 n \cdot \mathrm{OPT})$.

Fix an arbitrary ℓ. Let T^* be the optimal solution to our h-hop-constrained group Steiner tree problem on G and let $W := V(T_\ell)$ be all vertices embedded by T_ℓ. Let $T'_\ell := T_j(T^*, h)$ where

$T_j(T^*, h)$ is as defined in Theorem 47. By Theorem 48 we have $w_{T_j}(T_j(T^*, h)) \leq O(\log^2 n \cdot \mathrm{OPT})$ as desired.

We now argue that T'_ℓ is feasible with probability at least $\frac{1}{3}$. It suffices to show that some vertex from g'_ι is in $W \cap V(T^*)$ for every $\iota \in [k']$ with probability at least $\frac{1}{3}$.

Let $\mathcal{I} := \{i : g_i \cap W \cap V(T^*) \neq \emptyset\}$ be all groups with at least one embedded vertex from the optimal solution. We know by Theorem 46 and linearity of expectation that $\mathsf{E}[|\mathcal{I}|] \geq .9a$ but since $|\mathcal{I}| \leq a$, it follows by Markov's inequality that $\Pr(|\mathcal{I}| \geq .8a) \geq \frac{1}{2}$.

Fix a super-group g'_ι. For group g_i, let X_i be the indicator of whether $g_i \subseteq g'_\iota$. Similarly, let $\vec{\mathcal{I}}$ be a fixed value in the support of $\mathcal{I}$ and let $X_\iota^{(\vec{\mathcal{I}})} := \sum_{i \in \vec{\mathcal{I}}} X_i$. Notice that $\mathsf{E}[X_\iota^{(\vec{\mathcal{I}})}] \geq |\vec{\mathcal{I}}| \cdot \frac{1000 \log n}{a}$ and that for a fixed $\vec{\mathcal{I}}$ if $\mathcal{I} = \vec{\mathcal{I}}$ and $X_\iota^{(\vec{\mathcal{I}})} \geq 1$ then T'_ℓ will connect g'_ι to r.

Since for a fixed $\vec{\mathcal{I}}$ we know that each X_i in $\sum_{i \in \vec{\mathcal{I}}} X_i$ is independent, a Chernoff-bound shows that

$$\Pr\left(X_\iota^{(\vec{\mathcal{I}})} \leq |\vec{\mathcal{I}}| \cdot \frac{900 \log n}{a}\right) \leq \exp\left(-\frac{(.1)^2 \cdot 1000 \log n \cdot |\vec{\mathcal{I}}|}{3a}\right) \leq \exp\left(-\frac{3|\vec{\mathcal{I}}| \cdot \log n}{a}\right)$$

It follows that if $|\vec{\mathcal{I}}| \geq .8a$ we have that $\Pr\left(X_{\vec{\mathcal{I}}} = 0\right) \leq \frac{1}{n^2}$. Combining this with a union bound and the fact that $|\mathcal{I}|$ is at least $.8a$ with probability at least $\frac{1}{2}$, we have that T'_ℓ contains a vertex from every g'_ι except with probability

$$\begin{aligned}
\sum_{\vec{\mathcal{I}}} \Pr(\mathcal{I} = \vec{\mathcal{I}}) \cdot \Pr(X_\iota^{(\vec{\mathcal{I}})} = 0 \text{ for some } \iota) &\leq \sum_{(\vec{\mathcal{I}})} \Pr(\mathcal{I} = \vec{\mathcal{I}}) \sum_\iota \Pr(X_\iota^{(\vec{\mathcal{I}})} = 0) \\
&= \sum_{\vec{\mathcal{I}}:|\vec{I}<.8a|} \Pr(\mathcal{I} = \vec{\mathcal{I}}) \sum_\iota \Pr(X_\iota^{(\vec{\mathcal{I}})} = 0) \\
&\quad + \sum_{\vec{\mathcal{I}}:|\vec{I}\geq.8a|} \Pr(\mathcal{I} = \vec{\mathcal{I}}) \sum_\iota \Pr(X_\iota^{(\vec{\mathcal{I}})} = 0) \\
&\leq \frac{1}{2} + \frac{1}{2n^3} \\
&\leq \frac{2}{3}.
\end{aligned}$$

□

We conclude with our approximation algorithm for hop-constrained group Steiner tree.

Theorem 59. *There is a poly-time algorithm which given an instance of h-hop-constrained group Steiner tree returns a tree T such that $w(T) \leq O(\log^3 n \log N \log^2 k \cdot \mathrm{OPT})$ and $\mathrm{hop}_H(g_i, r) \leq O(h \cdot \log^3 n)$ with high probability for every g_i.*

Proof. We use the algorithm described above. We first argue our cost bound. By Theorem 58, Theorem 57 and a union bound over all phases we have that with high probability $w_{T_{jj^*}}(H_{jj^*}) \leq O(\log^2 n \log N \log^2 k \cdot \mathrm{OPT})$ for every j. Since each of the at most $10 \log n$ shortest paths we

buy cost at most OPT, we can apply the properties of our embeddings and sum up over all phases, to see that $w_G(T) \leq O(\log^3 n \log N \log^2 k \cdot \text{OPT})$.

Next, notice that T satisfies the stated hop bounds by Theorem 46.

To see why T connects all groups notice that in a given phase j where we have a unconnected groups and $a \geq 10 \log n$, we newly connect at least $\frac{a}{10 \log n}$ groups. Thus, the number of unconnected groups after this iteration is at most $(1 - \frac{1}{10 \log n})a$. Assume for the sake of contradiction that the number of unconnected groups after $1000 \log n \log k$ phases is more than $10 \log n$. Then, we have that the number of unconnected groups after $1000 \log n \log k$ phases is at most,

$$k \cdot \left(1 - \frac{1}{10 \log n}\right)^{1000 \log n \log k} \leq k \cdot e^{-\log k} \leq 1$$

a contradiction. □

4.5.3 Hop-Constrained k-Steiner Tree

In this section we give a bicriteria approximation algorithm for the hop-constrained k-Steiner tree problem and a relaxed version of it. Notably, unlike most other problems to which we apply our embeddings, the hop-constrained version of k-Steiner tree and its relaxed version have previously been studied under the name "Shallow-Light k-Steiner Trees" [116, 126]. While both our techniques and prior work yield bicriteria approximation algorithms with polylogarithmic guarantees, our techniques are simpler (i.e., follow directly from the theory of partial tree embeddings), and for the relaxed problem give the best known cost approximation (at the cost of a worse hop stretch than known results).

Hop-Constrained k-Steiner Tree: Let $G = (V, E, w_G)$ be a weighted graph. Given a terminal set $S \subseteq V$, an integer $1 \leq k \leq |S|$, and a root $r \in V$ we want to find the connected subgraph $H \subseteq G$ which minimizes $w_G(H) := \sum_{e \in E(H)} w_G(e)$ that contains r and has at least k terminals (i.e., $|V(H) \cap S| \geq k$). In the hop-constrained version, we are additionally given a hop constraint $h \geq 1$ and need to satisfy that the hop diameter of H is at most h (i.e., $\text{hop}_H(u, v) \leq h$ for all $u, v \in V(H)$). In the relaxed version, we must find an h-hop-diameter subgraph H with at least $k/8$ terminals, but we compare our cost to the optimal solution on k-terminals whose value we denote OPT.

Related work: Hajiaghayi et al. [116] solve the relaxed hop-constrained k-Steiner tree with $O(\log n)$ hop stretch and $O(\log^3 n)$ cost approximation. They show that the relaxed and non-relaxed problems are equivalent up to a $O(\log k)$ factor in the cost and they use a black-box reduction to reduce the relaxed problem to a new problem (without hop constraints) called the "buy-at-bulk k-Steiner tree problem" (which we do not define here), achieving a $O(\log^4 n)$ cost approximation and $O(\log^2 n)$ hop approximation. Khani and Salavatipour [126] improve the hop-constrained k-Steiner tree guarantee to $O(\log n)$ hop stretch and $O(\log^2 n)$ cost approximation by improving the buy-at-bulk k-Steiner tree cost approximations.

Algorithm for the h-hop relaxed k-Steiner tree problem: We sample an $8h$-hop partial tree embedding T of G with root r, hop stretch $O(\log^3 n)$, worst-case distance stretch $O(\log^2 n)$, and exclusion probability $\frac{1}{4}$ via Theorem 46. Let H' be the optimal $\frac{k}{8}$-Steiner tree solution (without hop constraints and containing the root) on T with the terminal set $V(T) \cap S$, which can be found

with a standard (poly-time) dynamic programming algorithm. We return $H := \bigcup_{e \in E(H')} T_e^G$, i.e., the projection of H' back to G.

Lemma 60. *There is a poly-time algorithm for relaxed hop-constrained k-Steiner tree which produces a solution H that contains r, at least $k/8$ terminals, and has hop diameter $O(\log^3 n \cdot h)$. With constant probability, H satisfies $w_G(H) \leq O(\log^2 n \cdot \mathrm{OPT})$.*

Proof. Suppose that H^* is the optimal solution of weight $w_G(H^*) = \mathrm{OPT}$ and hop diameter at most h. Furthermore, let $S_{\mathrm{OPT}} := S \cap V(H^*)$ be the set of terminals in the optimal solution. Since T was sampled from a distribution with exclusion probability $1/4$, we have that $\mathsf{E}[|V(T) \cap S_{\mathrm{OPT}}|] = k/4$, hence we have $\Pr[|V(T) \cap S_{\mathrm{OPT}}| \geq k/8] \geq \frac{k/4 - k/8}{k - k/8} = 1/7$. Furthermore, we can find a "projection" $F^* = T(H^*, h) \subseteq T$ of H^* to the tree T (using Theorem 48) where $w_G(F^*) \leq 4\alpha \cdot \mathrm{OPT}$ and $|V(F^*) \cap S| = |V(T) \cap S_{\mathrm{OPT}}| \geq k/8$ with constant probability. Therefore, there exists an optimal solution F^* on T solving the (un-hop-constrained) $k/8$-Steiner tree problem with value at most $4\alpha \cdot \mathrm{OPT}$. Hence $w_G(H') \leq 4\alpha \cdot \mathrm{OPT}$ with constant probability.

Finally, we project H' back to G to obtain the output H and deduce that the hop diameter is $O(\log^3 n \cdot h)$ and $w_G(H) \leq 4\alpha \cdot OPT$ with constant probability (since the partial embeddings are dominating, i.e., $w_G(T_e^G) \leq w_T(e)$), as required. □

Algorithm for the (non-relaxed) h-hop k-Steiner tree problem: We can easily boost the $k/8$-Steiner algorithm from constant probability to high probability by repeating it $O(\log n)$ times and taking the minimum solution. We can then apply this high-probability algorithm $O(\log k)$ times and take the union of the results as our solution. Note that in each iteration, a $1/8$-fraction of the remaining terminals will be added to the final solution and so by a standard covering argument $O(\log k)$ iterations suffice to cover all terminals. The hop diameter does not increase during the iterations since all solutions share a common root r, while the cost of our solution increases by $O(\mathrm{OPT} \cdot \log^2 n)$ in each iteration. This proves the following result.

Lemma 61. *There is a poly-time algorithm for the k-Steiner tree problem which outputs a solution H with $w_G(H) \leq O(\log^2 n \cdot \log k) \cdot \mathrm{OPT}$ and hop diameter at most $O(\log^3 n \cdot h)$ with high probability.*

4.5.4 Hop-Constrained Oblivious Network Design

In this section we give a bicriteria approximation algorithm for hop-constrained oblivious network design which generalizes the hop-constrained version of many well-studied oblivious network design problems.

(Hop-Constrained) Oblivious Network Design: In the oblivious network design problem we are given a weighted graph $G = (V(G), E(G), w_G)$, a monotone subadditive function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ (satisfying $f(a+b) \leq f(a) + f(b)$ and $f(a) \leq f(a+b)$ for all $a, b \geq 0$). For each pair of vertices $(u, v) \in V \bigtimes V$ we need to select a single path P_{uv} between u and v. An adversary then reveals a set of k demand pairs $\{(s_i, t_i)\}_{i=1}^k$, inducing our solution $\bigcup_i P_i$ where $P_i := P_{s_i t_i}$. For an edge $e \in E(G)$ let $\ell_e := |\{i : P_i \ni e\}|$ be the "load" of our induced solution: that is, the number of paths passing through e. The cost of our induced solution is $\sum_{e \in E(G)} w_G(e) \cdot f(\ell_e)$. In hop-constrained oblivious network design we are additionally given a hop constraint $h \geq 1$ and require that each P_{uv} satisfies $\mathrm{hop}(P_{uv}) \leq h$ for all i. (Non-oblivious) network design is identical but we are shown the demand pairs before we must fix our paths. We emphasize that OPT in this

section will refer to the cost of the optimal hop-constrained network design problem; that is, the cost of the optimal solution which knows the demand pairs *before* it fixes its paths.

Related Work: [105] introduced the oblivious network design problem as an oblivious generalization of many well-studied problems such as Steiner forest and buy-at-bulk network design [15].

Algorithm: We sample $5h$-hop partial tree embeddings with trees $T_1, T_2, \ldots, T_{O(\log n)}$ by applying Theorem 46 to G with $\epsilon = .1$ and an arbitrary root. Next, we fix a T_j and do the following. We let $\mathcal{S}_j := \{(u, v) : u, v \in V(T_j)\}$ be the pairs of vertices embedded by T_j. For each pair $(u, v) \in \mathcal{S}_j$, we let $P'_{uv} \leftarrow (T_j)_{uv}$ be the unique simple path between u and v in T_j (possibly overwriting a previous P'_{uv}). Lastly, we let $P_{uv} := (T_j)^G_{uv}$ be P'_{uv}'s projection onto G.

Our proof will use the usual partial tree-embedding template along with the idea of mixture metrics which we introduced in Section 4.3.3. We let $\mathcal{I}_j$ be the

Lemma 62. *Given the revealed demand pairs, let $\mathcal{I}_j$ be the indices of all pairs with vertices in T_j and let ALG_j be the optimal cost of the network design problem (without hop constraints) on the tree T_j with demand pairs $\mathcal{I}_j$. Then $\mathrm{ALG}_j \leq O(\log^3 n \cdot \mathrm{OPT})$ with high probability.*

Proof. We introduce some notation. We write $T := T_j$, $\mathrm{ALG} := ALG_j$, and $\mathcal{I} := \mathcal{I}_j$ for brevity. Let $\alpha = O(\log^2 n)$ be the worst-case distance stretch of T (as in Theorem 46). For a tree edge $e = \{u, v\} \in E(T)$ where u is the parent of v, we define $S_e \subseteq V(T)$ as the set of nodes in the subtree of e (including v, but excluding u). Furthermore, given a set $W \subseteq V(T)$ we write $\mathrm{unmatched}(W) := |\{i \in \mathcal{I} : \{s_i, t_i\} \cap W = 1\}|$ as the number of "unmatched terminals" in W.

Remember that T is well-separated (as stipulated by Theorem 46), meaning each root-to-leaf has edges of weights that are decreasing powers of 2. Specifically, $w_T(e) = 2^p$ for some p. From now on we fix a value p.

We define "mixture weights" w'_G on the graph G: for an edge $e \in E(G)$ we define $w'_G(e) := 1/h + w_G(e) \cdot \frac{5\alpha}{2^p}$. By $d'_G : V(G) \times V(G) \to \mathbb{R}_{\geq 0}$ we denote the distances induced by w'_G. Furthermore, given a node $v \in V(G)$, radius $r > 0$, we define the "ball" $B'_G(r, v)$ as the set of all (fractional) edges f such that there exists a path Q in G between v and (any endpoint of) f with $w'_G(Q) \leq r$. Here we consider an edge to be subdivided into infinitesimal pieces, hence one can talk about a fractional portion of an edge—while this can be made fully formal by considering a version of G where an edge $e \in E(G)$ is subdivided into $\xi \to \infty$ pieces of weight $w_G(e)/\xi$, hops $1/\xi$, and mixture weight $w'_G(e)/\xi$, we choose to keep it slightly informal for simplicity. Next, we extend $B'_G(r, W) := \bigcup_{v \in W} B'_G(r, v)$ for a subset $W \subseteq V(G)$.

For each tree edge $e \in E(T)$ of weight $w_T(e) = 2^p$ we associate the ball $A_e := B'_G(2, S_e)$ to e. We show that the balls assigned to different edges e and f of the same weight 2^p are disjoint. Clearly, since T is well-separated, there is no root-leaf path that contains both e and f. Note that $2^p \leq d_T(u, v) \leq \alpha \cdot d^{(5h)}_G(u, v)$ for $u \in S_e$ (subtree below e) and $v \notin S_e$, implying $d^{(5h)}_G(S_e, S_f) \geq \frac{2^i}{\alpha}$. Therefore, any path Q between (a node in) S_e and (a node in) S_f has $\mathrm{hop}(Q) > 5h$ or $w_G(Q) \leq \frac{2^i}{\alpha}$. Since $w'_G(Q) = \mathrm{hop}(Q)/h + w_G(Q) \cdot \frac{5\alpha}{2^p}$ we have that $w'_G(Q) \geq 5$. We conclude that $d'_G(S_e, S_f) \geq 5$. Therefore, the balls A_e and A_f (of radius 2) are disjoint.

Let $\{P^*_i\}_i$ be an optimal hop-constrained solution on G. Fix a tree edge $e \in E(T)$ of weight $w_T(e) = 2^p$. With a slight abuse of notation, let $P^*_i \cap A_e$ be the sub-path of P^*_i from its start in S_e to the first node (in the infinitesimal graph) not contained in the ball A_e. We claim that for each

unmatched terminal $s_i \in S_e$ (or t_i, but we will WLOG assume it is s_i) it holds $w_G(P_i^* \cap A_e) \geq \frac{2^p}{5\alpha}$. First, since by definition $s_i \in S_e$ and $t_i \notin S_e$, then $d'_G(s_i, t_i) \geq 5$ (as in the previous paragraph). Therefore, since the radius of A_e is $2 < 5$ we have that $w'_G(P_i^* \cap A_e) \geq 2$. Furthermore, let $Q := P_i^* \cap A_e$, and we have $2 \leq w'_G(Q) = \mathrm{hop}(Q)/h + w_G(Q) \cdot \frac{5\alpha}{2^p} \leq 1 + w_G(Q) \cdot \frac{5\alpha}{2^p}$ giving us $w_G(Q) \geq \frac{2^p}{5\alpha}$ as claimed.

Continuing to fix $\{P_i^*\}_i$ and $e \in E(T)$ with $w_T(e) = 2^p$, we define ℓ_f^* to be the load of $\{P_i^*\}_i$ on any edge $f \in E(G)$ and then define $\mathrm{OPT}(p, e) := \sum_{f \in A_e} w_G(f) \cdot \ell_f^*$. Furthermore, since for each p the balls $\{A_e\}_{e \in E(T), w_T(e) = 2^p}$ are disjoint, we conclude that (for each p)

$$\mathrm{OPT} = \sum_{f \in E(G)} w_G(f)\ell_f^* \geq \sum_{e \in E(T), w_T = 2^p} \sum_{f \in A_e} w_G(f)\ell_f^* = \sum_{e \in E(T), w_T(e) = 2^p} \mathrm{OPT}(p, e).$$

Fix tree edge $e \in E(T)$ with $w_T(e) = 2^p$. As proven before, for each i where s_i or t_i are an unmatched terminal in S_e we have that $w_G(P_i^* \cap A_e) \geq \frac{w_T(e)}{5\alpha}$ and all such $\{P_i^* \cap A_e)$ are contained within the same set of edges A_e. By subadditivity, the contribution to $\mathrm{OPT}(p, e)$ is minimized when the paths $\{P'_i \cap A_e\}_{\text{unmatched } i \text{ in } S_e}$ are identical for all i, leading to a contribution of at least $\mathrm{OPT}(p, e) \geq \frac{2^p}{5\alpha} f(\mathrm{unmatched}(S_e))$.

Note that there are at most $O(\log n)$ values for p since T is well-separated and the aspect ratio of G is $\mathsf{poly}(n)$. Therefore, the value of the algorithm can be written as

$$\begin{aligned}\mathrm{ALG} &\leq \sum_{p=1}^{O(\log n)} \sum_{e \in E(T), w_T(e) = 2^p} 2^p f(\mathrm{unmatched}(S_e)) \leq \sum_{p=1}^{O(\log n)} \sum_{e \in E(T), w_T(e) = 2^p} O(\alpha)\mathrm{OPT}(p, e) \\ &= O(\alpha) \sum_{p=1}^{O(\log n)} \mathrm{OPT}.\end{aligned}$$

Therefore, $\mathrm{ALG} \leq O(\alpha \log n \cdot \mathrm{OPT}) = O(\log^3 n \cdot \mathrm{OPT})$. □

We conclude the analysis of the above algorithm.

Theorem 63. *There is a poly-time algorithm for h-hop-constrained oblivious network design which with high probability outputs a selection of paths $\{P_{uv}\}_{(u,v) \in V \times V}$ each with at most $O(\log^3 n \cdot h)$ hops such that the induced solution for any set of demand pairs has cost at most $O(\log^4 n \cdot \mathrm{OPT})$.*

Proof. First, since we sample from a partial tree distribution $\mathcal{D}$ with $\Pr_{(T,\cdot) \sim \mathcal{D}}[v \in V(T)] \geq 0.9$, we conclude using a standard Chernoff and union bound that both nodes of each demand pair (s_i, t_i) appear in at least one partial tree embedding T_j, with high probability. Therefore, with high probability, each pair is assigned a valid path at least once. Furthermore, since the distribution is h-hop with hop stretch $O(\log^3 n)$, we have that $\mathrm{hop}(P_i) \leq O(\log^3 n)h$.

Let ALG_j be the cost of the (unique) solution in the tree T_j with respect to all demand pairs $\mathcal{I}_j$. We have $\mathrm{ALG}_j \leq O(\log^3 n \cdot \mathrm{OPT})$ by Theorem 62. Since for each j we purchase a subset of the paths corresponding to the cost ALG_j solution on T_j and since projecting such a solution back to G only decreases its cost and there are $O(\log n)$ trees T_j, we conclude that our cost is at most $O(\log^4 n)\mathrm{OPT}$. □

4.6 h-Hop Copy Tree Embeddings

In the preceding section we showed how many hop-constrained problems reduce to solving $O(\log n)$ non-hop-constrained problems on trees by sampling $O(\log n)$ partial tree embeddings as in Theorem 46. In this section, we show how to compactly represent $O(\log n)$ draws from Theorem 46 in a single "h-hop copy tree embedding" to reduce many hop-constrained problems to a *single* non-hop-constrained problem on a tree. This will allow us to give several online algorithms for hop-constrained problems and improve the approximation guarantees for offline hop-constrained group Steiner tree which we gave in the preceding section. Roughly, a copy tree embedding is a tree embedding where a vertex maps to many copies of itself.

To precisely define our h-hop copy tree embeddings we will use the idea of a copy mapping (Theorem 2) from the previous chapter, a function ϕ from the vertex set V to subsets of V'. $\phi(v)$ should be understood as the "copies" of v in V'.

Definition 64 (h-Hop Copy Tree Embedding)**.** *Let $G = (V, E, w)$ be a weighted graph with some distinguished root $r \in V$ and fix $h \geq 1$. An h-hop copy tree embedding with cost stretch α and hop stretch β consists of a weighted rooted tree $T = (V', E', w')$, a copy mapping $\phi : V \to 2^{V'}$ and monotone edge mappings $\pi_{G\to T} : 2^E \to 2^{E'}$ and $\pi_{T\to G} : 2^{E'} \to 2^E$ such that:*

1. ***α-Approximate Cost Preservation**: For any $F \subseteq E$ we have $w(F) \leq \alpha \cdot w'(\pi_{G\to T}(F))$ and for any $F' \subseteq E'$ we have $w'(F') \leq w(\pi_{T\to G}(F'))$.*
2. ***β-Approximate h-Hop-Connectivity Preservation:** For all $F \subseteq E$ and $u, v \in V$ if $\text{hop}_F(u, v) \leq h$, then $\phi(u), \phi(v) \subseteq V'$ are connected via $\pi_{G\to T}(F)$. Symmetrically, for all $F' \subseteq E'$ and $u', v' \in V'$ if u' and v' are connected by F' then $\text{hop}_{\pi_{T\to G}(F')}(\phi^{-1}(u'), \phi^{-1}(v')) \leq \beta h$.*
3. ***Root mapping:** $\phi(r) = \{r'\}$ where r' is the root of T.*

We say an h-hop copy tree embedding is efficient if ϕ, $\pi_{G\to T}$ and $\pi_{T\to G}$ are all deterministically poly-time computable.

A simple consequence of our main embedding theorem (Theorem 46), along with our projection mapping theorem (Theorem 48), shows that we can compute h-hop copy tree embeddings with poly-log hop and cost stretch.

Theorem 65. *Given $h \geq 1$, there is a poly-time algorithm which given any weighted graph $G = (V, E, w)$ and root vertex $r \in V$ computes an efficient h-hop copy tree embedding from G into some weighted and rooted tree T with hop stretch $O(\log^3 n)$ and cost stretch $O(\log^3 n)$ with high probability. Further, T is well-separated and satisfies $|\phi(v)| \leq O(\log n)$ for all v.*

Proof. We compute our h-hop copy tree embedding as follows. First, apply Theorem 46 to compute $\Theta(\log n)$ $8h$-hop partial tree embeddings $T_1, T_2, \ldots$ with exclusion probability $\epsilon = .01$, root r, worst-case distance stretch $O(\log^2 n)$ and hop stretch $O(\log^3 n)$. We let our copy tree embedding T be the result of identifying r in each of our T_i as the same vertex; that is, $V(T) = \{r\} \sqcup \bigsqcup_i V(T_i) \setminus \{r\}$. Let ϕ map from a vertex in V to its copies in the natural way. We let $\pi_{G\to T}(F) := \bigcup_i T_i(G[F], h)$ where $T_i(G[F], h)$ is as defined in Theorem 47. We also let $\pi_{T\to G}(F') := \bigcup_{e\in F'} T_e^G$. Notice that our mappings are monotone by definition. Also notice that our tree satisfies root mapping by construction. Lastly, our tree satisfies $O(\log^3 n)$-approximate cost preservation as an immediate consequence of Theorem 46, Theorem 48 and the fact that we

sampled $\Theta(\log n)$ trees. Our tree satisfies $O(\log^3)$-approximate h-hop-connectivity preservation by Theorem 46 and a Chernoff bound which shows that any u, v have some copies that appear in the same T_i with high probability. The well-separatedness and number of copies of each vertex trivially follow from Theorem 46. □

4.7 Applications of h-Hop Copy Tree Embeddings

In this section we apply our h-hop copy tree embeddings to give approximation algorithms for the hop-constrained versions of group Steiner tree, online group Steiner tree, group Steiner forest and online group Steiner forest. As in our previous section, we let OPT stand for the optimal value of the relevant hop-constrained problem throughout.

4.7.1 Hop-Constrained Group Steiner Tree

Here we give an approximation algorithm for hop-constrained group Steiner tree which improves over our result in Section 4.5.2 by using h-hop copy trees. For a problem definition and related work see Section 4.5.2.

Algorithm: We first sample an h-hop copy tree T with high probability as in Theorem 65 with mappings $\pi_{G\to T}$, $\pi_{T\to G}$ and ϕ and root r. Next, consider the group Steiner tree instance on T whose root is the one vertex in $\phi(r)$ and whose groups are $(g'_i)_i$ where $g'_i := \bigcup_{v\in g_i} \phi(v)$. We apply Theorem 57 to this group Steiner tree problem to get back tree $T' \subseteq T$ and let $H' := \pi_{T\to G}(T'_t)$ be its projection onto G. We let our solution H be a BFS tree of H' rooted at r where edges have unit cost in the BFS.

The properties of our h-hop copy tree embeddings immediately show that this algorithm is competitive.

Theorem 66. *There is a poly-time algorithm which with high probability given an instance of h-hop-constrained group Steiner tree returns a tree T such that $w(T) \le O(\log^3 n \log N \log k \cdot \mathrm{OPT})$ and $\mathrm{hop}_H(g_i, r) \le O(h \cdot \log^3 n)$ for every g_i.*

Proof. We use the above algorithm. The root mapping and β-approximate h-hop connectivity preservation properties of our h-hop copy tree embedding along with the feasibility of T' guarantees that H' connects g_i to r for every $i \le t$ with at most $O(h \cdot \log^3 n)$ hops; it follows that H does the same. The bound on our cost comes from combining the fact that $\pi_{G\to T}(H^*)$ is feasible for the group Steiner tree instance we solve on T where H^* is the optimal solution on G, the $O(\log^3 n)$-approximate cost preservation of our h-hop tree embedding and the cost guarantee of Theorem 67. □

4.7.2 Online Hop-Constrained Group Steiner Tree

In this section we show that our h-hop copy trees reduce solving online hop-constrained group Steiner tree to online group Steiner tree on a tree; we then apply a known solution for online group Steiner tree on trees.

Online Hop-Constrained Group Steiner Tree: Online group Steiner tree is defined in Section 2.0.1. Online hop-constrained group Steiner tree is the same as group Steiner tree but we

are also given a hop-constraint $h \geq 1$ and the optimal solution as well as each of our trees must satisfy $\text{hop}_{T_t}(r, g_i) \leq h$ for every $i \leq t$. We assume that the possible groups revealed by the adversary are known ahead of time as otherwise this problem is known to admit no sub-polynomial approximations [8] and let k be the number of possible groups revealed by the adversary.

We recall a result of [8] that solves online group Steiner tree on trees.

Theorem 67 ([8])**.** *There is a poly-time algorithm for online group Steiner tree on trees with expected competitive ratio* $O(\log^2 n \log k)$.

We then combine this result with our h-hop copy trees to get our algorithm for online hop-constrained group Steiner tree.

Algorithm: We first sample an h-hop copy tree T with high probability as in Theorem 65 with root r and mappings $\pi_{G\to T}$, $\pi_{T\to G}$ and ϕ. Next, consider the online group Steiner tree instance on T whose root is the one vertex in $\phi(r)$ where group $g'_t := \bigcup_{v\in g_t} \phi(v)$ is revealed in time step t. Apply Theorem 67 to maintain solution T'_t for this problem in time step t on T and let our solution in time step t on G be $T_t := \pi_{T\to G}(T'_t)$.

The properties of our h-hop copy tree embeddings immediately give the desired properties of our algorithm.

Theorem 68. *There is a poly-time algorithm for online hop-constrained group Steiner tree which with high probability maintains a solution* $\{T_t\}_t$ *that is* $O(\log k \cdot \log^5 n)$*-cost-competitive in expectation where* $\text{hop}_{T_t}(r, g_i) \leq O(\log^3 n \cdot h)$ *for all* t *and* $i \leq t$.

Proof. We use the above algorithm. The root mapping and β-approximate h-hop connectivity preservation properties of our h-hop copy tree embedding along with the feasibility of T'_t guarantees that our solution T_t connects g_i to r for every $i \leq t$ with at most $O(h \cdot \log^3 n)$ hops. The bound on our cost comes from combining the fact that $\pi_{G\to T}(T^*_t)$ is feasible for the group Steiner tree instance we solve on T in time step t where T^*_t is the optimal solution on G in time step t, the $O(\log^3 n)$-approximate cost preservation of our h-hop tree embedding and the cost guarantee of Theorem 67. □

4.7.3 Hop-Constrained Group Steiner Forest

As both group Steiner tree and Steiner forest are special cases of it, the group Steiner forest is one of the most general studied connectivity problems; for this reason it is also sometimes referred to as the "generalized connectivity problem."

Hop-Constrained Group Steiner Forest: Group Steiner forest is defined in Section 2.0.1. In hop-constrained group Steiner forest we are additionally given a hop bound $h \geq 1$ and for every i we must ensure that $\text{hop}_F(s_i, t_i) \leq h$ for some $s_i \in S_i$ and $t_i \in T_i$. We will use the shorthand $\text{hop}_F(S_i, T_i) := \min_{s_i\in S_i, t_i\in T_i} \text{hop}_F(s_i, t_i)$.

We use our copy tree embeddings to reduce hop-constrained group Steiner forest to the tree case. We then apply an algorithm of Naor et al. [149] which shows how to solve group Steiner forest on trees.

Theorem 69 ([149])**.** *There is a poly-time algorithm for group Steiner forest on trees of depth* d *which achieves an approximation ratio of* $O(d \cdot \log^2 n \log k)$ *with high probability.*

Algorithm: Formally, we first apply Theorem 65 to sample a copy tree T with depth $O(\log n)$, an arbitrary root and mappings ϕ, $\pi_{G\to T}$ and $\pi_{T\to G}$. Next, we apply Theorem 69 to solve the group Steiner forest on T with pairs to be connected $(S'_1, T'_1), \ldots (S'_k, T'_k)$ where $S'_i := \bigcup_{v\in S_i} \phi(v)$ and symmetrically $T'_i := \bigcup_{v\in T_i} \phi(v)$. Let F' be the resulting solution on T. We return as our solution $F := \pi_{T\to G}(F')$.

Theorem 70. *There is a poly-time algorithm for h-hop-constrained group Steiner forest which with high probability returns a solution F such that $w(F) \leq O(\mathrm{OPT}\cdot\log^6 n \log k)$ and $\mathrm{hop}_F(S_i, T_i) \leq O(h \cdot \log^3 n)$ for every i.*

Proof. We use the above algorithm. A polynomial runtime is immediate from Theorem 65 and Theorem 69. The hop guarantee is immediate from the correctness of the algorithm of Theorem 69 and the properties of $\pi_{T\to G}$ as given in Theorem 65. To see the bound on cost, notice that $\pi_{G\to T}(F^*)$ is feasible for the group Steiner forest problem that we solve on T and has cost at most $O(\log^3 n \cdot \mathrm{OPT})$ by the properties of $\pi_{G\to T}$ as specified in Theorem 65 where F^* is the optimal solution to the input group Steiner forest problem on G. The bound then follows from the fact that T is well-separated and so has depth at most $O(\log n)$. □

4.7.4 Online Hop-Constrained Group Steiner Forest

In this section we give our algorithm for online hop-constrained group Steiner forest. It follows, almost immediately, from our h-hop copy tree embeddings and a result of [149] for online group Steiner forest on trees.

Online Hop-Constrained Group Steiner Forest: Online group Steiner forest is defined in Section 2.0.1. Online hop-constrained group Steiner forest is the same as online group Steiner forest but we are given a hop constraint h and we must ensure that for each t and $i \leq t$ there is some $s_i \in S_i$ and $t_i \in T_i$ such that $\mathrm{hop}_{F_t}(s_i, t_i) \leq h$. We assume that the possible pairs revealed by the adversary are known ahead of time as otherwise this problem is known to admit no sub-polynomial approximations [8] and let k be the number of possible pairs.

We use our copy tree embeddings to reduce online hop-constrained group Steiner forest to the tree case. We then apply an algorithm of Naor et al. [149] which shows how to solve group Steiner forest on trees.

Theorem 71 ([149])**.** *There is a randomized poly-time algorithm for group Steiner forest on trees of depth d with expected competitive ratio $O(d \cdot \log^3 n \log k)$.*

Algorithm: Formally, we first apply Theorem 65 to sample a copy tree T with depth $O(\log n)$, an arbitrary root and mappings ϕ, $\pi_{G\to T}$ and $\pi_{T\to G}$. Next, we apply Theorem 71 to solve the online group Steiner forest on T with pairs to be connected $(S'_1, T'_1), \ldots (S'_t, T'_t)$ in time step t where $S'_i := \bigcup_{v\in S_i} \phi(v)$ and symmetrically $T'_i := \bigcup_{v\in T_i} \phi(v)$. Let F'_t be the resulting solution on T in time step t. In time step t we return as our solution on G the subgraph $F_t := \pi_{T\to G}(F'_t)$.

We conclude with the properties of our online group Steiner forest algorithm.

Theorem 72. *There is a poly-time algorithm for online h-hop-constrained group Steiner forest which with high probability maintains a solution $\{F_t\}_t$ that is $O(\log^7 n \log k)$-cost-competitive in expectation where $\mathrm{hop}_{F_t}(S_i, T_i) \leq O(h \cdot \log^3 n)$ for all t and $i \leq t$.*

Proof. We use the above algorithm. A polynomial runtime is immediate from Theorem 65 and Theorem 71. The hop guarantee is immediate from the correctness of the algorithm of Theorem 71 and the properties of $\pi_{T\to G}$ as given in Theorem 65. To see the bound on cost, notice that $\pi_{G\to T}(F_t^*)$ is feasible for the group Steiner forest problem that we solve on T and has cost at most $O(\log^3 n \cdot \mathrm{OPT})$ by the properties of $\pi_{G\to T}$ as specified in Theorem 65 where F_t^* is the optimal solution to the input group Steiner forest problem on G in time step t. The bound then follows from the fact that T is well-separated and so has depth at most $O(\log n)$. □

4.8 Conclusion and Future Work

In this chapter we showed that, while far from any metric, hop-constrained distances are well-approximated by partial tree metrics. We used this fact to develop new embeddings for hop-constrained distances which we then used to give the first bicriteria (poly-log, poly-log) approximation algorithms for many classic network design problems.

We conclude by giving directions for future work. Reducing the stretch in our embeddings, or proving lower bounds stronger than those immediately implied by the FRT lower bounds is our main open question. A recent work [85] has made significant progress in this direction. Another point to note is that we lose an $O(\log n)$ in the hop stretch when moving from partial tree metrics to partial tree embeddings. This loss does not seem to have an analogue in the (non-partial) tree embedding setting and it is not clear if such a loss is necessary.

Moreover, while tree embeddings have proven useful for many network design problems, there are many other problems such as k-server [20], metrical task systems [26] and requirement cuts [145] where tree embeddings enabled the first poly-log approximations. Thus, while the focus of this chapter has been on the hop-constrained versions of network design problems, we expect that our embeddings will prove useful for the hop-constrained versions of many of these other problems.

Lastly, as we discussed at the end of Section 4.4, our h-hop partial tree embeddings are built on the worst-case stretch guarantees of our partial metrics; it would be interesting if it were possible to construct embeddings based on the expected stretch guarantees of our partial metrics. Such a result would immediately give several randomized algorithms for hop-constrained problems with low expected cost.

4.9 Deferred Proofs of Section 4.3

Lemma 73. *For any hop constraint $h \geq 1$, distance stretch α, hop stretch β and any $L > 1$, there exists a graph $G = (V, E, w)$ with aspect ratio L such that if a metric $\tilde{d}$ approximates $d_G^{(h)}$ with distance stretch α and hop stretch β then $\alpha(\beta h + 1) \geq L$.*

Proof. Set $k := \beta h + 1$ and consider the path graph with vertices $v_0, v_1, \ldots, v_k$ where the edges have a uniform weight of 1 (and all other edges have length L). Note that $\tilde{d}(v_i, v_{i+1}) \leq \alpha \cdot d^{(h)}(v_i, v_{i+1}) = \alpha$. Applying the triangle inequality k times gives $\tilde{d}(v_0, v_k) \leq \alpha k$. However, $\alpha k \geq \tilde{d}(v_0, v_k) \geq d^{(h\beta)}(v_0, v_k) = L$, giving us that $\alpha(\beta h + 1) \geq L$. □

Lemma 74. *Given any graph $G = (V, E, w)$ with aspect ratio L and a distance stretch α and hop*

stretch β satisfying $\alpha(\beta h + 1) \geq L$, we have that $\alpha \cdot d_G$ approximates $d_G^{(h)}$ with distance stretch α and hop stretch β.

Proof. Set $\tilde{d}(u, v) := \alpha \cdot d_G(u, v)$ where d_G is the standard shortest-path metric on G. It remains to check that $\tilde{d}(u, v)$ satisfies the requirements of Theorem 31. The right hand side of the inequality in Theorem 31 clearly holds since $d_G(u, v) \leq d_G^{(h)}(u, v)$ implies that $\tilde{d}(u, v) = \alpha d_G(u, v) \leq \alpha d_G^{(h)}(u, v)$. We now argue the left hand side, i.e., $\tilde{d}(u, v) \geq d_G^{(\beta h)}(u, v)$, by considering two cases: on the one hand, if $d_G(u, v) > \beta h$ then $\tilde{d}(u, v) = \alpha d_G(u, v) \geq \alpha(\beta h + 1) \geq L \geq d_G^{(\beta h)}(u, v)$. On the other hand, if $d_G(u, v) \leq \beta h$, this value must come from an unconstrained shortest-path with hop distance (and length) of at most βh in which case $d_G^{(\beta h)}(u, v) = d_G(u, v)$ and therefore $\tilde{d}(u, v) = \alpha \cdot d_G(u, v) \geq d_G(u, v) = d_G^{(\beta h)}(u, v)$ as desired. □

Lemma 75. *Let $G = (V, E, w)$ be a weighted graph with padding parameter ρ_{pad}. For any hop constraint $h > 0$, weight diameter $b > 0$, and exclusion probability $\gamma > 0$, there exists a distribution $\mathcal{C}$ over partial vertex partitions where for every $C = C_1 \sqcup \ldots \sqcup C_k$ in the support of $\mathcal{C}$:*

1. ***Hop-Constrained Diameter:*** *$d_G^{(h)}(u, v) \leq b$ for $i \in [k]$ and $u, v \in C_i$;*
2. ***Hop-Constrained Paddedness:*** *$d_G^{(h\frac{\gamma}{2\rho_{\text{pad}}})}(u, v) \geq b \cdot \frac{\gamma}{2\rho_{\text{pad}}}$ for every $u \in C_i$ and $v \in C_j$ where $i \neq j$.*

And:

3. ***Exclusion probability:*** *$\Pr_{C \sim \mathcal{C}}[v \notin \bigcup_{i \in [k]} C_i] \leq \gamma$ for each $v \in V$ where $C = C_1 \sqcup \ldots \sqcup C_k$;*
4. ***Path preservation:*** *$\Pr_{C \sim \mathcal{C}}[V(P)$ is broken in $C] \leq (\text{hop}(P)/h + w(P)/b) \cdot \rho_{\text{pad}}$ for each path P.*

Proof. Let d' be the mixture metric of G with hop scale h and weight scale b and let $\Delta := 2\rho_{\text{pad}}$. We first take a (distribution over) $(\rho_{\text{pad}}, \Delta)$-padded decompositions $C' = C'_1 \sqcup C'_2 \sqcup \ldots \sqcup C'_k$ using d' as the underlying metric. Next, we construct $C_i \subseteq C'_i$ by starting with $C_i := C'_i$ and removing all vertices $v \in C'_i$ where $B_{d'}(v, 2\gamma) \not\subseteq C'_i$. Now $\Pr[v \notin \bigcup_{i \in [k]} C_i] \leq \frac{2\gamma \cdot \rho_{\text{pad}}}{\Delta} \leq \gamma$ for each vertex $v \in V$, as stipulated by (3).

Fix $u, v \in C_i$. Since every C'_i has d'-diameter at most Δ, there exists a sequence of edges $P = (e_1, e_2, \ldots, e_\ell)$ between u and v whose d'-length is at most Δ. Therefore:

$$\Delta \geq \sum_{i=1}^{\ell} \left(\frac{\Delta}{h} + \frac{\Delta \cdot w(e_i)}{b} \right) = \frac{\Delta \cdot \text{hop}(P)}{h} + \frac{\Delta \cdot w(P)}{b}.$$

In other words, $\text{hop}(P) \leq h$ and $w(P) \leq b$, implying that $d_G^{(h)}(u, v) \leq b$ for any $u, v \in C'_i$. Therefore, the same claim holds for $u, v \in C_i \subseteq C'_i$, giving (1).

For $u \in C_i$ and $v \in C_j$ where $i \neq j$ we argue that $d^{(\gamma h/\Delta)}(u, v) > \gamma b/\Delta$, i.e. (2). Suppose for the sake of contradiction that $d_G^{(\gamma h/\Delta)}(u, v) \leq \gamma b/\Delta$. It follows that there exists a path P with $\text{hop}(P) \leq \gamma h/\Delta$ and $w(P) \leq \gamma b/\Delta$. However, the d'-length of P is at most $\frac{\text{hop}(P)\Delta}{h} + \frac{w(P)\Delta}{b} \leq 2\gamma$. Thus, we have contradicted how we constructed C_i from C'_i. Hence $d_G^{(\gamma h/2\rho_{\text{pad}})}(u, v) > \gamma b/2\rho_{\text{pad}}$ since $\Delta = 2\rho_{\text{pad}}$.

Finally, consider a path P from u to v and let $\delta' := \mathrm{hop}(P)\Delta/h + w(P)\Delta/b$. If P is broken in $C_1 \sqcup \ldots \sqcup C_k$ then $B_{d'}(u, \delta') \not\subseteq P$. We therefore have (4), namely

$$\Pr[P \text{ is broken in } C_1 \sqcup \ldots \sqcup C_k] \leq \Pr[B_{d'}(u, \delta') \not\subseteq P] < \frac{\delta' \cdot \rho_{\mathrm{pad}}}{\Delta} = \frac{\delta'}{2} \leq (\mathrm{hop}(P)/h + w(P)/b) \cdot \rho_{\mathrm{pad}}.$$

□

Part II

New Primitives for Graph Decompositions

Chapter 5

Length-Constrained Flows

5.1 Introduction

In the previous chapter we saw several examples of how to incorporate both hop distances and metric distances into compact representations of graphs, namely tree embeddings. In this chapter, we further explore the interplay between classic graph structures and a notion of distances, namely length-constrained flows. These algorithms, in turn, give a variety of new results in expander decompositions, a well-studied graph decomposition.

Throughput and latency are two of the most fundamental quantities in a communication network. Given node sets S and T, throughput measures the rate at which bits can be delivered from S to T while the worst-case latency measures the maximum time it takes for a bit sent from S to arrive at T. Thus, a natural question in network optimization is:

How can we achieve high throughput while maintaining a low latency?

If we imagine that each edge in a graph incurs some latency and edges in a graph can only support limited bandwidth, then achieving high throughput subject to a latency constraint reduces to finding a large collection of paths that are both short and non-overlapping. One of the simplest and most well-studied ways of formalizing this is the maximal edge-disjoint paths problems (henceforth we use h-length to mean length *at most* h).

> **Maximal Edge-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of h-length edge-disjoint S to T paths $\mathcal{P}$ such that any h-length S to T path shares an edge with at least one path in $\mathcal{P}$.

The simplicity of the maximal edge-disjoint paths problem has made it a crucial primitive in numerous algorithms. For example, algorithms for maximal edge-disjoint paths are used in approximating maximum matchings [140] and computing expander decompositions [59, 160]. While efficient randomized algorithms are known for maximal edge-disjoint paths in the CONGEST model of distributed computation [45, 140], no deterministic CONGEST algorithms are known. Indeed, the existence of such algorithms was stated as an open question by Chang and Saranurak [45].

Of course, a maximal collection of routing paths need not be near-optimal in terms of cardinality and so a natural extension of the above problem is its *maximum* version.

Maximum Edge-Disjoint Paths: Given graph $G = (V, E)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of h-length edge-disjoint S to T paths.

While this problem and its variants have received considerable attention [29, 39, 129], it is unfortunately known to suffer from strong hardness results: the above problem has an $\Omega(h)$ integrality gap and is $\Omega(h)$-hard-to-approximate under standard complexity assumptions in the directed case [18, 107]. Indeed, as observed in several works [11, 113, 129], working in the presence of latency bounds in the form of a length constraint can make otherwise tractable problems computationally infeasible and render otherwise structured objects poorly behaved.

In large part, the above problems are common primitives because their solutions are special cases of a more general class of routing schemes that are central to distributed computing. Namely, they are special cases of length-constrained flows.

Maximum Length-Constrained Flow: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of h-length S to T paths $\mathcal{P}$ and a value $f_P \geq 0$ for $P \in \mathcal{P}$ where $\sum_{P \ni a} f_P \leq 1$ for every $a \in A$ and $\sum_P f_P$ is maximized.

In several formal senses, length-constrained flows are *the* problem that describes how to efficiently communicate in a network. Haeupler et al. [109] showed that, up to poly-log factors, the maximum length-constrained flow gives the minimum makespan of multiple unicasts in a network, even when (network) coding is allowed. Even stronger, the "best" length-constrained flow gives, up to poly-log factors, the optimal running time of a CONGEST algorithm for numerous distributed optimization problems, including minimum spanning tree (MST), approximate min-cut and approximate shortest paths [114]. Despite the key role these flows play in distributed computing, there are currently no known distributed (or even parallel!) algorithms for computing them. The need for algorithms for length-constrained flows is further highlighted by the fact that many classic optimization problems (such as matchings) reduce to length-constrained flows with small values of h.

Thus, in summary a well-studied class of routing problems aims to capture both latency and throughput concerns. These problems are known to serve as important algorithmic primitives as well as complete characterizations of the distributed complexity of many problems. However, even the simplest of these problems—maximal edge-disjoint paths—lacks good deterministic CONGEST algorithms; even worse virtually nothing is known about parallel or distributed algorithms for the maximum version of this problem and its fractional generalization, length-constrained flows.

5.1.1 Our Contributions

We give the first efficient algorithms for computing these objects in several models of computation.

Algorithms for Length-Constrained Flows

Given a digraph with n nodes and m arcs, our main theorem shows how to *deterministically* compute h-length flows that are $(1-\epsilon)$-approximate in $\tilde{O}(\mathsf{poly}(h, \frac{1}{\epsilon}))$ parallel time with m processors and $\tilde{O}(\mathsf{poly}(h, \frac{1}{\epsilon}) \cdot 2^{O(\sqrt{\log n})})$ distributed CONGEST time. We additionally give a randomized

CONGEST algorithm that succeeds with high probability and runs in time $\tilde{O}(\mathsf{poly}(h, \frac{1}{\epsilon}))$. Our distributed algorithms for length-constrained flows algorithms can be contrasted with the best distributed algorithms for (non-length-constrained) flows which run in $(D + \sqrt{n}) \cdot n^{o(1)}$ time [94], nearly matching an $\tilde{\Omega}(D + \sqrt{n})$ lower bound of Sarma et al. [161].[1]

As an immediate consequence of our parallel algorithms we also get deterministic sequential algorithms running in $\tilde{O}(m \cdot \mathsf{poly}(h, \frac{1}{\epsilon}))$ time. To our knowledge this is the first algorithm for sequentially computing length-constrained flows with a linear dependence on m; a recent work of Altmanová et al. [10] gave sequential algorithms with a dependence on m of $O(m^2)$ (but with a smaller polynomial dependence on h and $\frac{1}{\epsilon}$).

Additionally, our algorithms satisfy three desirable properties.

1. **General Capacities, Lengths and Multi-Commodity:** Our algorithms work for general arc capacities (i.e. connection bandwidths), general lengths (i.e. connection latencies) and multi-commodity flow variants.
2. **Dual Solution:** Not only do our algorithms compute a primal solution for length-constrained flows but they also compute a certifying dual solution; a so-called moving cut, which is an object of algorithmic utility in its own right; see, e.g. [57, 114].
3. **Optimal Integrality:** The flows we compute are "as integral as possible." In particular, for constant $\epsilon > 0$ (and unit capacities) they are a convex combinations of $\tilde{O}(h)$ sets of arc-disjoint paths. No near-optimal h-length flow can be a convex combination of $o(h)$ such sets since, by an averaging argument, this would violate the aforementioned $\Omega(h)$ integrality gap.

We give a more formal description of our results for length-constrained flows in Section 5.3.

Applications of our Length-Constrained Flow Algorithms

We give several applications of our length-constrained flow algorithms.

Maximal and Maximum Edge-Disjoint Paths First, as an almost immediate corollary of our length-constrained flow algorithms, we derive the first deterministic CONGEST algorithms for maximal edge-disjoint paths and essentially-optimal parallel and distributed algorithms for the maximum edge-disjoint paths problem as well as for many variants of these problems. The former result settles the open question of Chang and Saranurak [45]. The latter result crucially relies on the optimal integrality of our length-constrained flows and matches known hardness-of-approximation results. See Section 5.12 for details.

Simpler Distributed Expander Decompositions Deterministically. As a consequence of our maximal edge-disjoint paths algorithms, we are able to greatly simplify known distributed algorithms for deterministically computing expander decompositions.

We refer the reader to Chang and Saranurak [45] for a more thorough overview of the area, but provide a brief synopsis here. An (ϵ, ϕ) expander decomposition removes an ϵ fraction of edges from a graph so as to ensure that each remaining connected component has conductance at least ϕ.

[1]We use $\tilde{O}$ notation to suppress dependence on $\mathsf{poly}(\log n)$ factors, "with high probability" to mean with probability at least $1 - \frac{1}{\mathsf{poly}(n)}$ and D for the diameter of the input graph.

Expander decompositions have led to many recent exciting breakthroughs, including in solving linear systems [162], unique games [14, 154, 164], minimum cut [124], and dynamic algorithms [146].

Chang and Saranurak [45] gave the first deterministic CONGEST algorithms for constructing expander decompositions. However, while much of expander decomposition construction reduces to maximal edge-disjoint paths, the authors observe:

> *In the deterministic setting, we are not aware of an algorithm that can [efficiently] solve [maximal disjoint paths]... [A solution to this problem would] simplify our deterministic expander decomposition and routing quite a bit. [45]*

As a result of the lack of such algorithms, the authors employ significant technical work-arounds.

Our deterministic CONGEST algorithms for maximal edge-disjoint paths when plugged into the results of Chang and Saranurak [45] greatly simplify deterministic distributed algorithms for expander decompositions. See Section 5.13 for further details.

Bipartite b-Matching. Using our length-constrained flow algorithms, we give the first efficient $(1-\epsilon)$-approximations for bipartite b-matching in CONGEST. b-matching is a classical problem in combinatorial optimization which generalizes matching where we are given a graph $G = (V, E)$ and a function $b : V \to \mathbb{Z}_{>0}$. Our goal is to assign integer values to edges so that each vertex v has at most $b(v)$ assigned value across its incident edges. b-matching and its variants have been extensively studied in distributed settings [19, 37, 87, 87, 88, 133]. A standard folklore reduction which replaces vertex v with $b(v)$ non-adjacent copies and edge $e = \{u, v\}$ with a bipartite clique between the copies of u and v reduces b-matching to matching but requires overhead $\max_{\{u,v\}\in E} b(u) \cdot b(v)$ to run in CONGEST. Thus, the non-trivial goal here is a CONGEST algorithm whose running time does not depend on b. Currently, the best algorithm in CONGEST is a $(\frac{1}{2} - \epsilon)$-approximation of Fischer [87] running in time $\tilde{O}(\mathsf{poly}(\log \frac{1}{\epsilon}))$.

Similarly to classical matching, it is easy to reduce bipartite b-matching to an $O(1)$-length flow problem. Thus, applying our algorithms for length-constrained flows and some of the flow rounding techniques we develop in this chapter allows us to give the first $(1-\epsilon)$-approximation for b-matching in bipartite graphs running in CONGEST time $\tilde{O}(\mathsf{poly}(\frac{1}{\epsilon}) \cdot 2^{O(\sqrt{\log n})})$. Our algorithms are deterministic though similar results even for the randomized setting do not seem to be known. See Section 5.14 for further details.

Length-Constrained Cutmatches. Lastly, our results allow us to give the first efficient constructions of length-constrained cutmatches. Informally, an h-length cutmatch with congestion γ is a collection of h-length γ-congestion paths between two vertex subsets along with a moving cut that shows that adding any more h-length paths to this set would incur congestion greater than γ. See Section 5.15 for details.

A recent work [115] uses our algorithms for length-constrained cutmatches to give the first efficient constructions of a length-constrained version of expander decompositions. This work, in turn, uses these constructions to, among other things, give CONGEST algorithms for many problems including MST, $(1+\epsilon)$-min-cut and $(1+\epsilon)$-shortest paths that are guaranteed to run in sub-linear rounds as long as such algorithms exist on the input network.

5.2 Chapter-Specific Notation and Conventions

Before moving on to a formal statement of length-constrained flows, moving cuts and our results we introduce some notation and conventions. Suppose we are given a digraph $D = (V, A)$.

Digraph Notation. We will associate three functions with the arcs of D. We clarify these here.

1. **Lengths:** We will let $l = \{l_a\}_a$ be the *lengths* of arcs in A. These lengths will be input to our problem and determine the lengths of paths when we are computing length-constrained flows. Throughout this chapter we imagine each l_a is in $\mathbb{Z}_{>0}$. Informally, one may think of l as giving link latencies. We will assume l_a is always $\mathsf{poly}(n)$.
2. **Capacities:** We will let $U = \{U_a\}_a$ be the capacities of arcs in A. These capacities will specify a maximum amount of flow (either length-constrained or not) that is allowed over each arc. Throughout this chapter we imagine each U_a is in $\mathbb{Z}_{\geq 0}$ and we let $U_{\max}$ give $\max_a U_a$. We will assume $U_{\max}$ is $\mathsf{poly}(n)$. Informally, one may think of U as link bandwidths.
3. **Weights:** We will let $w = \{w_a\}_a$ stand for the weights of arcs in A. These weights will be given by our moving cut solutions. Throughout this chapter each w_a will be in $\mathbb{R}_{>0}$.

Unlike the rest of this thesis, here it will be convenient to treat a path $P = ((v_1, v_2), (v_2, v_3), \ldots)$ as series of consecutive arcs in A (all oriented consistently towards one endpoint). For any one of these weighting functions $\phi \in \{l, U, w\}$, we will let $d_\phi(u, v)$ give the minimum value of a path in D that connects u and v where the value of a path P is $\phi(P) := \sum_{a \in P} \phi(a)$. That is, we think of $d_\phi(u, v)$ as the distance from u to v with respect to ϕ. We will refer to paths which minimize w as lightest paths (so as to distinguish them from e.g. shortest paths with respect to l).

We let $\delta^+(v) := \{a : a = (v, u)\}$ and $N^+(v) := \{u : (v, u) \in A\}$ give the out arcs and out neighborhoods of vertex v. $\delta^-(v) := \{a : a = (u, v)\}$ and $N^- := \{u : (u, v) \in A\}$ are defined symmetrically. We let $\mathcal{P}(u, v)$ be all simple paths between u and v and for $W, W' \subseteq V$, we let $\mathcal{P}(W, W') := \bigcup_{w \in W, w' \in W'} \mathcal{P}(w, w')$ give all paths between vertex subsets W and W'.

Given sources $S \in V$ and sinks $T \in V$, we say that D is an S-T DAG if $\delta^-(v) = \emptyset$ iff $v \in S$ and $\delta^+(v) = \emptyset$ iff $v \in T$. We say that such an S-T DAG is an h-layer DAG if the vertex set V can be partitioned into $h + 1$ layers $S = V_1 \sqcup V_2 \sqcup \ldots \sqcup V_{h+1} = T$ where any arc $a = (u, v)$ is such that $u \in V_i$ and $v \in V_j$ for some i and $j > i$. We say that D has diameter at most d if in the graph where we forget about arc directions in D every pair of vertices is connected by a path of at most d edges. Notice that the diameter of an h-layer S-T DAG might be much larger than h, for example, when S and T are large sets of vertices.

(Non-Length Constrained) Flow Notation and Conventions. We will make extensive use of non-length constrained flows and so clarify our notation for such flows here.

Given a DAG $D = (V, A)$ with capacities U we will let a flow f be any assignment of non-negative values to arcs in a where f_a gives the value that f assigns to a and $f_a \leq U_a$ for every a. If it is ever the case that $f_a > U_a$ for some a, we will explicitly state that this "flow" does not respect capacities. We say that f is an integral flow if it assigns an integer value to each arc. We let $f(A') := \sum_{a \in A'} f_a$ for any $A' \subseteq A$. We define the deficit of a vertex v as $\text{deficit}(f, v) := |\sum_{a \in \delta^+(f,v)} f_a - \sum_{a \in \delta^-(v)} f_a|$. We will let $\text{supp}(f) := \{a : f_a > 0\}$ give the

support of flow f.

Given desired sources S and sinks T, we let deficit$(f) := \sum_{v \notin S \cup T}$ deficit(f, v) be the total amount of flow produced but not at S plus the amount of flow consumed but not at T; likewise, we say that a flow f is an S-T flow if $f$deficit$(f) = 0$. We let s.t. $(f) = \bigcup_{s \in S} f(\delta^+(s))$ be the amount of flow delivered by an S-T flow f and we say that f is α-approximate if s.t. $(f) \geq \alpha \cdot$ s.t. (f^*) where f^* is the S-T flow that maximizes s.t. . We say that f is α-blocking for $\alpha \in [0, 1]$ if for every path from S to T there is some $a \in P$ where $f_a \geq \alpha \cdot U_a$. We say that a 1-blocking flow is blocking. We say that flow f' is a subflow of f if $f'_a \leq f_a$ for every a.

Given a maximum capacity of $U_{\max}$, we may assume that every flow f is of the form $f = \sum_i f^{(i)}$ where $(f^{(i)})_a \in \{0, 2^{\log(U_{\max})-i}\}$ for every a and i; that is, a given flow can always be decomposed into its values on each bit. We call $f^{(i)}$ the ith bit flow of f and call the decomposition of f into these flows be the bitwise decomposition of f.

Length-Constrained Notation. Given a length function l, vertices $u, v \in V$ and length constraint $h \geq 1$, we let $\mathcal{P}_h(u, v) := \{P \in \mathcal{P}(u, v) : l(P) \leq h\}$ be all paths between u and v which have length at most h. For vertex sets W and W', we let $\mathcal{P}_h(W, W') := \{P \in \mathcal{P}(W, W') : l(P) \leq h\}$. If G also has weights w then we let $d_w^{(h)}(u, v) := \min_{P \in \mathcal{P}_h(u,v)} w(P)$ give the minimum weight of a length at most h path connecting u and v. For vertex sets $W, W' \subseteq V$ we define $d_w^{(h)}(W, W') := \min_{P \in \mathcal{P}_h(W,W')} w(P)$ analogously. As mentioned an h-length path is a path of length at most h.

Parallel and Distributed Models. Throughout this chapter the parallel model of computation we will make use of is the EREW PRAM model [123]. Here we imagine that we are given some number of processors as well as shared random access memory; every memory cell can be read or written to by only one processor at a time.

The distributed model we will make use of is the CONGEST model, defined as follows [152]. The network is modeled as a graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. Communication is conducted over discrete, synchronous rounds. During each round each node can send an $O(\log n)$-bit message along each of its incident edges. Every node has an arbitrary and unique ID of $O(\log n)$ bits, first only known to itself. The running time of a CONGEST algorithm is the number of rounds it uses. We will slightly abuse terminology and talk about running a CONGEST algorithm in digraph D; when we do so we mean that the algorithm runs in the (undirected) graph G which is identical to D but where we forget the directions of arcs. In this work, we will assume that if an arc a has capacity U_a then we allow nodes to send $O(U_a \cdot \log n)$ bits over the corresponding edge, though none of our applications rely on this assumption.[2]

5.3 Length-Constrained Flows, Moving Cuts and Main Result

We proceed to more formally define a length-constrained flow, moving cuts and our main result which computes them. While we have defined length-constrained flows in Section 5.1 for unit capacities, it will be convenient for us to formally define length-constrained flows for general lengths and capacities in terms of a relevant linear program (LP). We do so now.

[2]We only make use of this assumption once and only make use of it in our deterministic algorithms (in Theorem 109). Furthermore, we do not require this assumption if the underlying digraph is a DAG.

Suppose we are given a digraph $D = (V, A)$ with arc capacities U, lengths l and specified source and sink vertices S and T. A maximum S to T flow in D in the classic sense can be defined as a collection of paths between S and T where each path receives some value and the total value incident to an edge does not exceed its capacity. This definition naturally extends to the length-constrained setting where we imagine we are given some length constraint $h \geq 1$ and define a length-constrained flow as a collection of S to T paths each of length at most h where each such path P receives some some value f_P. Additionally, these values must respect the capacities of arcs. More precisely, we have the following LP with a variable f_P for each path $P \in \mathcal{P}_h(s, t)$.

$$\max \sum_{P \in \mathcal{P}_h(S,T)} f_P \quad \text{s.t.} \qquad \text{(Length-Constrained Flow LP)}$$
$$\sum_{P: a \in P} f_P \leq U_a \qquad \forall a \in A$$
$$0 \leq f_P \qquad \forall P \in \mathcal{P}_h(s, t)$$

For a length-constrained flow f, we will use the shorthand $f(a) := \sum_{P \ni a} f_P$ and $\text{supp}(f) := \{P : f_P > 0\}$ to give the support of f. We will let s.t. $(f) := \sum_{P \in \mathcal{P}_h(s,t)} f_P$ give the value of f. An h-length flow, then, is simply a feasible solution to this LP.

Definition 76 (h-Length Flow)**.** *Given digraph $D = (V, A)$ with lengths l, capacities U and vertices $S, T \subseteq V$, an h-length S-T flow is any feasible solution to Length-Constrained Flow LP.*

With the above definition of length-constrained flows we can now define moving cuts as the dual of length-constrained flows. In particular, taking the dual of the above LP we get the moving cut LP with a variable w_a for each $a \in A$.

$$\min \sum_{a \in A} U_a \cdot w_a \quad \text{s.t.} \qquad \text{(Moving Cut LP)}$$
$$\sum_{a \in P} w_a \geq 1 \qquad \forall P \in \mathcal{P}_h(S, T)$$
$$0 \leq w_a \qquad \forall a \in A$$

An h-length moving cut is simply a feasible solution to this LP.

Definition 77 (h-Length Moving Cut)**.** *Given digraph $D = (V, A)$ with lengths l, capacities U and vertices $S, T \subseteq V$, an h-length moving cut is any feasible solution to Moving Cut LP.*

We will use f and w to stand for solutions to Length-Constrained Flow LP and Moving Cut LP respectively. We say that (f, w) is a feasible pair if both f and w are feasible for their respective LPs and that (f, w) is $(1 \pm \epsilon)$-approximate for $\epsilon \geq 0$ if the moving cut certifies the value of the length-constrained flow up to a $(1 - \epsilon)$; i.e. if $(1 - \epsilon) \sum_a U_a \cdot w_a \leq \sum_P f_P$.

We clarify what it means to compute (f, w) in CONGEST. When we are working in CONGEST we will say that f is computed if each vertex v stores the value $f_a(h') := \sum_{P \in \mathcal{P}_{h,h'}(s,a,t)} f_P$ for every a incident to v and $h' \leq h$. Here, we let $\mathcal{P}_{h,h'}(s, a, t)$ be all paths in $\mathcal{P}_h(s, t)$ of the form $P' = (a_1, a_2, \ldots a, b_1, b_2, \ldots)$ where the path $(a, b_1, b_2, \ldots)$ has length exactly h' according to l. We say moving cut w is computed if each vertex v knows the value of w_a for its incident arcs.

Likewise, we imagine that each node initially knows the capacities and lengths of its incident arcs.

With the above notions, we can now state our main results which say that one can efficiently compute a feasible pair (f, w) in parallel and distributedly. In the following we say f is integral if f_P is an integer for every path in $\mathcal{P}_h(S, T)$. The notable aspect of our results is the polynomial dependence on h and $\frac{1}{\epsilon}$; the polynomials could be optimized to be much smaller.

Given a digraph $D = (V, A)$ with capacities U, lengths l, length constraint $h \geq 1$, $\epsilon > 0$ and source and sink vertices $S, T \subseteq V$, one can compute a feasible h-length flow, moving cut pair (f, w) that is $(1 \pm \epsilon)$-approximate in:

1. Deterministic parallel time $\tilde{O}(\frac{1}{\epsilon^9} \cdot h^{17})$ with m processors
2. Randomized CONGEST time $\tilde{O}(\frac{1}{\epsilon^9} \cdot h^{17})$ with high probability;
3. Deterministic CONGEST time $\tilde{O}\left(\frac{1}{\epsilon^9} \cdot h^{17} + \frac{1}{\epsilon^7} \cdot h^{16} \cdot (\rho_{CC})^{10}\right)$.

Also, $f = \eta \cdot \sum_{j=1}^{k} f_j$ where $\eta = \tilde{\Theta}(\epsilon^2)$, $k = \tilde{O}\left(\frac{h}{\epsilon^4}\right)$ and each f_j is an integral h-length S-T flow. All of our algorithms compute and separately store each f_j. The above result immediately gives the deterministic parallel and randomized CONGEST algorithms running in time $\tilde{O}(\mathsf{poly}(h, \frac{1}{\epsilon}))$ mentioned in Section 5.1.1. For our deterministic CONGEST algorithms, ρ_{CC} in the above gives the quality of the optimal deterministic CONGEST cycle cover algorithm. We formally define this parameter in Section 5.5 but for now we simply note that $\rho_{CC} \leq 2^{O(\sqrt{\log n})}$ by known results [118, 151]. Applying this bound on ρ_{CC} gives deterministic CONGEST algorithms running in time $\tilde{O}(\mathsf{poly}(h, \frac{1}{\epsilon}) \cdot 2^{O(\sqrt{\log n})})$. If ρ_{CC} is shown to be $\mathsf{poly}(\log n)$, we immediately would get an $\tilde{O}(\mathsf{poly}(h, \frac{1}{\epsilon}))$ time deterministic algorithm for solving $(1 - \epsilon)$-approximate h-length flow in CONGEST. Also, as mentioned in Section 5.1.1, k in the above result is optimal up to $\tilde{O}(1)$ factors by the results of Guruswami et al. [107] and Baier et al. [18].

5.4 Intuition and Overview of Approach

Before moving on to details, we give an overview of our strategy for computing length-constrained flows. For simplicity, we will assume that the capacity U_a of arc a is 1 in this section.

5.4.1 Using Lightest Path Blockers for Multiplicative Weights

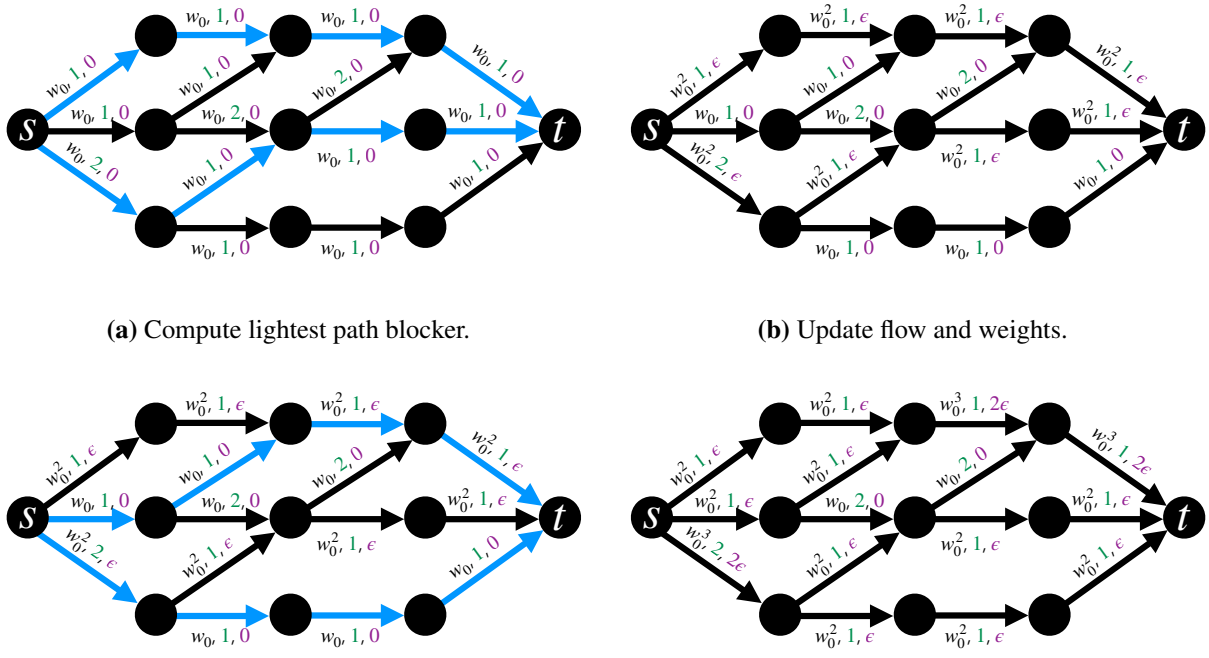
Computing a length-constrained flow, moving cut pair is naturally suggestive of the following multiplicative-weights-type approach. We initialize our moving cut value w_a to some very small value for every a. Then, we find a lightest h-length path from S to T according to w, send some small ($\approx \epsilon$) amount of flow along this path and multiplicatively increase the value of w on all arcs in this path by $\approx (1 + \epsilon)$. We repeat this until S and T are at least 1 apart according to $d_w^{(h)}$. This general idea was recently used by Altmanová et al. [10] to compute length-constrained flows sequentially and is an adaptation of ideas of Garg and Könemann [91].

The principle shortcoming of using such an algorithm for our setting is that it is easy to construct examples where there are polynomially-many arc-disjoint h-length paths between S and T and so we would clearly have to repeat the above process at least polynomially-many times until S and T are at least 1 apart according to $d_w^{(h)}$. This is not consistent with our goal of $\mathsf{poly}(h)$ complexities since h may be much smaller than n. To solve this issue, we use an algorithm similar to the

above but instead of sending flow along a single path at a time, we send it along a large batch of arc-disjoint paths.

What can we hope to say about how long such an algorithm takes to make S and T at least 1 apart according to $d_w^{(h)}$? If it were the case that every lightest (according to w) h-length path from S to T shared an arc with some path in our batch of paths then after each batch we would know that we increased $d_w^{(h)}(S,T)$ by some non-zero amount. However, there is no way to lower bound this amount; in principle we might only increase $d_w^{(h)}(S,T)$ by some tiny $\epsilon' > 0$. To solve this issue we find a batch of arc-disjoint paths which have weight essentially $d_w^{(h)}(S,T)$ but which share an arc with every h-length path with weight at most $(1+\epsilon) \cdot d_w^{(h)}(S,T)$. Thus, when we increment weights in our batch we know that all *near*-lightest h-length paths have their weights incremented and this, in turn, allows us to lower bound the rate at which $d_w^{(h)}(S,T)$ increases and therefore to argue that our algorithm completes quickly.

Thus, in summary we repeatedly find a batch of arc-disjoint h-length paths between S and T which have weight about $d_w^{(h)}(S,T)$; these paths satisfy the property that every h-length path from S to T with weight at most $(1+\epsilon) \cdot d_w^{(h)}(S,T)$ shares an edge with at least one of these paths; we call such a collection an h-length $(1+\epsilon)$-lightest path blocker. We then send a small amount of flow along these paths and multiplicatively increase the weight of all incident edges, appreciably increasing $d_w^{(h)}(S,T)$. We repeat this until our weights form a feasible moving cut. See Figure 5.1.



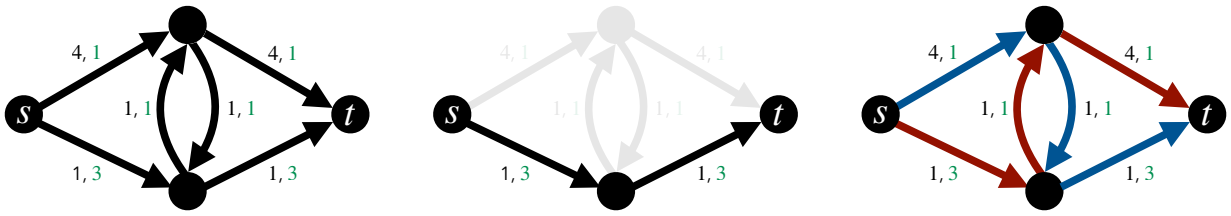
(a) Compute lightest path blocker.

(b) Update flow and weights.

(c) Compute lightest path blocker.

(d) Update flow and weights.

Figure 5.1: An illustration of the first two iterations of our multiplicative-weights-type algorithm where $h = 5$, $S = \{s\}$ and $T = \{t\}$ and capacities are all 1. Each arc is labelled with the value we multiply its initial weight by (initialized to $w_0 := 1 + \epsilon$) then length then flow. Our h-length shortest path blockers are in blue.



(a) Digraph D. **(b)** Lightest paths of D. **(c)** Lightest 5-length paths of D.

Figure 5.2: A digraph D with $S = \{s\}$ and $T = \{t\}$ where the 5-length lightest S-T paths do not induce a DAG. 5.2a gives D where each arc is labeled with its weight (in black) and length (in green). 5.2b shows how all lightest S-T paths have weight 2 and induce a DAG. 5.2c shows how the two 5-length lightest S-T paths (in blue and red) have weight 6 and induce a digraph with a cycle.

5.4.2 Length-Weight Expanded DAG to Approximate h-Length Lightest Paths

The above strategy relies on the computation of h-length lightest path blockers. Without the presence of a weight constraint computing such an object easily reduces to computing an integral blocking S-T flow on an h-layer S-T DAG. Specifically, consider the problem of computing a collection of paths from S to T so that every lightest S to T path shares an arc with one path in this collection. It is easy to see that all lightest paths between S and T induce an h'-layer S-T DAG where h' is the minimum weight of a path between S and T. One can then consider this DAG and compute an integral blocking S-T flow in it—i.e. a maximal arc-disjoint collection of h'-length S-T paths. By maximality of the flow, the paths corresponding to this flow will guarantee that every h'-length S to T path shares an arc with one path in this collection.

However, when we are working in the presence of both a length constraint and weight constraint computing such an object becomes significantly more tricky. Indeed, lightest paths subject to length constraints are known to be notoriously poorly behaved; not only do lightest paths subject to a length constraint not induce a metric but they are also arbitrarily far from any metric [11, 113]. As a consequence of this, all lightest paths subject to a length constraint from S to T do not induce a DAG, much less an h-layer S to T DAG; see Figure 5.2 for an example.

Our solution to this issue is to observe that, if we are allowed to duplicate vertices, then we can construct an S-T DAG with about h^2 layers that approximately captures the structure of all h-length $(1+\epsilon)$-lightest paths. Specifically, we discretize weights and then make a small number of copies of each vertex to compute a DAG $D^{(h,\lambda)}$—which we call the length-weight expanded DAG. $D^{(h,\lambda)}$ will satisfy the property that if we compute an integral blocking flow in it and then project this back into D as a set of paths $\mathcal{P}$, then $\mathcal{P}$ is *almost* a $(1+\epsilon)$-lightest path blocker. In particular, $\mathcal{P}$ will guarantee that some arc of any h-length path with weight at most $(1+\epsilon) \cdot d_w^{(h)}(S,T)$ is used by some path in $\mathcal{P}$; however, the paths of $\mathcal{P}$ may not be arc-disjoint which is required of our lightest path blockers. Nonetheless, by carefully choosing the capacities in $D^{(h,\lambda)}$, we will be able to argue that $\mathcal{P}$ is nearly arc-disjoint and these violations of arc-disjointness can be repaired with bounded loss by a "decongesting" procedure. Summarizing, these ideas reduce computing h-length $(1+\epsilon)$-lightest path blockers to computing integral blocking flows in layered S-T DAGs.

5.4.3 Deterministic Integral Blocking Flows Paths via Flow Rounding

Lastly, we describe how we compute integral blocking flows in layered S-T DAGs.

A somewhat straightforward adaptation of a *randomized* algorithms of Lotker et al. [140] solves this problem in $\tilde{O}(\text{poly}(h))$ time both in parallel and in CONGEST. This algorithm samples an integral S-T flow in D (i.e. a collection of arc-disjoint S to T paths) according to a carefully chosen distribution based on "path counts", deletes these paths and repeats. The returned solution is the flow induced by all paths that were ever deleted. Unfortunately Lotker et al. [140]'s algorithm seems inherently randomized and our goal is to solve this problem deterministically.

We derandomize the algorithm of Lotker et al. [140] in the following way. Rather than integrally sampling according to Lotker et al. [140]'s distribution and then deleting arcs that appear in sampled paths, we instead calculate the probability that an arc is in a path in this distribution and then "fractionally delete" it to this extent. We repeat this until every path between S and T has some arc which has been fully deleted. In other words, we run a smoothed version of Lotker et al. [140] which behaves (deterministically) like the algorithm of Lotker et al. [140] does in expectation. A simple counting argument shows that we need only iterate this process about h times to separate S and T. The fractional deletion values of arcs at the end of this process induce a blocking S-T flow but a blocking flow that *may be fractional*. We call this flow the "iterated path count flow."

However, recall that our goal is to compute an *integral* blocking flow in an S-T DAG. Thus, we may naturally hope to round the iterated path count flow. Indeed, drawing on some flow rounding techniques of Cohen [60], doing so is not too difficult in parallel. Unfortunately, it is less clear how to do so in CONGEST. Indeed, Chang and Saranurak [45] state:

> *...Cohen's algorithm that rounds a fractional flow into an integral flow does not seem to have an efficient implementation in CONGEST...*

Roughly, Cohen's technique relies on partitioning edges in a graph into cycles and paths and then rounding each cycle and path independently. The reason this seems infeasible in CONGEST is that the cycles and paths that Cohen's algorithm relies on can have unbounded diameter and so communicating within one of these cycles or paths is prohibitively slow. To get around this, we argue that, in fact, one may assume that these cycles and paths have low diameter *if* we allow ourselves to discard some small number of arcs. This, in turn allows us to orient these cycles and paths and use them in rounding flows. We formalize such a decomposition with the idea of a $(1-\epsilon)$-near Eulerian partition. Arguing that discarding these arcs does bounded damage to our rounding then allows us to make use of Cohen-type rounding to deterministically round the path count flow, ultimately allowing us to compute h-length $(1+\epsilon)$-lightest path blockers.

5.4.4 Overview of Chapter

In Section 5.6 we more formally define path counts. In Section 5.7 we describe how to use these path counts with randomization to compute integral blocking flows in an h-layer S-T DAG. In Section 5.9 we do the same but deterministically, employing the above flow rounding strategy and the idea of near Eulerian partitions as introduced and constructed in Section 5.8. Next, in Section 5.10 we show how to use these blocking flow primitives along with our length-weight expanded DAG to compute $(1+\epsilon)$-lightest path blockers. In Section 5.11 we formalize how to use these $(1+\epsilon)$-lightest path blockers to compute length-constrained flows and moving cuts by

applying multiplicative-weights-types arguments, thereby showing our main result.

In Section 5.12 we observe that our main result solves the aforementioned problem of Chang and Saranurak [45] by giving deterministic algorithms for many disjoint paths problems in CONGEST. We also observe that our algorithms give essentially optimal parallel and distributed algorithms for maximum arc-disjoint paths. In Section 5.13 we give more details of how our results simplify expander decomposition constructions. In Section 5.14 we give our new algorithms for bipartite b-matching based on our flow algorithms and in Section 5.15 we show how to compute length-constrained cutmatches using our main theorem. Lastly, in Section 5.17 we observe that our length-constrained flow algorithms generalize to the multi-commodity setting.

5.5 Preliminaries

Before moving on to our own technical content, we briefly review some well-known algorithmic tools and slight variants thereof (mostly for deterministic CONGEST).

5.5.1 Deterministic CONGEST Maximal and Maximum Independent Set

We will rely on deterministic CONGEST primitives for maximal and maximum independent sets. Given graph $G = (V, E)$, a subset of vertices $V' \subseteq V$ is independent if no two vertices in V' are adjacent in G. A maximal independent set (MIS) is an independent set V' such that any $w \in V \setminus V'$ is adjacent to at least one node in V'. If we are additionally given node weights $\{x_v\}_v$ where $x_v > 0$ for every v, then a maximum independent set is an independent set V' maximizing $\sum_{v \in V'} x_v$; we say that an independent set is α-approximate if its total weight is within α of that of the maximum independent set.

The following summarizes the deterministic CONGEST algorithm we will use for MIS.

Theorem 78 (Censor-Hillel et al. [42])**.** *There is a deterministic CONGEST algorithm which given a graph $G = (V, E)$ with diameter D, outputs a maximal independent set in time $O(D \cdot \log^2 n)$.*

The following gives the deterministic CONGEST algorithm we will use for maximum independent set.

Theorem 79 (Bar-Yehuda et al. [22])**.** *There is a deterministic CONGEST algorithm which given an instance of maximum independent in a graph $G = (V, E)$ with maximum degree Δ and node weights $\{x_v\}_v$, outputs a solution that is $\frac{1}{\Delta}$-approximate in time $O(\Delta + \log^* n)$.*

5.5.2 Deterministic Low Diameter Decompositions

A well-studied object in metric theory is the low diameter decomposition which is usually defined as a distribution over vertex partitions [139, 144]. For our deterministic algorithms, we will make use of a deterministic version of these objects defined as follows where $G[V_i] := (V_i, \{\{u, v\} \in E : u, v \in V_i\})$ gives the induced graph on V_i.

Definition 80 (Deterministic Low Diameter Decomposition)**.** *Given graph $G = (V, E)$, a deterministic low diameter decomposition (DLDD) with diameter d and cut fraction ϵ is a partition of V into sets $V_1, V_2, \ldots$ where:*

1. ***Low Diameter:*** *$G[V_i]$ has diameter at most d for every i;*
2. ***Cut Edges:*** *The number of cut edges is at most $\epsilon|E|$; i.e. $|\{e = (u, v) : u \in V_i \wedge v \in V_j \wedge i \neq j\}| \leq \epsilon|E|$.*

One can efficiently compute DLDDs deterministically in CONGEST as a consequence of many well-known results in distributed computing. We will use a result of Chang and Ghaffari [44] to do so.

Theorem 81. *Given a graph $G = (V, E)$ and desired diameter d, one can compute a DLDD with diameter d and cut fraction $\epsilon = \tilde{O}(\frac{1}{d})$ in deterministic CONGEST time $\tilde{O}(d)$.*

Proof. Theorem 1.2 of Chang and Ghaffari [44] states that there is a deterministic CONGEST algorithm which, given a graph $G = (V, E)$ and desired diameter d', computes a set $\bar{V} \subseteq V$ where $|\bar{V}| \leq \frac{1}{d'} \cdot |V|$ and $G[V \setminus \bar{V}]$ has connected components $C_1, C_2, \ldots, C_k$ where each C_i has diameter at most $\tilde{O}(d')$ in $\tilde{O}(d')$ rounds.

Given graph $G = (V, E)$ we can compute a DLDD in G by applying the above result in a new graph $G' = (V', E')$. For each vertex $v \in V$, G' will have a clique of $\Delta(v)$-many vertices where $\Delta(v)$ is the degree of v in G. We then connect these cliques in the natural way. More formally, to construct G' we do the following. For each v with edges to vertices $v_1, v_2, \ldots, v_{\Delta(v)}$ we create a clique of vertices $v(v_1), v(v_2), \ldots, v(v_{\Delta(v)})$. Next, for each edge $e = \{u, v\}$ in E, we add the edge $\{v(u), u(v)\}$ to G'. Observe that each vertex of G' corresponds to exactly one edge in G; that is, $v(u)$ in V' corresponds to the edge $\{u, v\} \in E$.

Next, we apply the above theorem of Chang and Ghaffari [44] to G' to get set $\bar{V}$. Let $\bar{E} \subseteq E$ be the set of edges to which these vertices correspond. We return as our solution $\bar{E}$. Observe that the size of $\bar{E}$ is

$$\begin{aligned} |\bar{E}| &\leq |\bar{V}| \\ &\leq \frac{1}{d'} \cdot |V'| \\ &= \frac{2}{d'}|E|. \end{aligned}$$

Letting $d' = \frac{1}{\tilde{\Theta}(1)} \cdot d$ for an appropriately large hidden poly-log in $\tilde{\Theta}(1)$ gives us that each component in G has diameter at most d since otherwise there would be a component in G' after deleting $\bar{v}$ with diameter more than d'. Likewise, the above gives us cut fraction at most $\tilde{O}(\frac{1}{d})$.

Simulating a CONGEST algorithm on G' on G is trivial since each vertex can simulate its corresponding clique and so the entire algorithm runs in time $\tilde{O}(d') = \tilde{O}(d)$. □

5.5.3 Sparse Neighborhood Covers

A closely related notion to low diameter decompositions is that of the sparse neighborhood cover [16]. We use the following definition phrased in terms of partitions.

Definition 82 (Sparse Neighborhood Cover)**.** *Given a simple graph $G = (V, E)$, an s-sparse k-neighborhood cover with weak-diameter d and overlap o is a set of partitions $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_s$ of V where each partition is a collection of disjoint vertex sets $V_i^{(j)} \subset V$ whose union is V, i.e.,*

$\mathcal{V}_i = \{V_i^{(1)}, V_i^{(2)}, \ldots\}$ *and:*

1. ***Weak-Diameter and Overlap:*** *Each $V_i^{(j)}$ comes with a rooted tree $T_i^{(j)}$ in G of diameter at most d that spans all nodes in $V_i^{(j)}$; Any node in G is contained in at most o trees overall.*
2. ***Neighborhood Covering:*** *For every node v its k neighborhood $B_k(v)$, containing all vertices in G within distance k of v, is fully covered by at least one cluster, i.e., $\forall v\ \exists i, j : B_k(v) \subseteq V_i^{(j)}$.*

The below summarizes the current state of the art in deterministic sparse neighborhood covers in CONGEST.

Lemma 83 ([44, 93, 159])**.** *There is a deterministic CONGEST algorithm which given any radius $r \geq 1$, computes an s-sparse r-neighborhood cover with $s, o = \tilde{O}(1)$ and diameter at most $\tilde{O}(r)$ in $\tilde{O}(r)$ time.*

Furthermore, there is a deterministic CONGEST algorithm which given an $O(k)$-bit value x_v for every v computes $x_{i,v}$ for every v and i in $\tilde{O}(r+k)$ rounds, where $x_{i,v}$ is the maximum x-value among nodes in the same cluster as v in the partition $\mathcal{V}_i$. That is, letting $\mathcal{V}_i(v)$ be the one cluster in $\mathcal{V}_i$ containing v, we have

$$x_{i,v} = \max_{u \in \mathcal{V}_i(v)} x_u.$$

5.5.4 Cycle Covers

Our flow rounding algorithm will make use of low diameter cycles. Thus, it will be useful for us to make use of some recent insights into distributely and deterministically decomposing graphs into low diameter cycles. We define the diameter of a cycle C as $|C|$ and the diameter of a collection of cycles $\mathcal{C}$ as the maximum diameter of any cycle in it. Likewise the congestion of $\mathcal{C}$ is $\max_e |\{C : e \in C\}|$.

The idea of covering a graph with low congestion cycles is well-studied [58, 118, 151] and formalized by the idea of a cycle cover.

Definition 84 (Cycle Cover)**.** *Given a simple graph $G = (V, E)$ where E_0 are all non-bridge edges[3] of G, a (d, c) cycle cover is a collection of (simple) cycles $\mathcal{C}$ in G such that:*

1. ***Covering:*** *Every $e \in E_0$ is contained in some cycle of $\mathcal{C}$;*
2. ***Low Diameter:*** $\max_{C \in \mathcal{C}} |C| \leq d$*;*
3. ***Low Congestion:*** $\max_{e \in E} |\{C : e \in C\}| \leq c$*.*

We now formally define the parameter ρ_{CC}; recall that this parameter appears in the running time of our deterministic CONGEST algorithm in our main theorem (Section 5.3).

Definition 85 (ρ_{CC})**.** *Given a deterministic CONGEST algorithm that constructs a (d, c) cycle cover in worst-case time T in graphs of diameter D, we say that the quality of this algorithm is $\max\{\frac{d}{D}, c, \frac{T}{D}\}$. We let ρ_{CC} be the smallest quality of any deterministic CONGEST algorithm for constructing cycle covers.*

[3] Recall that a bridge edge of a graph is one whose removal increases the number of connected components in the graph.

The following summarizes the current state-of-the-art in deterministic cycle cover computation in CONGEST.

Theorem 86 ([118, 151])**.** *There is a deterministic CONGEST algorithm that given a graph G with diameter D computes a (d, c) cycle cover with $d = 2^{O(\sqrt{\log n})} \cdot D$ and $c = 2^{O(\sqrt{\log n})}$ in time $2^{O(\sqrt{\log n})} \cdot D$. In other words, $\rho_{CC} \leq 2^{O(\sqrt{\log n})}$*

5.6 Path Counts for h-Layer S-T DAGs

We begin be recounting the notion idea of path counts which we will use for our randomized algorithm to sample flows and for our deterministic algorithms to compute the iterated path count flow. This idea has been used in several prior works [45, 60, 140].

Suppose we are given an h-layer S-T DAG D with capacities U. We define these path counts as follows. We define the capacity of a path as the product of its edge capacities, namely given a path P we let $U(P) := \prod_{a \in P} U_a$. Recall that we use $\mathcal{P}(S, T)$ to stand for all paths between S and T. We will slightly abuse notation and let $\mathcal{P}(v, T) = \mathcal{P}(\{v\}, T)$ and $\mathcal{P}(S, v) = \mathcal{P}(S, \{v\})$. For vertex v we let n_v^+ be the number of paths from v to T, weighted by U, namely $n_v^+ := \sum_{P \in \mathcal{P}(v,T)} U(P)$. Symmetrically, we let $n_v^- := \sum_{P \in \mathcal{P}(S,v)} U(P)$. For any arc $a = (u, v)$, we define n_a as

$$n_a := n_u^- \cdot U_a \cdot n_v^+.$$

Equivalently, we have that n_a is the number of paths in $\mathcal{P}(S, T)$ that use a weighted by capacities:

$$n_a = \sum_{P \in \mathcal{P}(S,T): a \in P} U(P).$$

It may be useful to notice that if we replace each arc a with U_a-many parallel arcs then n_a exactly counts the number of unique paths from S to T that use a in the resulting (multi) digraph. A simple dynamic-programming type algorithms that does a "sweep" from S to T and T to S shows that one can efficiently compute the path counts.

Lemma 87. *Let D be a capacitated h-layer S-T DAG. Then one can compute n_v^+ and n_v^- for every vertex v and n_a for every arc a in:*

1. *Parallel time $O(h)$ with m processors;*
2. *CONGEST time $\tilde{O}(h^2)$.*

Proof. To compute n_a it suffices to compute n_v^+ and n_v^-. We proceed to describe how to compute n_v^-; computing n_v^+ is symmetric.

First, notice that n_v^- can be described by the recurrence

$$n_v^- := \begin{cases} 1 & \text{if } v \in S \\ \sum_{(u,v) \in \delta^-(v)} U_{uv} \cdot n_u^- & \text{otherwise} \end{cases}$$

We repeat the following for iteration $i = 2, \ldots, h + 1$. Let V_i be all vertices in the ith layer of our graph. In iteration i we will compute n_v^- for every $v \in V_i$ by applying the above recurrence.

Running one of the above iterations in parallel is trivial to do in $O(1)$ parallel time with m processors, leading to the above parallel runtime. Running one iteration of this algorithm in CONGEST requires that every vertex in $v \in V_j$ for $j < i$ broadcast its n_v^-. Since $n_v^- \leq (n \cdot U_{\max})^h$ this can be done in $h\left(1 + \frac{\log U_{\max}}{\log n}\right)$ rounds of CONGEST, leading to the stated CONGEST runtime. □

5.7 Randomized Blocking Integral Flows in h-Layer DAGs

We now describe how to compute blocking integral flows in h-layer S-T DAGs with high probability by using the path counts of the previous section. This is the general capacities version of the problem described in Section 5.4.3. More or less, the algorithm we use is one of Chang and Saranurak [45] adapted to the general capacities case; the algorithm of Chang and Saranurak [45] is itself an adaptation of an algorithm of Lotker et al. [140]. We mostly include these results for the sake of completeness.

Our randomized algorithm will repeatedly sample an integral flow proportional to the path counts of Section 5.6, add this to our existing flow, reduce capacities and then repeat. We will argue that we need only iterate this process a small number of times until we get a blocking integral flow by appealing to the fact that "high degree" paths have their capacities reduced with decent probability.

One can see this as essentially running the randomized MIS algorithm of Luby [141] but with two caveats: (1) the underlying graph in which we compute an MIS has a node for every path between S and T and so has up to $O(n^h)$-many nodes; as such we cannot explicitly construct this graph but rather can only implicitly run Luby's algorithm on it; (2) Luby's analysis assumes nodes attempt to enter the MIS independently but our sampling will have some dependencies between nodes (i.e. paths) entering the MIS which must be addressed in our analysis.

More formally, suppose we are given a capacitated S-T DAG D. For a given path $P \in \mathcal{P}(S,T)$ we let Δ_P be $\sum_{P'} \prod_{a \in P' \setminus P} U_a$ be the "degree" of path P where the sum over P' ranges over all P' that share at least one arc with P and are in $\mathcal{P}(S,T)$. We let $\Delta = \max_{P \in \mathcal{P}(S,T)} \Delta_P$ be the maximum degree. Similarly, we let $\mathcal{P}_{\approx\max} := \{P : \Delta_P \geq \frac{\Delta}{2}\}$ be all paths with near-maximum degree. The following summarizes the flow we repeatedly compute; in this lemma the constant $\frac{2046}{2047}$ is arbitrary and could be optimized to be much smaller.

Lemma 88. *Given a h-layer S-T DAG D with capacities U and $\tilde{\Delta}$ satisfying $\frac{\Delta}{2} \leq \tilde{\Delta} \leq \Delta$, one can sample an integral S-T flow f where for each $P \in \mathcal{P}_{\approx\max}$ we have $\prod_{a \in P}(U_a - f_a) \leq \frac{2047}{2048} \cdot U(P)$ with probability at least $\Omega(1)$. This can be done in:*

1. *Parallel time $O(h)$ with m;*
2. *CONGEST time $\tilde{O}(h^2)$ with high probability.*

Proof. The basic idea is to have each path P sample about $U(P)/\tilde{\Delta}$ copies of itself.

More formally, we do the following. Consider the (multi) digraph D' that is created by starting with D and replacing each arc a with U_a copies. For a given path P in D' from S to T, we let Δ'_P be the number of distinct S to T paths in D' which share an arc with P. Likewise, we let $\Delta' = \max_P \Delta'_P$ where this max is taken over all S to T paths in D'. We let $\mathcal{P}'_{\approx\max}$ be all paths P for which $\Delta'_P \geq \Delta'/2$. By how we defined the degree of paths in D, if a given path P is in

$\mathcal{P}'_{\approx\max}$ then so too is its corresponding path in D in $\mathcal{P}_{\approx\max}$. Lastly, we let $N(P)$ be all paths from S to T in D' which share an arc with P other than P itself and let $N^+(P) := N(P) \cup \{P\}$.

In what follows we show how to sample a collection of arc-disjoint paths $\mathcal{P}_2$ in D' where each $P \in \mathcal{P}'_{\approx\max}$ is such that with probability at least $\frac{1}{1024}$ the set $\mathcal{P}_2 \cap N^+(P)$ is non-empty. Before doing so, we observe that this suffices to show our claim. In particular, we can construct a flow f by setting its value on arc a to be $|\{P \in \mathcal{P}_2 : a \in P\}|$. Observe that by the arc-disjointness of $\mathcal{P}_2$ and how we constructed D', f is indeed a feasible S-T flow. Moreover, we claim that for a given $\tilde{P} \in \mathcal{P}_{\approx\max}$ in D we have $\prod_{a\in\tilde{P}}(U_a - f_a) \leq \frac{1}{2}U(\tilde{P})$ with probability $\Omega(1)$. In particular, let X_P be the indicator of whether a given path P in D' from S to T is such that $N^+(P) \cap \mathcal{P}_2 = \emptyset$ so that $\mathsf{E}[X_P] \leq \frac{1023}{1024}$. Also, let $\tilde{\mathcal{P}}$ be all the paths in D' that visit the same vertices as $\tilde{P}$ in D. Then we have

$$\prod_{a\in\tilde{P}}(U_a - f_a) = \sum_{P\in\tilde{\mathcal{P}}} X_P.$$

But, looking at the expectation of this, we have

$$\begin{aligned}\mathsf{E}\left[\sum_{P\in\tilde{\mathcal{P}}} X_P\right] &\leq \sum_{P\in\tilde{\mathcal{P}}} \frac{1023}{1024} \\ &= \frac{1023}{1024} \cdot U(\tilde{P})\end{aligned}$$

Thus, by Markov's inequality we have that $\sum_{P\in\tilde{\mathcal{P}}} X_P \geq \frac{2047}{2046} \cdot \mathsf{E}\left[\sum_{P\in\tilde{\mathcal{P}}} X_P\right]$ with probability at most $\frac{2046}{2047}$ and so with probability $\Omega(1)$ we get that $\sum_{P\in\tilde{\mathcal{P}}} X_P \leq \frac{2047}{2046} \cdot \mathsf{E}\left[\sum_{P\in\tilde{\mathcal{P}}} X_P\right] \leq \frac{2047}{2048} \cdot U(\tilde{P})$.

Thus, it remains to show how to sample our collection of arc-disjoint paths $\mathcal{P}_2$ in D' where each $P \in \mathcal{P}'_{\approx\max}$ is such that with probability at least $\frac{1}{1024}$ the set $\mathcal{P}_2 \cap N^+(P)$ is non-empty. We will sample $\mathcal{P}_2$ as follows. Imagine that s initially receives $B\left(n_s^+, \frac{1}{64\tilde{\Delta}}\right)$-many balls where $B(n,p)$ is a binomial with n trials each with probability of success p. We let n_a and n_v^+ be as defined in Section 5.6 for D' where $U_{a'} = 1$ for every arc a' in D'.

When a vertex v receives a ball, it tosses it to vertex $u \in N^+(v)$ with probability n_u^+/n_v^+. As $n_v^+ = \sum_{w\in N^+(v)} n_w^+$ this induces a valid probability distribution. Let $\mathcal{P}_1$ be the (multi) set of all paths traced out by balls. We will let $\mathcal{P}_2$ be all paths in $\mathcal{P}_1$ which are arc-disjoint (in D') from all other paths in $\mathcal{P}_1$.

We first consider this process from the perspective of a single path P from S to T in D'. Specifically, notice that the probability that a ball traces out a path $P = (s = v_1, v_2, \ldots, v_{h+1} = t)$ where $s \in S$ and $t \in T$ is uniform over paths. In particular, the probability that a given ball traces out path P in D' from s to t nicely telescopes as

$$\begin{aligned}\frac{n_{v_2}^+}{n_{v_1}^+} \cdot \frac{n_{v_3}^+}{n_{v_2}^+} \cdot \cdots \cdot \frac{n_{v_{h+1}}^+}{n_{v_h}^+} &= \frac{n_{v_{h+1}}^+}{n_{v_1}^+} \\ &= \frac{1}{n_s^+}.\end{aligned}$$

Thus, each ball that starts at s traces out a uniformly random path incident to s in $\mathcal{P}(S,T)$. Applying the parameters of our binomial distribution, it follows that the expected number of times a given path P is included in $\mathcal{P}_1$ is $\frac{1}{64\cdot\tilde{\Delta}}$. Markov's inequality then shows that a given path has some copy in $\mathcal{P}_1$ with probability at most $\frac{1}{64\cdot\tilde{\Delta}} \leq \frac{1}{32\cdot\Delta}$. On the other hand, P has exactly one copy included in $\mathcal{P}_1$ with probability $\frac{1}{64\tilde{\Delta}} n_s^+ \cdot \frac{1}{n_s^+}\left(1-\frac{1}{n_s^+}\right)^{n^+-1} \geq \frac{1}{128\tilde{\Delta}}$. Thus, P has at least one copy in $\mathcal{P}_1$ with probability at least $\frac{1}{128\tilde{\Delta}} \geq \frac{1}{128\Delta}$.

We proceed to bound two simple probabilities regarding how paths are sampled. In particular, fix a path $P \in \mathcal{P}'_{\approx\max}$ in D' from S to T. Next, fix a $P' \in N^+(P)$. Then, let $\mathcal{E}_1(P')$ be the event that some copy of P' is in $\mathcal{P}_1$ and no other path in $N^+(P)$ has a copy in $\mathcal{P}_1$. Likewise, let $\mathcal{E}_2(P')$ be the event that no path in $N(P')$ is in $\mathcal{P}_1$. Notice that if $\mathcal{E}_1(P')$ and $\mathcal{E}_2(P')$ hold then we have $P' \in \mathcal{P}_2$.

- **Bounding** $\Pr(\mathcal{E}_1(P'))$. We will argue that $\Pr(\mathcal{E}_1(P')) \geq \frac{1}{256\Delta}$.

  Notice that since $N^+(P) \setminus \{P'\}$ consists of at most Δ-many paths, the expected number of copies of paths in $N^+(P) \setminus \{P'\}$ in $\mathcal{P}_1$ is at most $\frac{1}{32}$. It follows by a Markov bound that with probability at least $\frac{1}{2}$ we have $N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset$.

  Next, imagine that we condition on the event $N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset$. Conditioning on this event can only increase the probability that a ball traces out P'. Since some copy of P' is included in $\mathcal{P}_1$ with probability at least $\frac{1}{128\Delta}$ when we don't condition on this event, we conclude that

$$\begin{aligned}\Pr(\mathcal{E}_1(P')) &= \Pr(N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset) \cdot \Pr(P' \in \mathcal{P}_1 \mid N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset) \\ &\geq \Pr(N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset) \cdot \Pr(P' \in \mathcal{P}_1) \\ &\geq \frac{1}{256\Delta}.\end{aligned}$$

- **Bounding** $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P'))$. We argue that $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P')) \geq \frac{1}{2}$.

  Notice that $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P'))$ is minimized when $N^+(P)$ is of size exactly $\Delta + 1$. However, in this case we have $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P')) \geq \Pr(\mathcal{E}_2(P'))$. Thus, we conclude by a union bound that in general $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P')) \geq \Pr(\mathcal{E}_2(P')) \geq 1 - \Delta \cdot \frac{1}{32\Delta} \geq \frac{1}{2}$.

Putting these facts together and applying the fact that $P \in \mathcal{P}'_{\approx\max}$, we have that there is path in $N^+(P)$ included in $\mathcal{P}_2$ with probability at least

$$\begin{aligned}\sum_{P' \in N^+(P)} \Pr(\mathcal{E}_1(P')) \cdot \Pr(\mathcal{E}_2(P') \mid \mathcal{E}_2(P')) &\geq \sum_{P' \in N^+(P)} \frac{1}{512\Delta} \\ &\geq \frac{1}{1024}.\end{aligned}$$

as required.

It remains to argue that we can accomplish the above sampling of $\mathcal{P}_1$ and the construction of our flow f in the stated times. Constructing f from $\mathcal{P}_1$ is trivial to do in parallel and CONGEST so we focus on sampling $\mathcal{P}_1$. By Theorem 87 we can compute n_v^+ in the stated times. Passing balls to construct $\mathcal{P}_1$ and then $\mathcal{P}_2$ and constructing the above flow is trivial to do in the stated parallel time. For the CONGEST algorithm, we note that expected number of balls to cross any

one arc in D' when constructing $\mathcal{P}_1$ is at most 1 and so a Chernoff and union bound shows that with high probability we never need to transmit more than $O(\log n)$ balls across an arc in D' when constructing $\mathcal{P}_1$, with high probability. It follows that we never need to transmit more than $\tilde{O}(U_{\max})$ balls across any one arc in D. Since it suffices to just transmit the number of balls, this can be done in $\tilde{O}(\log U_{\max}) = \tilde{O}(1)$ rounds with high probability. Thus we can pass all balls from one layer to the next in $\tilde{O}(1)$ rounds of CONGEST with high probability. Lastly, constructing $\mathcal{P}_2$ from $\mathcal{P}_1$ is trivial to do in $O(h)$ rounds of CONGEST. □

Lemma 89. *There is an algorithm which, given an h-layer S-T DAG D with capacities U, computes an integral S-T flow that is blocking in:*

1. *Parallel time $\tilde{O}(h^3)$ with m processors with high probability;*
2. *CONGEST time $\tilde{O}(h^4)$ with high probability.*

Proof. Our algorithm simply repeatedly calls Theorem 88. In particular we initialize our output flow $\hat{f}$ to be 0 on all arcs and our working capacities on D to be $\hat{U} = U$. Then for each $\tilde{\Delta} = (n \cdot U_{\max})^h, (n \cdot U_{\max})^h/2, (n \cdot U_{\max})^h/4, \ldots$ we repeat the following $\Theta(h \cdot \log n \cdot \log U_{\max})$ times. Let f be the flow computed according to Theorem 88. Update $\hat{U}_a = \hat{U}_a - f_a$ for every a and update $\hat{f} = \hat{f} + f$. Clearly $\hat{f}$ is an integral S-T flow.

We need only verify that $\hat{f}$ is blocking. Since initially $\Delta \leq (n \cdot U_{\max})^h$, to do so it suffices to argue that when we fix a value of $\tilde{\Delta}$ for which $\frac{\Delta}{2} \leq \tilde{\Delta} \leq \Delta$, then over the course of the $\Theta(h \cdot \log n \cdot \log U_{\max})$ iterations where we use this value of $\tilde{\Delta}$ we have that Δ decreases by at least a factor of 2 with high probability.

Consider $\Theta(h \cdot \log n \cdot \log U_{\max})$ contiguous iterations of the above with a $\tilde{\Delta}$ that satisfies $\frac{\Delta}{2} \leq \tilde{\Delta} \leq \Delta$ at the beginning of these iterations. Let $\mathcal{P}_0$ be $\mathcal{P}_{\approx\max}$ at the beginning of these iterations. To show that Δ decreases by at least a factor of 2 over the course of these $\Theta(h \cdot \log n \cdot \log U_{\max})$ iterations it suffices to show that no path in $\mathcal{P}_0$ is in $\mathcal{P}_{\approx\max}$ for all of these iterations. Suppose for the sake of contradiction that some path $P \in \mathcal{P}_0$ is in $\mathcal{P}_{\approx\max}$ for all of these iterations. Then, applying the guarantees of Theorem 88, we get that with high probability $U(P)$ decreases by a $\frac{2047}{2048}$ factor at least $\Theta(h \cdot \log U_{\max}))$ times. However, since $U(P) \leq O((U_{\max})^h)$, we get that after these iterations we would have reduced $U(P)$ to 0 with high probability by a union bound, i.e. Δ must have reduced by at least a factor of 2.

The running time of our algorithm is immediate from the fact that we simply invoke Theorem 88 $\tilde{O}(h^2)$ times. □

5.8 Deterministic and Distributed Near Eulerian Partitions

In the previous section we showed how to efficiently compute blocking integral flows in h-layer DAGs with high probability. In this section, we introduce the key idea we make use of in doing so deterministically, a near Eulerian partition.

Informally, a near Eulerian partition will discard a small number of edges and then partition the remaining edges into cycles and paths. Because these cycles and paths will have small diameter in our construction, we will be able to efficiently orient them in CONGEST. In Section 5.9 we will see how to use these oriented cycles and paths to efficiently round flows in a distributed fashion in order to computer a blocking integral flow in h-layer DAGs.

We now formalize the idea of a $(1-\epsilon)$-near Eulerian partition.

Definition 90 ($(1-\epsilon)$-Near Eulerian Partition)**.** *Let $G = (V, E)$ be an undirected graph and $\epsilon \geq 0$. A $(1-\epsilon)$-near Eulerian partition $\mathcal{H}$ is a collection of edge-disjoint cycles and paths in G, where*

1. *$(1-\epsilon)$-**Near Covering:** The number of edges in $E[\mathcal{H}]$ is at least $(1-\epsilon) \cdot |E|$;*
2. ***Eulerian Partition:** Each vertex is the endpoint of at most one path in $\mathcal{H}$.*

The following is the main result of this section and summarizes our algorithms for construction $(1-\epsilon)$-near Eulerian partitions. In what follows we say that a cycle is oriented if every edge is directed so that every vertex in the cycle has in and out degree 1; a path P is oriented if it has some designated source and sink s_P and t_P. We say that a collection of paths and cycles $\mathcal{H}$ is oriented if each element of $\mathcal{H}$ is oriented. In CONGEST we will imagine that a cycle is oriented if each vertex knows the orientation of its incident arcs and a path is oriented if every vertex knows which of its neighbors are closer to s_P.

Lemma 91. *One can deterministically compute an oriented $(1-\epsilon)$-near Eulerian partitions in:*

1. *Parallel time $\tilde{O}(1)$ with m processors and $\epsilon = 0$;*
2. *CONGEST time $\tilde{O}(\frac{1}{\epsilon^5} \cdot (\rho_{CC})^{10})$ for any $\epsilon > 0$.*

Again, see Section 5.5.4 for a definition of ρ_{CC}.

5.8.1 High-Girth Cycle Decompositions

In order to compute our near Eulerian partitions we will make use of a slight variant of cycle covers which we call high-girth cycle decompositions (as introduced in Section 5.5.4). The ideas underpinning these decompositions seem to be known in the literature but there does not seem to be a readily citable version of quite what we need; hence we give details below.

To begin, in our near-Eulerian partitions we would like for our cycles to be edge-disjoint so that each cycle can be rounded independently. Thus, we give a subroutine for taking a collection of cycles and computing a large edge-disjoint subset of this collection. This result comes easily from applying a deterministic approximation algorithm for maximal independent set (MIS). Congestion and dilation in what follows are defined in Section 5.5.4.

Lemma 92. *There is a deterministic CONGEST algorithm that, given a graph $G = (V, E)$ and a collection of (not necessarily edge-disjoint) cycles $\mathcal{C}$ with congestion c and diameter d, outputs a set of edge disjoint cycles $\mathcal{C}' \subseteq \mathcal{C}$ which satisfies $|E[\mathcal{C}']| \geq \frac{1}{d^2c^2} \cdot |E[\mathcal{C}]|$ in time $\tilde{O}(c^3d^3)$.*

Proof. Our algorithm simply computes an approximately-maximum independent set in the conflict graph which has a node for each cycle. In particular, we construct conflict graph $G' = (\mathcal{C}, E')$ as follows. Our vertex set is $\mathcal{C}$. We include edge $\{C, C'\}$ in E' if $C \in \mathcal{C}$ and $C' \in \mathcal{C}$ overlap on an edge; that is, if $E[C] \cap E[C'] \neq \emptyset$.

Observe that since each cycle in $\mathcal{C}$ has at most d-many edges and since each edge is in at most c-many cycles, we have that the maximum degree of G' is cd. Next, we let the "node-weight" of cycle $C \in \mathcal{C}$ be $|C|$. We apply Theorem 79 with these node-weights to compute a $\frac{1}{cd}$-approximate maximum independent set $\mathcal{C}'$. We return $\mathcal{C}'$ as our solution.

First, observe that since $\mathcal{C}'$ is an independent set in G', we have that the cycles of $\mathcal{C}'$ are indeed edge-disjoint.

Next, we claim that $|E[\mathcal{C}']| \geq \frac{1}{d^2c^2} \cdot |E[\mathcal{C}]|$. Since Theorem 79 guarantees that $\mathcal{C}'$ is a $\frac{1}{dc}$-approximate solution, to show this, it suffices to argue that $|E[\mathcal{C}^*]| \geq \frac{1}{dc} \cdot |E[\mathcal{C}]|$ where $\mathcal{C}^* \subseteq \mathcal{C}$ is the set of edge-disjoint cycles of maximum edge cardinality, i.e. the maximum node-weight independent set in G'. However, notice that since the total node weight in G' is $\sum_{C \in \mathcal{C}} |E[C]|$ and the max degree in G' is at most cd, we have that the maximum node-weight independent set in G' must have node-weight at least $\frac{1}{cd} \sum_{C \in \mathcal{C}} |E[C]| \geq \frac{1}{cd} |E[\mathcal{C}]|$. Thus, we conclude that $|E[\mathcal{C}']| \geq \frac{1}{d^2c^2} \cdot |E[\mathcal{C}]|$.

Next, we argue that we can implement the above in the stated running times. Computing our $\frac{1}{cd}$-approximate maximum independent set on G' takes deterministic CONGEST time $\tilde{O}(cd)$ on G' by Theorem 79. Furthermore, we claim that we can simulate a CONGEST algorithm on G' in G with only an overhead of $O(c^2d^2)$. In particular, since the maximum degree on G' is cd, in each CONGEST round on G' each node (i.e. cycle in G) receives at most cd-many messages. Fix a single round of CONGEST on G'. We will maintain the invariant that if $v \in V$ is a node in a cycle $C \in \mathcal{C}$, then in our simulation v receives all the same messages as C in our CONGEST algorithm on G'. We do so by broadcasting all messages that C receives in this one round on G' to all nodes in C. As a cycle in G' receives at most cd messages in one round of CONGEST on G' and each edge is in at most c-many cycles, it follows that in such a broadcast the number of messages that need to cross any one edge is at most c^2d. Since the diameter of each cycle is at most d, we conclude that this entire broadcast can be done deterministically in time $O(c^2d^2)$, giving us our simulation.

Combining this $O(c^2d^2)$-overhead simulation with the $\tilde{O}(cd)$ running time of our approximate maximum independent set algorithm on G' gives an overall running time of $O(c^3d^3)$. □

Recall that the girth of a graph is the minimum length of a cycle in it. The following formalizes the notion of high-girth cycle decompositions that we will need.

Definition 93 (High-Girth Cycle Decomposition)**.** *Given a graph $G = (V, E)$ and $\epsilon > 0$ where E_0 are all non-bridge edges of G, a high-girth cycle decomposition with diameter d and deletion girth k is a collection of edge-disjoint (simple) cycles $\mathcal{C}$ such that:*

1. ***High Deletion Girth:*** *The graph $(V, E \setminus E[\mathcal{C}])$ has girth at least k.*
2. ***Low Diameter:*** $\max_{C \in \mathcal{C}} |C| \leq d$*;*

The following theorem gives the construction of high-girth cycle decompositions that we will use.

Theorem 94. *There is a deterministic CONGEST algorithm that, given a graph $G = (V, E)$ and desired girth $k \geq 0$, computes a high-girth cycle decomposition with diameter $\tilde{O}(k \cdot \rho_{CC})$ and girth k in time $\tilde{O}(k^5 \cdot (\rho_{CC})^{10})$.*

Proof. The basic idea is: take a sparse neighborhood cover; compute cycle covers on each part of our neighborhood cover; combine all of these into a single cycle cover; decongest this cycle cover into a collection of edge-disjoint cycles; delete these cycles; repeat.

More formally, our algorithm is as follows, We initialize our collection of cycles $\mathcal{C}$ to $\emptyset$.

Next, we repeat the following $\tilde{\Theta}\left(k^2 \cdot (\rho_{CC})^4\right)$ times. Apply Theorem 83 to compute an $\tilde{O}(1)$-sparse k-neighborhood cover of G with diameter $\tilde{O}(k)$ and overlap $\tilde{O}(1)$. Let $\mathcal{V}_1, \mathcal{V}_2, \ldots$ be the

partitions of this neighborhood cover. By definition of a neighborhood cover, for each $\mathcal{V}_i$ and each $V_i^{(j)} \in \mathcal{V}_i$, we have that $V_i^{(j)}$ comes with a tree $T_i^{(j)}$ where each node in the tree is in $\tilde{O}(1)$ other $V_i^{(j)}$. We let $H_i^{(j)} := G[V_i^{(j)}] \cup T_i^{(j)}$ be the union of this tree and the graph induced on $V_i^{(j)}$. By the guarantees of our neighborhood cover we have that the diameter of $H_i^{(j)}$ is at most $\tilde{O}(k)$. We then compute a cycle cover $\mathcal{C}_i^{(j)}$ of each $H_i^{(j)}$ with diameter $\tilde{O}(k \cdot \rho_{CC})$ and congestion ρ_{CC} (we may do so by definition of ρ_{CC}). We let $\mathcal{C}_0 = \bigcup_{i,j} \mathcal{C}_i^{(j)}$ be the union of all of these cycle covers. Next, we apply Theorem 92 to compute a large edge-disjoint subset $\mathcal{C}_0' \subseteq \mathcal{C}_0$ of $\mathcal{C}_0$. We add $\mathcal{C}_0'$ to $\mathcal{C}$ and delete from G any edge that occurs in a cycle in $\mathcal{C}_0'$.

We first argue that the solution we return is indeed a high-girth cycle decomposition. Our solution consists of edge-disjoint cycles by construction. Next, consider one iteration of our algorithm. Observe that since each $\mathcal{C}_i^{(j)}$ has congestion at most ρ_{CC}, it follows by the $\tilde{O}(1)$ overlap and $\tilde{O}(1)$ sparsity of our neighborhood cover that $\mathcal{C}_0$ has congestion $\tilde{O}(\rho_{CC})$. Likewise, since each $H_i^{(j)}$ has diameter $\tilde{O}(k)$, it follows that each $\mathcal{C}_i^{(j)}$ has diameter at most $\tilde{O}(k \cdot \rho_{CC})$ and so $\mathcal{C}_0$ has diameter at most $\tilde{O}(k \cdot \rho_{CC})$. Thus, $\mathcal{C}_0$ has congestion at most $\tilde{O}(\rho_{CC})$ and diameter at most $\tilde{O}(k \cdot \rho_{CC})$. Since $\mathcal{C}_0' \subseteq \mathcal{C}_0$, it immediately follows that the solution we return has diameter at most $\tilde{O}(k \cdot \rho_{CC})$.

It remains to show that the deletion of our solution induces a graph with high girth. Towards this, observe that applying the congestion and diameter of $\mathcal{C}_0$ and the guarantees of Theorem 92, it follows that

$$|E[\mathcal{C}_0']| \geq \tilde{\Omega}\left(\frac{1}{k^2(\rho_{CC})^4}\right) \cdot |E[\mathcal{C}_0]|. \tag{5.8.1}$$

On the other hand, let E_0 be all edges in cycle of diameter at most k at the beginning of this iteration. Consider an $e \in E_0$. Since $\mathcal{V}_1, \mathcal{V}_2, \ldots$ is a k-neighborhood cover we know that there is some $\mathcal{C}_i^{(j)}$ which contains a cycle which contains e. Thus, we have

$$|E[\mathcal{C}_0]| \geq |E_0|. \tag{5.8.2}$$

Combining Equation (5.8.1) and Equation (5.8.2), we conclude that

$$|E[\mathcal{C}_0']| \geq \tilde{\Omega}\left(\frac{1}{k^2(\rho_{CC})^4}\right) \cdot |E_0|.$$

However, since in this iteration we delete every edge in $E[\mathcal{C}_0']$, it follows that we reduce the number of edges that are in a cycle of diameter at most k by at least a $1 - \tilde{\Omega}\left(\frac{1}{k^2(\rho_{CC})^4}\right)$ multiplicative factor. Since initially the number of such edges is at most $|E|$, it follows that after $\tilde{O}(k^2 \cdot (\rho_{CC})^4)$-many iterations we have reduced the number of edges in a cycle of diameter at most k to 0; in other words, our graph has girth at most k. This shows the high girth of our solution, namely that $(V, E \setminus E[\mathcal{C}])$ has girth at least k after the last iteration of our algorithm.

Next, we argue that we achieve the stated running times. Fix an iteration.

- By the guarantees of Theorem 83, the sparse neighborhood cover that we compute takes time $\tilde{O}(k)$.
- We claim that by definition of ρ_{CC}, the $\tilde{O}(k)$ diameter of each part in our sparse neighborhood cover and the $\tilde{O}(1)$ overlap of our sparse neighborhood cover, we can compute

every $\mathcal{C}_i^{(j)}$ in time $\tilde{O}(k \cdot \rho_{CC})$. Specifically, for a fixed i we run the cycle cover algorithm simultaneously in meta-rounds, each consisting of $\tilde{\Theta}(1)$ rounds. In each meta-round a node can send the messages that it must send for the cycle cover algorithm of each of the $H_i^{(j)}$ to which it is incident by our overlap guarantees. Since the total number of i is $\tilde{O}(1)$ by our sparsity guarantee, we conclude that we can compute all $\mathcal{C}_i^{(j)}$ in a single iteration in at most $\tilde{O}(k \cdot \rho_{CC})$ time.

- Lastly, by the guarantees of Theorem 92 and the fact that $\mathcal{C}_0$ has congestion at most $\tilde{O}(\rho_{CC})$ and diameter at most $\tilde{O}(k \cdot \rho_{CC})$, we can compute $\mathcal{C}_0'$ in time $\tilde{O}(k^3 \cdot (\rho_{CC})^6)$.

Combining the above running times with the fact that we have $\tilde{\Theta}\left(k^2 \cdot (\rho_{CC})^4\right)$-many iterations gives us a running time of $\tilde{O}(k^5 \cdot (\rho_{CC})^{10})$. □

5.8.2 Efficient Algorithms for Computing Near Eulerian Partitions

We conclude by proving the main section of this theorem, namely the following which shows how to efficiently compute near Eulerian partitions in deterministic CONGEST by making use of our high-girth cycle decomposition construction and DLDDs.

Lemma 95. *One can deterministically compute an oriented $(1-\epsilon)$-near Eulerian partitions in:*

1. *Parallel time $\tilde{O}(1)$ with m processors and $\epsilon = 0$;*
2. *CONGEST time $\tilde{O}(\frac{1}{\epsilon^5} \cdot (\rho_{CC})^{10})$ for any $\epsilon > 0$.*

Proof. The parallel result is well-known since a 1-near Eulerian partition is just a so-called Eulerian partition; see e.g. Karp and Ramachandran [123].

The rough idea of our CONGEST algorithm is as follows. First we compute a high-girth cycle decomposition (Theorem 93), orient these cycles and remove all edges covered by this decomposition. The remaining graph has high girth by assumption. Next we compute a DLDD (Theorem 80) on the remaining graph; by the high girth of our graph each part of our DLDD is a low diameter tree. Lastly, we decompose each such tree into a collection of paths.

More formally, our CONGEST algorithm to return cycles $\mathcal{C}$ and paths $\mathcal{P}$ is as follows. Apply Theorem 94 to compute a high-girth cycle decomposition $\mathcal{C}$ with deletion girth $\tilde{\Theta}(\frac{1}{\epsilon})$ and diameter $\tilde{O}(\frac{1}{\epsilon} \cdot \rho_{CC})$. Orient each cycle in $\mathcal{C}$ and delete from G any edge in a cycle in $\mathcal{C}$. Next, apply Theorem 81 to compute a DLDD with diameter $\tilde{\Theta}(\frac{1}{\epsilon})$ and cut fraction ϵ. Delete all edges cut by this DLDD. Since $\mathcal{C}$ has deletion girth $\tilde{\Theta}(\frac{1}{\epsilon})$, by appropriately setting our hidden constant and poly-logs, it follows that no connected component in the remaining graph contains a cycle; in other words, each connected component is a tree with diameter $\tilde{\Theta}(\frac{1}{\epsilon})$.

We decompose each tree T in the remaining forest as follows. Fix an arbitrary root r of T. We imagine that each vertex of odd degree in T starts with a ball. Each vertex waits until it has received a ball from each of its children. Once a vertex has received all such balls, it pairs off the balls of its children arbitrarily, deletes these balls and adds to $\mathcal{P}$ the concatenation of the two paths traced by these balls in the tree. It then passes its up to one remaining ball to its parent. Lastly, we orient each path in $\mathcal{P}$ arbitrarily.

We begin by arguing that the above results in a $(1-\epsilon)$-near Eulerian partition. Our paths and cycles are edge-disjoint by construction. The only edges that are not included in some element of $\mathcal{C} \sqcup \mathcal{P}$ are those that are cut by our DLDD; by our choice of parameters this is at most an ϵ

fraction of all edges in E. To see the Eulerian partition property, observe that every vertex of odd degree in $G[\mathcal{C} \sqcup \mathcal{P}]$ is an endpoint of exactly one path in $\mathcal{P}$ since each odd degree vertex starts with exactly one ball. Likewise, a vertex of even degree will never be the endpoint of a path since no such vertex starts with a ball.

It remains to argue that the above algorithm achieves the stated CONGEST running time.

- Computing $\mathcal{C}$ takes time at most $\tilde{O}(\frac{1}{\epsilon^5} \cdot (\rho_{CC})^{10})$ by Theorem 94. Furthermore, by Theorem 94, each cycle in $\mathcal{C}$ has diameter $\tilde{O}(\frac{1}{\epsilon} \cdot \rho_{CC})$ and so can be oriented in time $\tilde{O}(\frac{1}{\epsilon} \cdot \rho_{CC})$.
- Computing our DLDD takes time $\tilde{O}(\frac{1}{\epsilon})$ by Theorem 81.
- Since our DLDD has diameter $\tilde{O}(\frac{1}{\epsilon})$, we have that the above ball-passing to comptue $\mathcal{P}$ can be implemented in time at most $\tilde{O}(\frac{1}{\epsilon})$.

Thus, overall our CONGEST algorithm takes time $\tilde{O}(\frac{1}{\epsilon^5} \cdot (\rho_{CC})^{10})$. □

5.9 Deterministic Blocking Integral Flows in h-Layer DAGs

In Section 5.7 we showed how to efficiently compute blocking integral flows in h-layer DAGs with high probability. In this section, we show how to do so deterministically by making use of the near Eulerian partitions of Section 5.8. Specifically, we show the following.

Lemma 96. *There is a deterministic algorithm which, given a capacitated h-layer S-T DAG D, computes an integral S-T flow that is blocking in:*

1. *Parallel time $\tilde{O}(h^3)$ with m processors;*
2. *CONGEST time $\tilde{O}(h^6 \cdot (\rho_{CC})^{10})$.*

The above parallel algorithm is more or less implied by the work of Cohen [60]. However, the key technical challenge we solve in this section is a distributed implementation of the above. Nonetheless, for the sake of completeness we will include the parallel result as well alongside our distributed implementation.

Our strategy for showing the above lemma has two key ingredients.

Iterated Path Count Flow. First, we construct the iterated path count flow. This corresponds to repeatedly taking the expected flow induced by the sampling of our randomized algorithm (as given by Theorem 88). As the flow we compute is the expected flow of the aforementioned sampling, this process is deterministic. The result of this is a $\tilde{\Omega}(\frac{1}{h})$-blocking but not necessarily integral flow. We argue that any such flow is also $\tilde{\Omega}(\frac{1}{h^2})$-approximate and so the iterated path count flow is nearly optimal but fractional.

Flow Rounding. Next, we provide a generic way of rounding a fractional flow to be in integral in an h-layer DAG while approximately preserving its value. Here, the main challenge is implementing such a rounding in CONGEST; the key idea we use is that of a $(1-\epsilon)$-near Eulerian partition from Section 5.8 which discards a small number of edges and then partitions the remaining graph into cycles and paths.

These partitions enables us to implement a rounding in the style of Cohen [60]. In particular, we start with the least significant bit of our flow, compute a $(1-\epsilon)$-near Eulerian partition of the graph induced by all arcs which set this bit to 1 and then use this partition to round all these bits to 0. Working our way from least to most significant bit results in an integral flow. The last major hurdle to this strategy is showing that discarding a small number of edges does not damage our resulting integral flow too much; in particular discarding edges in the above way can increase the deficit of our flow. However, by always discarding an appropriately small number of edges we show that this deficit is small and so after deleting all flow that originates or ends at vertices not in S or T, we are left with a flow of essentially the same value of the input fraction flow. The end result of this is a rounding procedure which rounds the input fractional flow to an integral flow while preserving the value of the flow up to a constant.

Our algorithm to compute blocking integral flows in h-layer DAGs deterministically combines the above two tools. Specifically, we repeatedly compute the iterated path count flow, round it to be integral and add the resulting flow to our output. As the iterated path count flow is $\tilde{\Omega}(\frac{1}{h^2})$-approximate, we can only repeat this about h^2 time (otherwise we would end up with a flow of value greater than that of the optimal flow).

5.9.1 Iterated Path Count Flows

In this section we define our iterated path count flows and prove that they are $\tilde{\Omega}(\frac{1}{h})$-approximate.

Specifically, the path counts of Section 5.6 naturally induce a flow. In particular, they induce what we will call the path count flow where the flow on arc (u,v) is defined as:

$$f_a = U_a \cdot \frac{n_a}{\max_{a \in A} n_a}.$$

It is easy to see these path counts induce an S-T flow.

Lemma 97. *For a given capacitated S-T DAG the path count flow is an S-T flow.*

Proof. The above flow does not violate capacities by construction. Moreover, it obeys flow conservation for all vertices other than those in S and T since it is a convex combination of paths between S and T. More formally, for any vertex $v \notin S \cup T$ we have flow conservation by the calculation:

$$\begin{aligned}
\sum_{a=(u,v)\in\delta^-(v)} f_a &= \frac{U_a}{\max_{a\in A} n_a} \sum_{a=(u,v)\in\delta^-(v)} \sum_{P\in\mathcal{P}(S,T):a\in P} U(P) \\
&= \frac{U_a}{\max_{a\in A} n_a} \sum_{a=(u,v)\in\delta^+(v)} \sum_{P\in\mathcal{P}(S,T):a\in P} U(P) \\
&= \sum_{a=(u,v)\in\delta^+(v)} f_a
\end{aligned}$$

where the second line follows from the fact that every path from S to T which enters v must also exit v. □

Path count flows were first introduced by Cohen [60]. Our notion of an iterated path count flow is

closely related to Cohen [60]'s algorithm for computing blocking flows in parallel. In particular, in order to compute an integral blocking flow, Cohen [60] iteratively computes a path count flow, rounds it, decrements capacities and then iterates. For us it will be more convenient to do something slightly different; namely, we will compute a path count flow, decrement capacities and iterate; once we have *a single* blocking fractional flow we will apply our rounding *once*. Nonetheless, we note that many of the ideas of this section appear implicitly in Cohen [60].

We proceed to define the iterated path count flow which is always guaranteed to be near-optimal. The iterated path count flow will be a sum of several path count flows. More formally, suppose we are given an h-layer capacitated S-T DAG $D = (V, A)$ with capacities U. In such a DAG we have $n_a \leq (n \cdot U_{\max})^h$. We initialize f_0 to be the flow that assigns 0 to every arc and $U_0 = U$. We then let $D_i = (V, A)$ with capacities U_i where $U_i = U_{i-1} - f_{i-1}$ and f_{i-1} is the path count flow of D_{i-1}. Lastly, we define the iterated path count flow as a convex combination of these path count flows iterated $k = \Theta(h \cdot (\log n) \cdot \log(n \cdot U_{\max}))$ times. That is, the iterated path count flow is

$$\tilde{f} := \sum_{i=0}^{k} f_i.$$

We begin by observing that the iterated path count flow is reasonably blocking.

Lemma 98. *The iterated path count flow $\tilde{f}$ is a (not necessarily integral) blocking S-T flow.*

Proof. Since each path count flow is an S-T flow by Theorem 97, by how we reduce capacities it immediately follows that $\tilde{f}$ is an S-T flow.

Thus, it remains to argue that $\tilde{f}$ is blocking. Towards this, consider computing the ith path count flow when the current path counts are $\{n_a\}_a$ and the flow over arc a is $(f_i)_a = (U_i)_a \cdot \frac{n_a}{\max_a n_a}$. Letting $A_{\approx\max}$ be all arcs for which $n_a \geq \frac{1}{2} \max_a n_a$, we get that $(U_{i+1})_a \leq \frac{1}{2} \cdot (U_i)_a$ for all $a \in A_{\approx\max}$. It follows that after $\Theta(\log n)$ iterations we will reduce $\max_a n_a$ by at least a multiplicative factor of 2. Since initially $n_a \leq (n \cdot U_{\max})^h$, it follows that after $k = \Theta(h \cdot (\log n) \cdot \log(n \cdot U_{\max}))$ iterations we have reduced n_a to 0 for every arc which is to say that for any path P between S and T we have that there is some arc $a \in P$ it holds that $\sum_i (f_i)_a = U_a$. Since $\tilde{f}_a = \sum_i (f_i)_a$, we conclude that $\tilde{f}$ is blocking. □

Next, we observe that any blocking flow is near-optimal.

Lemma 99. *Any α-blocking S-T flow in an h-layer S-T DAG is $\left(\frac{\alpha}{h}\right)$-approximate.*

Proof. Let f be our α-blocking flow and let D be the input graph. Let f^* be the optimal S-T flow in the input DAG and let $\sum_P f_P$ be it's flow decomposition into path flows where each P is a directed path from S to T and $(f_P)_a$ is 1 if $a \in P$ and 0 otherwise.

Since f is blocking, for each such path there is some arc, a_P where $f_{a_P} \geq \alpha \cdot U_{a_P} \geq \alpha \cdot f^*_{a_P}$. Let $A' = \{a_P : P \text{ in flow decomposition of } f^*\}$ be the union of all such blocked arcs. Thus, s.t. $(f^*) \leq \sum_{a \in A'} f^*_a \leq \sum_{a \in A'} \frac{f_a}{\alpha}$. However, since D is h-layered, by an averaging argument we have that there must be some j such that $f(\delta^+(V_j) \cap A') \geq \frac{1}{h} \sum_{a \in A'} f_a$ where V_j is the jth layer of our digraph. On the other hand, s.t. $(f) \geq f(\delta^+(V_j)) \geq f(\delta^+(V_j) \cap A')$ and so we conclude that

$$\text{s.t. } (f) \geq f(\delta^+(V_j) \cap A')$$

$$\geq \frac{1}{h} \cdot \sum_{a \in A'} f_a$$
$$\geq \frac{\alpha}{h} \cdot \text{ s.t. } (f^*),$$

showing that f is $\left(\frac{\alpha}{h}\right)$-approximate as desired. □

We conclude that the iterated path count flow is near-optimal and efficiently computable; our CONGEST algorithm will make use of sparse neighborhood covers to deal with potentially large diameter graphs.

Lemma 100. *Let D be a capacitated h-layer S-T DAG with diameter at most $\tilde{O}(h)$. Then one can deterministically compute a (possibly non-integral) flow $\tilde{f}$:*

1. *In parallel that is $\Omega\left(\frac{1}{h}\right)$-approximate in time $\tilde{O}(h^2)$ with m processors;*
2. *In CONGEST that is $\tilde{\Omega}\left(\frac{1}{h}\right)$-approximate in time $\tilde{O}\left(h^4\right)$.*

Proof. Combining Theorem 98 and Theorem 99 shows that the iterated path count flow is an S-T flow that is $\Omega(\frac{1}{h})$-approximate.

For our parallel algorithm, we simply return the iterated path count flow. The iterated path count flow is simply a sum of $k = \Theta(h \cdot (\log n) \cdot \log(n \cdot U_{\max})$-many path count flows. Thus, it suffices to argue that we can compute path count flows in $O(h)$ parallel time with m processors. By Theorem 87 we can compute n_a for every a in these times and so to then compute the corresponding path count flows we need only compute $\max_a n_a$ which is trivial to do in parallel in the stated time.

For our CONGEST algorithm we do something similar but must make use of sparse neighborhood covers, because we cannot outright compute $\max_a n_a$ as the diameter of D might be very large. Specifically, we do the following. Apply Theorem 83 to compute an s-sparse h-neighborhood cover with diameter $\tilde{O}(h)$ and partition $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_s$ for $s = \tilde{O}(1)$. Then we iterate through each of these partitions for $i = 1, 2, \ldots, s$. For each part $V_i^{(j)} \in \mathcal{V}_i$, we let $\tilde{f}_i^{(j)}$ be the iterated path count flow of $D[V_i^{(j)}]$ with source set $S \cap V_i^{(j)}$ and sink set $T \cap V_i^{(j)}$. We let $\tilde{f}_i := \sum_j f_i^{[j]}$ be the path count flows associated with the ith partition and return as our solution the average path count flow across partitions; namely we return

$$\tilde{f} = \frac{1}{s} \cdot \sum_i \tilde{f}_i.$$

This flow is an S-T flow since it is a convex combination of S-T flows. We now argue that this flow is $\tilde{\Omega}(\frac{1}{h})$-optimal. Let $\hat{f}_i^{[j]}$ be the optimal flow on $D[V_i^{(j)}]$ with source set $S \cap V_i^{(j)}$ and sink set $T \cap V_i^{(j)}$. As our path count flows are $\Omega(\frac{1}{h})$-approximate, we know that

$$\text{s.t. } (\tilde{f}_i^{[j]}) \geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot \text{ s.t. } (\hat{f}_i^{[j]}).$$

Moreover, since every h-neighborhood is contained in one of the $V_i^{[j]}$, it follows that $\sum_{i,j} \text{ s.t. } (\hat{f}_i^{[j]}) \geq \text{ s.t. } (f^*)$ where f^* is the optimal S-T flow on D with source set S and sink set T. Thus, we

conclude that

$$\begin{aligned}\text{s.t. } (\tilde{f}) &= \frac{1}{s} \cdot \sum_{i,j} \tilde{f}_i^{[j]} \\ &\geq \Omega\left(\frac{1}{h}\right) \cdot \frac{1}{s} \cdot \sum_{i,j} \hat{f}_i^{[j]} \\ &\geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot \text{ s.t. } (f^*).\end{aligned}$$

Lastly, we argue the running time of our CONGEST algorithm. We describe how to compute $\tilde{f}_i$ for a fixed i. Again, $\tilde{f}_i$ on each part is simply a sum of $k = \tilde{\Theta}(h)$-many path count flows. To compute one of these path count flows we first compute the path counts $\{n_a\}_a$ on each part by applying Theorem 87 which takes $\tilde{O}(h^2)$ time. Next, we compute $\max_a n_a$ in $\tilde{O}(h)$ time by appealing to Theorem 83 and the fact that $\max_a n_a \leq O(n^h)$. Thus, computing each $\tilde{f}_i$ takes time $\tilde{O}(h^3)$ and since there are $\tilde{O}(h)$ of these, overall this takes $\tilde{O}(h^4)$ time. □

5.9.2 Deterministic Rounding of Flows in h-Layer DAGs

In the previous section we showed how to construct our iterated path count flows and that they were near-optimal but possibly fractional. In this section, we give the flow rounding algorithm that we will use to round our iterated path count flows to be integral. Specifically, in this section we show the following flow rounding algorithm.

Lemma 101. *There is a deterministic algorithm which, given a capacitated h-layer S-T DAG D, $\epsilon = \Omega(\frac{1}{\text{poly}(n)})$ and (possibly fractional) flow f, computes an integral S-T flow $\hat{f}$ in:*

1. *Parallel time $\tilde{O}(h)$ with m processors;*
2. *CONGEST time $\tilde{O}(\frac{1}{\epsilon^5} \cdot h^5 \cdot (\rho_{CC})^{10})$.*

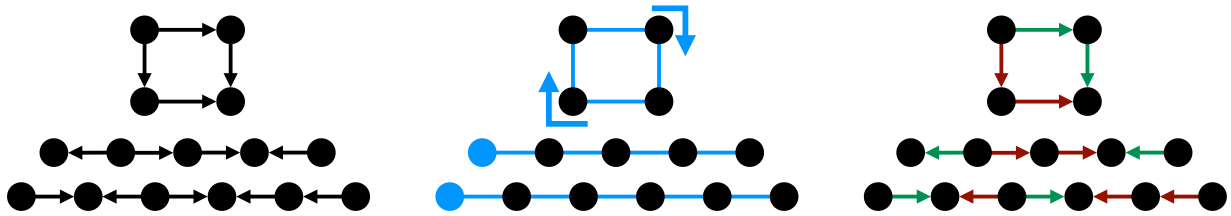
Furthermore, s.t. $(\hat{f}) \geq (1-\epsilon) \cdot$ s.t. (f).

Parts of the above parallel result are implied by the work of Cohen [60] while the CONGEST result is entirely new.

Turning Flows on $(1-\epsilon)$-Near Eulerian Partitions

As discussed earlier, our rounding will round our flow from the least to most significant bit. To round the input flow on a particular bit we will consider the graph induced by the arcs which set this bit to 1. We then compute an oriented near-Eulerian partition of these edges and "turn" flow along each cycle and path consistently with its orientation. We will always turn flow so as to not increase the deficit of our flow.

We now formalize how we use our $(1-\epsilon)$-near Eulerian partitions to update our flow. Given a path or cycle H, our flow update will carefully choose a subset of arcs of H along which to increase flow (denoted H^+) and decrease flow along all other arcs of H. Specifically, let H be an oriented cycle or path of a graph produced by forgetting about the directions in a digraph $D = (V, A)$. Then H^+ is illustrated in Figure 5.3 and defined as follows:



(a) Near Eulerian Partition $\mathcal{H}$. **(b)** Orientation of $\mathcal{H}$. **(c)** H^+ for each $H \in \mathcal{H}$.

Figure 5.3: An illustration of a near Eulerian partition $\mathcal{H}$ and H^+ for each $H \in \mathcal{H}$. 5.3a gives $\mathcal{H}$ which consists of one cycle and two paths. 5.3b gives the orientation of $\mathcal{H}$ where the source of each path is in blue. 5.3c gives H^+ (in green) and $H \setminus H^+$ (in red) for each $H \in \mathcal{H}$.

- Suppose H is an oriented cycle. Then, we let H^+ be all arcs of D in this cycle that point in the same direction as their orientation.
- Suppose $H = (s_H = v_0, v_1, v_2, \ldots)$ is an oriented path. We let H^+ be all arcs of D in this path that point in the same direction as the one arc in D incident to s_H (i.e. the designated source of the path). That is either (v_0, v_1) or (v_1, v_0) are in D. In the former case we let H^+ be all arcs in D of the form (v_i, v_{i+1}) for some i. In the latter case we let H^+ be all arcs in D of the form (v_{i+1}, v_i) for some i.

With our definition of H^+ in hand, we now define our flow updates as follows.

Definition 102 ($(1-\epsilon)$-Near Eulerian Partition Flow Update)**.** *Let f be a flow in a capacitated DAG for which $f_a \in \{0, c\}$ for every $a \in A$ for some c and let $\mathcal{H}$ be an oriented $(1-\epsilon)$-near Eulerian partition of $\mathrm{supp}(f)$ after forgetting about edge directions. Then if $H \in \mathcal{H}$, we define the flow f_H on arc a:*

$$(f_H)_a := \begin{cases} 2c & \text{if } a \in H^+ \\ 0 & \text{otherwise} \end{cases}$$

Likewise, we define the flow corresponding to $(f, \mathcal{H})$ as

$$f_{\mathcal{H}} := \sum_{H \in \mathcal{H}} f_H.$$

The following shows that our flow update will indeed zero out the value of each bit on each edge while incurring a negligible deficit.

Lemma 103. *Let f be a flow in a capacitated DAG D with specified source and sink vertices S and T where $f_a \in \{0, c\}$ for every $a \in A$ for some c. Let $\mathcal{H}$ be an oriented $(1-\epsilon)$-near Eulerian partition of $\mathrm{supp}(f)$ after forgetting about edge directions. Then $f_{\mathcal{H}}$ (as defined in Theorem 102) satisfies:*

1. *$(f_{\mathcal{H}})_a \in \{0, 2c\}$ for every $a \in A$;*
2. *deficit$(f_{\mathcal{H}}) \leq$ deficit$(f) + 2\epsilon \cdot \sum_a f_a$.*

Proof. $(f_{\mathcal{H}})_a \in \{0, 2c\}$ holds by the definition of $f_{\mathcal{H}}$ and the fact that the elements of $\mathcal{H}$ are edge-disjoint.

We next argue that $\text{deficit}(f') \leq \text{deficit}(f) + 2\epsilon \cdot \sum_a f_a$. The basic idea is that each edge in the support of f which does not appear in $A[\mathcal{H}]$ contributes its value to the deficit but any way of turning a cycle in $\mathcal{H}$ leaves the deficit unchanged and the way we chose to turn paths also leaves the deficit unchanged.

We let f' be f projected onto the arcs in $A[\mathcal{H}]$. That is, on arc a the flow f' takes value

$$f'_a =: \begin{cases} f_a & \text{if } a \in A[\mathcal{H}] \\ 0 & \text{otherwise} \end{cases}$$

We have that $\text{deficit}(f') \leq \text{deficit}(f) + 2\epsilon \cdot \sum_a f_a$ since each arc $a \notin A[\mathcal{H}]$ increases the deficit of f' by at most $2f_a$ and, from Theorem 90, there are at most ϵ-fraction of arcs not in $A[\mathcal{H}]$. Thus, to show our claim it suffices to argue that $\text{deficit}(f_{\mathcal{H}}) \leq \text{deficit}(f')$. For a given vertex v, we let $n_i(v)$ be the number of elements of $\mathcal{H}$ in which v has in-degree 2. Similarly, we let $n_o(v)$ be the number of elements of $\mathcal{H}$ for which v has out-degree 2. Lastly, we let $s(v)$ be the indicator of whether v is the source of some path in $\mathcal{H}$ and $t(v)$ be the indicator of whether v is the sink of a path in $\mathcal{H}$. Thus, we have

$$\text{deficit}(f', v) = 2c \cdot |n_i(v) - n_o(v)| + c \cdot (s(v) + t(v))$$

and so

$$\begin{aligned} \text{deficit}(f') &= \sum_v 2c \cdot |n_i(v) - n_o(v)| + c \cdot (s(v) + t(v)) \\ &= 2c|\mathcal{P}| + \sum_v 2c \cdot |n_i(v) - n_o(v)| \end{aligned}$$

On the other hand, we have

$$\text{deficit}(f_{\mathcal{H}}, v) \leq 2c \cdot |n_i - n_o| + 2c \cdot t(v)$$

and so

$$\begin{aligned} \text{deficit}(f_{\mathcal{H}}) &\leq \sum_v 2c \cdot |n_i(v) - n_o(v)| + 2c \cdot t(v) \\ &= 2c|\mathcal{P}| + \sum_v 2c \cdot |n_i(v) - n_o(v)| \end{aligned}$$

showing $\text{deficit}(f_{\mathcal{H}}) \leq \text{deficit}(f')$ as required. □

Extracting Integral S-T Subflows

The last piece of our rounding deals with how to fix the damage that the accumulating deficit incurs. Specifically, as we round each bit we discard some edges, increasing our deficit. This means that after rounding all bits we are left with some (small) deficit. In this section we show how to delete flows that originate or end at vertices not in S or T, thereby reducing the value of our flow by the deficit but guaranteeing that we are left with a legitimate S-T flow.

Lemma 104. *Let $\hat{f}$ be an integral (not necessarily S-T) flow on an h-layer S-T DAG. Then one can compute an S-T integral flow f' which is a subflow of $\hat{f}$ and satisfies s.t. $(f') \geq$ s.t. $(\hat{f}) -$ deficit$(\hat{f})$ in:*

1. *Parallel time $O(h)$ with m processors;*
2. *CONGEST time $\tilde{O}(h)$.*

Proof. Our algorithm will simply delete out flow that originates not in S or ends at vertices not in T. More formally, we do the following. We initialize our flow f' to $\hat{f}$. Let $S = V_1, V_2, \ldots, V_{h+1} = T$ be the vertices in each layer of our input S-T DAG $D = (V, A)$. Recall that we defined a flow $\hat{f}$ as an arbitrary function on the arcs so that $\hat{f}_a \leq U_a$ for every a. The basic idea of our algorithm is to first push all "positive" deficit from left to right and then to push all "negative" deficit from right to left. The deficit will be non-increasing under both of these processes.

More formally, we push positive deficit as follows. For $i = 2, 3, \ldots h$ we do the following. For each $v \in V_i$, let

$$\text{deficit}^+(v) := \max\left(0, \sum_{a \in \delta^+(v)} f'_a - \sum_{a \in \delta^-(v)} f'_a\right)$$

be the positive deficit of v. Then, we reduce $\sum_{a \in \delta^+(v)} f'_a$ to be equal to $\sum_{a \in \delta^-(v)} f'_a$ by arbitrarily (integrally) reducing f'_a for some subset of $a \in \delta^+(v)$.

It is easy to see by induction that at this point we have $\text{deficit}^+(v) = 0$ for all $v \notin S \cup T$. Likewise, we have that $\sum_{v \notin S \cup T} \text{deficit}^+(v)$ is non-increasing each time we iterate the above. Thus, if deficit$^+$ is the initial value of $\sum_{v \notin S \cup T} \text{deficit}^+(v)$ then in the last iteration of the above we may decrease the flow into T by at most deficit$(\hat{f})$.

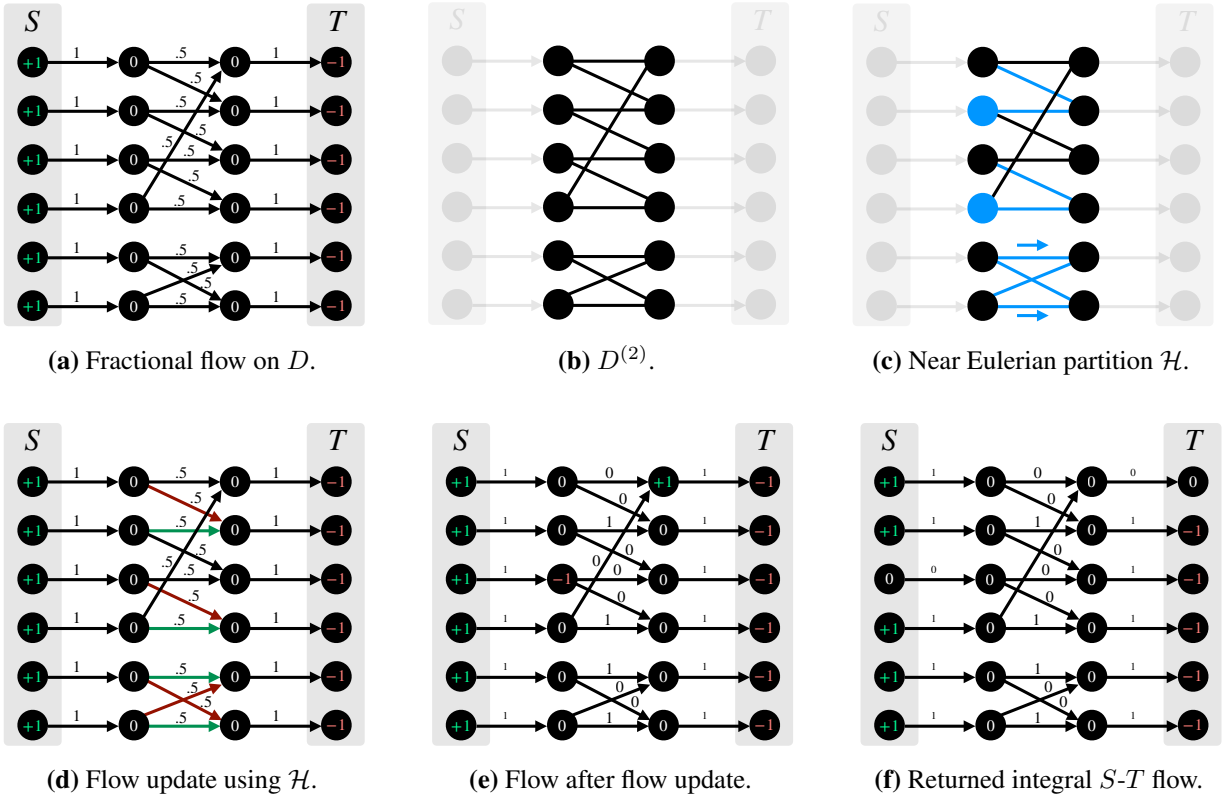
Next, we do the same thing symmetrically to reduce the negative deficits. For $i = h, h-1, \ldots, 2$ we do the following for each $v \in V_i$. Let

$$\text{deficit}^-(v) := \max\left(0, \sum_{a \in \delta^-(v)} f'_a - \sum_{a \in \delta^+(v)} f'_a\right)$$

be the negative deficit of v. Then, we reduce $\sum_{a \in \delta^-(v)} f'_a$ to be equal to $\sum_{a \in \delta^+(v)} f'_a$ by arbitrarily (integrally) reducing f'_a for some subset of $a \in \delta^-(v)$. Notice that this does not increase $\text{deficit}^+(v)$ for any $v \notin S \cup T$.

Symmetrically to the positive deficit case, it is easy to see that at the end of this process we have reduced $\text{deficit}^-(v)$ to 0 for every $v \notin S \cup T$ while reducing the flow out of S by at most deficit$(\hat{f})$.

Thus, at the end of this process we have an S-T integral flow f' whose value is at least s.t. $(\hat{f}) -$ deficit$(\hat{f})$. Implementing the above in the stated running times is trivial; the only caveat is that updating a flow in CONGEST requires updating it for both endpoints but since the flow is integral and we reduce it integrally, this can be done along a single arc in time $O(\log U_{\max}) = \tilde{O}(1)$ by assumption. □



(a) Fractional flow on D. **(b)** $D^{(2)}$. **(c)** Near Eulerian partition $\mathcal{H}$.

(d) Flow update using $\mathcal{H}$. **(e)** Flow after flow update. **(f)** Returned integral S-T flow.

Figure 5.4: An example of our flow rounding algorithm on digraph D with unit capacities. 5.4a gives the input flow where arcs are labelled with their flow and vertices are labelled with their deficit. 5.4b gives $D^{(2)}$, the graph induced by all arcs with flow value .5. 5.4c gives our oriented near Eulerian partition of $D^{(2)}$ (in blue). 5.4d shows how we update our flow based on the near Eulerian partition. 5.4e gives the result of this flow update; notice that some vertices not in S and T have non-zero deficit. 5.4f gives the S-T subflow we return where only vertices in S and T have non-zero deficit.

Flow Rounding Algorithm

Having defined the flow update we use for each $(1-\epsilon)$-near Eulerian partition and how to extract a legitimate S-T flow from the resulting rounding, we conclude with our algorithm for rounding flows from least to most significant bit. Our algorithm is given in Algorithm 1 and illustrated in Figure 5.4.

We conclude that the above rounding algorithm rounds with negligible loss in the value.

Lemma 105. *There is a deterministic algorithm which, given a capacitated h-layer S-T DAG D, $\epsilon = \Omega(\frac{1}{\mathsf{poly}(n)})$ and (possibly fractional) flow f, computes an integral S-T flow $\hat{f}$ in:*

1. *Parallel time $\tilde{O}(h)$ with m processors;*
2. *CONGEST time $\tilde{O}(\frac{1}{\epsilon^5} \cdot h^5 \cdot (\rho_{CC})^{10})$.*

Furthermore, s.t. $(\hat{f}) \geq (1-\epsilon) \cdot$ s.t. (f).

Proof. We use Algorithm 1.

We first argue that the above algorithm returns an integral flow. Notice that by the fact that we initialize $\hat{f}$ to $\sum_{i=0}^{k}$ it follows that for $j > k$ on every a we have $\hat{f}_a^{(j)} = 0$ just before the first

Algorithm 1 Deterministic Flow Rounding

Input: h-layer DAG D, S-T flow $f = \sum_{i=0} f^{(i)}$ where $(f^{(i)})_a \in \{0, 2^{\log(U_{\max})-i}\}$ for every a, i.

Output: integral S-T flow $\hat{f}$.

$\hat{f} \leftarrow \sum_{i=0}^{k} f^{(i)}$ for $k = \Theta(\log n + \log(U_{\max}))$.{Truncate lower order bits of input flow} $i = k, \ldots, \log(U_{\max})$

Let $\hat{f} = \sum_j \hat{f}^{(j)}$ be the bitwise flow decomposition of $\hat{f}$ (defined in Section 5.2) and let $D^{(i)}$ be the undirected graph induced by the support of $f^{(i)}$.

Compute an oriented $(1-\epsilon')$-near Eulerian partition $\mathcal{H}$ of $D^{(i)}$ (using Theorem 91 with $\epsilon' = 0$ for the parallel algorithm and $\epsilon' = \Theta\left(\frac{\epsilon}{h \cdot \log n}\right)$ for the CONGEST algorithm).

$\hat{f} \leftarrow \hat{f}^{(i)}_{\mathcal{H}} + \sum_{j<i} \hat{f}^{(j)}$ (as defined in Theorem 102). {Turn flow along $\mathcal{H}$}

Let $\hat{f}$ be an S-T subflow of $\hat{f}$ (compute using Theorem 104).

$\hat{f}$.

iteration of our algorithm. Thus, to argue that the returned flow is integral it suffices to argue that if $\hat{f}^{(j)}$ is the jth bit flow of $\hat{f}$ just after the ith iteration then for $j \leq i$ we have $f_a^{(j)} = 0$ for every a. However, notice that, by Theorem 103, after we update $\hat{f}$ each $\hat{f}_a^{(i)}$ value is either doubled or set to 0, meaning that $\hat{f}_a^{(i)} = 0$ after this update.

Next, we argue that s.t. $(\hat{f}) \geq (1-\epsilon) \cdot$ s.t. (f). By Theorem 104 it suffices to argue that just before we compute our S-T subflow of $\hat{f}$ we have deficit$(\hat{f}) \leq \epsilon \cdot$ s.t. (f). We may set the constant in $k = \Theta(\log n + \log(U_{\max}))$ to be appropriately large so that when we initialize $\hat{f}$ we reduce the flow value on each arc by at most $\frac{1}{\text{poly}(n)}$. It follows that at this point deficit$(\hat{f}) \leq \frac{2}{\text{poly}(n)} \sum_a f_a$. Similarly, by Theorem 103 in the ith iteration of our algorithm we increase the deficit of $\hat{f}$ by at most $2\epsilon' \sum_a \hat{f}_a^{(i)} \leq 2\epsilon' \sum_a f_a$.

For our parallel algorithm, since we have $\epsilon' = 0$, it immediately then follows that deficit$(\hat{f}) \leq \frac{2}{\text{poly}(n)} \sum_a f_a \leq \epsilon \cdot$ s.t. (f) by our assumption that $\epsilon = \Omega(\frac{1}{\text{poly}(n)})$. For our CONGEST algorithm we choose $\epsilon' = \Theta(\frac{\epsilon}{h \log n})$ for some appropriately small constant. Since we have $\Theta(\log n)$ iterations it follows that after all of our iterations (but before we compute an S-T subflow) it holds that deficit$(\hat{f}) \leq \frac{\epsilon}{h} \cdot \sum_a f_a \leq \epsilon \cdot$ s.t. (f) where the last inequality follows from the fact that our flow is h-length.

Lastly, we argue that the algorithm achieves the stated running times. The above algorithm runs for $k = \Theta(\log n)$ iterations. The computation in each iteration is dominated by computing a $(1-\epsilon')$-near Eulerian partition. For our parallel algorithm, computing each $(1-\epsilon')$-near Eulerian partition takes time at most $\tilde{O}(1)$ with m processors by Theorem 91. For our CONGEST algorithm computing each $(1-\epsilon')$-near Eulerian partition takes time at most $\tilde{O}(\frac{1}{\epsilon^5} \cdot h^5 \cdot (\rho_{CC})^{10})$ by Theorem 91. Lastly, we must compute an S-T subflow of $\hat{f}$ which by Theorem 104 takes $O(h)$ parallel time with m processors or $\tilde{O}(h)$ CONGEST time. □

5.9.3 Deterministic Blocking Integral Flows

Having shown that the iterated path count flow is near-optimal and fractional but that we can efficiently round fractional flows to be integral, we conclude with our algorithm to compute a

blocking integral flow by repeatedly rounding iterated path count flows.

Lemma 106. *There is a deterministic algorithm which, given a capacitated h-layer S-T DAG D, computes an integral S-T flow that is blocking in:*

1. *Parallel time $\tilde{O}(h^3)$ with m processors;*
2. *CONGEST time $\tilde{O}(h^6 \cdot (\rho_{CC})^{10})$.*

Proof. We repeatedly compute the iterated path count flow, round it to be integral, reduce capacities appropriately and repeat. We will return flow f initialized to 0 on all arcs.

Specifically, we repeat the following $\tilde{\Theta}(h)$ times. Apply Theorem 100 to compute a $\tilde{\Omega}(1/h)$-approximate (possibly fractional) flow $\tilde{f}$. Next, apply Theorem 101 with $\epsilon = .5$ to round this to an integral flow $\hat{f}$ where s.t. $(\hat{f}) \geq \frac{1}{2}$ s.t. $(\tilde{f})$. Next, we update f to $f + \hat{f}$ and for each arc a we reduce U_a by $\hat{f}_a$.

After each time we iterate the above $\tilde{\Theta}(h)$ times we must reduce the value of the optimal solution by at least a multiplicative $\frac{1}{2}$ since otherwise f would be a flow with value greater than the max S-T flow in the graph at the beginning of these iterations. Since the optimal solution is at most $m \cdot U_{\max}$, it follows that we need only iterate the above $\tilde{\Theta}(h)$ times until the value of the optimal S-T flow is 0 which is to say that f is a blocking flow.

By Theorem 100 and Theorem 101 each of the above iterations takes parallel time $\tilde{O}(h^2)$ with m processors and CONGEST time $\tilde{O}(h^5 \cdot (\rho_{CC})^{10})$, giving the stated running times. □

5.10 h-Length $(1+\epsilon)$-Lightest Path Blockers

In this section we show how to efficiently compute our main subroutine for our multiplicative-weights-type algorithm; what we call h-length $(1+\epsilon)$-lightest path blockers. We will use the blocking integral flow primitives of Section 5.7 for our randomized algorithm and that of Section 5.9 for our deterministic algorithm.

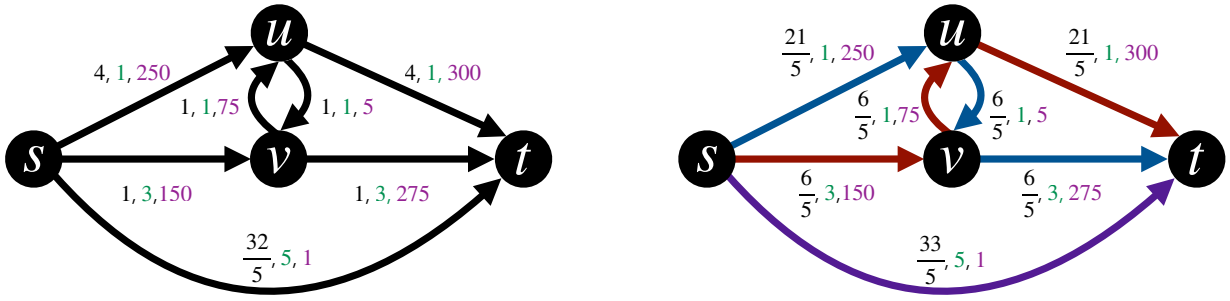
Our $(1+\epsilon)$-lightest path blockers are defined below. In what follows, λ is intuitively a guess of $d_w^{(h)}(S,T)$. Also, in the following recall that if f is an h-length flow then f assigns flow values to entire paths (rather than just arcs as a non-length-constrained flow does). As such the support of f, $\text{supp}(f)$, is a collection of paths. However, as mentioned earlier, for an h-length flow f, we will use $f(a)$ as shorthand for $\sum_{P \ni a} f_P$.

Definition 107 (h-length $(1+\epsilon)$-Lightest Path Blockers)**.** *Let $G = (V, E)$ be a graph with lengths l, weights w and capacities U. Fix $\epsilon > 0$, $h \geq 1$, $\lambda \leq d_w^{(h)}(S,T)$ and $S, T \subseteq V$. Let f be an h-length integral S-T flow. f is an h-length $(1+\epsilon)$-lightest path blocker if:*

1. ***Near-Lightest:** $P \in \text{supp}(f)$ has weight at most $(1+2\epsilon) \cdot \lambda$;*
2. ***Near-Lightest Path Blocking:** If $P' \in \mathcal{P}_h(S,T)$ has weight at most $(1+\epsilon) \cdot \lambda$ then there is some $a \in P'$ where $f(a) = U_a$.*

The main theorem of this section we show is how to compute our $(1+\epsilon)$-lightest path blockers efficiently.

Given digraph $D = (V, A)$ with lengths l, weights w, capacities U, length constraint $h \geq 1$, $\epsilon > 0$, $S, T \subseteq V$ and $\lambda \leq d_w^{(h)}(S,T)$, one can compute an h-length $(1+\epsilon)$-lightest path blocker in:



(a) Digraph D with w.

(b) Digraph D with $\tilde{w}$.

Figure 5.5: An illustration of how we round weights according to ϵ, λ and h. Here $h = 5$, $\lambda = 6$ and $\epsilon = .5$ and so we round to multiples of $\frac{\epsilon}{h}\lambda = \frac{3}{5}$. 5.5a gives our input DAG where each arc is labeled with its weight, then length then capacity and 5.5b gives the weights after we round them where we color each lightest 5-length path from s to t.

1. Deterministic parallel time $\tilde{O}(\frac{1}{\epsilon^5} \cdot h^{16})$ with m processors
2. Randomized CONGEST time $\tilde{O}(\frac{1}{\epsilon^5} \cdot h^{16})$ with high probability;
3. Deterministic CONGEST time $\tilde{O}\left(\frac{1}{\epsilon^5} \cdot h^{16} + \frac{1}{\epsilon^3} \cdot h^{15} \cdot (\rho_{CC})^{10}\right)$.

The main idea for computing these objects is to reduce finding them to computing a series of blocking flows in a carefully constructed "length-weight expanded DAG." In particular, by rounding arc weights up to multiples of $\frac{\epsilon}{h}\lambda$ we can essentially discretize the space of weights. Since each path has at most h arcs, it follows that this increases the length of a path by at most only $\lambda\epsilon$. This discretization allows us to construct DAGs from which we may extract blocking flows which we then project back into D and then "decongest" so as to ensure they are feasible flows.

5.10.1 Length-Weight Expanded DAG

We now formally define the length-weight-expanded DAGs on which we compute blocking integral flows. Roughly, the length-weight expanded graph will create many copies of vertices and organize them into a grid where moving further down in rows corresponds to increases in length and moving further along in columns corresponds to increases in weight.

Let $D = (V, A)$ be a digraph with specified source and sink vertices S and T, lengths l, weights w, capacities U and a parameter $\lambda \leq d_w^{(h)}(S, T)$. We let $\tilde{w}$ be w but rounded up to the nearest multiple of $\frac{\epsilon}{h} \cdot \lambda$. That is, for each $a \in A$ we have

$$\tilde{w}_a = \frac{\epsilon \cdot \lambda}{h} \cdot \left\lceil w_a \cdot \frac{h}{\epsilon \cdot \lambda} \right\rceil$$

See Figure 5.5 for an illustration of $\tilde{w}$.

Next, we define the length-weight expanded DAG $D^{(h,\lambda)} = (V', A')$ with capacities U'. See Figure 5.6 for an illustration of $D^{(h,\lambda)}$.

- **Vertices:** We construct the vertices V' as follows. For each each vertex $v \in V$ we make $\kappa = h \cdot (\frac{h}{\epsilon} + 2h)$ copies of v, where we let $v(x, h')$ be one of these vertices; here x ranges over all multiples of $\frac{\epsilon}{h} \cdot \lambda$ up to $(1+2\epsilon) \cdot \lambda$ (of which there are $\frac{h}{\epsilon} + 2h$) and $h' \leq h$. Intuitively,

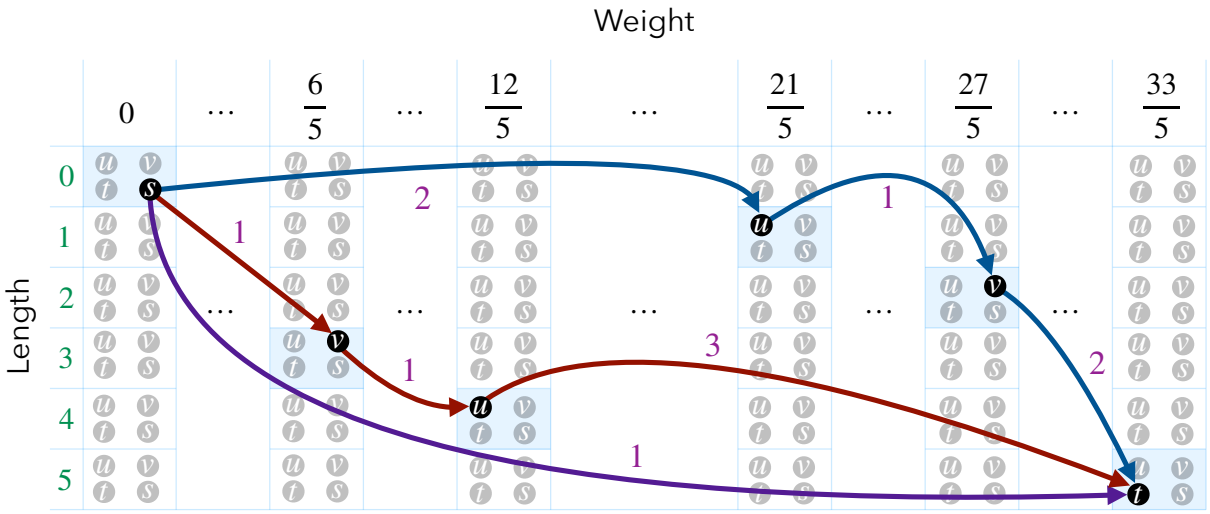


Figure 5.6: An illustration of $D^{(h,\lambda)}$ where D and the parameters we use are given by Figure 5.5, $\kappa = 100$, $S = \{s\}$ and $T = \{t\}$. Copy $v(x, h')$ of vertex v is in the (x, h')th grid cell and each arc is labelled with its capacity. We only illustrate the subgraph between $s(0, 0)$ and $t(\frac{33}{5}, 5)$. Each path is colored according to the path in Figure 5.5b of which it is a copy. Notice that the graph induced by all 5-length lightest paths in Figure 5.5b is not a DAG but $D^{(h,\lambda)}$ is.

there will be a path from a copy of a vertex $s \in S$ to a vertex $v(x, h')$ iff there is a path with exactly x weight (according to $\tilde{w}$) and h'-length from s to v in D.

- **Arcs:** We construct the arcs A' as follows. For each each vertex $v \notin T$ and each $a = (v, u) \in \delta^+(v)$ we do the following. For each copy $v(x, h')$ of v we add an arc to A' from $v(x, h')$ to $u(x + \tilde{w}_a, h' + l_a)$ provided $u(x + \tilde{w}_a, h' + l_a)$ is actually a vertex in V'. That is, provided $x + \tilde{w}_a \leq (1 + 2\epsilon) \cdot \lambda$ and $h' + l_a \leq h$. We say that the arc $v(x, h')$ to $u(x + \tilde{w}_a, h' + l_a)$ in A' is a copy of arc a. For a given $a \in A$, we let $A'(a)$ give all copies of arc a that are in A'.

- **Capacities:** We construct the capacities U' as follows. For low capacity arcs we set the capacity of all copies to 1; for high capacity arcs we evenly distribute the capacity across all copies. Specifically, suppose arc $a' \in A'$ is a copy of arc $a \in A$. Then if $0 < U_{uv} \leq \kappa$ we let $U'_{a'} = 1$. Otherwise, we let $U'_{a'}$ have capacity $\lfloor U_a/\kappa \rfloor$. As we will see later in our proofs, this rebalancing of flows will guarantee that when we "project" a flow from $D^{(h,\lambda)}$ to D, the only arcs that end up overcapacitated in D are arcs with capacity at most κ. This, in turn, will allow us to argue that the conflict graph on which we compute an MIS is small.

We let $V'(S)$ and $V'(T)$ be all copies of S and T in $D^{(h,\lambda)}$ and we delete any vertex from $D^{(h,\lambda)}$ that does not lie on a $V'(S)$ to $V'(T)$ path. This will guarantee that the resulting digraph is indeed an $V'(S)$-$V'(T)$ DAG.

Lastly, we clarify what it means for a path to have its copy in $D^{(h,\lambda)}$. Suppose $P = (a_1, a_2, \ldots)$ is a path in D that visits vertices $s = v_1, v_2, \ldots, v_k = t$ in D and let $\tilde{w}_i$ and l_i be the weight (according to $\tilde{w}$) and length of P summed up to the ith vertex it visits. Then we let a'_i be the arc from $v_i(\tilde{w}_i, l_i)$ to $v_{i+1}(\tilde{w}_i + \tilde{w}_{a_i}, l_i + l_{a_i})$. If a'_i is in $D^{(h,\lambda)}$ for every i then we call $P' = (a'_1, a'_2, \ldots)$ the copy of P in $D^{(h,\lambda)}$. Observe that a path in D has at most one copy in $D^{(h,\lambda)}$ but every path in $D^{(h,\lambda)}$ is the copy of some path in D.

The following summarizes the key properties of our length-weight expanded digraphs.

Lemma 108. *Let $D = (V, A)$ be a digraph with weights w, $S, T \subseteq V$ and some $\lambda \leq d_w^{(h)}(S, T)$. Let $D^{(h,\lambda)} = (V', A)$ be the length-weight expanded digraph of D. Then $D^{(h,\lambda)}$ is an h-layer $V'(S)$-$V'(T)$ DAG which satisfies*

1. ***Few Arc Copies**: $|A'(a)| \leq O(\frac{h^2}{\epsilon})$.*
2. ***Forward Path Projection**: For each path P in D from S to T of weight at most $\lambda \cdot (1+\epsilon)$ according to w, there is a copy of P in $D^{(h,\lambda)}$ from $V'(S)$ to $V'(T)$.*
3. ***Backward Path Projection**: If P' is a $V'(S)$ to $V'(T)$ path in $D^{(h,\lambda)}$ then it is a copy of a path with weight at most $(1+2\epsilon) \cdot \lambda$ according to w.*
4. ***Optimal Flow Preserving**: the maximum S-T flow on $D^{(h,\lambda)}$ has value at least $\Omega(\frac{\epsilon}{h^2})$ times that of the maximum flow on D.*

Proof. First, we argue that $D^{(h,\lambda)}$ is indeed a DAG. To see this, observe that if a' is an arc in A' from $v(x_1, h_1)$ to $v(x_2, h_2)$ then by construction it must be the case that $h_1 < h_2$. It follows that $D^{(h,\lambda)}$ has no cycles and has at most h layers. Next, observe that $D^{(h,\lambda)}$ is a $V'(S)$-$V'(T)$ DAG by construction since we deleted any any vertices that do not lie on a path between $V'(S)$ and $V'(T)$. Additionally, we have $|A'(a)| \leq O(\frac{h^2}{\epsilon})$ for every a since each vertex has at most $O(\frac{h^2}{\epsilon})$-many copies.

Next, consider an arc a with weight w_a according to w and weight $\tilde{w}_a$ according to $\tilde{w}$. Observe that since we are rounding arc weights up we have $w_a \leq \tilde{w}_a$. Combining this with the fact that we are rounding to multiples of $\frac{\epsilon}{h} \cdot \lambda$ we have that

$$w_a \leq \tilde{w}_a \leq w_a + \frac{\epsilon}{h} \cdot \lambda \tag{5.10.1}$$

We next argue our forward path projection property. That is, for each h-length path P in D from S to T of weight at most $\lambda \cdot (1+\epsilon)$ according to w, there is a copy of P in $D^{(h,\lambda)}$ from $V'(S)$ to $V'(T)$. First, observe that P consists of at most h-many edges and so applying Equation (5.10.1), its weight according to $\tilde{w}$ is at most $\lambda \cdot (1+\epsilon) + h \cdot \frac{\epsilon}{h} \cdot \lambda = \lambda \cdot (1+2\epsilon)$. Next, observe that since P has weight at most $\lambda \cdot (1+2\epsilon)$ according to $\tilde{w}$, it must have a copy in $D^{(h,\lambda)}$. In particular, suppose $P = (a_1, a_2, \ldots)$ visits vertices $s = v_1, v_2, \ldots, v_k = t$ in D and let $\tilde{w}_i$ and l_i be the weight (according to $\tilde{w}$) and length of P up to the ith vertex it visits. Then $D^{(h,\lambda)}$ always includes the arc from $v_i(\tilde{w}_i, l_i)$ to $v_{i+1}(\tilde{w}_i + \tilde{w}_{a_i}, l_i + l_{a_i})$ since $\tilde{w}_i \leq (1+2\epsilon)\lambda$ and $l_i \leq h$ for every i.

We argue our backward path projection property. That is, if P' is a $V'(S)$ to $V'(T)$ path in $D^{(h,\lambda)}$ then it is a copy of a path with weight at most $(1+2\epsilon) \cdot \lambda$ in D according to w. Since each arc in $D^{(h,\lambda)}$ is a copy of some arc in D, we know that P' is a copy of *some* path in D. Moreover, since we let $v(x, h')$ only range over $x \in \frac{h}{\epsilon} + 2h$, it follows that the weight of this path according to $\tilde{w}$ is at most $(1+2\epsilon) \cdot \lambda$. However, since weights according to $\tilde{w}$ are only larger than those according to w by Equation (5.10.1), it follows that P' is a copy of a path with weight at most $(1+2\epsilon) \cdot \lambda$ according to w.

Lastly, to see the optimal flow preserving property notice that if f^* is the optimal flow on D then by how chose the capacities of $D^{(h,\lambda)}$ we have that the flow that gives path P' in $D^{(h,\lambda)}$ value $\Theta(\frac{\epsilon}{h^2}) \cdot f_P^*$ where P' is the copy of P is indeed a feasible flow in $D^{(h,\lambda)}$. □

5.10.2 Decongesting Flows

Part of what makes using our length-weight expanded digraph non-trivial is that when we compute a flow in it and then project this flow back into D, the projected flow might not respect capacities. However, this flow will only violate capacities to a bounded extent and so in this section we show how to resolve such flows at a bounded loss in the value of the flow. In the below we say that an h-length flow $\hat{f}$ is α-congested if any arc a where $\hat{f}(a) > U_a$ satisfies $\hat{f}(a) \leq \alpha$.

Lemma 109. *There is a deterministic algorithm that, given a digraph $D = (V, A)$ with capacities U, a length constraint $h \geq 1$, $S, T \subseteq V$ and an h-length α-congested S-T integral flow $\hat{f}$, computes an S-T h-length integral flow f where s.t. $(f) \geq \frac{1}{\alpha^2 h^2} \cdot$ s.t. $(\hat{f})$ in:*

1. *Parallel time $\tilde{O}(\alpha^2 \cdot h)$ with m processors;*
2. *CONGEST time $\tilde{O}(\alpha^3 \cdot h^3)$.*

Proof. The basic idea is to consider the conflict graph induced by our flow paths and then to compute a approximate maximum-weighted independent set among these flow paths where flow paths are weighted according to their flow value.

Specifically, construct our conflict graph $G' = (V', E')$ of $\text{supp}(\hat{f})$ as follows. $V' = \text{supp}(\hat{f})$ has a vertex for each path in the support of $\hat{f}$. We say that P_1 and P_2 in $\text{supp}(\hat{f})$ *conflict* if there is some arc a in both P_1 and P_2 such that $\hat{f}(a) > U_a$. Then we add edge $\{P_1, P_2\}$ to E' iff P_1 and P_2 conflict.

Observe that since each path in $\text{supp}(\hat{f})$ consists of at most h arcs and since $\hat{f}$ is α-congested, we know that the maximum degree in G' is at most $h \cdot \alpha$.

We then apply Theorem 79 to G' to compute a $\frac{1}{h\alpha}$-approximate maximum independent set in G' in deterministic CONGEST time $\tilde{O}(h\alpha)$ with the node weight of $P \in \text{supp}(\hat{f})$ as $\hat{f}_P$. Let $\mathcal{I}$ be this independent set and let $f = \sum_{P \in \mathcal{I}} \hat{f}_P$ be the flow corresponding to this set. We return f.

We first argue that s.t. $(f) \geq \frac{\text{s.t. } (\hat{f})}{\alpha^2 h^2}$ Since the total node weight in G' is s.t. $(\hat{f})$ and the maximum degree in G' is $\alpha \cdot h$, it follows that the maximum independent set in G' has node weight at least $\frac{\text{s.t. } (\hat{f})}{\alpha h}$. Since $\mathcal{I}$ is $\frac{1}{\alpha h}$-approximate, we conclude that f has s.t. $(f) \geq \frac{\text{s.t. } (\hat{f})}{\alpha^2 h^2}$.

Lastly, we argue that we achieve the claimed running times. Notice that the total number of vertices in G' is at most $m \cdot \alpha$ because each congested arc a where $U_a < \hat{f}(a) \leq \alpha$ is contained in at most α integral flow paths. Hence, we can simulate any CONGEST algorithm in G' with at most α overhead. Theorem 79 tells us that we can compute $\mathcal{I}$ in time at most $\tilde{O}(\alpha \cdot h)$ in G', giving our parallel running time.

It remains to describe how to simulate G' in D in CONGEST. We keep the following invariant: if a node P_1 in G' receives a message, we make sure that all vertices $v \in P_1$ in G receive the same message too. Because of this, any vertex $v \in P_1$ in G can determine what P_1 as a node in G' will do next. Let us assume that each message in G' from P_1 to P_2 is of the form $(\texttt{msg}, P_1, P_2)$. To simulate sending $(\texttt{msg}, P_1, P_2)$ in G, a vertex $v_1 \in P_1$ first forwards $(\texttt{msg}, P_1, P_2)$ through P_1 to make sure that every node in P_1 gets this message. Let $v_2 \in P_1 \cap P_2$ be a common vertex in both P_1 and P_2. Then, v_2 forwards $(\texttt{msg}, P_1, P_2)$ through P_2. After we are done simulating all messages sent in G', our invariant is maintained.

Now, we analyze the overhead of simulating one round of G' in G. The dilation for simulating sending each message in G' is clearly $O(h)$. Next, we analyze the congestion. Each arc a is

contained in at most $\max\{U_a, \alpha\} \leq \alpha U_a$ paths. For each such path P, there are at most αh messages needed to sent through a because the maximum degree in G' at most αh. Therefore, the congestion is at most $\frac{\alpha U_a \cdot \alpha h}{U_a} = \alpha^2 h$. Note that, here (and nowhere else in this work) we rely on the fact that we may send $O(U_a)$ messages over an arc a with capacity U_a in one round of CONGEST.

To conclude, the deterministic simulation overhead is at most dilation times congestion which is at most $O(h) \cdot \alpha^2 h = O(\alpha^2 h^2)$. Combining this simulation with the $\tilde{O}(\alpha \cdot h)$ running time of our approximate maximal independent set algorithm gives our CONGEST running time. □

5.10.3 Computing h-Length $(1+\epsilon)$-Lightest Path Blockers

Having described our length-weight expanded DAGs, their properties and how to decongest flows that we compute using them, we now use these primitives to build our h-length $(1+\epsilon)$-lightest path blockers. Again, the basic idea is to compute the length-weight expanded DAG $D^{(h,\lambda)}$, compute blocking flows in $D^{(h,\lambda)}$, project these back into D, decongest the resulting flows and then repeat. Algorithm 2 gives our algorithm. We prove its properties below.

Algorithm 2 $(1+\epsilon)$-Lightest Path Blocker

Input: $D = (V, A)$ with weights w, lengths l, capacities U, $h \geq 1$, $S, T \subseteq V$, $\lambda > 0$ and $\epsilon > 0$.
Output: h-length $(1+\epsilon)$-lightest path blocker f.
Initialize solution f to be 0 on all arcs.
Let $D^{(h,\lambda)} = (V', A')$ be the length-weight expanded digraph of D with capacities $\hat{U} = U$ $\tilde{\Theta}(\frac{h^7}{\epsilon^2})$ repetitions
Blocking Flows: Let f' be a blocking integral flow in $D^{(h,\lambda)}$ with capacities $\hat{U}$ (compute using Theorem 89 with randomness or Theorem 96 deterministically).
Project Into D: Let $\tilde{f}$ be the h-length flow that gives path P value $f'_{P'}$ where P' is the copy of P in $D^{(h,\lambda)}$.
Decongest Flow: Let $\hat{f}$ be the result of decongesting $\tilde{f}$ with Theorem 109.
For each copy $a' \in A'$ of $a \in A$ update capacities as $\hat{U}_{a'} = \hat{U}_{a'} - \hat{f}(a)$.
Update $f = f + \hat{f}$.
f.

Given digraph $D = (V, A)$ with lengths l, weights w, capacities U, length constraint $h \geq 1$, $\epsilon > 0$, $S, T \subseteq V$ and $\lambda \leq d_w^{(h)}(S, T)$, one can compute an h-length $(1+\epsilon)$-lightest path blocker in:

1. Deterministic parallel time $\tilde{O}(\frac{1}{\epsilon^5} \cdot h^{16})$ with m processors
2. Randomized CONGEST time $\tilde{O}(\frac{1}{\epsilon^5} \cdot h^{16})$ with high probability;
3. Deterministic CONGEST time $\tilde{O}\left(\frac{1}{\epsilon^5} \cdot h^{16} + \frac{1}{\epsilon^3} \cdot h^{15} \cdot (\rho_{CC})^{10}\right)$.

Proof. We first argue that f is a h-length $(1+\epsilon)$-lightest path blocker (Theorem 107). f is an integral h-length S-T flow by construction. Moreover, the support of f is near-lightest by the backward path projection property of $D^{(h,\lambda)}$, as stated in Theorem 108.

Thus, it remains to argue the near-lightest path blocking property of f and, in particular that if $P \in \mathcal{P}_h(S, T)$ is a path in D and P has weight at most $(1+\epsilon) \cdot \lambda$ according to w then there is

some $a \in P$ where $f(a) = U_a$. Towards this, observe that by the forward path projection property as stated in Theorem 108, such a path P has copy in $D^{(h,\lambda)}$. By how we construct f, it follows that to show $f(a) = U_a$ for some a, it suffices to show that $\hat{U}_a = 0$ by the end of our algorithm. To show that such an a exists, it suffices to show that the maximum flow in $D^{(h,\lambda)}$ under the capacities $\hat{U}$ is 0 by the end of our algorithm.

We do so now. Our strategy will be to show that we have implicitly computed a flow on $D^{(h,\lambda)}$ of near-optimal value and so after just a few iterations it must be the case that the optimal flow on $D^{(h,\lambda)}$ is reduced to 0.

Consider a fixed iteration of our algorithm and let $\mathrm{OPT}^{(h,\lambda)}$ be the value of the maximum $V'(S)$-$V'(T)$ flow on $D^{(h,\lambda)}$. Since f' is a blocking flow in $D^{(h,\lambda)}$ and $D^{(h,\lambda)}$ is an h-layer DAG by Theorem 108, it follows from Theorem 99 that

$$\text{s.t. } (f') \geq \frac{1}{h} \cdot \mathrm{OPT}^{(h,\lambda)}. \tag{5.10.2}$$

Continuing, we claim that $\tilde{f}$ is an $O(\frac{h^2}{\epsilon})$-congested flow. In particular, any arc a with capacity in D greater than $O(\frac{h^2}{\epsilon})$ is such that the sum of its capacities across copies in $D^{(h,\lambda)}$ is at most $\hat{U}_a$. Thus, such an arc is never overcongested by $\tilde{f}$. Any arc with capacity less than $O(\frac{h^2}{\epsilon})$ in D' has up to $O(\frac{h^2}{\epsilon})$ copies in $D^{(h,\lambda)}$ each of which has capacity 1; thus, such an arc may have flow value up to $O(\frac{h^2}{\epsilon})$ in $\tilde{f}$. Thus, by $\text{s.t. } (\tilde{f}) = \text{ s.t. } (f')$ and this bound on the congestedness of $\tilde{f}$, we have from Theorem 109 that

$$\begin{aligned} \text{s.t. } (\hat{f}) &\geq \frac{\epsilon^2}{h^6} \cdot \text{ s.t. } (\tilde{f}) \\ &= \frac{\epsilon^2}{h^6} \cdot \text{ s.t. } (f'). \end{aligned} \tag{5.10.3}$$

Combining Equation (5.10.2) and Equation (5.10.3), we get

$$\text{s.t. } (\hat{f}) \geq \frac{\epsilon^2}{h^7} \cdot \mathrm{OPT}^{(h,\lambda)}. \tag{5.10.4}$$

Lastly, let f'' be $\hat{f}$ projected back into $D^{(h,\lambda)}$. That is, if arc a' is a copy of arc a then f'' assigns to a' the flow value $\sum_{P \ni a} \hat{f}_P$. Observe that by construction of $\hat{f}$, we know that f'' is a $V'(S)$-$V'(T)$ flow in $D^{(h,\lambda)}$ of value $\text{s.t. } (f'') = \text{ s.t. } (\hat{f})$. Thus, applying this and Equation (5.10.4) we get

$$\text{s.t. } (f'') \geq \frac{\epsilon^2}{h^7} \cdot \mathrm{OPT}^{(h,\lambda)}.$$

Since we decrement the value of $\hat{U}_a$ by f''_a in each iteration, it follows that after $\tilde{O}(\frac{h^7}{\epsilon^2})$ many repetitions of Algorithm 2, we must decrease the value of the optimal flow in $D^{(h,\lambda)}$ by at least a constant fraction since otherwise we would have computed a flow with value greater than that of the optimal flow. Since initially $\mathrm{OPT}^{(h,\lambda)} \leq \mathrm{poly}(n)$, we get that after $\tilde{O}(\frac{h^7}{\epsilon^2})$-many repetitions we have reduced the value of the optimal flow to 0 on $D^{(h,\lambda)}$, therefore showing that f satisfies

the near-lightest path blocking property.

It remains to show our running times. The computation in each of our iterations is dominated by constructing the length-expanded digraph $D^{(h,\lambda)}$, computing our maximal integral flow $f^{(h)}$ in $D^{(h,\lambda)}$ and decongesting our flow.

- We can construct $D^{(h,\lambda)}$ by e.g. Bellman-Ford for $\tilde{O}(h)$ rounds for a total running time of $\tilde{O}(h)$ in either CONGEST or parallel. Likewise projecting flows back from $D^{(h,\lambda)}$ is trivial.
- It is is easy to simulate $D^{(h,\lambda)}$ in either CONGEST or in parallel with an overhead of $O(\frac{h^2}{\epsilon})$ since this is a bound on the number copies of each vertex.

  With randomization, by Theorem 89 computing f' takes time $\tilde{O}(h^3)$ in parallel with m processors or $\tilde{O}(h^4)$ in CONGEST on $D^{(h,\lambda)}$ and so $\tilde{O}(\frac{h^5}{\epsilon})$ in parallel or $\tilde{O}(\frac{h^6}{\epsilon})$ in CONGEST on D.

  For our deterministic algorithm, by Theorem 96 doing so takes $\tilde{O}(h^3)$ in parallel with m processors and CONGEST time $\tilde{O}(h^6 \cdot (\rho_{CC})^{10})$ on $D^{(h,\lambda)}$ and so $\tilde{O}(\frac{1}{\epsilon} \cdot h^5)$ parallel time on D or $\tilde{O}(\frac{1}{\epsilon} \cdot h^8 \cdot (\rho_{CC})^{10})$ CONGEST time on D.
- Lastly, decongesting our flow by Theorem 109 and the fact that $\tilde{f}$ is $O(\frac{h^2}{\epsilon})$-congested takes deterministic parallel time $\tilde{O}(\frac{h^5}{\epsilon^2})$ and deterministic CONGEST time $\tilde{O}(\frac{h^9}{\epsilon^3})$.

Combining these running times with our $\tilde{O}(\frac{h^7}{\epsilon^2})$-many repetitions gives the stated running times. □

5.11 Computing Length-Constrained Flows and Moving Cuts

Having shown how to compute an h-length $(1+\epsilon)$-lightest path blocker, we now use a series of these as batches to which we apply multiplicative-weights-type updates. The result is our algorithm which returns both a length-constrained flow and a (nearly) certifying moving cut.

Algorithm 3 Length-Constrained Flows and Moving Cuts

Input: digraph $D = (V, A)$ with lengths l, capacities U, $h \geq 1$, $S, T \subseteq V$ and $\epsilon \in (0, 1)$.
Output: $(1 \pm \epsilon)$-approximate h-length flow f and moving cut w.
Let $\epsilon_0 = \frac{\epsilon}{6}$, let $\zeta = \frac{1+2\epsilon_0}{\epsilon_0} + 1$ and let $\eta = \frac{\epsilon_0}{(1+\epsilon_0)\cdot\zeta} \cdot \frac{1}{\log m}$.
Initialize $w_a \leftarrow \left(\frac{1}{m}\right)^{\zeta}$ for all $a \in A$.
Initialize $\lambda \leftarrow \left(\frac{1}{m}\right)^{\zeta}$.
Initialize $f_P \leftarrow 0$ for all $P \in \mathcal{P}_h(S, T)$. $\lambda < 1$: $\Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations:
Compute h-length $(1+\epsilon_0)$-lightest path blocker $\hat{f}$ (using Section 5.10 with current λ).
Length-Constrained Flow (Primal) Update: $f \leftarrow f + \eta \cdot \hat{f}$.
Moving Cut (Dual) Update: $w_a \leftarrow (1+\epsilon_0)^{\hat{f}(a)/U_a} \cdot w_a$ for every $a \in A$.
$\lambda \leftarrow (1+\epsilon_0) \cdot \lambda$
(f, w).

As a reminder for an h-length flow f, we let $f(a) := \sum_{P \ni a} f_P$. Throughout our analysis we will refer to the innermost loop of Algorithm 3 as one "iteration." We begin by observing that λ always lower bounds $d_w^{(h)}(S, T)$ in our algorithm.

Lemma 110. *At the beginning of each iteration of Algorithm 3 we have* $\lambda \leq d_w^{(h)}(S,T)$

Proof. Our proof is by induction. The statement trivially holds at the beginning of our algorithm. Let λ_i be the value of λ at the beginning of the ith iteration. We argue that if $d_w^{(h)}(S,T) = \lambda_i$ then after $\Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ additional iterations we must have $d_w^{(h)}(S,T) \geq (1+\epsilon_0)\cdot\lambda_i$. Let $\lambda_i' = (1+\epsilon_0)\cdot\lambda$ be λ after these iterations. Let $\hat{f}_j$ be our lightest path blocker in the jth iteration. Assume for the sake of contradiction that $d_w^{(h)}(S,T) < \lambda_i'$ after $i + \Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations. It follows that there is some path $P \in \mathcal{P}_h(S,T)$ with weight at most λ_i' after $i + \Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ many iterations. However, notice that by definition of an h-length $(1+\epsilon_0)$-lightest path blocker (Theorem 107), we know that for every $j \in \left[i, i+\Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)\right]$ there is some $a \in P$ for which $\hat{f}_j(a) = U_a$. By averaging, it follows that there is some single arc $a \in P$ for which $\hat{f}_j(a) = U_a$ for at least $\Theta\left(\frac{\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ of these $j \in [i, i+\Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)]$. Since every such arc starts with dual value $(\frac{1}{m})^\zeta$ and multiplicatively increases by a $(1+\epsilon_0)$ factor in each of these updates, such an arc after $i+\Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ many iterations must have w_a value at least $(\frac{1}{m})^\zeta\cdot(1+\epsilon_0)^{\Theta\left(\frac{\log_{1+\epsilon_0} n}{\epsilon_0}\right)} \geq n^2$ for an appropriately large hidden constant in our Θ. However, by assumption, the weight of P is at most λ_i' after $i+\Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations and this is at most 2 since $\lambda_i < 1$ since otherwise our algorithm would have halted. But $2 < n^2$ and so we have arrived at a contradiction.

Repeatedly applying the fact that $\lambda_i' = (1+\epsilon_0)\lambda_i$ gives that λ is always a lower bound on $d_w^{(h)}(S,T)$. □

We next prove the feasibility of our solution.

Lemma 111. *The pair* (f,w) *returned by Algorithm 3 are feasible for Length-Constrained Flow LP and Moving Cut LP respectively.*

Proof. First, observe that by Theorem 110 we know that λ is always a lower bound on $d_w^{(h)}(S,T)$ and so since we only return once $\lambda > 1$, the w we return is always feasible.

To see that f is feasible it suffices to argue that for each arc a, the number of times a path containing a has its primal value increased is at most $\frac{U_a}{\eta}$. Notice that each time we increase the primal value on a path containing arc a by η we increase the dual value of this edge by a multiplicative $(1+\epsilon_0)^{1/U_a}$. Since the weight of our arcs according to w start at $(\frac{1}{m})^\zeta$, it follows that if we increase the primal value of k paths incident to arc a then $w_a = (1+\epsilon_0)^{k/U_a}\cdot(\frac{1}{m})^\zeta$. On the other hand, by assumption when we increase the dual value of an arc a it must be the case that $w_a < 1$ since otherwise $d_w^{(h)}(S,T) \geq 1$, contradicting the fact that λ always lower bounds $d_w^{(h)}(S,T)$. It follows that $(1+\epsilon_0)^{k/U_a}\cdot(\frac{1}{m})^\zeta \leq 1$ and so applying the fact that $\ln(1+\epsilon_0) \geq \frac{\epsilon_0}{1+\epsilon_0}$ for $\epsilon_0 > -1$ and our definition of ζ and η we get

$$\begin{aligned} k &\leq \frac{\zeta\cdot(1+\epsilon_0)}{\epsilon_0}\cdot U_a\log m \\ &= \frac{U_a}{\eta} \end{aligned}$$

as desired. □

We next prove the near-optimality of our solution.

Lemma 112. *The pair (f, w) returned by Algorithm 3 satisfies $(1-\epsilon)\sum_a w_a \leq \sum_P f_P$.*

Proof. Fix an iteration i of the above while loop and let $\hat{f}$ be our lightest path blocker in this iteration. Let k_i be s.t. $(\hat{f})$, let λ_i be λ at the start of this iteration and let $D_i := \sum_a w_a$ be our total dual value at the start of this iteration. Notice that $\frac{1}{\lambda_i} \cdot w$ is dual feasible and has cost $\frac{D_i}{\lambda_i}$ by Theorem 110. If β is the optimal dual value then by optimality it follows that $\beta \leq \frac{D_i}{\lambda_i}$, giving us the upper bound on λ_i of $\frac{D_i}{\beta}$. By how we update our dual, our bound on λ_i and $(1+x)^r \leq 1 + xr$ for any $x \geq 0$ and $r \in (0,1)$ we have that

$$\begin{aligned}
D_{i+1} &= \sum_a (1+\epsilon_0)^{\hat{f}(a)/U_a} \cdot w_a \cdot U_a \\
&\leq \sum_a \left(1 + \frac{\epsilon_0 \hat{f}(a)}{U_a}\right) \cdot w_a \cdot U_a \\
&= D_i + \epsilon_0 \sum_a \hat{f}(a) w_a \\
&\leq D_i + \epsilon_0(1+2\epsilon_0) \cdot k_i \lambda_i \\
&\leq D_i \left(1 + \frac{(1+2\epsilon_0)\epsilon_0 \cdot k_i}{\beta}\right) \\
&\leq D_i \cdot \exp\left(\frac{(1+2\epsilon_0)\epsilon_0 \cdot k_i}{\beta}\right).
\end{aligned}$$

Let $T-1$ be the index of the last iteration of our algorithm; notice that D_T is the value of w in our returned solution. Let $K := \sum_i k_i$. Then, repeatedly applying this recurrence gives us

$$\begin{aligned}
D_T &\leq D_0 \cdot \exp\left(\frac{(1+2\epsilon_0)\epsilon_0 \cdot K}{\beta}\right) \\
&= \left(\frac{1}{m}\right)^{\zeta - 1} \exp\left(\frac{(1+2\epsilon_0)\epsilon_0 \cdot K}{\beta}\right)
\end{aligned}$$

On the other hand, we know that w is dual feasible when we return it, so it must be the case that $D_T \geq 1$. Combining this with the above upper bound on D_T gives us $1 \leq \left(\frac{1}{m}\right)^{\zeta} \exp\left(\frac{(1+2\epsilon_0)\epsilon_0 \cdot K}{\beta}\right)$. Solving for K and using our definition of ζ gives us

$$\begin{aligned}
\beta \log m \cdot \frac{\zeta - 1}{(1+2\epsilon_0)\cdot\epsilon_0} &\leq K \\
\beta \log m \cdot \frac{1}{\epsilon_0^2} &\leq K.
\end{aligned}$$

However, notice that $K\eta$ is the primal value of our solution so using our choice of η and rewriting this inequality in terms of $K\eta$ by multiplying by $\eta = \frac{\epsilon_0}{(1+\epsilon_0)\cdot\zeta} \cdot \frac{1}{\log m}$ and applying our definition of

$\zeta = \frac{1+2\epsilon_0}{\epsilon_0} + 1$ gives us

$$\begin{aligned} \frac{\beta}{\epsilon_0 \cdot (1+\epsilon_0) \cdot \zeta} &\leq K\eta \\ \frac{\beta}{(1+\epsilon_0)(1+3\epsilon_0)} &\leq K\eta. \end{aligned} \tag{5.11.1}$$

Moreover, by our choice of $\epsilon_0 = \frac{\epsilon}{6}$ and the fact that $\frac{1}{1+x+x^2} \geq 1 - x$ for $x \in (0,1)$ we get

$$\begin{aligned} 1 - \epsilon &\leq \frac{1}{1+\epsilon+\epsilon^2} \\ &\leq \frac{1}{(1+\frac{1}{2}\epsilon)^2} \\ &\leq \frac{1}{(1+3\epsilon_0)^2} \\ &\leq \frac{1}{(1+\epsilon_0)(1+3\epsilon_0)}. \end{aligned} \tag{5.11.2}$$

Combining Equation (5.11.1) and Equation (5.11.2) we conclude that

$$(1-\epsilon) \cdot \beta \leq K\eta.$$

□

We conclude with our main theorem by proving that we need only iterate our algorithm $\tilde{O}\left(\frac{h}{\epsilon^4}\right)$ times. Given a digraph $D = (V, A)$ with capacities U, lengths l, length constraint $h \geq 1$, $\epsilon > 0$ and source and sink vertices $S, T \subseteq V$, one can compute a feasible h-length flow, moving cut pair (f, w) that is $(1 \pm \epsilon)$-approximate in:

1. Deterministic parallel time $\tilde{O}(\frac{1}{\epsilon^9} \cdot h^{17})$ with m processors
2. Randomized CONGEST time $\tilde{O}(\frac{1}{\epsilon^9} \cdot h^{17})$ with high probability;
3. Deterministic CONGEST time $\tilde{O}\left(\frac{1}{\epsilon^9} \cdot h^{17} + \frac{1}{\epsilon^7} \cdot h^{16} \cdot (\rho_{CC})^{10}\right)$.

Also, $f = \eta \cdot \sum_{j=1}^{k} f_j$ where $\eta = \tilde{\Theta}(\epsilon^2)$, $k = \tilde{O}\left(\frac{h}{\epsilon^4}\right)$ and each f_j is an integral h-length S-T flow.

Proof. We use Algorithm 3. By Theorem 111 and Theorem 112 we know that our solution is feasible and $(1 \pm \epsilon)$-optimal so it only remains to argue the runtime of our algorithm and that the returned flow decomposes in the stated way.

We argue that we must only run for $O\left(\frac{h \log^2 n}{\epsilon^4}\right)$ total iterations. Since λ increases by a multiplicative $(1+\epsilon_0)$ after every $\Theta\left(\frac{h \log n}{\epsilon_0^2}\right)$ iterations and starts at at least $\left(\frac{1}{m}\right)^{\Theta(1/\epsilon_0)}$, it follows by Theorem 110 that after $y \cdot \Theta\left(\frac{h \log n}{\epsilon_0^2}\right)$ total iterations the h-length distance between S and T is at least $(1+\epsilon_0)^y \cdot \left(\frac{1}{m}\right)^{\Theta(1/\epsilon_0)}$. Thus, for $y \geq \Omega\left(\frac{\log_{1+\epsilon_0} m}{\epsilon_0}\right) = \Omega\left(\frac{\log n}{\epsilon_0^2}\right)$ we have that S and T are at least 1 apart in h-length distance. Consequently, our algorithm must run for at most $O\left(\frac{h \log^2 n}{\epsilon_0^4}\right) = O\left(\frac{h \log^2 n}{\epsilon^4}\right)$ many iterations.

Our running time is immediate from the the bound of $O\left(\frac{h\log^2 n}{\epsilon^4}\right)$ on the number of iterations of the while loop and the running times given in Section 5.10 for computing our h-length $(1+\epsilon_0)$-lightest path blocker.

Lastly, the flow decomposes in the stated way because we have at most $O\left(\frac{h\log^2 n}{\epsilon^4}\right)$ iterations and each f_j is an integral S-T flow by Section 5.10. Thus, our final solution is $\eta \cdot \sum_{j=1}^{k} f_j$ and $k = \tilde{O}\left(\frac{h}{\epsilon^4}\right)$. □

5.12 Application: Maximal and Maximum Disjoint Paths

In this section we show that our main theorem (Section 5.3) almost immediately gives deterministic CONGEST algorithms for many varieties of maximal disjoint path problems as well as essentially-optimal algorithms for many maximum disjoint path problems. In Section 5.12.1 we give the variants we study. In Section 5.12.2 we observe that it suffices to solve the arc-disjoint directed variants of these problems. Lastly, we give our results for maximal and maximum disjoint path problems in Section 5.12.3 and Section 5.12.4 respectively where we observe in Section 5.12.5 that our algorithms for the latter are essentially optimal.

5.12.1 Maximal and Maximum Disjoint Path Variants

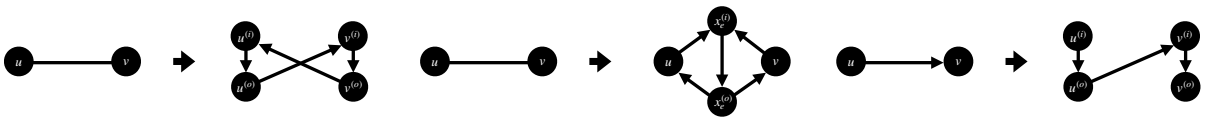
We consider the following maximal disjoint path variants.

> **Maximal Vertex-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of h-length vertex-disjoint S to T paths $\mathcal{P}$ such that any h-length S to T path shares a vertex with at least one path in $\mathcal{P}$.
>
> **Maximal Edge-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of h-length edge-disjoint S to T paths $\mathcal{P}$ such that any h-length S to T path shares an edge with at least one path in $\mathcal{P}$.
>
> **Maximal Vertex-Disjoint Directed Paths**: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of h-length vertex-disjoint S to T paths $\mathcal{P}$ such that any h-length S to T path shares a vertex with at least one path in $\mathcal{P}$.
>
> **Maximal Arc-Disjoint Directed Paths**: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of h-length arc-disjoint S to T paths $\mathcal{P}$ such that any h-length S to T path shares an arc with at least one path in $\mathcal{P}$.

As discussed in Section 5.1.1, the existence of efficient deterministic algorithms for the above problems (specifically the maximal vertex-disjoint paths problem) in CONGEST was stated as an open question by Chang and Saranurak [45] and the lack of these algorithms is a major barrier to simple deterministic constructions of expander decompositions.

We consider the following maximum disjoint path variants.

> **Maximum Vertex-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of h-length vertex-disjoint S to T paths.



(a) Vertex-disjoint paths. **(b)** Edge-disjoint paths. **(c)** Vertex-disjoint directed paths.

Figure 5.7: Illustration of our reduction on a single edge or arc between u and v for reducing maximal or maximum vertex-disjoint paths, edge-disjoint paths or vertex-disjoint directed paths to arc-disjoint directed paths.

Maximum Edge-Disjoint Paths: Given graph $G = (V, E)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of h-length edge-disjoint S to T paths.

Maximum Vertex-Disjoint Directed Paths: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of h-length vertex-disjoint S to T paths.

Maximum Arc-Disjoint Directed Paths: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of h-length arc-disjoint S to T paths.

5.12.2 Reducing Among Variants

We begin by observing that the arc-disjoint directed paths problem is the hardest of the above variants and so it will suffice to solve this problem. The reductions we use are illustrated in Figure 5.7.

Lemma 113. *If there is a deterministic algorithm for maximal arc-disjoint directed paths in CONGEST running in time T then there are deterministic CONGEST algorithms for maximal vertex-disjoint paths, edge-disjoint paths and vertex-disjoint directed paths all running in time $O(T)$.*

Likewise, if there is a deterministic (resp. randomized) parallel with m processors or CONGEST algorithm for maximum arc-disjoint directed paths in CONGEST running in time T with approximation ratio $\tilde{O}(h)$ then there are deterministic (resp. randomized) parallel with m processors and CONGEST algorithms for maximum vertex-disjoint paths, edge-disjoint paths and vertex-disjoint directed paths all running in time $O(T)$ with approximation ratio $\tilde{O}(h)$.

Proof. We reduce each of maximal vertex-disjoint paths, maximal edge-disjoint paths and maximal vertex-disjoint directed paths to maximal arc-disjoint directed paths and do the same for the maximum variants of these problems.

Reducing from maximal/maximum vertex-disjoint paths. Consider an instance of maximal or maximum vertex-disjoint paths on graph $G = (V, E)$ with length constraint h and vertex sets S and T. We create a digraph $D = (V', A)$ as follows:

- **Vertices:** V' is constructed as follows: for each $v \in V$ we add to V' vertex $v^{(i)}$ and $v^{(o)}$.
- **Arcs:** For each $v \in V$ we add an arc from $v^{(i)}$ to $v^{(o)}$. Furthermore, for each $e = \{u, v\} \in E$ we add to A the arcs $(u^{(o)}, v^{(i)})$ and $(v^{(o)}, u^{(i)})$.

A collection of arc-disjoint paths in D from $S' = \{s^{(i)} : s \in S\}$ to $T' = \{t^{(o)} : t \in T\}$ with length constraint $2h - 1$ uniquely corresponds to an equal cardinality collection of S-T vertex-disjoint paths in G with length constraint h. Thus, an $\tilde{O}(h)$ approximation on D for the maximum S'-T' arc-disjoint directed paths problem gives an $\tilde{O}(h)$ approximation for the maximum vertex-disjoint paths problem on G. Likewise, a maximal collection of arc-disjoint S'-T' paths on D with length constraint $2h - 1$ corresponds to a maximal collection of vertex-disjoint S-T paths with length constraint h. Lastly, a T-time CONGEST algorithm on D can be simulated on G in time $O(T)$ since each $v \in V$ can simulate $v^{(o)}$ and $v^{(i)}$.

Reducing from maximal/maximum edge-disjoint paths. Consider an instance of maximal or maximum edge-disjoint paths on graph $G = (V, E)$ with length constraint h and vertex sets S and T. We create a digraph $D = (V', A)$ as follows:

- **Vertices:** V' consists of V along with two vertices for each edge e, namely $x_e^{(i)}$ and $v_e^{(o)}$ for each $e \in E$.
- **Arcs:** For each $e \in \{u, v\} \in E$ we add to A an arc from $x_e^{(i)}$ to $x_e^{(o)}$ as well as an arc from u and v to $x_e^{(i)}$ and an arc from $x_e^{(o)}$ to u and v.

A collection of arc-disjoint S-T paths in D with length constraint $3h$ uniquely corresponds to an equal cardinality collection of S-T edge-disjoint paths in G with length constraint h. Thus, an $\tilde{O}(h)$ approximation on D for the maximum S-T arc-disjoint directed paths problem gives an $\tilde{O}(h)$ approximation for the maximum edge-disjoint paths problem on G. Likewise, a maximal collection of arc-disjoint S-T paths on D with length constraint $3h$ corresponds to a maximal collection of edge-disjoint S-T paths with length constraint h on G. Lastly, a T-time CONGEST algorithm on D can be simulated on G in time $O(T)$ since the endpoints of $e \in E$ can simulate $x_e^{(i)}$ and $x_e^{(o)}$ with constant overhead.

Reducing from maximal/maximum vertex-disjoint directed paths. Consider an instance of maximal or maximum vertex-disjoint directed paths on graph $D = (V, A)$ with length constraint h and vertex sets S and T. We create a digraph $D' = (V', A')$ as follows:

- **Vertices:** V' consists of vertices $v^{(o)}$ and $v^{(i)}$ for each $v \in V$.
- **Arcs:** For each $v \in V$ we add to A' the arc $(v^{(i)}, v^{(o)})$. For each arc $a = (u, v) \in A$ we add to A' the arc $(u^{(o)}, v^{(i)})$.

A collection of arc-disjoint paths in D' from $S' = \{s^{(i)} : s \in S\}$ to $T' = \{t^{(o)} : t \in T\}$ with length constraint $2h - 1$ uniquely corresponds to an equal cardinality collection of S-T vertex-disjoint paths in D with length constraint h. Thus, an $\tilde{O}(h)$ approximation on D' for the maximum S'-T' arc-disjoint directed paths problem gives an $\tilde{O}(h)$ approximation for the maximum S-T vertex-disjoint directed paths problem on D. Likewise, a maximal collection of arc-disjoint S'-T' paths on D' with length constraint $2h - 1$ corresponds to a maximal collection of vertex-disjoint S-T paths with length constraint h on D. Lastly, a T-time CONGEST algorithm on D' can be simulated on D in time T each $v \in V$ can simulate $v^{(i)}$ and $v^{(o)}$. □

5.12.3 Maximal Disjoint Path Algorithms

We now observe that our length-constrained flow algorithms allow us to solve maximal arc-disjoint directed paths and therefore all of the above variants efficiently.

Theorem 114. *There are deterministic CONGEST algorithms for maximal vertex-disjoint paths, edge-disjoint paths, vertex-disjoint directed paths and arc-disjoint directed paths running in time* $\tilde{O}\left(h^{18} + h^{17} \cdot (\rho_{CC})^{10}\right)$.

Proof. By Theorem 113, it suffices to show that maximal arc-disjoint directed paths can be solved in time $\tilde{O}\left(h^{18} + h^{17} \cdot (\rho_{CC})^{10}\right)$. We proceed to do so on digraph D with length constraint h and vertex sets S and T for the rest of this proof.

Specifically, we repeat the following until no path between S and T consists of h or fewer edges. Apply Section 5.3 to compute a $(1-\epsilon)$-approximate h-length S-T flow f in D for $\epsilon = .5$ (any constant would suffice) with unit capacities. By the properties of f as guaranteed by Section 5.3, we have that $f = \eta \cdot \sum_{j=1}^{k} f_j$ for $\eta = \tilde{\Theta}(1)$ and $k = \tilde{O}(h)$ where each f_j is an integral flow. For each vertex v we let $f_j^{(v)}$ be f_j restricted to its flow paths out of v and let $f_{j^*}^{(v)} := \arg\max_{f_j^{(v)}} \text{ s.t. } (f_j^{(v)})$. Then, we let $f_{j^*} := \sum_v f_{j^*}^{(v)}$ (notice that we cannot simply define f_{j^*} as $\arg\max_{f_j} \text{ s.t. } (f_j)$ since we cannot compute $\text{ s.t. } (f_j)$ effiicently in CONGEST because D may have diameter much larger than h). Observe that since f_{j^*} is integral and h-length, it exactly corresponds to an arc-disjoint collection of S-T paths $\mathcal{P}'$ in D each of which consists of at most h edges. We add $\mathcal{P}'$ to $\mathcal{P}$, delete from D any arc incident to a path of $\mathcal{P}'$ and continue to the next iteration.

As the above algorithm removes at least one path from S to T each time, it clearly terminates with a feasible solution for the maximal arc-disjoint directed paths problem.

Stronger, though, we claim that we need only iterate the above $\tilde{O}(h)$-many times until S and T are disconnected. Specifically, fix one iteration and let $\mathcal{P}^*$ be the collection of vertex-disjoint paths from S to T of maximum cardinality at the beginning of this iteration. By the $(1-\epsilon)$-optimality of our flow and an averaging argument we have that $\text{ s.t. } (f_{j^*}) \geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot |\mathcal{P}^*|$ which is to say that $|\mathcal{P}'| \geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot |\mathcal{P}^*|$. However, it follows that after $\tilde{\Theta}(h)$-many iterations for a large hidden constant we must at least halve $|P^*|$ since otherwise we would have computed a collection of vertex-disjoint S-T paths whose cardinality is larger than the largest cardinality of any set of vertex-disjoint S-T paths. Since initially $|P^*| \leq n$, it follows that after iterating the above $\tilde{O}(h)$-many times we have reduced $|P^*|$ to 0 which is to say we have solved the maximal arc-disjoint directed paths problem.

Our running time is immediate from Section 5.3 and the above bound we provide on the number of required iterations of $\tilde{O}(h)$ as well as the fact that each vertex can easily compute $f_{j^*}^{(v)}$ and $\mathcal{P}$ deterministically in parallel or CONGEST time $\tilde{O}(h)$ since our flows are h-length. □

Applying the fact that it is known that $\rho_{CC} \leq 2^{O(\sqrt{\log n})}$ (see Section 5.5.4), the above gives deterministic CONGEST algorithms running in time $\tilde{O}(\text{poly}(h) \cdot 2^{O(\sqrt{\log n})})$. If ρ_{CC} where improved to be poly-log in n then we would get a $\tilde{O}(\text{poly}(h))$ running time.

5.12.4 Maximum Disjoint Path Algorithms

Lastly, we observe that our length-constrained flow algorithms allow us to $\tilde{O}(h)$-approximate maximum arc-disjoint directed paths and therefore all of the above variants efficiently.

Theorem 115. *There are $\tilde{O}(h)$-approximation algorithms for maximum vertex-disjoint paths, edge-disjoint paths, vertex-disjoint directed paths and arc-disjoint directed paths running in:*

- *Deterministic parallel time $\tilde{O}(h^{17})$ with m processors;*
- *Randomized CONGEST time $\tilde{O}(h^{17})$ with high probability;*
- *Deterministic CONGEST time $\tilde{O}\left(h^{17} + h^{16} \cdot (\rho_{CC})^{10}\right)$.*

Proof. By Theorem 113, it suffices to provide a $\tilde{O}(h)$-approximate algorithm for maximum arc-disjoint directed paths with the stated running times. We do so for the rest of this proof. Let the input be digraph $D = (V, A)$ with length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$.

We apply Section 5.3 to compute an ϵ-approximate h-length constrained flow f in D for $\epsilon = .5$ (any constant would suffice) and capacities $U_a = 1$ for every a. By the properties of f as guaranteed by Section 5.3, we have that $f = \eta \cdot \sum_{j=1}^{k} f_j$ for $\eta = \Theta(1)$ and $k = \tilde{O}(h)$ where each f_j is an integral flow. For each vertex v we let $f_j^{(v)}$ be f_j restricted to its flow paths out of v and let $f_{j^*}^{(v)} := \arg\max_{f_j^{(v)}} \text{ s.t. } (f_j^{(v)})$. Then, we let $f_{j^*} := \sum_v f_{j^*}^{(v)}$. Observe that since f_{j^*} is integral and h-length, it exactly corresponds to an arc-disjoint collection of paths $\mathcal{P}$ in D each of which consists of at most h edges. We return $\mathcal{P}$ as our solution.

Letting $\mathcal{P}^*$ be the optimal solution to the input problem we have by $k = \tilde{O}(h)$ and an averaging argument that

$$|\mathcal{P}| = \text{ s.t. } (f_{j^*}) \geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot |\mathcal{P}^*|$$

and so our solution is $\tilde{\Omega}(\frac{1}{h})$-approximate.

For our running time, observe that each vertex can easily compute $f_{j^*}^{(v)}$ and $\mathcal{P}$ deterministically in parallel or CONGEST time $\tilde{O}(h)$ since our flows are h-length. Thus, our running time is dominated by Section 5.3. □

5.12.5 On the Hardness of Maximum Disjoint Paths

Guruswami et al. [107] give hardness results for a variety of length-constrained maximum disjoint path problems. In their work they state hardness of approximation result in terms of m, the number of edges in the graph. In the following we restate these results but in terms of h, the length-constraint.

Theorem 116 (Adaptation of Theorem 1 of Guruswami et al. [107])**.** *Assume the strong exponential time hypothesis (SETH). Then there does not exist a polynomial-time $O(h)$-approximation algorithm solving the maximum arc-disjoint directed paths problem for instances where $h = \Omega(\log n)$.*

Observe that it follows that assuming SETH, the parallel algorithm in Theorem 115 is optimal up to poly-logs.

5.13 Application: Simple Distributed Expander Decompositions

In this section, we explain how our maximal disjoint path algorithm can significantly simplify the distributed deterministic expander decomposition of Chang and Saranurak [45].

The key algorithmic primitive of [45] in their distributed deterministic expander decomposition is their Lemma D.8. Instead of computing maximal bounded-length disjoint paths, they were only be able to compute a set of paths that are "nearly maximal". The formal statement is as follows:

Lemma 117 (Nearly maximal disjoint paths (Lemma D.8 of [45])**.** *Consider a graph $G = (V, E)$ of maximum degree Δ. Let $S \subseteq V$ and $T \subseteq V$ be two subsets. There is an $O(d^3\beta^{-1}\log^2\Delta\log n)$-round deterministic algorithm that finds a set P of $S-T$ vertex-disjoint paths of length at most d, together with a vertex set B of size at most $\beta|V\setminus T| < \beta|V|$, such that any $S-T$ path of length at most d that is vertex-disjoint to all paths in P must contain a vertex in B.*

The set P from the lemma is nearly maximal in the sense that if B is deleted from G, then P would be maximal. However, we can see that there might possibly be many additional disjoint paths that go through B. This set B complicates all of their later algorithmic steps.

The high-level summary of the issue is that all their flow primitives that are based on Lemma D.8 must work with source/sink sets that are very big only. Otherwise, the guarantee becomes meaningless or the running time becomes very slow.

Now, we explain in more details. Given two sets S and T where $|S| \leq |T|$, normally if the matching player from the cut-matching game does not return a sparse cut, then it returns an embedding of a matching where every vertex in S is matched to some vertex in T. However, in Lemma D.9 of [45], the matching player based on Lemma D.8 may return an embedding that leaves as many as $\approx \beta|V\setminus T|$ vertices in S unmatched. This is called the "left-over" set. We think of $\beta \geq 1/n^{o(1)}$ as the round complexity of Lemma D.8 is proportional to β^{-1}. Therefore, it is only when $|S|, |T| \geq 2\beta|V| \geq |V|/n^{o(1)}$ that Lemma D.9 in [45] may give some meaningful guarantee, yet this is still weaker than normal.

The same issue holds for their multi-commodity version of the matching player (i.e. Lemma D.11 of [45]). For the same reasoning, the lemma is meaningful only when the total number of source and sink is at least $\Omega(\beta|V|)$. The issue propagates to their important subroutine (Theorem 4.1 of [45]) for computing most balanced sparse cut. The guarantee holds when only the returned cut C is such that $|C| \geq \Omega(\beta|V|)$. At the end, they managed to obtain an deterministic expander decomposition (just treat the edges incident to the left-over part as inter-cluster edges at the end). However, they need to keep track of this left-over parameter from the first basic primitive until the end result.

In contrast, in their randomized algorithm for computing expander decomposition, this issues does not appear anyway because of the randomized maximal disjoint path algorithm. Therefore, by plugging in our deterministic maximal disjoint path algorithm into the expander decomposition of [45], all these issue will be resolved immediately.

5.14 Application: $(1-\epsilon)$-Approximate Distributed Bipartite b-Matching

In this section we give the first efficient $(1-\epsilon)$-approximate CONGEST algorithms for maximum cardinality bipartite b-matching. In fact, our results are for the slightly more general edge-capacitated maximum bipartite b-matching problem, defined as follow.

> **Edge-Capacitated Maximum Bipartite b-Matching**: Given bipartite graph $G = (V, E)$, edge capacities U and function $b : V \to \mathbb{Z}_{>0}$ compute an integer $x_e \in [0, U_e]$ for each $e \in E$ maximizing $\sum_e x_e$ so that for each $v \in V$ we have $\sum_{e \in \delta(v)} x_e \leq b(v)$.

Notice that the case where $b(v) = 1$ for every v is just the classic maximum cardinality matching problem. "b-matching" seems to refer to two different problems in the literature depending on whether edges can be chosen with multiplicity: either it is the above problem where $U_e = 1$ for every $e \in E$ or it is the above problem where $U_e = \max_v b_v$ for each $e \in E$. Our algorithms will work for both of these variants since they solve the above problem which generalizes both of these problems.

The following theorem summarizes our main result for bipartite b-matching in CONGEST. Again, recall that ρ_{CC} is defined in Theorem 85 and is known to be at most $2^{O(\sqrt{\log n})}$.

Theorem 118. *There is a deterministic $(1-\epsilon)$-approximation for edge-capacitated maximum bipartite b-matching running in CONGEST time $\tilde{O}\left(\frac{1}{\epsilon^9} + \frac{1}{\epsilon^7} \cdot (\rho_{CC})^{10}\right)$.*

Proof. Our algorithm works in two steps. First, we reduce edge-capacitated b-matching to length-constrained flow and use our length constrained flow algorithm to efficiently compute a fractional flow. Then, we apply the flow rounding technology we developed in Section 5.9.2 to round this flow to an integral flow which, in turn, corresponds to an integral b-matching.

More formally our algorithm is as follows. Suppose we are given an instance of edge-capacitated b-matching on bipartite graph $G = (V, E)$. Let L and R be the corresponding bipartition of vertices of G. We construct the following instance of length-constrained flow on digraph $D = (V', A)$ with $h = 3$ as follows. Each $v \in V$ has two copies $v^{(i)}$ and $v^{(o)}$ in V'. We add arc $(v^{(i)}, v^{(o)})$ to A with capacity $b(v)$. If $\{u, v\} \in E$ where $u \in L$ and $v \in R$ then we add arc $(u^{(o)}, v^{(i)})$ with capacity U_e to A. Lastly, we let $S = \{u^{(i)} : u \in L\}$, $T = \{v^{(o)} : v \in R\}$ and the length of each arc in D be 1. Next, we apply Section 5.3 to compute a $(1-\epsilon_1)$-approximate maximum 3-length S-T flow f on D for some small ϵ_1 to be chosen later. Since D is a 3-layer S-T DAG we may interpret this as a (non-length-constrained) flow where the flow value on arc a is $f(a)$.

We then apply Theorem 101 to this non-length-constrained flow to get integral S-T flow f' satisfying s.t. $(f') \geq (1-\epsilon_2) \cdot$ s.t. (f) for some small ϵ_2 to be chosen later. We return as our solution the b-matching which naturally corresponds to f'. Namely, if $e = \{u, v\}$ then since f' is integral it assigns arc $(u^{(o)}, v^{(i)})$ a value in $\{0, 1, \ldots, U_e\}$. We let x_e be this value for $e = \{u, v\}$ and we return as our b-matching solution $\{x_e\}_e$.

f' is a $(1-\epsilon_1)(1-\epsilon_2)$-approximate maximum S-T flow. Letting OPT be the value of the optimal b-matching solution, it is easy to see that the maximum S-T flow has value OPT and so the solution we return has value at least $(1-\epsilon_1)(1-\epsilon_2) \cdot \text{OPT}$. Letting $\epsilon_1 = \epsilon_2 = \Theta(\epsilon)$ for an appropriately small hidden constant we get that $(1-\epsilon_1)(1-\epsilon_2) \cdot \text{OPT} \geq (1-\epsilon) \cdot \text{OPT}$.

Lastly, we argue our running time. Our running time is dominated by one call to Section 5.3

with $\epsilon_1 = \Theta(\epsilon)$ which takes $\tilde{O}\left(\frac{1}{\epsilon^9} + \frac{1}{\epsilon^7} \cdot (\rho_{CC})^{10}\right)$ and one call to Theorem 101 with $\epsilon_2 = \Theta(\epsilon)$ which takes $\tilde{O}(\frac{1}{\epsilon^5} \cdot (\rho_{CC})^{10})$. Combining these running times gives the overall running time of our algorithm. □

5.15 Application: Length-Constrained Cutmatches

As it captures low-latency communication subject to bandwidth constraints, the problem of computing low-congestion h-length paths between two set of nodes S and T occurs often in network optimization.

In this section we give algorithms that either finds a low-congestion h-length collection of paths between two sets of nodes or, if this is not possible, finds as large of such a collection of paths as possible together with a moving cut that (approximately) certifies that there is no low-congestion way of extending the current collection of paths. Such a construction is called a length-constrained cutmatch.

A recent work [115] uses the algorithms we give for cutmatches to give the first efficient constructions of a length-constrained version of expander decompositions. These constructions were then used to give the first distributed CONGEST algorithms for many problems including MST, $(1+\epsilon)$-min-cut and $(1+\epsilon)$-lightest paths that are guaranteed to run in sub-linear rounds as long as such algorithms exist on the input network.

In what follows, for a vertex subset $W \subseteq V$ we let $U^+(W) = \sum_{v \in W} \sum_{a \in \delta^+(v)} U_a$ and $U^-(W) = \sum_{v \in W} \sum_{a \in \delta^-(v)} U_a$. We also let $\delta^{\pm}(S,T) := \bigcup_{v \in S} \delta^+(v) \cup \bigcup_{v \in T} \delta^-(T)$

Definition 119 (h-Length Cutmatch)**.** *Given digraph $D = (V, A)$ with capacities U and lengths l, an h-length ϕ-sparse cutmatch of congestion γ between two node sets $S, T \subseteq V$ with $U^+(S) \leq U^-(T)$ consists of:*

- *An integral h-length S-T flow f in D with capacities $\{U_a\}_{a \in \delta^{\pm}(S,T)} \cup \{\gamma \cdot U_a\}_{a \notin \delta^{\pm}(S,T)}$;*
- *A moving cut w of S and T in D with capacities $\{U_a - f_a\}_{a \in \delta^{\pm}(S,T)} \cup \{U_a\}_{a \notin \delta^{\pm}(S,T)}$ of value $\sum_a w_a \leq \phi\,(U^+(S) - \text{ s.t. } (f))$.*

We proceed to show how to efficiently compute a cutmatch using our previous algorithm. As a reminder ρ_{CC} is defined in Section 5.5.4 and is known to be at most $2^{O(\sqrt{\log n})}$.

Theorem 120. *Suppose we are given a digraph $D = (V, A)$ with capacities U and lengths l. There is an algorithm that, given two node sets $S, T \subseteq V$ with $U^+(S) \leq U^-(T)$ and two integer parameters $h \geq 1$ and $\phi \leq 1$, outputs an h-length ϕ-sparse cutmatch of congestion γ between S and T, where $\gamma = \tilde{O}(\frac{1}{\phi})$. This algorithm runs in:*

1. *Deterministic parallel time $\tilde{O}(\gamma \cdot h^{18})$ with m processors*
2. *Randomized CONGEST time $\tilde{O}(\gamma \cdot h^{18})$ with high probability;*
3. *Deterministic CONGEST time $\tilde{O}\left(\gamma \cdot h^{18} + \gamma \cdot h^{17} \cdot (\rho_{CC})^{10}\right)$.*

Proof. We initialize the flow we return f to be 0 on all arcs. We set our working capacities to be $\hat{U} = U$ initially. The algorithm runs for at most $O(h \cdot \gamma)$ iterations for a small hidden constant. In each iteration $i \in [1, O(h \cdot \gamma)]$ we use Section 5.3 with $\epsilon = .5$ (any constant would suffice) to find a length-constrained flow, moving cut pair, $(\hat{f}, \hat{w})$ where $\delta = \tilde{\Theta}(1)$, $k = \tilde{O}(h)$, $f = \delta \cdot \sum_{j=1}^{k} f_j$

and f_j is an integral h-length flow from S to T using capacities $\hat{U}$. By averaging there must be some f_j such that s.t. $(f_j) \geq$ s.t. $(\hat{f})/k$. We let $\tilde{f}$ be this f_j.

If s.t. $(\tilde{f}) > \Omega\left(\frac{\log n \cdot \hat{U}^+(S)}{h\gamma}\right)$ then we update our solution as $f = f + \tilde{f}$ and decrement $\hat{U}_a$ by $\tilde{f}(a)$ for every $a \in \delta^{\pm}(S,T)$. Otherwise, we return the pair $(f, \hat{w})$ as our solution.

In each iteration i in which s.t. $(\tilde{f}) > \Omega\left(\frac{\log n \cdot \hat{U}^+(S)}{h\gamma}\right)$, we have that $\hat{U}^+(S)$ decreases multiplicatively by at least a $1 - \frac{2\log n}{h\gamma}$ factor. Such a shrinking can happen at most $O(h \cdot \gamma)$ times until $\hat{U}^+(S)$ is reduced to 0. Thus, our algorithm requires at most $O(h \cdot \gamma)$ iterations until terminating. Furthermore, notice when we return a moving cut $\hat{w}$ we have

$$\begin{aligned}
\sum_a \hat{w}_a &\leq 2 \cdot \text{ s.t. } (\hat{f}) \\
&\leq h \cdot \text{ s.t. } (\tilde{f}) \\
&\leq \tilde{O}\left(\frac{\hat{U}^+(S)}{\gamma}\right) \\
&= \tilde{O}\left(\frac{U^+(S) - \text{ s.t. } (f)}{\gamma}\right)
\end{aligned}$$

as desired. Also, observe that f is indeed an integral S-T flow in D with the stated capacities since we always have $\tilde{f}(a) \leq \hat{U}_a$.

The running time is exactly that of running at most $O(h \cdot \gamma)$ invocations of Section 5.3 with $\epsilon = .5$. □

5.16 Conclusion and Future Work

In this chapter we gave the first efficient randomized and deterministic algorithms for computing $(1-\epsilon)$-approximate length-constrained flows both in parallel and in the CONGEST model of distributed computation. We used these algorithms to give new results in maximal and maximum disjoint path problems, expander decompositions, bipartite b-matching and length-constrained cutmatches. We conclude with several open questions and directions for future work.

1. Our length-constrained flow algorithms have a dependence of $\text{poly}(h)$ which when plugged into the techniques of Haeupler et al. [115] give CONGEST algorithms for many distributed problems, e.g. MST, whose running time is $\text{poly}(\text{OPT})$ (up to sub-linear factors) where OPT is the optimal CONGEST running time for the input problem. It would be exciting to improve the dependence on h of our algorithms to, say, $O(h)$ as this result when combined with those of Haeupler et al. [115] would give CONGEST algorithms running in time $O(\text{OPT})$ (up to sub-linear factors).
2. The running time of many of our algorithms depends on ρ_{CC}, the best quality of a CONGEST algorithm for cycle cover (as defined in Theorem 85). It is known that $\rho_{CC} \leq 2^{O(\sqrt{\log n})}$ but it would be extremely interesting to show that $\rho_{CC} \leq \tilde{O}(1)$. Such an improvement would immediately improve the dependency on n from $n^{o(1)}$ to $\tilde{O}(1)$ for our CONGEST algorithms for deterministic length-constrained flows, deterministic maxi-

mal and maximum disjoint paths, $(1-\epsilon)$-approximate b-matching and length-constrained cutmatches. Such a result does not seem to be known even for the randomized case.

3. Lastly, many classic problems can be efficiently solved by reducing to flows but, in particular, by reducing to length-constrained flows with a length-constraint $h = O(1)$. Indeed this is how we were able to give new algorithms for b-matching in this work. It would be interesting to understand which additional classic problems our length-constrained flow algorithms give new algorithms for in CONGEST.

5.17 Generalizing Our Results to Multi-Commodity

In this section we generalize our main result for computing length-constrained flows and moving cuts to the setting where have many source sink pairs and we are trying to maximize the total flow between corresponding pairs subject to congestion constraints.

5.17.1 Multi-Commodity Flows, Cutmatches and Results

We now more formally define a multi-commodity length-constrained flow and moving cut. Suppose we are given a digraph $D = (V, A)$ with arc capacities U, lengths l and κ source set, sink set pairs $\{(S_i, T_i)\}_i$. Then, we have the following LP with a variable $f_P^{\{i\}}$ for each i and path $P \in \mathcal{P}_h(S_i, T_i)$. We let $f^{\{i\}}$ gives the entire flow for commodity i.

$$\max \sum_i \sum_{P \in \mathcal{P}_h(S_i,T_i)} f_P^{\{i\}} \text{ s.t.} \qquad \text{(Multi Length-Constrained Flow LP)}$$

$$\sum_i \sum_{P \ni a} f_P^{\{i\}} \leq U_a \qquad \forall a \in A$$

$$0 \leq f_P^{\{i\}} \qquad \forall i \in [\kappa], P \in \mathcal{P}_h(S_i, T_i)$$

For a multi-commodity length-constrained flow f, we will use the shorthand $f(a) = \sum_i \sum_{P \ni a} f_P^{\{i\}}$. Likewise we let s.t. $(f) = \sum_i$ s.t. $(f^{\{i\}})$ be the total flow we send. An h-length multi-commodity flow, then, is simply a feasible solution to this LP.

Definition 121 (h-Length Multi-Commodity Flow)**.** *Given digraph $D = (V, A)$ with lengths l, capacities U and source, sink pairs $\{(S_i, T_i)\}_i$, an h-length $\{(S_i, T_i)\}_i$ flow is any feasible solution to Multi Length-Constrained Flow LP.*

With the above definition of multi-commodity length-constrained flows we can now define moving cuts as the dual of length-constrained flows. In particular, taking the dual of the above LP we get the multi-commodity moving cut LP with a variable w_a for each $a \in A$ and a variable y_i for every $i \in [\kappa]$.

$$\min \sum_{a \in A} U_a \cdot w_a \quad \text{s.t.} \qquad \text{(Multi Moving Cut LP)}$$

$$\sum_{a \in P} w_a \geq 1 \qquad \forall i \in [\kappa], P \in \mathcal{P}_h(S_i, T_i)$$

$$0 \leq w_a \qquad \forall a \in A, i \in [\kappa]$$

A multi-commodity h-length moving cut is simply a feasible solution to this LP.

Definition 122 (h-Length Moving Cut)**.** *Given digraph $D = (V, A)$ with lengths l, capacities U and source, sink pairs $\{(S_i, T_i)\}_i$, a multi-commodity h-length moving cut is any feasible solution to Multi Moving Cut LP.*

We will use f and w to stand for solutions to Multi Length-Constrained Flow LP and Multi Moving Cut LP respectively. We say that (f, w) is a feasible pair if both f and w are feasible for their respective LPs and that (f, w) is $(1 \pm \epsilon)$-approximate for $\epsilon > 0$ if the moving cut certifies the value of the length-constrained flow up to a $(1 - \epsilon)$; i.e. if $(1 - \epsilon) \sum_a U_a \cdot w_a \leq \min_i$ s.t. $(f^{\{i\}})$.

When we are working in CONGEST we will say that f is computed if each vertex v stores the value $f_a^{(h',i)} := \sum_{P \in \mathcal{P}_{h,h'}(s,a,t)} f_P^{\{i\}}$. Here, we let $\mathcal{P}_{h,h'}(s, a, t)$ be all paths in $\mathcal{P}_h(s, t)$ of the form $P' = (a_1, a_2, \ldots a, b_1, b_2, \ldots)$ where the path $(a, b_1, b_2, \ldots)$ has length exactly h' according to l. We say multi-commodity moving cut w is computed in CONGEST if each vertex v knows the value of w_a for every arc incident to v. Likewise, we imagine that each node in the first round knows the capacities and lengths of its incident edges.

With the above notions, we can now state our main result for multi-commodity length-constrained flows and moving cuts which say that one can compute a feasible pair (f, w) in parallel and distributedly. In the following we say that length-constrained flow f is integral if $f_P^{\{i\}}$ is an integer for every path in $\mathcal{P}_h(S_i, T_i)$ for every i.

More generally than κ commodities, we solve the problem provided our commodoties can be grouped into κ batches that are far apart.

Definition 123 (κ-Batchable)**.** *Given digraph D with lengths l and source, sink set pairs $\{S_i, T_i\}_i$ we say that a $\{S_i, T_i\}_i$ is κ-batchable if the pairs of $\{S_i, T_i\}_i$ can be partitioned into batches $\{\mathcal{S}_j, \mathcal{T}_j\}_j$ if*

1. *For each i there some j such that $S_i \in \mathcal{S}_j$ and $T_i \in \mathcal{T}_j$;*
2. *For each i and i', if $v \in S_i \cup T_i$ and $v' \in S_{i'} \cup T_{i'}$ and $S_i, S_{i'} \in \mathcal{S}_j$ for some j then $d_l(v, v') > 2h$.*

Observe that if the number of commodities is κ then the set of source, sink pairs is trivially κ-batchable.

The following summarizes our main result for computing multi-commodity length-constrained flows and moving cuts. Given a digraph $D = (V, A)$ with capacities U, lengths l, length constraint $h \geq 1$, $0 < \epsilon < 1$ and source and sink vertices $S, T \subseteq V$, and κ-batchable source, sink pairs $\{S_i, T_i\}_i$, one can compute a feasible multi-commodity h-length flow, moving cut pair (f, w) that is $(1 \pm \epsilon)$-approximate in:

1. Deterministic parallel time $\tilde{O}(\kappa \cdot \frac{1}{\epsilon^9} \cdot h^{17})$ with m processors
2. Randomized CONGEST time $\tilde{O}(\kappa \cdot \frac{1}{\epsilon^9} \cdot h^{17})$ with high probability;
3. Deterministic CONGEST time $\tilde{O}\left(\kappa \cdot \frac{1}{\epsilon^9} \cdot h^{17} + \kappa \cdot \frac{1}{\epsilon^7} \cdot h^{16} \cdot (\rho_{CC})^{10}\right)$.

Furthermore, $f = \eta \cdot \sum_{j=1}^{k} f_j$ where $\eta = \tilde{\Theta}(\epsilon^2)$, $k = \tilde{O}\left(\kappa \cdot \frac{h}{\epsilon^4}\right)$ and f_j is an integral h-length S_i-T_i flow for some i.

Using the above algorithm, we can compute a multi-commodity version of cutmatches. As

before, for a vertex subset $W \subseteq V$ we let $U^+(W) = \sum_{v \in W} \sum_{a \in \delta^+(v)} U_a$ and $U^-(W) = \sum_{v \in W} \sum_{a \in \delta^-(v)} U_a$. We also let $\delta^{\pm}(S,T) := \bigcup_{v \in S} \delta^+(v) \cup \bigcup_{v \in T} \delta^-(T)$. The following formalizes the object we compute.

Definition 124 (Multi-Commodity h-Length Cutmatch)**.** *Given digraph $D = (V, A)$ with capacities U and lengths l, an h-length ϕ-sparse cutmatch of congestion γ between sink source node sets $\{(S_i, T_i)\}_i$ where $S_i, T_i \subseteq V$ for each i with $U^+(S_i) \leq U^-(T_i)$ consists of:*

- *An integral multi-commodity h-length $\{(S_i, T_i)\}_i$ flow f in D where for each arc a we have*

$$\sum_{i: a \in \delta^{\pm}(S_i, T_i)} f^{\{i\}}(a) + \frac{1}{\gamma} \cdot \sum_{i: a \notin \delta^{\pm}(S_i, T_i)} f^{\{i\}}(a) \leq U_a;$$

- *A multi-commodity moving cut w of $\{(S_i, T_i)\}_i$ in D where arc a has capacity*

$$U_a - \sum_{i: a \in \delta^{\pm}(S_i, T_i)} f^{\{i\}}(a)$$

and w has value

$$\sum_a w_a \leq \phi \left(\sum_i U^+(S_i) - \textit{ s.t. } (f) \right).$$

Using the above algorithm for multi-commodity h-length flows, we can efficiently compute multi-commodity h-length cutmatches. Suppose we are given a digraph $D = (V, A)$ with capacities U and lengths l. There is an algorithm that, given κ-batchable sink source pairs $\{(S_i, T_i)\}_i$ where $S_i, T_i \subseteq V$ with $U^+(S_i) \leq U^-(T_i)$ for every i and two integer parameters $h \geq 1$ and $\phi \leq 1$, outputs a multi-commodity h-length ϕ-sparse cutmatch of congestion γ between S and T, where $\gamma = \tilde{O}(\frac{1}{\phi})$. This algorithm runs in:

1. Deterministic parallel time $\tilde{O}(\kappa^2 \cdot \gamma \cdot h^{18})$ with m processors
2. Randomized CONGEST time $\tilde{O}(\kappa^2 \cdot \gamma \cdot h^{18})$ with high probability;
3. Deterministic CONGEST time $\tilde{O}\left(\kappa^2 \cdot \gamma \cdot h^{18} + \kappa \cdot \gamma \cdot h^{17} \cdot (\rho_{CC})^{10}\right)$.

5.17.2 Computing Multi-Commodity Length-Constrained Flows and Moving Cuts

We proceed to use our $(1+\epsilon)$-lightest path blockers and multiplicative weights to compute multi-commodity length-constrained flows and moving cuts. Our strategy is more or less that of Section 5.11 but now we iterate through our batches of commodities; our analysis is mostly unchanged but we include it here for completeness.

Formally, our algorithm is given in Algorithm 4. Throughout our analysis we will refer to the innermost loop of Algorithm 4 as one "iteration."

We begin by observing that λ always lower bounds $d_w^{(h)}(S_i, T_i)$ for every i.

Lemma 125. *It always holds that $\lambda \leq d_w^{(h)}(S_x, T_x)$ for every x in Algorithm 4.*

Algorithm 4 Multi-Commodity Length-Constrained Flows and Moving Cuts

Input: digraph $D = (V, A)$ with lengths l, capacities U, length constraint h and κ-batchable source, sink pairs $\{S_i, T_i\}_i$ where $S_i, T_i \subseteq V$ for every i and an $\epsilon \in (0, 1)$.
Output: $(1 \pm \epsilon)$-approximate h-length multi-commodity flow f and moving cut w.
Let $\epsilon_0 = \frac{\epsilon}{6}$, let $\zeta = \frac{1+2\epsilon_0}{\epsilon_0} + 1$ and let $\eta = \frac{\epsilon_0}{(1+\epsilon_0)\cdot\zeta} \cdot \frac{1}{\log m}$.
Initialize $w_a \leftarrow \left(\frac{1}{m}\right)^{\zeta}$ for all $a \in A$.
Initialize $\lambda \leftarrow \left(\frac{1}{m}\right)^{\zeta}$.
Initialize $f_P^{\{i\}} \leftarrow 0$ for all i and $P \in \mathcal{P}_h(S_i, T_i)$. $\lambda < 1$:
  $j \in [\kappa]$ and each batch $(\mathcal{S}_j, \mathcal{S}_j)$ each (S_i, T_i) with $S_i \in \mathcal{S}_j$ and $T_i \in \mathcal{T}_j$ in parallel $\Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ repetitions
Compute an h-length $(1+\epsilon_0)$-lightest path blocker $\hat{f}$ (using Section 5.10 with λ).
Length-Constrained Flow (Primal) Update: $f^{\{i\}} \leftarrow f^{\{i\}} + \eta \cdot \hat{f}$.
Moving Cut (Dual) Update: $w_a \leftarrow (1+\epsilon_0)^{\hat{f}(a)/U_a} \cdot w_a$ for every $a \in A$.
$\lambda \leftarrow (1+\epsilon_0) \cdot \lambda$
(f, w).

Proof. Fix an x and a value of λ and let $S = S_x$ and $T = T_x$. Our proof is by induction. The statement trivially holds at the beginning of our algorithm.

Let λ_i be the value of λ at the beginning of the ith iteration. We argue that if $d_w^{(h)}(S, T) = \lambda_i$ then after $\Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ additional iterations we must have $d_w^{(h)}(S, T) \geq (1+\epsilon_0) \cdot \lambda_i$. Let $\lambda_i' = (1+\epsilon_0) \cdot \lambda$ be λ after these iterations. Let $\hat{f}_j$ be our lightest path blocker in the jth iteration for (S_x, T_x).

Assume for the sake of contradiction that $d_w^{(h)}(S, T) < \lambda_i'$ after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations. It follows that there is some path $P \in \mathcal{P}_h(S, T)$ with weight at most λ_i' after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ many iterations. However, notice that by definition of an h-length $(1+\epsilon_0)$-lightest path blocker $\hat{f}_j$ (Theorem 107), we know that for every $j \in \left[i, i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)\right]$ there is some $a \in P$ for which $\hat{f}_j(a) = U_a$. By averaging, it follows that there is some single arc $a \in P$ for which $\hat{f}_j(a) = U_a$ for at least $\Theta\left(\frac{\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ of these $j \in [i, i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)]$. Since every such arc starts with dual value $(\frac{1}{m})^{\zeta}$ and multiplicatively increases by a $(1+\epsilon_0)$ factor in each of these updates, such an arc after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ many iterations must have w_a value at least $(\frac{1}{m})^{\zeta} \cdot (1+\epsilon_0)^{\Theta\left(\frac{\log_{1+\epsilon_0} n}{\epsilon_0}\right)} \geq n^2$ for an appropriately large hidden constant in our Θ. However, by assumption, the weight of P is at most λ_i' after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations and this is at most 2 since $\lambda_i < 1$ since otherwise our algorithm would have halted. But $2 < n^2$ and so we have arrived at a contradiction.

Repeatedly applying the fact that $\lambda_i' = (1+\epsilon_0)\lambda_i$ gives that λ is always a lower bound on $d_w^{(h)}(S, T)$. □

We next prove the feasibility of our solution.

Lemma 126. *The pair (f, w) returned by Algorithm 4 are feasible for Multi Length-Constrained*

Flow LP and Multi Moving Cut LP respectively.

Proof. First, observe that by Theorem 125 we know that λ is always a lower bound on $d_w^{(h)}(S_i, T_i)$ for every i and so since we only return once $\lambda > 1$, the w we return is always feasible.

To see that f is feasible it suffices to argue that for each arc a, the number of times a path containing a has its primal value increased is at most $\frac{U_a}{\eta}$. Notice that each time we increase the primal value on a path containing arc a by η we increase the dual value of this edge by a multiplicative $(1+\epsilon_0)^{1/U_a}$. Since the weight of our arcs according to w start at $(\frac{1}{m})^\zeta$, it follows that if we increase the primal value of k paths incident to arc a then $w_a = (1+\epsilon_0)^{k/U_a} \cdot (\frac{1}{m})^\zeta$. On the other hand, by assumption when we increase the dual value of an arc a it must be the case that $w_a < 1$ since otherwise $d_w^{(h)}(S,T) \geq 1$, contradicting the fact that λ always lower bounds $d_w^{(h)}(S,T)$. It follows that $(1+\epsilon_0)^{k/U_a} \cdot (\frac{1}{m})^\zeta \leq 1$ and so applying the fact that $\ln(1+\epsilon_0) \geq \frac{\epsilon_0}{1+\epsilon_0}$ for $\epsilon_0 > -1$ and our definition of ζ and η we get

$$\begin{aligned} k &\leq \frac{\zeta \cdot (1+\epsilon_0)}{\epsilon_0} \cdot U_a \log m \\ &= \frac{U_a}{\eta} \end{aligned}$$

as desired. □

We next prove the near-optimality of our solution.

Lemma 127. *The pair (f, w) returned by Algorithm 4 satisfies $(1-\epsilon)\sum_a w_a \leq \sum_i \sum_{P \in \mathcal{P}_h(S_i,T_i)} f_P$.*

Proof. Fix an iteration i which is an iteration for the jth batch and let $\hat{f}$ be the sum of all lightest path blockers that we compute in parallel for each $(S_i, T_i) \in (\mathcal{S}_j, \mathcal{T}_j)$ in this iteration. Let k_i be s.t. $(\hat{f})$, let λ_i be λ at the start of this iteration and let $D_i := \sum_a w_a$ be our total dual value at the start of this iteration. Notice that $\frac{1}{\lambda_i} \cdot w$ is dual feasible and has cost $\frac{D_i}{\lambda_i}$ by Theorem 125. If β is the optimal dual value then by optimality it follows that $\beta \leq \frac{D_i}{\lambda_i}$, giving us the upper bound on λ_i of $\frac{D_i}{\beta}$. By how we update our dual, our bound on λ_i and $(1+x)^r \leq 1 + xr$ for any $x \geq 0$ and $r \in (0,1)$ we have that

$$\begin{aligned} D_{i+1} &= \sum_a (1+\epsilon_0)^{\hat{f}(a)/U_a} \cdot w_a \cdot U_a \\ &\leq \sum_a \left(1 + \frac{\epsilon_0 \hat{f}(a)}{U_a}\right) \cdot w_a \cdot U_a \\ &= D_i + \epsilon_0 \sum_a \hat{f}(a) w_a \\ &\leq D_i + \epsilon_0(1+2\epsilon_0) \cdot k_i \lambda_i \\ &\leq D_i \left(1 + \frac{(1+2\epsilon_0)\epsilon_0 \cdot k_i}{\beta}\right) \\ &\leq D_i \cdot \exp\left(\frac{(1+2\epsilon_0)\epsilon_0 \cdot k_i}{\beta}\right). \end{aligned}$$

Let $T-1$ be the index of the last iteration of our algorithm; notice that D_T is the value of w in our returned solution. Let $K := \sum_i k_i$. Then, repeatedly applying this recurrence gives us

$$\begin{aligned} D_T &\leq D_0 \cdot \exp\left(\frac{(1+2\epsilon_0)\epsilon_0 \cdot K}{\beta}\right) \\ &= \left(\frac{1}{m}\right)^{\zeta-1} \exp\left(\frac{(1+2\epsilon_0)\epsilon_0 \cdot K}{\beta}\right) \end{aligned}$$

On the other hand, we know that w is dual feasible when we return it, so it must be the case that $D_T \geq 1$. Combining this with the above upper bound on D_T gives us $1 \leq \left(\frac{1}{m}\right)^{\zeta} \exp\left(\frac{(1+2\epsilon_0)\epsilon_0 \cdot K}{\beta}\right)$. Solving for K and using our definition of ζ gives us

$$\begin{aligned} \beta \log m \cdot \frac{\zeta - 1}{(1+2\epsilon_0)\cdot\epsilon_0} &\leq K \\ \beta \log m \cdot \frac{1}{\epsilon_0^2} &\leq K. \end{aligned}$$

However, notice that $K\eta$ is the primal value of our solution so using our choice of η and rewriting this inequality in terms of $K\eta$ by multiplying by $\eta = \frac{\epsilon_0}{(1+\epsilon_0)\cdot\zeta} \cdot \frac{1}{\log m}$ and applying our definition of $\zeta = \frac{1+2\epsilon_0}{\epsilon_0} + 1$ gives us

$$\begin{aligned} \frac{\beta}{\epsilon_0 \cdot (1+\epsilon_0) \cdot \zeta} &\leq K\eta \\ \frac{\beta}{(1+\epsilon_0)(1+3\epsilon_0)} &\leq K\eta. \end{aligned} \tag{5.17.1}$$

Moreover, by our choice of $\epsilon_0 = \frac{\epsilon}{6}$ and the fact that $\frac{1}{1+x+x^2} \geq 1 - x$ for $x \in (0,1)$ we get

$$\begin{aligned} 1 - \epsilon &\leq \frac{1}{1+\epsilon+\epsilon^2} \\ &\leq \frac{1}{(1+\frac{1}{2}\epsilon)^2} \\ &\leq \frac{1}{(1+3\epsilon_0)^2} \\ &\leq \frac{1}{(1+\epsilon_0)(1+3\epsilon_0)}. \end{aligned} \tag{5.17.2}$$

Combining Equation (5.17.1) and Equation (5.17.2) we conclude that

$$(1-\epsilon)\cdot\beta \leq K\eta.$$

□

We conclude with our main theorem by proving that we need only iterate our algorithm $\tilde{O}\left(\kappa \cdot \frac{h}{\epsilon^4}\right)$ times. Given a digraph $D = (V, A)$ with capacities U, lengths l, length constraint $h \geq 1$, $0 < \epsilon < 1$ and source and sink vertices $S, T \subseteq V$, and κ-batchable source, sink pairs $\{S_i, T_i\}_i$, one can compute a feasible multi-commodity h-length flow, moving cut pair (f, w) that is $(1 \pm \epsilon)$-

approximate in:

1. Deterministic parallel time $\tilde{O}(\kappa \cdot \frac{1}{\epsilon^9} \cdot h^{17})$ with m processors
2. Randomized CONGEST time $\tilde{O}(\kappa \cdot \frac{1}{\epsilon^9} \cdot h^{17})$ with high probability;
3. Deterministic CONGEST time $\tilde{O}\left(\kappa \cdot \frac{1}{\epsilon^9} \cdot h^{17} + \kappa \cdot \frac{1}{\epsilon^7} \cdot h^{16} \cdot (\rho_{CC})^{10}\right)$.

Furthermore, $f = \eta \cdot \sum_{j=1}^{k} f_j$ where $\eta = \tilde{\Theta}(\epsilon^2)$, $k = \tilde{O}\left(\kappa \cdot \frac{h}{\epsilon^4}\right)$ and f_j is an integral h-length S_i-T_i flow for some i.

Proof. By Theorem 126 and Theorem 127 we know that our solution is feasible and $(1 \pm \epsilon)$-optimal so it only remains to argue the runtime of our algorithm and that the returned flow decomposes in the stated way.

We argue that we must only run for $O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon^4}\right)$ total iterations. Since λ increases by a multiplicative $(1 + \epsilon_0)$ after every $\Theta\left(\kappa \cdot \frac{h \log n}{\epsilon_0^2}\right)$ iterations and starts at least $\left(\frac{1}{m}\right)^{\Theta(\frac{1}{\epsilon_0})}$, it follows by Theorem 125 that after $y \cdot \Theta\left(\kappa \cdot \frac{h \log n}{\epsilon_0^2}\right)$ total iterations the h-length distance between every S_i and T_i is at least $(1 + \epsilon_0)^y \cdot \left(\frac{1}{m}\right)^{\Theta(1/\epsilon_0)}$. Thus, for $y \geq \Omega\left(\frac{\ln_{1+\epsilon_0} m}{\epsilon_0}\right) = \Omega\left(\frac{\ln n}{\epsilon_0^2}\right)$ we have that every S_i and T_i are at least 1 apart in h-length distance. Consequently, our algorithm must run for at most $O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon_0^4}\right) = O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon^4}\right)$ many iterations.

Our running time is immediate from the the bound of $O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon^4}\right)$ on the number of iterations of the while loop, the fact that commodities in the same batch can be updated in parallel and the running times given in Section 5.10 for computing our h-length $(1 + \epsilon_0)$-lightest path blocker.

Lastly, the flow decomposes in the stated way because we have at most $O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon^4}\right)$ iterations and each f_j is an integral S-T flow by Section 5.10. Thus, our final solution is $\eta \cdot \sum_{j=1}^{k} f_j$ and $k = \tilde{O}\left(\frac{h}{\epsilon^4}\right)$. □

5.17.3 Computing Multi-Commodity Length-Constrained Cutmatches

We proceed to use our multi-commodity length-constrained flow algorithms to compute multi-commodity length-constrained cutmatches.

Suppose we are given a digraph $D = (V, A)$ with capacities U and lengths l. There is an algorithm that, given κ-batchable sink source pairs $\{(S_i, T_i)\}_i$ where $S_i, T_i \subseteq V$ with $U^+(S_i) \leq U^-(T_i)$ for every i and two integer parameters $h \geq 1$ and $\phi \leq 1$, outputs a multi-commodity h-length ϕ-sparse cutmatch of congestion γ between S and T, where $\gamma = \tilde{O}(\frac{1}{\phi})$. This algorithm runs in:

1. Deterministic parallel time $\tilde{O}(\kappa^2 \cdot \gamma \cdot h^{18})$ with m processors
2. Randomized CONGEST time $\tilde{O}(\kappa^2 \cdot \gamma \cdot h^{18})$ with high probability;
3. Deterministic CONGEST time $\tilde{O}\left(\kappa^2 \cdot \gamma \cdot h^{18} + \kappa \cdot \gamma \cdot h^{17} \cdot (\rho_{CC})^{10}\right)$.

Proof. We initialize the flow we return f to be 0 on all arcs. We set our working capacities to be $\hat{U} = U$ initially. The algorithm runs for at most $O(h \cdot \kappa \cdot \gamma)$ iterations for a small hidden constant. In each iteration $l \in [1, O(h \cdot \kappa \cdot \gamma)]$ we use Section 5.17.1 with $\epsilon = .5$ (any constant would suffice) to find a multi-commodity length-constrained flow, moving cut pair, $(\hat{f}, \hat{w})$ where $\delta = \tilde{\Theta}(1)$,

$k = \tilde{O}(\kappa \cdot h)$, $f = \delta \cdot \sum_{j=1}^{k} f_j$ and f_j is an integral multi-commodity h-length flow on $\{(S_i, T_i)\}_i$ using capacities $\hat{U}$. By averaging there must be some f_j such that s.t. $(f_j) \geq$ s.t. $(\hat{f})/k$. We let $\tilde{f}$ be this f_j.

If s.t. $(\tilde{f}) > \Omega\left(\frac{\log n \cdot \sum_i \hat{U}^+(S_i)}{\kappa h \gamma}\right)$ then we update our solution as $f = f + \tilde{f}$ and decrement $\hat{U}_a$ by $\sum_{i:a \in \delta^{\pm}(S_i,T_i)} \tilde{f}^{\{i\}}(a)$ for each a. Otherwise, we return the pair $(f, \hat{w})$ as our solution.

In each iteration l in which s.t. $(\tilde{f}) > \Omega\left(\frac{\log n \cdot \sum_i \hat{U}^+(S_i)}{\kappa h \gamma}\right)$, we have that $\sum_i \hat{U}^+(S_i)$ decreases multiplicatively by at least a $1 - \frac{2 \log n}{\kappa h \gamma}$ factor. Such a shrinking can happen at most $O(\kappa \cdot h \cdot \gamma)$ times until $\sum_i \hat{U}^+(S_i)$ is reduced to 0. Thus, our algorithm requires at most $O(\kappa \cdot h \cdot \gamma)$ iterations until terminating. Furthermore, notice when we return a moving cut $\hat{w}$ we have

$$\begin{aligned}
\sum_a \hat{w}_a &\leq 2 \cdot \text{ s.t. } (\hat{f}) \\
&\leq \kappa h \cdot \text{ s.t. } (\tilde{f}) \\
&\leq \tilde{O}\left(\frac{\sum_i \hat{U}^+(S_i)}{\gamma}\right) \\
&= \tilde{O}\left(\frac{\sum_i U^+(S_i) - \text{ s.t. } (f)}{\gamma}\right)
\end{aligned}$$

as desired. Also, observe that f is indeed an integral $\{(S_i, T_i)\}_i$ flow in D with the stated capacities since we always have $\tilde{f}(a) \leq \hat{U}_a$.

The running time is exactly that of running at most $O(\kappa h \cdot \gamma)$ invocations of Section 5.17.1 with $\epsilon = .5$. □

Part III

Steiner Point Removal

Chapter 6

Series-Parallel Steiner Point Removal

6.1 Introduction

Compact representations of graphs are particularly interesting when we assume that G is a member of a minor-closed graph family such as tree, cactus, series-parallel or planar graphs.[1] As many algorithmic problems are significantly easier on such families—see e.g. [12, 108, 166]—it is desirable that G' is not only a simple approximation of G's metric but that it also belongs to the same family as G.

Steiner point removal (SPR) formalizes the problem of producing a simple G' in the same graph family as G that preserves G's metric. In SPR we are given a weighted graph $G = (V, E, w)$ and a terminal set $V' \subseteq V$ where $V \setminus V'$ are called "Steiner points." We must return a weighted graph $G' = (V', E', w')$ where:

1. G' is a minor of G;
2. $d_G(u, v) \leq d_{G'}(u, v) \leq \alpha \cdot d_G(u, v)$ for every $u, v \in V'$;

and our aim is to minimize the multiplicative distortion α. We refer to a G' with distortion α as an α-SPR solution. In the above d_G and $d_{G'}$ give the distances in G and G' respectively.

If we only required that G' satisfies the second condition then we could always achieve $\alpha = 1$ by letting G' be the complete graph on V' where $w'(\{u, v\}) = d_G(u, v)$ for every $u, v \in V'$. However, such a G' forfeits any nice structure that G may have exhibited. Thus, the first condition ensures that if G belongs to a minor-closed family then so does G'. The second condition ensures that G''s metric is a good proxy for G's metric. G' is simpler than G since it is a graph only on V' while G' is a proxy for G's metric by approximately preserving distances on V'.

As Gupta [101] observed, even for the simple case of trees we must have $\alpha > 1$. For example, consider the star graph with unit weight edges where V' consists of the leaves of the star. Any tree

[1] A graph G' is a minor of a graph G if G' can be attained (up to isomorphism) from G by edge contractions as well as vertex and edge deletions. A graph is F-minor-free if it does not have F as a minor. A family of graphs $\mathcal{G}$ is said to be minor-closed if for any $G \in \mathcal{G}$ if G' is a minor of G then $G' \in \mathcal{G}$. A seminal work of Robertson and Seymour [157] demonstrated that every minor-closed family of graphs is fully characterized by a finite collection of "forbidden" minors. In particular, if $\mathcal{G}$ is a minor-closed family then there exists a finite collection of graphs $\mathcal{M}$ where $G \in \mathcal{G}$ iff G does not have any graph in $\mathcal{M}$ as a minor. Here and throughout this chapter we will use "minor-closed" to refer to all non-trivial minor-closed families of graphs; in particular, we exclude the family of all graphs which is minor-closed but trivially so.

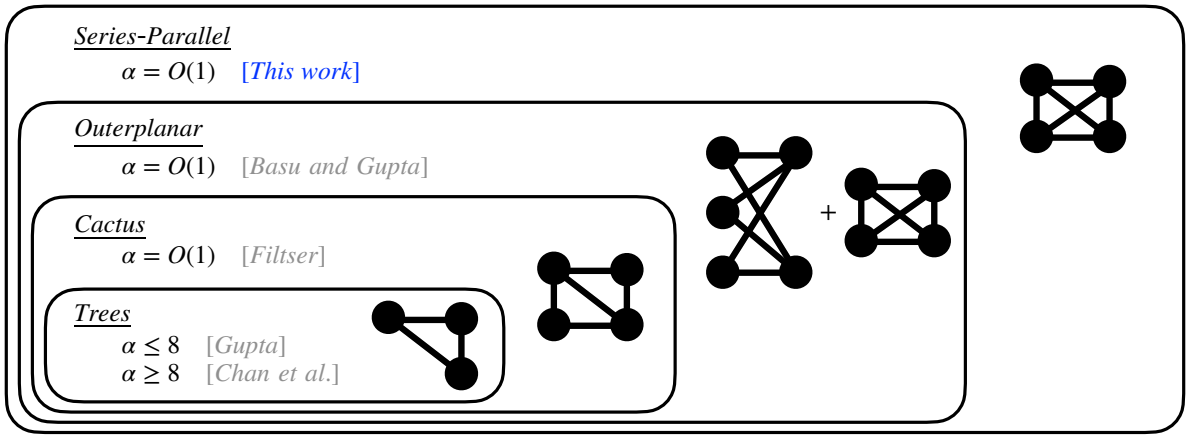


Figure 6.1: A summary of the SPR distortion for (connected) K_h-minor-free graphs achieved in prior work and our own. Graph classes illustrated according to containment. We also give the forbidden minors for each graph family.

$G' = (V', E', w')$ has at least two vertices u and v whose connecting path consists of at least two edges. On the other hand, the length of any edge in G' is at least 2 and so $d_{G'}(u, v) \geq 4$. Since $d_G(u, v) = 2$ it follows that $\alpha \geq 2$. While this simple example rules out the possibility of 1-SPR solutions on trees, it leaves open the possibility of small distortion solutions for minor-closed families.

In this vein several works have posed the existence of $O(1)$-SPR solutions for minor-closed families as an open question: see, for example, [28, 43, 55, 83, 136] among other works. A line of work (summarized in Figure 6.1) has been steadily making progress on this question for the past two decades. Gupta [101] showed that trees (i.e. connected K_3-minor-free graphs) admit 8-SPR solutions.Filtser et al. [86] recently gave a simpler proof of this result. Chan et al. [43] proved this was tight by showing that $\alpha \geq 8$ for trees which remains the best known lower bound for K_h-minor-free graphs. In an exciting recent work, Filtser [83] reduced $O(1)$-SPR in K_h-minor-free graphs to computing "$O(1)$ scattering partitions" and showed how to compute these partitions for several graph classes, including cactus graphs (i.e. all connected F-minor-free graphs where F is K_4 missing one edge). Lastly, a work of Basu and Gupta [28] generalizes these results by showing that outerplanar graphs (i.e. graphs which are both K_4 and $K_{2,3}$-minor-free) have $\alpha = O(1)$ solutions.

6.1.1 Our Contributions

In this work, we advance the state-of-the-art for Steiner point removal in minor-closed graph families. We show that series-parallel graphs (i.e. graphs which are K_4-minor-free) have $O(1)$-SPR solutions. The following theorem summarizes the main result of this chapter.

Theorem 128. *Every series-parallel graph $G = (V, E, w)$ with terminal set $V' \subseteq V$ has a weighted minor $G' = (V', E', w')$ such that for any $u, v \in V'$ we have*

$$d_G(u, v) \leq d_{G'}(u, v) \leq O(1) \cdot d_G(u, v).$$

Moreover, G' is poly-time computable by a deterministic algorithm.

Series-parallel graphs are a strict superset of all of the aforementioned graph classes for which $O(1)$-SPR solutions were previously known; again, see Figure 6.1. Series-parallel graphs are one of the most well-studied graph classes in metric embeddings and serve as a frequent test bed for making progress on long-standing open questions. For example, series-parallel graphs are one of the few graph classes for which the well-known GNRS conjecture in metric embeddings has been successfully proven [104]. For further examples see, among many other works, those of Brinkman et al. [38], Gupta et al. [103] and Emek and Peleg [74].

Relation to Prior Results. From a metric-embeddings perspective, series-parallel graphs are significantly more complex than outerplanar graphs (the largest minor-free graph class for which $O(1)$-distortion Steiner point removal was known prior to our work). For example, Gupta et al. [102] showed that outerplanar graphs can be embedded into "dominating tree metrics" with constant distortion but that such an embedding for series-parallel graphs incurs $\Omega(\log n)$ distortion. Likewise, outerplanar graphs embed isometrically into l_1 which is known to not be possible for series-parallel graphs; see Okamura and Seymour [150] and Chekuri et al. [50] for details. Thus, the metrics induced by series-parallel graphs often behave very differently and in less well-structured ways than those induced by outerplanar graphs.

Furthermore, the techniques on which we rely are quite different than those of Basu and Gupta [28] for the outerplanar case. At least two aspects of these techniques may be of independent interest. We defer a more thorough overview of our techniques to Section 6.4 but briefly highlight these two points now.

A New Approach for Steiner Point Removal. First, much of our approach generalizes to any K_h-minor-free graph so our approach seems like a promising avenue for future work on $O(1)$-SPR in minor-closed families. Specifically, we prove our result by beginning with the "chops" used by Klein et al. [128] to build low diameter decompositions for K_h-minor-free graphs. For input $\Delta > 0$ and root vertex r, these chops consist of deleting any edge which for some $i \in \mathbb{Z}$ has endpoints at distance $i\Delta$ and $i\Delta + 1$ from r; removing such edges partitions the input graph into width Δ "annuli." We begin with these chops but then slightly perturb them to respect the shortest path structure of the graph, resulting in what we call $O(1)$-scattering chops. We argue that the result of repeating such scattering chops is a scattering partition which by the results of Filtser [83] can be used to construct an $O(1)$-SPR solution.

The key to this strategy is arguing that series-parallel graphs admit a certain structure—which we call a hammock decomposition—that enables one to perform these perturbations in a principled way. If one could demonstrate a similar structure for K_h-minor-free graphs or otherwise demonstrate the existence of $O(1)$-scattering chops for such graphs, then the techniques laid out in our work would immediately give $O(1)$-SPR solutions for all K_h-minor-free graphs.

New Geometric Structure for Series-Parallel Graphs. Second, our hammock decompositions are a new metric decomposition for series-parallel graphs which may be interesting in their own right. We give significantly more detail in Section 6.7 but briefly summarize our decomposition for now. We show that for any fixed BFS tree T_{BFS} there is a forest-like subgraph which contains all shortest paths between cross edges of T_{BFS}.[2] Specifically, the "nodes" of this forest are not

[2]Here and throughout this chapter a cross edge is an edge that is in the input graph but not in T_{BFS}.

vertices but highly structured subgraphs of the input series-parallel graph which we call hammock graphs. A hammock graph consists of two subtrees of the BFS tree and the cross edges between them. See Figure 6.7 for a visual preview of our decomposition.

Our hammock decompositions stand in contrast to the fact that the usual way in which one embeds a graph into a tree—by way of dominating tree metrics—are known to incur distortion $\Omega(\log n)$ in series-parallel graphs [104]. Furthermore, our decomposition can be seen as a metric-strengthening of the classic nested ear decompositions for series-parallel graphs of Khuller [127] and Eppstein [78]. In general, a nested ear decomposition need not reflect the input metric. However, not only can one almost immediately recover a nested ear decomposition from a hammock decomposition, but the output nested ear decomposition interacts with the graph's metric in a highly structured way (see Section 6.11).

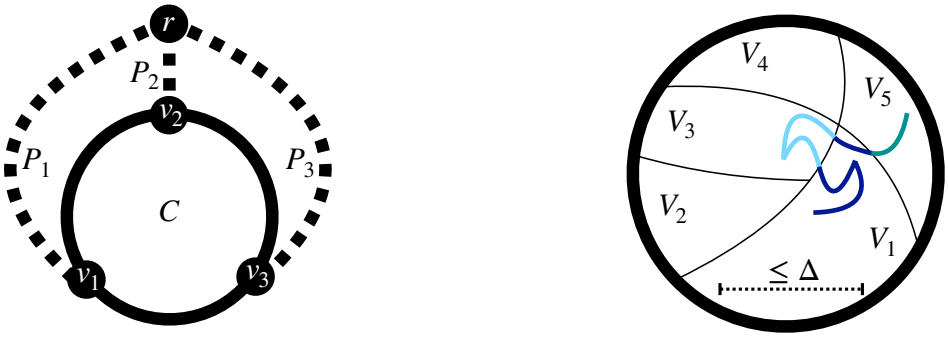
Open Questions Resolved. Lastly, we note that, in addition to making progress on the existence of $O(1)$-SPR solutions for every minor-closed family, our work also settles several open questions. The existence of $O(1)$-SPR solutions for series-parallel graphs was stated as an open question by both Basu and Gupta [28] and Chan et al. [43]; our result answers this question in the affirmative. Furthermore, Filtser et al. [86] posed the existence of $O(1)$ scattering partitions for outerplanar and series-parallel graphs as an open question; we prove our main result by showing that series-parallel graphs admit $O(1)$ scattering partitions, settling both of these questions.

6.2 Related Work

We briefly review additional related work.

Since the introduction of SPR by Gupta [101], a variety of works have studied the bounds achievable for well-behaved families of graphs for several very similar problems. Krauthgamer et al. [136] studied a problem like SPR but where distances in G must be exactly preserved by G' and the number of Steiner vertices—that is, vertices not in V'—must be made as small as possible; this work showed that while $O(k^4)$ Steiner vertices suffice (where $k = |V'|$) for general graphs, better bounds are possible for well-behaved families of graphs. More generally, Cheung et al. [55] studied how to trade off between the number of terminals and distortion of G', notably showing $(1+\epsilon)$ distortion is possible in planar graphs with $\tilde{O}(k^2/\epsilon^2)$ Steiner vertices. Englert et al. [77] showed that in minor-closed graphs distances can be preserved up to $O(1)$ multiplicative distortion in expectation by a distribution over minors as opposed to preserving distances deterministically with a single minor as in SPR.

A variety of recent works have also studied how to find minors which preserve properties of G other than G's metric. Englert et al. [77] studied a flow/cut version of SPR where the goal is for G' to be a minor of G just on the specified terminals while preserving the congestion of multicommodity flows between terminals: this work showed that a convex combination of planar graphs can preserve congestion on V' up to a constant while for general graphs a convex combination of trees preserves congestion up to an $O(\log k)$. Similarly, Krauthgamer and Rika [135] studied how to find minimum-size planar graphs which preserve terminal cuts. Goranci et al. [95] studied how to find a minor of a directed graph with as few Steiner vertices and which preserves the reachability relationships between all k terminals, showing that $O(k^3)$ vertices suffices for general graphs but $O(\log k \cdot k^2)$ vertices suffices for planar graphs.



(a) A clawed cycle.

(b) A scattering partition.

Figure 6.2: In (a) we illustrate a clawed cycle where the cycle C is given in solid black and each path is given in dotted black. In (b) we illustrate a scattering partition with $\tau = 3$ and how one path P of length at most Δ is incident to at most three parts where we color the subpaths of P according to the incident part.

There has been considerable effort in the past few years on developing good SPR solutions for general graphs. Kamma et al. [120] gave $O(\log^5 k)$-SPR solutions for general graphs. This was improved by Cheung [54] who gave $O(\log^2 k)$-SPR solutions which was, in turn, improved by Filtser [81] and Filtser [82] who gave $O(\log k)$-SPR solutions for general graphs. We also note that Filtser [83] also achieved similar results by way of scattering partitions, albeit with a worse poly-log factor as well as the first $O(1)$-SPR solutions for bounded pathwidth graphs.

6.3 Preliminaries

Before giving an overview of our approach we summarize the characterization of series-parallel graphs we use throughout this chapter as well as the scattering partition framework of Filtser [83] on which we build.

6.3.1 Characterizations of Series-Parallel Graphs

There are some minor inconsistencies in the literature regarding what is considered a series-parallel graph and so we clarify which notion of series-parallel we use throughout this chapter. Some works—e.g. Eppstein [78]—take series parallel graphs to be those which can be computed by iterating parallel and series compositions of graphs. Call these series-parallel A graphs.[3] Strictly speaking, series-parallel A graphs are not even minor-closed as they are not closed under edge or vertex deletion. Other works—e.g. Filtser [83]—take series-parallel graphs to be graphs whose biconnected[4] components are each series-parallel A graphs; call these series-parallel B graphs. Series-parallel B graphs clearly contain series-parallel A graphs and, moreover, are minor-closed.

[3]The following is a definition of series-parallel A graphs due to Eppstein [78]. A graph is two-terminal if it has a distinct source s and sink t. Let G and H be two two-terminal graphs with sources s and s' and sinks t and t'. Then the series composition of G and H is the graph resulting from identifying t and s' as the same vertex. The parallel composition of G and H is the graph resulting from identifying s and s' as the same vertex and t and t' as the same vertex. A two-terminal series-parallel graph is a two-terminal graph which is either a single edge or the graph resulting from the series or parallel composition of two two-terminal series-parallel graphs. A graph is series-parallel A if it has some pair of vertices with respect to which it is two-terminal series-parallel.

[4]A connected component C is biconnected if C remains connected even after the deletion of any one vertex in C.

For the rest of this chapter we will use the more expansive series-parallel B notion; henceforth we use "series-parallel" to mean series-parallel B.

It is well-known that a graph is K_4-minor-free iff it is series-parallel [34]. Similarly a graph has treewidth at most 2 iff it is series-parallel [34]. In this chapter we will use an alternate definition in terms of "clawed cycles" which we illustrate in Figure 6.2a.[5]

Definition 129 (Clawed Cycle)**.** *A clawed cycle is a graph consisting of a root r, a cycle C and three paths P_1 P_2 and P_3 from r to vertices $v_1, v_2, v_3 \in C$ where $v_1 \neq v_2 \neq v_3$*

The fact that series-parallel graphs are exactly those that do not have any clawed cycles as a subgraph was proven by Duffin [69]; we give a proof for completeness.

Lemma 130 (Duffin [69])**.** *A graph G is series-parallel iff it does not contain a clawed cycle as a subgraph.*

Proof. K_4 is itself a clawed cycle and so a graph with no clawed cycle subgraphs is K_4-minor-free and therefore series-parallel. If a graph contains a clawed cycle then we can construct a K_4 minor by arbitrarily contracting the graph into v_1, v_2, v_3 and r, as defined in Theorem 129. □

6.3.2 Scattering Partitions

Our result will be based on a new graph partition introduced by Filtser [83], the scattering partition. Roughly speaking, a scattering partition of a graph is a low-diameter partition which respects the shortest path structure of the graph; see Figure 6.2b.[6]

Definition 131 (Scattering Partition)**.** *Given a weighted graph $G = (V, E, w)$, a partition $\mathcal{P} = \{V_i\}_i$ of V is a (τ, Δ) scattering partition if:*

1. ***Connected:** Each $V_i \in \mathcal{P}$ is connected;*
2. ***Low Weak Diameter:** For each $V_i \in \mathcal{P}$ and $u, v \in V_i$ we have $d_G(u, v) \leq \Delta$;*
3. ***Scattering:** Every shortest path P in G of length at most Δ satisfies $|\{V_i : V_i \cap P \neq \emptyset\}| \leq \tau$.*

Filtser [83] extended these partitions to the notion of a scatterable graph.

Definition 132 (Scatterable Graph)**.** *A weighted graph $G = (V, E, w)$ is τ-scatterable if it has a (τ, Δ)-scattering partition for every $\Delta \geq 0$.*

We will say that G is deterministic poly-time τ-scatterable if for every $\Delta \geq 0$ a (τ, Δ)-scattering partition is computable in deterministic poly-time.

As a concrete example of a τ-scatterable graph and as observed by Filtser [83] notice that all trees are $O(1)$-scatterable. In particular, suppose we are given a tree and a $\Delta > 0$. If we fix a root vertex r and then delete any edge which for some $i \in \mathbb{Z}$ has endpoints at distance $\frac{i \cdot \Delta}{2}$ and $\frac{i \cdot \Delta}{2} + 1$ from r this breaks the input tree into connected components. Each component has diameter at most Δ by construction. Furthermore, it is easy to see that any path of length at most Δ is incident to a constant number of these components and so these components indeed form a scattering partition with $\tau = O(1)$. This construction is essentially a single chop of the aforementioned KPR strategy. However, while a KPR chop can be used to construct scattering partitions on trees,

[5] We note that clawed cycles are also called "embedded Wheatstone bridge."

[6] We drop one of the parameters of the definition of Filtser [83] as it will not be necessary for our purposes.

as we will see shortly, KPR chops on series-parallel graphs do not, in general, result in scattering partitions.

Lastly, the main result of Filtser [83] is that solving SPR reduces to showing that every induced subgraph is scatterable. In the following $G[A]$ is the subgraph of G induced by the vertex set A.

Theorem 133 (Filtser [83])**.** *A weighted graph $G = (V, E, w)$ with terminal set $V' \subseteq V$ has an $O(\tau^3)$-SPR solution if $G[A]$ is τ-scatterable for every $A \subseteq V$. Furthermore, if $G[A]$ is deterministic poly-time scatterable for every $A \subseteq V$ then the $O(\tau^3)$-SPR solution is computable in deterministic poly-time.*

6.4 Intuition and Overview of Techniques

We now give intuition and a high-level overview of our techniques. As discussed in the previous section, solving SPR with $O(1)$ distortion for any fixed graph reduces to showing that the subgraph induced by every subset of vertices is $O(1)$-scatterable. Moreover, since every subgraph of a K_h-minor-free graph is itself a K_h-minor-free graph, it follows that in order to solve SPR on any fixed K_h-minor-free graph, it suffices to argue that every K_h-minor-free graph is $O(1)$-scatterable.

Thus, the fact that we dedicate the rest of this document to showing is as follows.

Theorem 134. *Every series-parallel graph G is deterministic, poly-time $O(1)$-scatterable.*

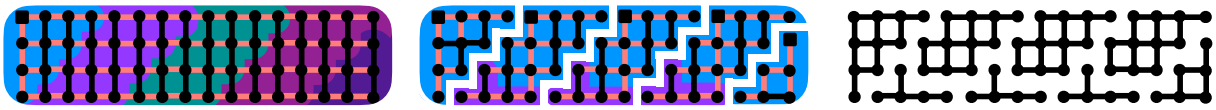
Combining this with Theorem 133 immediately implies Theorem 128.

6.4.1 General Approach

Given a series-parallel graph G and some $\Delta \geq 1$, our goal is to compute an $(O(1), \Delta)$-scattering partition for G. Such a partition has two non-trivial properties to satisfy: (1) each constituent part must have weak diameter at most Δ and (2) each shortest path of length at most Δ must be in at most $O(1)$ parts (a property we will call "scattering").

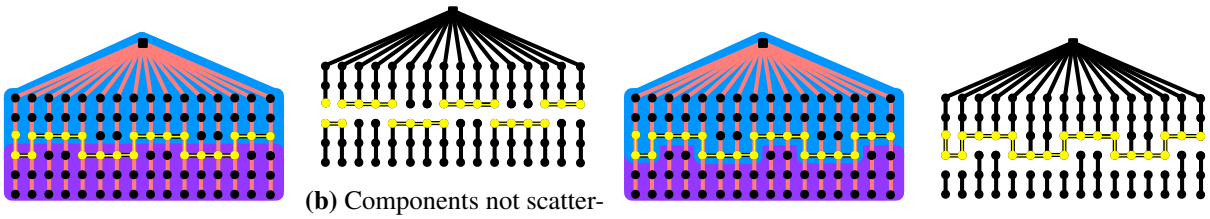
A well-known technique of Klein et al. [128]—henceforth "KPR"—has proven useful in finding so-called low diameter decompositions for K_h-minor-free graphs and so one might reasonably expect these techniques to prove useful for finding scattering partitions. Specifically, KPR shows that computing low diameter decompositions in a K_h-minor-free graph can be accomplished by $O(h)$ levels of recursive "Δ-chops". Fix a root r and a BFS tree T_{BFS} rooted at r. Then, a Δ-chop consists of the deletion of every edge with one vertex at depth $i \cdot \Delta$ and another vertex at depth $i \cdot \Delta + 1$ for every $i \in \mathbb{Z}_{\geq 1}$; that is, it consists of cutting edges between each pair of adjacent Δ-width annuli. KPR proved that if one performs a Δ-chop and then recurses on each of the resulting connected component then after $O(h)$ levels of recursive depth in a K_h-minor free graph the resulting components all have diameter at most $O(\Delta)$. We illustrate KPR on the grid graph in Figure 6.3.

Thus, we could simply apply Δ-chops $O(h)$ times to satisfy our diameter constraints (up to constants) and hope that the resulting partition is also scattering. Unfortunately, it is quite easy to see that (even after just one Δ-chop!) a path of length at most Δ can end up in arbitrarily many parts of the resulting partition. For example, the highlighted shortest path in Figures 6.4a and 6.4b repeatedly moves back and forth between two annuli and ends up in arbitrarily many parts after



(a) First Δ-chop. **(b)** Second Δ-chop. **(c)** Resulting connected components.

Figure 6.3: Two levels of Δ-chops on the grid graph for $\Delta = 3$. We give the edges of the BFS trees we use in pink; roots of these trees are given as squares. Background colors give the annuli of nodes.



(a) A Δ-chop **(b)** Components not scattering. **(c)** A perturbed Δ-chop. **(d)** Components scattering.

Figure 6.4: An example (of an outerplanar graph) where a Δ-chop does not produce a scattering partition but how perturbing said chop does. Here, we imagine that the root is at the top of the graph and each edge incident to the root has length $\Delta - 3$. We highlight the path P that either ends up in many or one connected component depending on whether we perturb our Δ-chop in yellow.

a single Δ-chop. Nonetheless, this example is suggestive of the basic approach of our work. In particular, if we merely perturbed our first Δ-chop to cut "around" said path as in Figures 6.4c and 6.4d then we could ensure that this path ends up in a small number of partitions.

More generally, the approach we take in this chapter is to start with the KPR chops but then slightly perturb these chops so that they do not cut *any* shortest path of length at most Δ more than $O(1)$ times. That is, all but $O(1)$ edges of any such path will have both vertices in the same (perturbed) annulus. We then repeat this recursively on each of the resulting connected components to a constant recursion depth. Since each subpath of a shortest path of length at most Δ is itself a shortest path with length at most Δ, we know that each such shortest path is broken into a constantly-many-more shortest paths at each level of recursion. Moreover, since we recurse a constant number of times, each path ends up in a constant number of components.

Implementing this strategy requires meeting two challenges. First, it is not clear that the components resulting from KPR still have low diameter if we allow ourselves to perturb our chops. Second, it is not clear how to perturb a chop so that it works *simultaneously* for every shortest path. Solving the first challenge will be somewhat straightforward while solving the second will be significantly more involved. In particular, what makes the second challenge difficult is that we cannot, in general, perturb a chop on the basis of one violated shortest path as in the previous example; doing so might cause other paths to be cut too many times which will then require additional, possibly conflicting, perturbations and so on. Rather, we must somehow perturb our chops in a way that takes every shortest path into account all at once.

6.4.2 Scattering Chops

The easier issue to solve will be how to ensure that our components have low diameter even if we perturb our chops. Here, by closely tracking various constants through a known analysis of KPR we show that the components resulting from KPR with (boundedly) perturbed cuts are still low

diameter.

We summarize this fact and the above discussion with the idea of a scattering chop. A (τ, Δ)-scattering chop consists of cutting all edges at *about* every Δ levels in the BFS tree in such a way that no shortest path of length at most Δ is cut more than τ times. Our analysis shows that if all K_h-minor-free graphs admit $(O(1), \Delta)$-scattering chops for every Δ then they are also $O(1)$-scatterable and therefore also admit $O(1)$-SPR solutions; this holds even for $h > 4$.

6.4.3 Hammock Decompositions and How to Use Them

The more challenging issue we must overcome is how to perturb our chops so that every shortest path of length at most Δ is only cut $O(1)$ times. Moreover, we must do so in a way that does not perturb our boundaries by too much so as to meet the requirements of a scattering chop. We solve this issue with our new metric decomposition for series-parallel graphs, the hammock decomposition.

Consider a shortest path P of length at most Δ. Such a path can be partitioned into a (possibly empty) prefix consisting of only edges in T_{BFS}, a middle portion whose first and last edges are cross edges of T_{BFS} and a (possibly empty) suffix which also only has edges in T_{BFS}. Thus, if we want to compute a scattering chop, it suffices to guarantee that any shortest path of length at most Δ which is either fully contained in T_{BFS} or which is between two cross edges of T_{BFS} is only cut $O(1)$ times by our chop; call the former a BFS path and the latter a cross edge path.

Next, notice that all BFS paths are only cut $O(1)$ times by our initial KPR chops. Specifically, each BFS path can be partitioned into a subpath which goes "up" in the BFS tree and a subpath which goes "down" in the BFS tree. As our initial KPR chops are Δ apart and each such subpath is of length at most Δ, each such subpath is cut at most $O(1)$ times. Thus provided our perturbations do not interfere *too much* with the initial structure of our KPR chops we should expect that our BFS paths will only be cut $O(1)$ times.

Thus, our goal will be to perturb our KPR chops to not cut any cross edge path more than $O(1)$ times while mostly preserving the initial structure of our KPR chops. Our hammock decompositions will allow us to do exactly this. They will have two key components.

The first part is a "forest of hammocks." Suppose for a moment that our input graph had a forest subgraph F that contained all cross edge paths of our graph which were also shortest paths. Then, it is not too hard to see how to use F to perturb our chops to be scattering for all cross edge paths. Specifically, for each tree T in our forest F we fix an arbitrary root and then process edges in a BFS order. Edges which we process will be marked or unmarked where initially all edges are unmarked. To process an edge $e = \{u, v\}$ we do the following. If e is marked or u and v both belong to the same annulus then we do nothing. Otherwise, e is unmarked and u is in some annulus A but v is in some other annulus A' (before any perturbation). We then propagate A an additional $\Theta(\Delta)$ deeper into T; that is if we imagine that v is the child of u in F then we move all descendants of u in F within $\Theta(\Delta)$ of u into A. We then mark all edges in T whose endpoints are descendants of u and within $\Theta(\Delta)$ of u. A simple amortized analysis shows that after performing these perturbations every cross edge path is cut $O(1)$ times: if we think of following a cross edge path from one endpoint to the other, then each time this path is cut there must be at least $\Omega(\Delta)$ many edges we get to traverse until the next time it is cut again.

Unfortunately, it is relatively easy to see that such an F may not exist in a series-parallel graph.

The forest of hammocks component of our decompositions is a subgraph which will be "close enough" to such an F, thereby allowing us to perturb our chops similarly to the above strategy. As mentioned in the introduction, a hammock graph consists of two subtrees of a BFS tree and the cross edges between them. A forest of hammocks is a graph partitioned into hammocks where every cycle is fully contained in one of the constituent hammocks. While the above perturbation will guarantee that our cross edge paths are not cut too often, it is not clear that such a perturbation does not change the structure of our initial chops in a way that causes our BFS paths to be cut too many times.

The second part of our hammock decompositions is what we use to guarantee that our BFS paths are not cut too many times by preserving the structure of our initial KPR chops. Specifically, the forest structure of our hammocks will reflect the structure of T_{BFS}. In particular, we can naturally associate each hammock H_i with a single vertex, namely the LCA of any u and v where u is in one tree of H_i and v is in the other. Then, our forest of hammocks will satisfy the property that if hammock H_i is a "parent" of hammock H_j in our forest of hammocks then the LCA corresponding to H_i is an ancestor of the LCA corresponding to H_j in T_{BFS}; even stronger, the LCA of H_j will be contained in H_i. Roughly, the fact that our forest of hammocks mimics the structure of T_{BFS} in this way will allow us to argue that the above perturbation does not alter the initial structure of our KPR chops too much, thereby ensuring that BFS paths are not cut too many times.

The computation of our hammock decompositions constitutes the bulk of our technical work but is somewhat involved. The basic idea is as follows. We will partition all cross edges into equivalence classes where each cross edge in an equivalence class shares an LCA in T_{BFS} (though there may be multiple, distinct equivalence classes with the same LCA). Each such equivalence class will eventually correspond to one hammock in our forest of hammocks. To compute our forest of hammocks we first connect up all cross edges in the same equivalence class. Next we connect our equivalence classes to one another by cross edge paths which run between them. We then extend our hammocks along paths towards their LCAs to ensure the above-mentioned LCA properties. Finally, we add so far unassigned subtrees of T_{BFS} to our hammocks. We will argue that when this process fails it shows the existence of a K_4-minor and, in particular, a clawed cycle.

6.5 Chapter-Specific Notation and Conventions

Before proceeding to our formal results we specify the notation we use throughout this work as well as some of the assumptions we make on our input series-parallel graph without loss of generality (WLOG).

Assumption of Unique Shortest Paths and Unit Weights: We will assume throughout this work that in our input series-parallel graph for any vertices u and v the shortest path between u and v is unique and that $w(e) = 1$ for every e. It is easy to see that our algorithms extend to non-unique shortest paths and the non-unit weight edge cases by standard techniques. In particular, one can randomly perturb the initial weights of the input graph so as to guarantee the uniqueness of shortest paths. Similarly, one can expand each edge of weight $w(e)$ into a path of $w(e)$ edges while preserving series-parallelness and the metric on the nodes from the original graph which suffices for our purposes.

Tree Paths: For a tree T, we will let $T(u, v)$ stand for the unique path between u and v in T for

$u, v \in V(T)$. We will sometimes assume that a path from a vertex set to another vertex set is directed in the natural way.

BFS Tree Notation: For much of this chapter we will fix a series-parallel graph $G = (V, E)$ along with a fixed but arbitrary root $r \in V$ and a fixed but arbitrary BFS tree T_{BFS} with respect to r. When we do so we will let $E_c := E \setminus E(T_{\text{BFS}})$ be all cross edges of T_{BFS}. For $u, v \in V$, if $u \in T_{\text{BFS}}(r, v) \setminus \{v\}$ then we say that u is an ancestor of v. In this case, we also say that v is a descendant of u. If u is an ancestor of v or v is an ancestor of u then we say that u and v are related; otherwise, we say that u and v are unrelated. For two vertices $u, v \in V$ we will use the notation $u \prec v$ to indicate that v is an ancestor of u and we will use the notation $u \preceq v$ to indicate that v is an ancestor of or equal to u. It is easy to verify that $\preceq$ induces a partial order. We let $T_{\text{BFS}}(v) := T_{\text{BFS}}[\{v\} \cup \{u \in V : u \text{ is a descendant of } v\}]$ be the subtree of T_{BFS} rooted at v. Given a connected subgraph $T \subseteq T_{\text{BFS}}$, we will let $\text{high}(T)$ be the vertex in $V(T)$ which is an ancestor of all vertices in $V(T)$. Given a path $P \subseteq T_{\text{BFS}}$ we will say that P is monotone if $\text{high}(P)$ is an ancestor of all vertices in P and there is some vertex $\text{low}(P)$ which is a descendant of all vertices in P. We let $h(v)$ give the height of a vertex in T_{BFS} (where we imagine that the nodes furthest from r are at height 0). We let $\text{LCA}(e)$ be the least common ancestor of u and v in T_{BFS} for each $e = \{u, v\} \in E$.

6.6 Perturbing KPR and Scattering Chops

In this section we show that KPR still gives low diameter components even if its boundaries are perturbed and therefore somewhat "fuzzy." We then observe that this fact shows that "$O(1)$-scattering chops" imply the existence of $O(1)$-scattering partitions for K_h-minor-free graphs and therefore $O(1)$-SPR solutions.

6.6.1 Perturbing KPR

We will repeatedly take the connected components of annuli with "fuzzy" boundaries. We formalize this with the idea of a c-fuzzy Δ-chop; see Figure 6.5a for an illustration.

Definition 135 (c-Fuzzy Δ-Chop)**.** *Let $G = (V, E, w)$ be a weighted graph with root r and fix $0 \leq c < 1$ and $\Delta \geq 1$. Then a c-fuzzy Δ-chop is a partition of V into "fuzzy annuli" $\mathcal{A} = \{A_1, A_2, \ldots\}$ where for every i and $v \in A_i$ we have*

$$(i-1)\Delta - \frac{c\Delta}{2} \leq d(r, v) < i \cdot \Delta + \frac{c\Delta}{2}.$$

As each fuzzy annulus in a fuzzy chop may contain many connected components we must be careful when specifying how recursive application of these chops break a graph into connected components; hence the following definitions. Given fuzzy annulus A_i, we let $\mathcal{C}_i$ be the connected components of A_i.

Definition 136 (Components Resulting from a c-Fuzzy Δ-Chop)**.** *Let $G = (V, E, w)$ be a weighted graph and let $\mathcal{C}$ be a partition of V into connected components. Then we say that $\mathcal{C}$ results from one level of c-fuzzy Δ-chops if there is a c-fuzzy Δ-chop $\mathcal{A}$ with respect to some root $r \in V$ satisfying $\mathcal{C} = \bigcup_{i:A_i \in \mathcal{A}} \mathcal{C}_i$. Similarly, for $h \geq 2$ we say that $\mathcal{C}$ results from h-levels of*

c-fuzzy Δ-chops if there is some $\mathcal{C}'$ which results from one level of c-fuzzy Δ-chops and $\mathcal{C}$ is the union of the result of $h-1$ levels of c-fuzzy Δ-chops on each $C' \in \mathcal{C}'$.

We will now claim that taking $h-1$ levels of fuzzy chops in a K_h-minor-free graph will result in a connected, low weak diameter partition. In particular, we show the following lemma, the main result of this section.

Lemma 137. *Let Δ and h satisfy $2 \leq h$, $\Delta \geq 1$ and fix constant $0 \leq c < 1$. Suppose $\mathcal{C}$ is the result of $h-1$ levels of c-fuzzy Δ-chops in a K_h-minor-free weighted graph G. Then, the weak diameter of every $C \in \mathcal{C}$ is at most $O(h \cdot \Delta)$.*

For the rest of this section we identify the nodes of a minor of graph G with "supernodes." In particular, we will think of each of the vertices of the minor as corresponding to a disjoint, connected subset of vertices in G (a supernode) where the minor can be formed from G (up to isomorphism) by contracting the constituent nodes of each such supernodes.

Our proof will closely track a known analysis of KPR [137]. The sketch of this strategy is as follows. We will argue that if we fail to produce parts with low diameter then we have found K_h as a minor. Our proof will be by induction on the number of levels of fuzzy chops. Suppose $\mathcal{C}$ is produced by $h-1$ levels of fuzzy chops; in particular, suppose $\mathcal{C}$ is produced by taking some fuzzy chop to get $\mathcal{C}'$ and then taking $h-2$ levels of fuzzy chops on each $C' \in \mathcal{C}'$. Also assume that there is some $C \in \mathcal{C}$ which has *large* diameter. Then, C must result from taking $h-2$ levels of fuzzy chops on some $C' \in \mathcal{C}$ where C' lies in some fuzzy annulus A_i of G. By our inductive hypothesis it follows that C contains K_{h-1} as a minor. Our goal is to add one more supernode to this minor to get a K_h minor. We will do so by finding disjoint paths of length $O(\Delta)$ in the annulus above A_i from each of the K_{h-1} supernodes all of which converge on a single connected component. By adding these paths to their respective supernodes in the K_{h-1} minor and adding the connected component on which these paths converge to our collection of supernodes, we will end up with a K_h minor.

The main challenge in this strategy is to show how to find paths as above which are disjoint. We will do so by choosing these paths from a "representative" from each supernode where initially the representatives are $\Omega(h\Delta)$ far-apart and grow at most $O(\Delta)$ closer at each level of chops; since we do at most $O(h)$ levels of chops, the paths we choose will never intersect.

To formalize this strategy we must state a few definitions which will aid in arguing that these representatives are far apart.

Definition 138 (Δ-Dense)**.** *Given sets $S, U \subseteq V$ we say S is Δ-dense in U if $d(u, S) \leq \Delta$ for every $u \in U$.*

Definition 139 (R-Represented)**.** *A K_h minor is R-represented by set $S \subseteq V$ if each supernode $V_i \subseteq V$ of the minor in G contains a representative $v_i \in S \cap V_i$ and these representatives are pairwise at least R apart in G.*

Since V is clearly $(1+c)\Delta$-dense in V, we can set S to V and j to $h-1$ in the following lemma to get Theorem 137.

Lemma 140. *Fix $0 \leq c < 1$ and $h > j \geq 0$. Let S be any set which is $(1+c)\Delta$-dense in V and suppose $\mathcal{C}$ is the result of j levels of c-fuzzy Δ-chops and some $C \in \mathcal{C}$ has weak diameter more than $22h\Delta$. Then there exists a K_{j+1} minor which is $8(h-j)\Delta$-represented by S.*

Proof. Our proof is by induction on j. The base case of $j = 0$ is trivial as K_1 is a minor of any graph with a supernode $+\infty$-represented by any single vertex in V.

Now consider the inductive step on graph $G = (V, E)$. Fix some set S which is $(1 + c)\Delta$-dense in V and let $\mathcal{C}$ be the result of j levels of c-fuzzy chops using root r with some $C' \in \mathcal{C}$ of diameter more than $22h\Delta$. Suppose C' is in fuzzy annulus A_k and suppose that C' is the result of applying $j - 1$ levels of c-fuzzy chops to some C which resulted from 1 level of c-fuzzy chops in G; note that C is a connected component of A_k and that C' is contained in C.

As an inductive hypothesis we suppose that any $j - 1$ levels of c-fuzzy Δ-chops on any graph H which results in a cluster of weak diameter more than $22h\Delta$ demonstrates the existence of a K_j minor in H which is $8(h - j + 1)\Delta$-represented by any set S' which is $(1 + c)\Delta$-dense in $V(H)$. Here weak diameter is with respect to the distances induced by the original input graph.

Thus, by our inductive hypothesis we therefore know that C contains a K_j minor which is $8(h - j + 1)\Delta$-represented by any $S' \subseteq V(C)$ which is $(1 + c)\Delta$-dense in $V(C)$. In particular, we may let S' be the "upper boundary" of C; that is, we let S' be all vertices v in C such that the shortest path from v to r does not contain any vertices in C. Clearly the shortest path from any vertex in C to r intersects a node in S'; moreover, when restricted to C this shortest path has length at most $\Delta + c\Delta$ (since C is contained in A_k) which is to say that S' is $(1 + c)\Delta$-dense in C. Thus, by our induction hypothesis there is a K_j minor in C which is $8(h - j + 1)\Delta$-represented by S'. Let $V_1, \ldots, V_j$ be the nodes in the supernodes of the K_j minor.

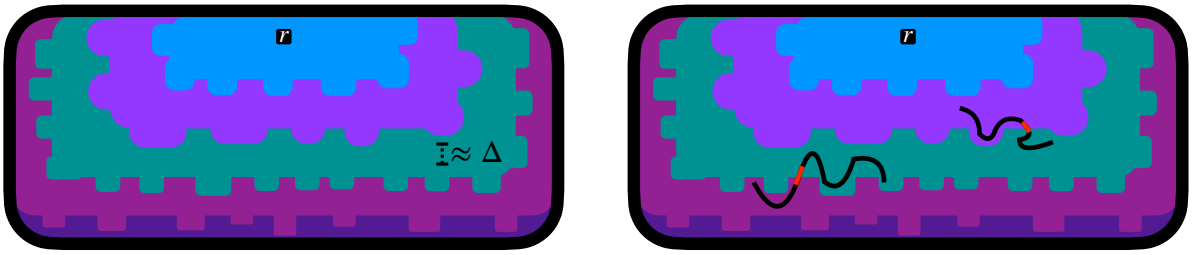
We now describe how to extend the K_j minor to a K_{j+1} minor which is $8(h - j)\Delta$-represented by S. We may assume that $k \geq 9h + 1$; otherwise the distance from every node in A_k to r would be at most $(9h + 1)\Delta + \frac{c\Delta}{2} \leq (9h + \frac{3}{2})\Delta$ and so the weak diameter of C' would be at most $(18h + 3)\Delta \leq 21h\Delta$, contradicting our assumption on C''s diameter. It follows that for every $v \in A_k$ we have

$$d(v, r) \geq (k - 1)\Delta - \frac{c\Delta}{2} \geq 9h\Delta - \frac{c\Delta}{2} \geq 8h\Delta. \tag{6.6.1}$$

We first describe how we grow each supernode V_i from the K_j minor to a new supernode V_i'. Let v_i be the representative in S' for V_i. Consider the path which consists of following the shortest path from v_i to r for distance 2Δ and then continuing on to the nearest node in S; let v_i' be this nearest node; this path from v_i to v_i' has length at most $(3 + c)\Delta$ since S is $(1 + c)\Delta$-dense in $V(G)$. Let V_i' be the union of V_i with the vertices in this path. Since each of these paths is of length at most $(3 + c)\Delta \leq 4\Delta$, it follows that each of these paths for each i must be disjoint since each v_i is at least $8(h - j + 1)\Delta > 8\Delta$ apart. Further, every v_i' must also, therefore, be at least $8(h - j)\Delta$ apart. Therefore, we let these v_i' form the representatives in S for each of the V_i'.

We now describe how we construct the additional supernode, V_0, which we add to our minor to get a K_{j+1} minor. V_0 will "grow" from the root to S and each of the V_i'. In particular, let $u_i \in V_i'$ be the node in V_i' which is closest to r and let P_i be the shortest path from r to u_i, excluding u_i. Similarly, let v_0' be the node in S closest to r and let P_0 be the shortest path from r to v_0', including v_0'. Then, we let V_0 be $P_0 \cup P_1 \cup \ldots \cup P_j$ and we let v_0' be the representative for V_0 in S. We claim that for every $i \geq 1$ we have

$$d(P_0, V_i') \geq 8(h - j)\Delta. \tag{6.6.2}$$



(a) A c-fuzzy Δ-chop. **(b)** Visualizing some paths.

Figure 6.5: A c-fuzzy Δ-chop that is 1-scattering. We draw each fuzzy annulus in a distinct color. In (b) we visualize some shortest paths of length at most Δ and highlight cut edges in red.

In particular, notice that since S is $(1+c)\Delta$-dense in $V(G)$ we know that $d(r, v_0') \leq (1+c)\Delta \leq 2\Delta$ and since $d(v, r) \geq 8h\Delta$ for every $v \in A_k$ by Equation (6.6.1) and $d(v_i', A_k) \leq (3+c)\Delta \leq 4\Delta$, it follows that $d(P_0, V_i') \geq (8h-6)\Delta \geq 8(h-j)\Delta$. Consequently, $d(v_0', v_i') \geq 8(h-j)\Delta$ for every $i \geq 1$. Thus, our representatives of each supernode are appropriately far apart.

It remains to show that our supernodes indeed form a K_{j+1} minor; clearly by construction they are all pair-wise adjacent and so it remains only to show that they are all disjoint from one another. We already argued above that for $i, j \geq 1$ any V_i' and V_j' are disjoint so we need only argue that V_0' is disjoint from each V_i' for $i \geq 1$. P_0 must be disjoint from each V_i' for $i \geq 1$ by Equation (6.6.2) and so we need only verify that P_i is disjoint from V_j' for $i, j \geq 1$;

By construction if $i = j$ we know that P_i is disjoint from V_j' so we assume $i \neq j$ and that P_i intersects V_j' for the sake of contradiction. Notice that each P_i has length at most $k\Delta + \frac{c\Delta}{2} - 2\Delta = k\Delta + c\Delta - 2\Delta - \frac{c\Delta}{2} < (k-1)\Delta - \frac{c\Delta}{2}$ by how we construct V_i'. Thus, P_i must be disjoint from A_k. It follows that if P_i intersects V_j' then it must intersect $V_j' \setminus V_j$. However, since $d(v_i, v_j) \geq 8(h-j+1)\Delta \geq 16\Delta$ and the length of paths $V_i' \setminus V_i$ and $V_j' \setminus V_j$ are at most 4Δ we know that $d(V_i' \setminus V_i, V_j' \setminus V_j) \geq 8(h-j)\Delta \geq 8\Delta$. Thus, after intersecting $V_j' \setminus V_j$ and then continuing on to a vertex adjacent to $V_i' \setminus V_i$, we know P_i must travel at least 8Δ; since the vertices of P_i are monotonically further and further from r, and the vertex in $V_j' \setminus V_j$ that P_i intersects must be distance at least $(k-1)\Delta - \frac{c\Delta}{2} - 4\Delta \geq (k-5)\Delta$ from r, then the last vertex of P_i must be distance at least $(k+3)\Delta$ from r, meaning P_i must intersect annulus A_k, a contradiction. □

6.6.2 Scattering Chops

Using Theorem 137 we can reduce computing a good scattering partition and therefore computing a good SPR solution to finding what we call a scattering chop. The following definitions are somewhat analogous to Theorem 131 and Theorem 132. However, notice that the second definition is for a family of graphs (as opposed to a single graph as in Theorem 132). We illustrate a τ-scattering chop in Figure 6.5.

Definition 141 (τ-Scattering Chop)**.** *Given a weighted graph $G = (V, E, w)$, let $\mathcal{A}$ be a c-fuzzy Δ-chop with respect to some root $r \in V$. $\mathcal{A}$ is a τ-scattering chop if each shortest path of length at most Δ has at most τ edges cut by $\mathcal{A}$ where we say that an edge is cut by $\mathcal{A}$ if it has endpoints in different fuzzy annuli of $\mathcal{A}$.*

Definition 142 (τ-Scatter-Choppable Graphs)**.** *A family of graphs $\mathcal{G}$ is τ-scatter-choppable if*

there exists a constant $0 \leq c < 1$ *such that for any* $G \in \mathcal{G}$ *and* $\Delta \geq 1$ *there is some* τ*-scattering and* c*-fuzzy* Δ*-chop* $\mathcal{A}$ *with respect to some root.*

We will say that $\mathcal{G}$ is deterministic poly-time τ-scatter-choppable if the above chop $\mathcal{A}$ for each $G \in \mathcal{G}$ can be computed in deterministic poly-time.

Lastly, we conclude that to give an $O(1)$-scattering partition—and therefore to give an $O(1)$-SPR solution—for a K_h-minor-free graph family it suffices to show that such a family is $O(1)$-scatter choppable.

Lemma 143. *Fix a constant* $h \geq 2$ *and let* $\mathcal{G}_h$ *be all* K_h*-minor-free graphs. Then, if* $\mathcal{G}_h$ *is* τ*-scatter-choppable then every* $G \in \mathcal{G}_h$ *is* $O(\tau^{h-1})$*-scatterable.*

Proof. The claim is almost immediate from Theorem 137 and the fact that all subpaths of a shortest path are themselves shortest paths.

In particular, first fix a sufficiently small constant c' to be chosen later. Then, consider a $G \in \mathcal{G}_h$ and fix a Δ. By assumption we know that G is τ-scatter-choppable and since each subgraph of G is in $\mathcal{G}_h$ so too is each subgraph of G. Thus, we may let $\mathcal{C}$ be the connected components resulting from $h-1$ levels of c-fuzzy and $(c'\Delta)$-chops which are τ-scattering.

We claim that for sufficiently small c' we have that $\mathcal{C}$ is a $\left(\frac{\tau^{h-1}}{c'}, \Delta\right)$-scattering partition. By Theorem 137 the diameter of each part in $\mathcal{C}$ is at most $O(c' \cdot h \cdot \Delta) \leq \Delta$ for sufficiently small c'. Next, consider a shortest path P of length at most Δ. We can partition the edges of P into at most $\frac{1}{c'}$ shortest paths $P_1, P_2, \ldots$, each of length at most $c' \cdot \Delta$. Thus, it suffices to show that each P_i satisfies $|\{C \in \mathcal{C} : P_i \cap C \neq \emptyset\}| \leq \tau^{h-1}$.

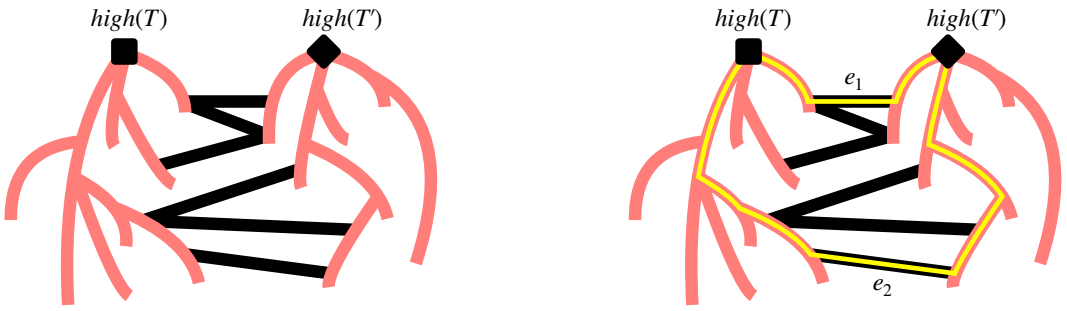
We argue by induction on the number of levels of chops that after $h' < h$ chops we have $|\{C \in \mathcal{C} : P_i \cap C \neq \emptyset\}| \leq \tau^{h'}$. Suppose we perform just one chop; i.e. $h' = 1$. Then, since our chops are τ-scattering we know that P will be cut at most τ times and so be incident to at most τ components of $\mathcal{C}$ as required. Next, suppose we perform $h' > 1$ levels of chops. Then our top-level chop will partition the vertices of P_i into at most τ components. By induction and the fact that each subpath of P_i is itself a shortest path of length at most $c'\Delta$, we know that the vertices of P_i in each such component are broken into at most $\tau^{h'-1}$ components and so P_i will be incident to at most $\tau^{h'}$ components as required. As we perform $h-1$ levels of chops, it follows that $\mathcal{C}$ is indeed a $\left(\frac{\tau^{h-1}}{c'}, \Delta\right)$-scattering partition. □

6.7 Hammock Decompositions

In this section we formally define our hammock decompositions and give some of their properties. For the rest of this section we will assume we are working with a fixed graph $G = (V, E)$ with a fixed but arbitrary root $r \in V$ and a fixed but arbitrary BFS tree T_{BFS} rooted at r. Throughout this section we will extensively use the notational conventions specified in Section 6.5, especially the BFS tree notation.

6.7.1 Trees of Hammocks

We begin by formalizing and establishing the properties of the key component of our hammock decompositions, the tree of hammocks. Roughly speaking, a tree of hammocks will be a tree in



(a) A hammock.

(b) A hammock-fundamental cycle.

Figure 6.6: An illustration of a hammock and a hammock-fundamental cycle for $e_1, e_2 \in E_c$. Edges of T_{BFS} in pink, cross edges in black, hammock roots are a black square and diamond and the hammock-fundamental cycle is in yellow.

the usual graph-theoretic sense whose nodes are structured subgraphs which we call hammocks. Much of this section will be concerned with showing that a tree of hammocks, despite having subgraphs as its nodes, satisfies nice properties analogous to those of a tree of nodes.

We begin by defining a hammock graph as two subtrees of T_{BFS} along with all cross edges between them. We illustrate a hammock in Figure 6.6a. Recall that $E_c := E \setminus E(T_{\text{BFS}})$ gives the "cross" edges of T_{BFS} and high gives the vertex closest to the root of T_{BFS}.

Definition 144 (Hammock Graph)**.** *We say that subgraph $H \subseteq G$ is a hammock if $H = G[V(H)]$ and $V(H)$ can be partitioned into sets V_1 and V_2 where its "hammock trees" $T := T_{BFS}[V_1]$ and $T' := T_{BFS}[V_2]$ are connected, $E_c(T, T') \neq \emptyset$ and the "hammock roots" high(T) and high(T') are unrelated.*

Notice that given a hammock H and two distinct edges $e_1 = \{v, v'\}, e_2 = \{u, u'\} \in E_c(T, T')$ where $v, u \in T$ and $v', u' \in T'$, we have that there is a unique cycle containing e_1 and e_2, namely $T(v, u) \oplus e_1 \oplus T'(u', v') \oplus e_2$. We will refer to this cycle as the hammock-fundamental cycle of e_1 and e_2. We illustrate a hammock-fundamental cycle in Figure 6.6b.

For the following definition of a tree of hammocks, recall that if $\mathcal{H}$ is a collection of subgraphs of G then $G[\mathcal{H}]$ gives the graph induced by the union of the subgraphs contained in $\mathcal{H}$.

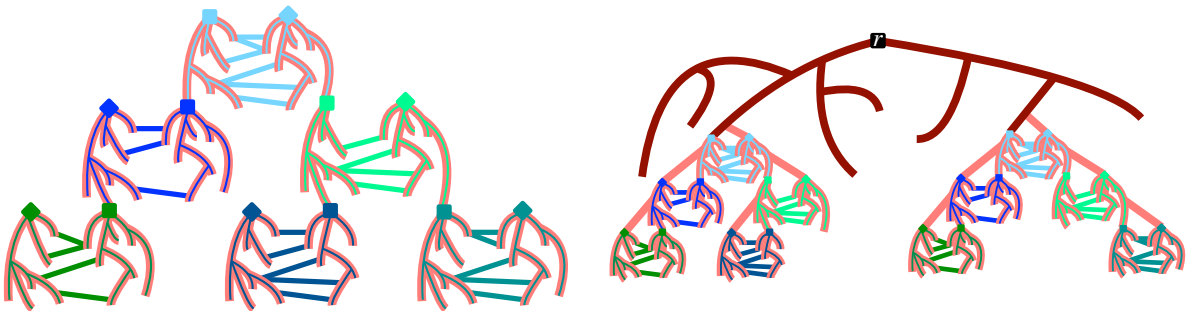
Definition 145 (Tree of Hammocks)**.** *We say that a collection of edge-disjoint hammocks $\mathcal{H} = \{H_i \subseteq G\}_i$ forms a tree of hammocks if every simple cycle C in $G[\mathcal{H}]$ satisfies $|H_i : E(C) \cap H_i \neq \emptyset| = 1$ and $G[\mathcal{H}]$ has a single connected component.*

We illustrate a tree of hammocks in Figure 6.7a. Just as it is often useful to root a tree of vertices, so too will it be useful for us to root our trees of hammocks.

Definition 146 (Rooted Tree of Hammocks)**.** *Suppose $\mathcal{H} = \{H_i\}_i$ forms a tree of hammocks. Then we say that $\mathcal{H}$ forms a rooted tree of hammocks if some $H_k \in \mathcal{H}$ is designated a "root hammock."*

While the above definitions are concerned with trees of hammocks, they extend naturally to a notion of forests of hammocks.

Definition 147 (Rooted Forests of Hammocks)**.** *We say that a collection of edge-disjoint hammocks*



(a) A tree of hammocks. **(b)** A hammock decomposition.

Figure 6.7: An illustration of one tree of hammocks in a hammock decomposition $\{H_i\}_i$ and a hammock decomposition consisting of two trees of hammocks. Each r_i and r'_i given as a square and diamond colored according to corresponding H_i. Edges in T_{BFS} in pink and highlighted according to the H_i which contains them. Edges of E_c colored according to the H_i which contains them. T_0 in the hammock decomposition given in dark red.

$\mathcal{H} = \{H_i\}_i$ forms a (rooted) forest of hammocks if for each connected component W of $G[\mathcal{H}]$ we have that $\{H_i : V(H_i) \cap V(W) \neq \emptyset\}$ forms a (rooted) tree of hammocks. We call the trees of hammocks formed by each such connected component the trees of hammocks of $\mathcal{H}$.

All of our notation and definitions will extend from trees of hammocks to forests of hammocks in the natural way.

Parents and Ancestors in Trees of Hammocks

While defining a root for a tree of nodes immediately determines the parents and ancestors of each node, it is not so clear that defining a root hammock in a tree of hammocks determines a reasonable notion of parents and ancestors of hammocks. In this section, we provide some simple observations that, in turn, will allow us to coherently define parent and ancestor relationships in a tree of hammock. Along the way we will provide an explicit tree representation of a tree of hammocks $\mathcal{H}$ as a tree $T_{\mathcal{H}}$.

We begin by showing how, given a fixed root hammock in a tree of hammocks, we can define what it means for one hammock to be the parent of another hammock. To do so we need the following simple observation. Let $\mathcal{H}$ be a forest of hammocks and let P be a path in the graph induced by $\mathcal{H}$ between hammocks H_i and H_j. We say that P *passes through* hammock $H_l \in \mathcal{H}$ if it contains at least one edge of H_l. Then, we have the following fact which is analogous to the fact that there is a unique simple path in a tree between any two vertices in a tree. We remind the reader that a path is from subgraph H_i to subgraph H_j (in what follows these subgraphs are hammocks) iff its first and last vertices are in H_i and H_j and these are the only vertices of the path in H_i and H_j.

Lemma 148. *Suppose $\mathcal{H}$ forms a tree of hammocks. Then for any $H_i, H_j \in \mathcal{H}$ if P and P' are both from H_i to H_j then both P and P' pass through the same hammocks of $\mathcal{H}$ in the same order.*

Proof. Suppose that P passes through $H_1, H_2, \ldots$ and P' passes through $H'_1, H'_2, \ldots$. Moreover, let $H_0, H'_0 := H_i$ and let $H_k, H'_{k'} := H_j$. We will consider the paths induced by P and P' from when they diverge to when they come back together and then argue that this gives a cycle with edges in multiple distinct hammocks. Specifically, let x be any index such that $H_x = H'_x$ but $H_{x+1} \neq H'_{x+1}$ and let y be the lowest index larger than x such that $H_y = H'_{y'}$ for some $y' > x$.

By our choice of H_0 and our assumption that both paths pass through different hammocks we know that x is well-defined while our choice of H_k and $H'_{k'}$ shows that y and y' are well-defined. Let $\bar{P}$ and $\bar{P}'$ be P and P' restricted to the corresponding subpath between H_x and H_y and let v_x and v_y and v'_x and v'_y be the endpoints of $\bar{P}$ and $\bar{P}'$ in H_x and H_y respectively. Then we have that $\bar{P}$ combined with $\bar{P}'$ along with the path between v_x and v'_x in H_x and the path between v_y and v'_y in H_y is a cycle; call this cycle C. Moreover, since $\bar{P}$ and $\bar{P}'$ each pass through at least 1 distinct hammock we have that such a cycle satisfies $|H_l : E(C) \cap H_l \neq \emptyset| \geq 2$, contradicting our assumption that $\mathcal{H}$ is a forest of hammocks as defined in Theorem 145. □

The above lemma now allows us to formally define what it means for a hammock to be the parent of another hammock in a tree of hammocks.

Definition 149 (Parent/Child Hammocks)**.** *Suppose $\mathcal{H}$ is a rooted tree of hammocks with root H_k. Then, we say that H_k is the parent of all H_j for $j \neq k$ which share a vertex with H_k. Similarly, for any other $H_j \neq H_k$ we let the parent of H_j be H_i where H_i is the unique hammock any path from H_j to H_k first passes through (as guaranteed to exist by Theorem 148). Symmetrically, H_j is a child of hammock H_i if H_i is a parent of H_j.*

More generally, we can use this notion of what it means for a hammock to be a parent of another hammock in order to make the "treeness" of a tree of hammocks more explicit and to define what it means for a hammock to be the ancestor of another hammock.

Definition 150 (Tree Representation $T_{\mathcal{H}}$)**.** *Suppose $\mathcal{H} = \{H_i\}_i$ forms a rooted tree of hammocks with root H_k. Then $T_{\mathcal{H}}$ is the graph with vertex set $\{H_i\}_i$ and root H_k which has an edge between H_j and H_i iff H_i is the parent of H_j in $\mathcal{H}$.*

Lemma 151. *If $\mathcal{H}$ is a rooted tree of hammocks then $T_{\mathcal{H}}$ is a tree where H_i is a parent of H_j in $T_{\mathcal{H}}$ iff H_i is a parent of H_j in $\mathcal{H}$.*

Proof. Let H_k be the designated root hammock in $\mathcal{H}$. Now assume for the sake of contradiction that we have a cycle $H_0, H_1, \ldots, H_{x-1}$ in $\mathcal{H}$. Since each vertex/hammock in $\mathcal{H}$ has at most one parent, it follows that $H_{i+1 \mod x}$ is the parent of H_i for every $i \in [x-1]$ (where we have assumed WLOG that $H_{i+1 \mod x}$ is the parent of H_i and not that H_i is the parent of $H_{i+1 \mod x}$). By definition of what it means for $H_{i+1 \mod x}$ to be the parent of H_i, it follows that every path from H_i to H_k must pass through $H_{i+1 \mod x}$, $H_{i+2 \mod x}$ and so on. However, since we have a cycle it then follows that every path from H_i to H_k must pass through H_i itself, contradicting the fact that there is a path in $G[\mathcal{H}]$ between H_i and H_k which does not pass through H_i since $G[\mathcal{H}]$ is connected.

Moreover, by how we define $T_{\mathcal{H}}$ it follows that H_i is a parent of H_j in $T_{\mathcal{H}}$ iff H_i is a parent of H_j in $\mathcal{H}$. □

As we have established that $T_{\mathcal{H}}$ is a tree, we henceforth assume it is rooted at H_k, the designated root hammock of $\mathcal{H}$. With this in mind, we can define ancestors and descendants in $\mathcal{H}$.

Definition 152 (Ancestor/Descendant Hammock)**.** *Let $\mathcal{H} = \{H_i\}_i$ be a rooted tree of hammocks. H_i is an ancestor (resp. descendant) of H_j in $\mathcal{H}$ iff H_i is an ancestor (resp. descendant) of H_j in $T_{\mathcal{H}}$.*

Similarly to our BFS tree notation, we will use the notation $H_j \preceq_{\mathcal{H}} H_i$ to indicate that H_j is a descendant of H_i in tree of hammocks $\mathcal{H}$.

The Structure of Paths in Trees of Hammocks

We now observe that if $\mathcal{H}$ is a tree of hammocks then any path in $G[\mathcal{H}]$ coincides with the corresponding path in $T_{\mathcal{H}}$.

We begin by observing the following simple technical lemma.

Lemma 153. *If $\mathcal{H} = \{H_i\}_i$ is a tree of hammocks then for every i, j we have $H_i \cap H_j \neq \emptyset$ implies that $H_i \cap H_j$ consists of a single vertex which is an articulation vertex for $G[\mathcal{H}]$.*

Proof. Suppose for the sake of contradiction that two hammocks H_i and H_j share a vertex v which is not an articulation point for the graph induced by $\mathcal{H}$ and let v_i and v_j be vertices adjacent to v in H_i and H_j respectively. Then, since v is not an articulation point there is a path P from v_i to v_j not containing v. It follows that $C := \{v, v_i\} \oplus P \oplus \{v_j, v\}$ is a cycle. However, we then have $\{H_i, H_j\} \subseteq \{H_l : E(C) \cap H_l \neq \emptyset\}$, violating Theorem 145. □

Concluding, we have the fact that the paths in $G[\mathcal{H}]$ adhere to the structure of $T_{\mathcal{H}}$. This fact is obvious if one inspects a picture of a tree of hammocks (see e.g. Figure 6.7a), but formalizing it requires a little bit of careful thought and notation. Again recall that if a path is between two subgraphs H_i and H_j then by definition the only vertices of H_i and H_j in this path are its first and last vertices; for this reason the indices of P_x begin at 1 and end at $l - 1$ in the following definition.

Lemma 154. *Let $\mathcal{H}$ be a rooted tree of hammocks, let P be a path in $G[\mathcal{H}]$ between hammocks H_i and H_j and let $T_{\mathcal{H}}(H_i, H_j) = (H_i = H_0, H_1, H_2, \ldots, H_l = H_j)$ be the path between H_i and H_j in the tree representation $T_{\mathcal{H}}$ of $\mathcal{H}$. Then P is of the form $P_1 \oplus P_2 \ldots \oplus P_{l-1}$ where for each $x \in [l-1]$ we have:*

1. *P_x is a subpath of P gotten by restricting P to $V(H_x)$ where $E(P_x) \subseteq E(H_x)$ and;*
2. *The first and last vertices of P_x are articulation vertices $b_{(x-1)x}$ and $b_{x(x+1)}$ where $V(H_{x-1}) \cap V(H_x) = \{b_{(x-1)x}\}$ and $V(H_x) \cap V(H_{x+1}) = \{b_{x(x+1)}\}$.*

Proof. We begin with the following simple observation. Suppose $e = \{u, v\} \in P$. Then if u is in some H_x but $v \notin H_x$ then the suffix of P after and including v cannot include any vertices of H_x. Suppose for the sake of contradiction that it did and let P' be the subpath of P from u back to some $u' \in H_x$. Then combining P' with the path in H_x connecting u and u' gives a cycle with edges in more than one hammock, a contradiction. The symmetric statement holds for prefixes of P.

As our hammocks in $\mathcal{H}$ are edge-disjoint we know that we can partition the edges of P into their constituent hammocks. By our above observation we know that for each $H_x \in \mathcal{H}$ it holds that $H_x \cap E(P)$ is a connected (possibly singleton) subpath of P. We let $P'_x := H_x \cap E(P)$ be each such subpath and let $\mathcal{P}'$ be all such induced non-empty subpaths by hammocks along $T_{\mathcal{H}}(H_i, H_j)$.

We proceed to show that P'_x is the xth path in P among all paths in $\mathcal{P}'$. Let H_m be the hammock at maximum height in $T_{\mathcal{H}}$ among all hammocks in $T_{\mathcal{H}}(H_i, H_j)$. By definition of a parent, we know that in any path incident to a vertex in H_x, if said path has a vertex of H_x's parent's parent then such a path must pass through H_x's parent. Moreover, as each hammock is connected and shares a vertex with its parent, we know that there is a path from H_i to H_j. It follows that there is a path P' from H_i to H_j which passes through every hammock in $T_{\mathcal{H}}(H_i, H_j)$ except for possibly H_m. If P' passes through H_m then our claim holds by Theorem 148 and our above observation. If

P' does not pass through H_m then by Theorem 148 we must verify that the vertex in both P'_{m-1} and P'_{m+1} is in H_m. However, since H_m is the parent of both H_{m-1} and H_{m+1} we know that any vertices shared by H_{m-1} and H_{m+1} must be in H_m as it is easy to see that otherwise we could construct a cycle with edges in multiple hammocks by connecting H_{m-1} and H_{m+1} via this vertex as well as through H_m.

Lastly, we note that each $b_{x(x+1)}$ is an articulation vertex and the only vertex in $V(H_x) \cap V(H_{x+1})$ by Theorem 153. □

6.7.2 Hammock Decompositions

We now define a hammock decomposition. Roughly, a hammock decomposition is a forest of hammocks which both contains all shortest paths which start and end with cross edges and whose forest structure reflects the LCA structure of the BFS tree T_{BFS}. Recall that, ultimately, we will show that every series-parallel graph has such a decomposition and use this decomposition to perturb our KPR chops.

We will call a path a cross edge path if its first and last edges are in the cross edges of T_{BFS} (that is, in E_c) and a shortest cross edge path if it is both a cross edge path and it is a shortest path in G. For a hammock H_i, we will henceforth use T_i and T'_i to stand for the hammock trees of H_i and use $r_i = \text{high}(T_i)$ and $r'_i = \text{high}(T'_i)$ to stand for the hammock roots of H_i. In the below definitions we will without loss of generality (WLOG) assume that each r_i satisfies certain properties and that r'_i satisfies certain other properties.

We begin by formalizing the sense in which a forest of hammocks can reflect the LCA structure of the BFS tree. Specifically, recall that given a hammock H_i with hammock roots r_i and r'_i, by definition we know r_i and r'_i are unrelated in T_{BFS}. Thus, we can naturally associate hammock H_i with the vertex $\text{LCA}(r_i, r'_i)$. Then the condition we would like to enforce is that if hammock H_i is an ancestor of H_j in a forest of hammocks $\mathcal{H}$ then $\text{LCA}(r_i, r'_i)$ is an ancestor of $\text{LCA}(r_j, r'_j)$ in T_{BFS}. In fact, we will be able to enforce an even stronger condition than this in our hammock decompositions, as formalized by the following notion of an LCA-respecting forest of hammocks.

Definition 155 (LCA-Respecting Forest of Hammocks)**.** *Let $\mathcal{H} = \{H_i\}_i$ be a rooted forest of hammocks . Then, we say that $\mathcal{H}$ is LCA-respecting if for every pair H_i, H_j where H_i is the parent of H_j in $\mathcal{H}$ then:*

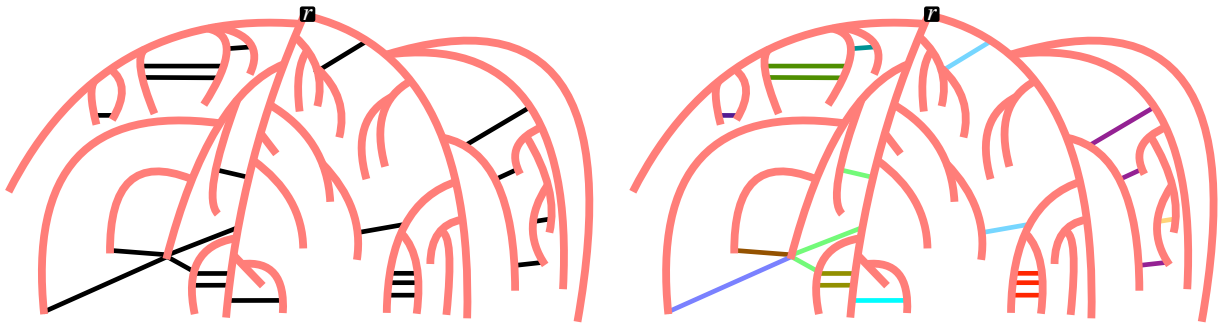
1. *$V(H_i) \cap V(H_j) = \{r_j\}$;*
2. *The parent of r'_j in T_{BFS} is $LCA(r_j, r'_j)$ and $LCA(r_j, r'_j) \in V(H_i) \cup \{LCA(r_i, r'_i)\}$.*

Furthermore, we say that $\mathcal{H}$ is LCA-respecting with base tree $T_0 \subseteq T_{BFS}$ if for each root hammock $H_k \in \mathcal{H}$ we also have

1. *$r_k \in V(T_0)$;*
2. *The parent of r_k and r'_k in T_{BFS} is $LCA(r_k, r'_k)$ and $LCA(r_k, r'_k) \in V(T_0)$.*

Notice that it follows that if $\mathcal{H}$ is LCA-respecting then if H_i is an ancestor of H_j in $\mathcal{H}$ then $\text{LCA}(r_i, r'_i)$ is an ancestor of $\text{LCA}(r_j, r'_j)$ in T_{BFS}. Even stronger, though, we know that if H_i is a parent of H_j then $\text{LCA}(r_j, r'_j) \in V(H_i)$ or $\text{LCA}(r_j, r'_j)$ is equal to $\text{LCA}(r_i, r'_i)$.

Concluding we may now give our definition of a hammock decomposition. Since every hammock contains a cross edge we cannot, in general, expect to decompose a graph into hammocks. For



(a) Series-parallel graph G.

(b) LCA-equivalence classes.

Figure 6.8: An illustration of the LCA-equivalence classes of a series-parallel graph G. Edges of T_{BFS} in pink. Edges of E_c in black in Figure 6.8a and colored according to their LCA-equivalence class in Figure 6.8b. Notice that edges with the same LCA can belong to distinct equivalence classes.

example, if the input graph was just a tree then said graph would contain no hammocks. Thus, our hammock decomposition will partition a series-parallel graph into a forest of hammocks along with a tree T_0 and a "parent edge" for each hammock in our forest of hammocks.

Definition 156 (Hammock Decomposition)**.** *A hammock decomposition of graph $G = (V, E)$ with root r and BFS tree T_{BFS} is a partition of E into $E(T_0) \sqcup E(\mathcal{H}) \sqcup E_p$ where:*

1. *T_0 is a subtree of T_{BFS} containing r;*
2. *$\mathcal{H}$ is an LCA-respecting rooted forest of hammocks with base tree T_0 such that $G[\mathcal{H}]$ contains every shortest cross edge path in G;*
3. *$E_p := \{e_i : H_i \in \mathcal{H}\}$ where e_i is the parent edge of r_i' in T_{BFS}.*

Thus, roughly, a hammock decomposition consists of a base tree T_0 with trees of hammocks "hanging off" of T_0. We illustrate a hammock decomposition where $\mathcal{H}$ consists of two trees of hammocks in Figure 6.7b.

6.8 Hammock Decompositions for Series-Parallel Graphs

In this section we show how to construct hammock decomposition for a series-parallel graph. For the rest of this section we will assume we are working with a fixed series-parallel graph $G = (V, E)$ with a fixed but arbitrary root $r \in V$ and a fixed but arbitrary BFS tree T_{BFS} rooted at r. As in the previous section we will extensively make use of the notation laid out in Section 6.5. The following theorem summarizes the main result of this section.

Theorem 157. *Every series-parallel graph has a hammock decomposition which can be computed in deterministic poly-time.*

The main idea of our construction is to partition all cross edges into equivalence classes based on the behavior of the cross edges' least common ancestors. Notably, two cross edges with the same least common ancestors may end up in different equivalence classes. Each such equivalence class will correspond to one hammock in each of our hammock decomposition.

More specifically, we build a series of forests of hammocks where at each step we add on to our

forest of hammocks to guarantee one of the required properties of our hammock decomposition. We will use different notation for each of these forests of hammocks. For example $\hat{\mathcal{H}}$ will be our first forest of hammocks where a constituent hammock will be notated $\hat{H}_i$ with hammock trees $\hat{T}_i$ and $\hat{T}'_i$ and hammock roots $\hat{r}_i$ and $\hat{r}'_i$. We will also let $\hat{H}$ be the subgraph induced by $\hat{\mathcal{H}}$; and use symmetric notation for our other forests of hammocks. Our progression of forests of hammocks is as follows.

1. $\hat{\mathcal{H}}$: we will initialize our forest of hammocks by connecting each cross edge that falls into the same equivalence class. The result of this will be one hammock for each of our equivalence classes.
2. $\bar{\mathcal{H}}$: Next, we will extend each $\hat{H}_i$ to $\bar{H}_i$ by connecting these hammocks to one another along "hammock-joining" paths. The result of this will be a forest of hammocks $\bar{\mathcal{H}}$ which contains every shortest cross edge path.
3. $\tilde{\mathcal{H}}$: Next we will extend each $\bar{H}_i$ along paths towards $\text{LCA}(\bar{r}_i, \bar{r}'_i)$, resulting in $\tilde{H}_i$, to make our forest of hammocks LCA-respecting
4. $\mathcal{H}$: Lastly, we will add any edges of T_{BFS} which do not appear in a hammock to an incident hammock (with the exception of the connected component of unassigned edges which contain r). This step will ensure that our hammock decomposition indeed partitions all edges of the graph.

We will illustrate this process on the series-parallel graph given in Figure 6.8a throughout this section where Figure 6.15b gives the final hammock decomposition we compute for this graph. We also give all illustrations of the construction in a single figure in the appendix in Figure 6.18.

Most of our proofs will revolve around finding clawed cycles when the above procedure fails to produce a forest of hammocks with the desired properties. As our proofs are quite lengthy, we will briefly highlight three of the major conceptual milestones of this section before proceeding.

1. **Connected Below Subgraphs:** Connected below subgraphs will be a useful abstraction to help us find clawed cycles. An edge $e = \{u, v\} \in H$ where u is the parent of v in T_{BFS} is connected below if there is a path from r to $T_{\text{BFS}}(v)$ which only intersects $T_{\text{BFS}}(u)$ at $T_{\text{BFS}}(v)$. A subgraph will be connected below if each of its edges are. We will argue that the subgraph induced by our construction is connected below. As the paths guaranteed to exist for any connected below edges $\{u, v_l\}$ and $\{u, v_r\}$ (for v_l and v_r children of u and $v_l \neq v_r$) have distinct endpoints in $T_{\text{BFS}}(u)$, we will often use these paths along with $T_{\text{BFS}}(r, u)$ as the paths of our clawed cycles. Theorem 163 gives a concise summary of this technique, showing that any connected below subgraph with at most one edge in each C_i is a forest. The forest structure given by this lemma will be at the core of our proof of why our construction gives a *forest* of hammocks.
2. **Assignments of Hammock-Joining Path Components:** As alluded to above, after computing our $\hat{H}_i$s we will then connect these subgraphs to one another. We will do this by considering the graph induced by all "hammock-joining paths"—roughly, the shortest paths which connect cross edges in different equivalence classes. We will then assign each connected component in this graph to one of its incident hammocks. The main idea here is to first argue that any assignment which is valid (in some later-described technical sense) will result in a forest of hammocks; see Theorem 173. Next, we will argue that no matter which valid assignment we use, for each collection of hammocks incident to one of the

components which we are assigning there is some hammock which will always end up as an ancestor hammock of the other incident hammocks in the resulting forest of hammocks (Theorem 174). We will therefore assign each component to this always-ancestor hammock. That we may assume that each component is assigned to a hammock which is the ancestor of all other incident hammocks will further assist in arguing that our construction gives a forest of hammocks.

3. **Tree of Hammock Ancestry $\leftrightarrow$ LCA Ancestry:** The idea that forms the foundation of how we extend our $\bar{H}_i$s to our $\tilde{H}_i$s to be LCA-respecting is as follows. We will argue that if $\bar{H}_j$ is a descendant of $\bar{H}_i$ in $\bar{\mathcal{H}}$ then the LCA corresponding to the cross edges of $\bar{H}_j$ must be a descendant of the LCA corresponding to the cross edges of $\bar{H}_i$ in T_{BFS}. Theorem 182 summarizes this fact. The fact that $\bar{\mathcal{H}}$'s forest structure reflects the LCA structure of T_{BFS} will be what allows us to ensure that our hammocks are LCA-respecting.

6.8.1 Initial Hammocks $\hat{\mathcal{H}}$ by Connecting Equivalence Classes

In this section we describe our initial forest of hammocks $\hat{\mathcal{H}}$. Roughly, we will define an equivalence relation for cross edges and then if two cross edges fall into the same equivalence we will connect the cross edges to one another. We will also introduce the notion of connected below subgraphs which will help us to argue that the result of this (and the next step in our construction) is a forest of hammocks.

LCA-Equivalent Edges

In this section we define the behavior of cross edges' least common ancestors which we will use to partition our cross edges into equivalence classes. Formally, we partition our cross edges based on "LCA-equivalence" which we define as follows. In the following we WLOG distinguish between the endpoints of e and e'.

Definition 158 (LCA-Equivalent Edges)**.** *Let $e = \{u, v\}$ and $e' = \{u', v'\}$. Then we say that e and e' are LCA-equivalent if $l(e) = l(e')$, $l(u, u') \prec l(e)$ and $l(v, v') \prec l(e)$.*

We emphasize that two cross edges with the same LCA may end up in different equivalence classes. We illustrate these equivalence classes in Figure 6.8b. Next, we verify that, indeed, these sets form an equivalence relation.

Lemma 159. *The set of LCA-equivalent edges forms an equivalence relation.*

Proof. Reflexivity and symmetry are trivial. We prove transitivity. Suppose $e = \{u, v\}$ and $e' = \{u', v'\}$ are LCA-equivalent and e' and $e'' = \{u'', v''\}$ are LCA-equivalent. We claim that e and e'' are LCA-equivalent. In particular, we have $l(e) = l(e')$ and $l(e') = l(e'')$ so $l(e) = l(e'')$. Now consider the (monotone) path from u' to $l(e)$; this path contains both $l(u, u')$ and $l(u', u'')$; WLOG suppose $l(u', u'')$ occurs higher in T_{BFS} in this path. It follows that $l(u', u'')$ has both u and u'' as a descendant and so $l(u, u'') \preceq l(u', u'') \prec l(e)$. A symmetric argument shows $l(v, v'') \prec l(e)$ and so we conclude that e and e'' are LCA-equivalent. □

For the remainder of this section we will let $\mathcal{C} := \{C_i\}_i$ be the equivalence classes of the above equivalence relation. We will let $l(C_i)$ give the LCA of any edge in C_i. Similarly, we will let $h(C_i)$ give the height of $l(C_i)$ in T_{BFS}. We will also slightly abuse notation and let $\preceq$ be the partial

ordering of these equivalence classes according to their LCAs: $C_j \preceq C_i$ iff $l(C_j) \preceq l(C_i)$ and $C_j \prec C_i$ iff $l(C_j) \prec l(C_i)$ where again $\prec$ and $\preceq$ give the "ancestry" partial ordering in T_{BFS}.

Connected Below Subgraphs

The crucial property of the edges which we use to connect edges in the same LCA-equivalence class and to connect our $\hat{H}_i$s will be the following notion of "connected below." This property of these edges will aid us in finding disjoint paths which will, in turn, allow us to construct clawed cycles when our hammock decomposition construction fails.

Definition 160 (Connected Below)**.** *We say that $e = \{u, v\} \in T_{BFS}$ where $v \prec u$ is connected below if there is a path P in G between r and $T_{BFS}(v)$ which satisfies $P \cap T_{BFS}(u) \subseteq T_{BFS}(v)$. We say that a subgraph $G_U = (U, E_U) \subseteq G$ is connected below if each edge of $E_U \cap T_{BFS}$ is connected below.*

The following gives a slightly different but useful characterization of what it means for an edge to be connected below.

Lemma 161. *Let $e = \{u, v\} \in T_{BFS}$ be connected below where $v \prec u$ and let F be a non-empty subgraph of G with vertices contained in $T_{BFS}(v)$. Then there is a path P in G between r and F where $P \cap T_{BFS}(u) \subseteq T_{BFS}(v)$.*

Proof. By the definition of e being connected below we know that there is a path P' from r to $T_{\text{BFS}}(v)$ satisfying $P' \cap T_{\text{BFS}}(u) \subseteq T_{\text{BFS}}(v)$. Extending this path through $T_{\text{BFS}}(v)$ to F gives P. □

A simple proof by contradiction shows that two adjacent connected-below edges cannot also have a path connecting their children vertices in a series-parallel graph.

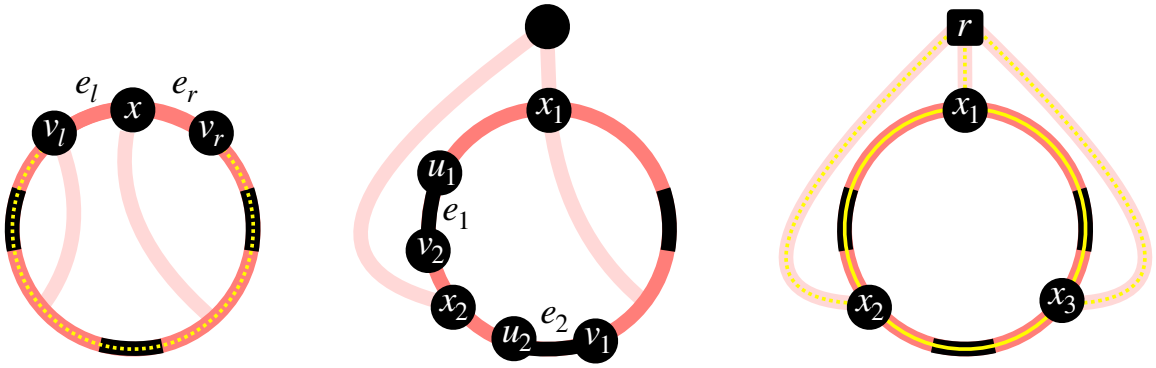
Lemma 162. *There does not exist a vertex x with distinct children v_l and v_r in T_{BFS} such that $\{x, v_l\}$ and $\{x, v_r\}$ are connected below and there exists a path in G contained in $T_{BFS}(x) \setminus \{x\}$ between v_l and v_r*

Proof. Suppose for the sake of contradiction that the stated path exists and call it P. Let C be the cycle created by taking the union of P, $\{x, v_l\}$ and $\{x, v_r\}$. Applying Theorem 161 to the fact that e_l and e_r are connected below, we have that there exists a path P_l from r to C which does not intersect $T_{\text{BFS}}(x) \setminus T_{\text{BFS}}(v_r)$. Symmetrically, there is a path P_r from r to C_e which does not intersect $T_{\text{BFS}}(x) \setminus T_{\text{BFS}}(v_l)$. Letting $P_x := T_{\text{BFS}}(r, x)$, we have that C_e with paths P_x, P_l and P_r forms a clawed cycle, a contradiction. □

Building on the previous lemma, we have our main fact for this section: any subgraph connected below with at most one edge from each equivalence class is a forest. That such a graph is a forest will form the basis of our proof that our hammock decompositions are indeed forests of hammocks.

Lemma 163. *Suppose $G_U = (U, E_U)$ is a subgraph of G which is connected below and where $|C_i \cap E_U| \leq 1$ for every i. Then G_U is a forest.*

Proof. Assume for the sake of contradiction that G_U has a cycle C. Since G_U has at most one edge from each C_i, we know that $|C_i \cap C| \leq 1$ for all i and that C contains at least one edge from E_c. Say that vertex $v \in C$ is a local max if no vertex in C is an ancestor of v; clearly there is at



(a) One local max. **(b)** Two local maxes. **(c)** At least three local maxes.

Figure 6.9: The three cases of the proof of Theorem 163. Edges in C solid, edges outside of C transparent. T_{BFS} in pink and E_c in black. In (a) we give the path between v_l and v_r contained in $T_{\text{BFS}}(x) \setminus \{x\}$ with a dotted yellow path. In (b) we illustrate give $l(e_1) = l(e_2)$ at the top. In (c) we highlight the cycle and the paths of the clawed cycle in solid and dotted yellow respectively.

least one such local max in C. We case on the number of local maxes in C. For each of our three cases we illustrate the contradiction we arrive at in Figures 6.9a, 6.9b and 6.9c respectively.

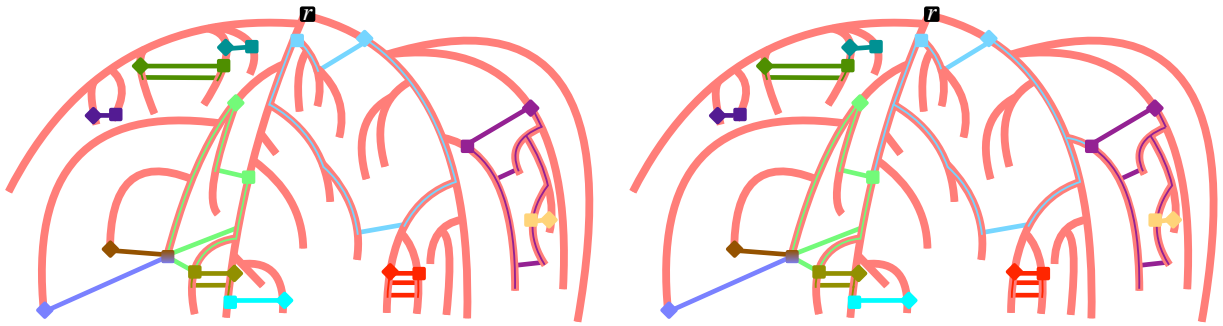
1. Suppose there is 1 local max x in C. Let $e_l = \{v_l, x\}$ and $e_r = \{v_r, x\}$ be the two edges of C incident to x. Since every vertex in C is a descendant of x we know that e_l and e_r are in T_{BFS} and are therefore connected below since otherwise we would have an edge in E_c from a vertex to one of its descendants. However, it follows that the subgraph of C connecting v_r and v_l which does not contain x—call it P—is contained in $T_{\text{BFS}}(x) \setminus \{x\}$. Thus P along with e_l and e_r contradicts Theorem 162.
2. Suppose there are 2 local maxes x_1 and x_2 in C. We will contradict the fact that $|C \cap C_i| \leq 1$ for every i. To do so, we first claim that there are distinct edges $e_1 = \{u_1, v_2\}, e_2 = \{u_2, v_1\} \in E_c \cap C$ where $u_1, v_1 \in T_{\text{BFS}}(x_1)$ and $v_2, u_2 \in T_{\text{BFS}}(x_2)$. To see this, notice that each subpath of C in T_{BFS} is contained in either $T_{\text{BFS}}(x_1)$ or $T_{\text{BFS}}(x_2)$ and so since C is a cycle two such edges must exist. Next, notice that it therefore follows that e_1 and e_2 are LCA-equivalent: $l(u_2, v_2) \in T_{\text{BFS}}(x_2)$ and $l(u_1, v_1) \in T_{\text{BFS}}(x_1)$ but $l(x_1, x_2) = l(e_1) = l(e_2)$ where $x_1, x_2 \prec l(x_1, x_2)$ by our assumption that x_1 and x_2 are local maxes. Thus, we know that e_1 and e_2 lie in the same C_i, a contradiction to the fact that $|C \cap C_i| \leq 1$ for every i.
3. Suppose there are at least 3 local maxes; let x_1, x_2 and x_3 be an arbitrary but distinct three of these maxes. Then cycle C along with paths $P_1 := T_{\text{BFS}}(r, x_1)$, $P_2 := T_{\text{BFS}}(r, x_2)$ and $P_3 := T_{\text{BFS}}(r, x_3)$ form a clawed cycle since each x_i is an ancestor of or unrelated to every other vertex in C. □

Constructing $\hat{\mathcal{H}}$

We now define subgraph $\hat{H}_i$ and argue that $\hat{H}_i$ is indeed a hammock. Our initial hammocks will connect all "LCA-free minimal" cross edge paths.

Definition 164 (Minimal, LCA-Free Cross Edge Paths)**.** *We say that a cross edge path $P \subseteq G$ is LCA-free if $l(e_f), l(e_l) \notin P$. We say that P is minimal if only its first and last edges are in E_c.*

Definition 165 ($\hat{H}_i$)**.** *$\hat{H}_i$ is the subgraph of G induced by all LCA-free minimal cross edge paths*



(a) Initial hammocks $\hat{\mathcal{H}}$. **(b)** Extending $\hat{\mathcal{H}}$ to $\bar{\mathcal{H}}$.

Figure 6.10: An illustration of our initial hammocks $\hat{\mathcal{H}} = \{\hat{H}_i\}_i$ and how we extend them to our final hammocks $\bar{\mathcal{H}}$. Roots and edges of initial hammocks colored according to i. Notice that one vertex is the root of two hammocks (and so colored with two colors).

between edges in C_i.

As a reminder we let $\hat{T}_i$, $\hat{T}'_i$, $\hat{r}_i$ and $\hat{r}'_i$ refer to the two hammock trees and hammock roots of hammock $\hat{H}_i$. Similarly, we let $\hat{\mathcal{H}} := \{\hat{H}_i\}_i$ be the collection of all of our initial hammocks and let $\hat{H}$ be its corresponding induced subgraph of G. We illustrate $\hat{\mathcal{H}}$ in Figure 6.10a.

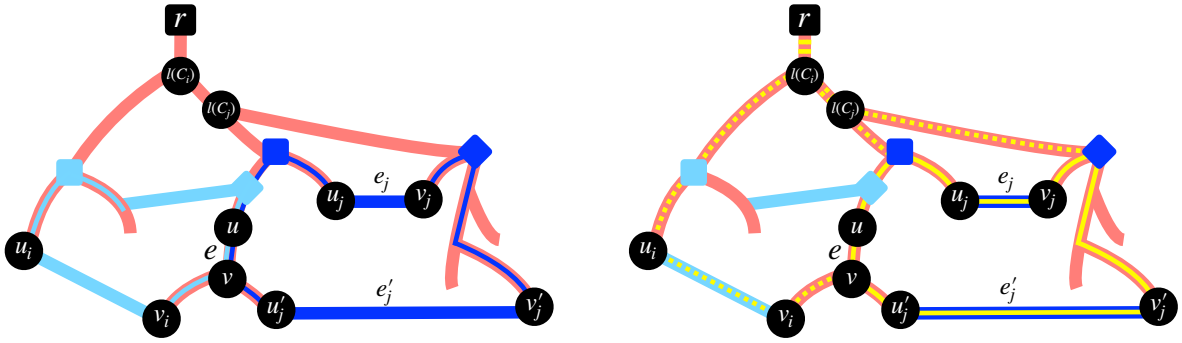
Lemma 166. *Each $\hat{H}_i \in \hat{\mathcal{H}}$ is a hammock.*

Proof. Consider a fixed $\hat{H}_i$. Fix an arbitrary edge $e_0 = (u_0, v_0) \in C_i$. Now consider an arbitrary $e = \{u, v\} \in C_i$. By definition of C_i we know that (WLOG) u_0 and u have an LCA which is a descendant of $l(e) = l(e')$; thus, there is a minimal LCA-free cross edge path of the form $(v_0, u_0, \ldots, u, v)$ which connects e_0 and e which is included in $\hat{H}_i$. Symmetrically, there is a minimal cross edge connecting path of the form $(u_0, v_0, \ldots, v, u)$. We therefore know that u and v are in the same connected components in $\hat{H}_i \setminus E_c$ as u_0 and v_0 respectively. Moreover, since $\hat{H}_i \setminus E_c \subseteq T_{\text{BFS}}$, it follows that $\hat{H}_i \setminus E_c$ consists of at most two trees (the tree containing u_0 and the tree containing v_0). Even stronger, since $l(C_i)$ is not contained in any of the constituent paths of $\hat{H}_i$, we know that $\hat{H}_i \setminus E_c$ consists of exactly two trees and these trees are vertex-disjoint. We let these two trees T and T' be the hammock trees of $\hat{H}_i$. Next, to argue that $\hat{H}_i$ is a hammock we must argue that every cross edge between T and T' is included in $\hat{H}_i$. However, any such edge $e' = \{u', v'\}$ has $l(e') = l(C_i)$. Moreover, since $u', v' \neq l(C_i)$, we have that (WLOG) $l(u_0, u')$ and $l(v_0, v')$ are descendants of $l(C_i)$ and so $e' \in C_i$ and therefore $e' \in \hat{H}_i$. Lastly we must show that high(T) and high(T') are unrelated. However, notice that if (WLOG) high(T') where a descendant of high(T) then there would have been be a minimal LCA-free cross edge path included in $\hat{H}_i$ which would have connected T and T', contradicting the fact that they are vertex disjoint. □

As our H_is must ultimately be edge-disjoint, we will need that our $\hat{H}_i$s are edge-disjoint.

Lemma 167. *$\hat{H}_i$ and $\hat{H}_j$ are edge-disjoint for $i \neq j$.*

Proof. By Theorem 166 we may assume that $\hat{H}_i$ and $\hat{H}_j$ are hammocks. Suppose for the sake of contradiction that $E(\hat{H}_i) \cap E(\hat{H}_j) \neq \emptyset$ for $i \neq j$ and let $e = \{u, v\}$ be an edge included in $\hat{H}_i$ and $\hat{H}_j$. We illustrate the resulting situation, described below, in Figure 6.11a. We will argue that



(a) Theorem 167 proof setup.

(b) Theorem 167 proof contradiction.

Figure 6.11: The contradiction in the proof of Theorem 167. In (a) we illustrate $\bar{H}_i$ in light blue and H_j in dark blue. In (b) we highlight F_j in solid yellow and P_1, P_2 and P_3 in dotted yellow.

this situation leads to a clawed cycle, illustrated in Figure 6.11b. By definition of $\hat{H}_i$ and $\hat{H}_j$, such an edge must an edge in T_{BFS} and so we assume WLOG $v \prec u$.

We claim that in this case $l(C_i)$ and $l(C_j)$ are related; to see this notice that $l(C_i)$ and $l(C_j)$ must both be ancestors of or equal to u by virtue of the fact that $e \in \hat{H}_i, \hat{H}_j$ and that $\hat{H}_i$ and $\hat{H}_j$ are hammocks. WLOG we assume that $C_j \preceq C_i$. Since C_j contains e we know that $|C_j| \geq 2$ and, in particular, that $\hat{H}_j$ contains a hammock-fundamental cycle containing e; let F_j be this cycle with edges $e_j = \{u_j, v_j\}$ and $e'_j = \{u'_j, v'_j\}$ in E_c where we imagine WLOG that u_j and u'_j are in the same hammock tree of $\hat{H}_j$ as e; we will construct a clawed cycle with this cycle. Let $P_1 := T_{\text{BFS}}(r, l(u_j, u'_j))$ and $P_2 := T_{\text{BFS}}(r, l(v_j, v'_j))$ be our first two paths to F_j. By definition of a hammock we know that $l(u_j, u'_j) \neq l(v_j, v'_j)$; also notice that $u \preceq l(u_j, u'_j)$ and $l(v_j, v'_j) \notin T_{\text{BFS}}(l(u_j, u'_j))$.

Next, notice that since $e \in \hat{H}_i$, there must be some edge $e_i = \{u_i, v_i\} \in C_i$ where $v_i \in T_{\text{BFS}}(v)$. Let P'_3 be the path from v_i to F_j along $T_{\text{BFS}}(v_i, v)$ and let $P_3 := T_{\text{BFS}}(r, u_i) \oplus e_i \oplus P'_3$.

To show that F_j along with P_1, P_2 and P_3 are a clawed cycle it suffices to argue that P_3 does not contain $l(u_j, u'_j)$ or $l(v_j, v'_j)$ and so we verify that none of e_i, P'_3 or $T_{\text{BFS}}(r, u_i)$ contain $l(u_j, u'_j)$ or $l(v_j, v'_j)$.

- First, we argue that $l(u_j, u'_j), l(v_j, v'_j) \notin e_i$. Since $v_i \in T_{\text{BFS}}(v)$ and $l(v_j, v'_j) \notin T_{\text{BFS}}(v)$ we know that $v_i \neq l(v_j, v'_j)$. Similarly, since $v_i \preceq v \prec u \preceq l(u_j, u'_j)$, we know that $v_i \neq l(u_j, u'_j)$. Now, consider u_i. It cannot be the case that $u_i \in T_{\text{BFS}}(l(u_j, u'_j))$ since otherwise we would have $l(C_i) \preceq l(u_j, u'_j)$ and so $C_i \prec C_j$, a contradiction. It follows that $u_i \neq l(u_j, u'_j)$. Similarly, we cannot have $u_i = l(v_j, v'_j)$ since then we would have that $C_i = C_j$.
- Since $P'_3 \subseteq T_{\text{BFS}}(v) \subseteq T_{\text{BFS}}(l(u_j, u'_j)) \setminus \{l(u_j, u'_j)\}$, we know that P'_3 contains neither $l(u_j, u'_j)$ nor $l(v_j, v'_j)$.
- Lastly, suppose for the sake of contradiction that $l(u_j, u'_j) \in T_{\text{BFS}}(r, u_i)$. It follows that $l(C_i) \preceq l(u_j, u'_j)$, contradicting the fact that $l(u_j, u'_j) \prec l(C_j) \preceq l(C_i)$. Lastly, suppose for the sake of contradiction that $l(v_j, v'_j) \in T_{\text{BFS}}(r, u_i)$. It follows that $l(C_i) = l(v_i, u_i) \preceq l(l(u_j, u'_j), l(v_j, v'_j)) = l(C_j)$ and since $l(C_j) \preceq l(C_i)$, it follows that $l(C_i) = l(C_j)$. Since $u_i \in T_{\text{BFS}}(l(u_j, u'_j))$ and $v_i \in l(v_j, v'_j)$ we then would have $C_i = C_j$, a contradiction. □

Lastly, as alluded to above, we must establish that each of our $\hat{H}_i$s are connected below.

Lemma 168. *Every $\hat{H}_i$ is connected below for every i.*

Proof. Consider an edge $e \in \hat{H}_i \cap T_{\text{BFS}}$. By definition of $\hat{H}_i$, we know that $e \in \hat{H}_i$ because there are edges $e_1 = \{u_1, v_1\}, e_2 = \{u_2, v_2\} \in C_i$ which are connected by an LCA-free minimal cross edge path P_{12} between e_1 and e_2 containing e. Thus, one of e_1 and e_2 has an endpoint in $T_{\text{BFS}}(v)$; WLOG suppose e_1 does. We may also assume that e_1 has at most one endpoint in $T_{\text{BFS}}(u)$ (and so has exactly one endpoint in $T_{\text{BFS}}(v)$) since if both endpoints of e_1 were in $T_{\text{BFS}}(u)$ then we would have $l(e_1) \in T_{\text{BFS}}(u)$ and so P_{12} would contain $l(e_1)$, contradicting its LCA-freeness. Thus, we assume that $v_1 \in T_{\text{BFS}}(v)$ but $u_1 \notin T_{\text{BFS}}(u)$. Next, we let $P := T_{\text{BFS}}(r, u_1) \oplus e_1$; we know that P is from r to $T_{\text{BFS}}(v)$ and is internally vertex disjoint from $T_{\text{BFS}}(u)$ since $u_1 \notin T_{\text{BFS}}(u)$, thereby demonstrating that e is connected below. □

Notice that it immediately follow from Theorem 163, Theorem 166, Theorem 167 and Theorem 168 that $\hat{\mathcal{H}}$ is a forest of hammocks.

6.8.2 Extending $\hat{\mathcal{H}}$ to $\bar{\mathcal{H}}$ by Hammock-Joining Paths

We now describe how we extend $\hat{\mathcal{H}}$ to $\bar{\mathcal{H}}$ via what we call "hammock-joining" paths. We do so to ensure that our final forest of hammocks contains all shortest cross edge paths. We illustrate this process in Figure 6.10b.

Definition 169 (Hammock-Joining Paths)**.** *We say that path $P \subseteq T_{BFS}$ is a hammock-joining path if*

1. *P is between distinct hammocks $\hat{H}_i, \hat{H}_j \in \hat{\mathcal{H}}$;*
2. *$l(C_i), l(C_j) \notin P$.*

We let H_{HJ} be the subgraph induced by all hammock-joining paths for the rest of this section.

Lemma 170. *Every $e \in H_{HJ}$ is connected below.*

Proof. Consider an edge $e = \{u, v\}$ where u is the parent of v in T_{BFS} which is part of a hammock-joining path P_{ij} between $\hat{H}_i$ and $\hat{H}_j$. We will construct a path P from r to $T_{\text{BFS}}(v)$ where $P \cap T_{\text{BFS}}(u) \subseteq T_{\text{BFS}}(v)$ as required by the definition of connected below. WLOG we assume P_{ij}'s endpoint in $\hat{H}_i$ is in $T_{\text{BFS}}(u)$. Since every vertex in $\hat{H}_i$ is a descendant of $l(C_i)$ and $l(C_i) \notin P$ since P is hammock-joining, we know that $l(C_i) \notin T_{\text{BFS}}(u)$. Notice that, by construction, each leaf of each of $\hat{H}_i$'s hammock trees is incident to an edge of C_i. Thus, we know that there is some $e_i = \{u_i, v_i\} \in C_i$ where WLOG $v_i \in T_{\text{BFS}}(u)$ but $u_i \notin T_{\text{BFS}}(u)$. Then, we can let P be $T_{\text{BFS}}(r, u_i)$. We know that P is internally vertex-disjoint from $T_{\text{BFS}}(u)$ since $u_i \notin T_{\text{BFS}}(u)$. Thus, P is from r to F and $P \cap T_{\text{BFS}}(u) \subseteq T_{\text{BFS}}(v)$ as required. □

Next, we describe how we will assign each collection of connected components in H_{HJ} to one of our hammocks. We always assign such a component to an incident hammock, hence the following definition.

Definition 171 ($I(T)$)**.** *For a connected component T of $H_{HJ} \setminus \hat{H}$, we let $I(T) := \{i : V(\hat{H}_i) \cap V(T) \neq \emptyset\}$ be the indices of initial hammocks which intersect T.*

The following definition formalizes the notion of a valid assignment of connected components of H_{HJ} to hammocks.

Definition 172 (Valid Assignment of Connected Components)**.** *Let π be a mapping from components of $H_{HJ} \setminus \hat{H}$ to non-negative integers. Then we say that π is an assignment of the connected components of $H_{HJ} \setminus \hat{H}$ if it maps each component $T \subseteq H_{HJ} \setminus \hat{H}$ to an index in $I(T)$. We say that $\bar{\mathcal{H}} = \{\bar{H}_i\}_i$ results from π if $\bar{H}_i = \hat{H}_i \cup \bigcup_{T:\pi(T)=i} T$. We say that π is valid if it holds that when $\pi(T) = i$ then $l(C_i) \notin T$ for every component $T \subseteq H_{HJ} \setminus \hat{H}$.*

We now show that, provided connected components of $H_{\text{HJ}} \setminus \hat{H}$ are assigned in a valid way, the result is a forest of hammocks.

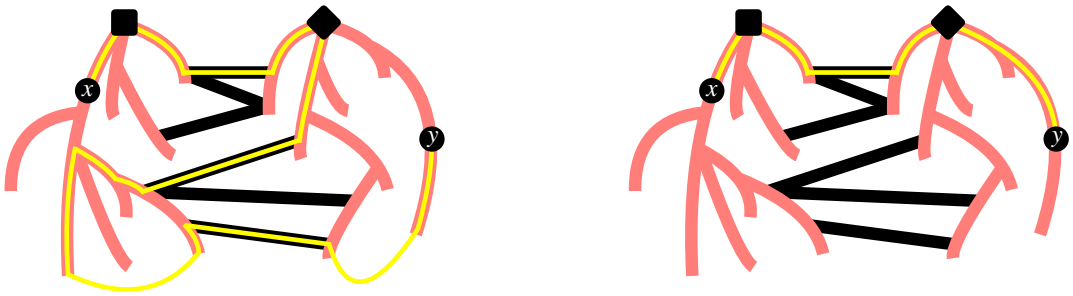
Lemma 173. *Let π be a valid assignment of the connected components of $H_{HJ} \setminus \hat{H}$. Then if $\bar{\mathcal{H}}$ results from π then $\bar{\mathcal{H}}$ is a forest of hammocks.*

Proof. We let $\bar{H}$ be the subgraph induced by $\bar{\mathcal{H}}$ for the remainder of this proof. We first note that by Theorem 168 and Theorem 170, $\bar{H}$ is connected below; we will use this fact several times in this proof.

We begin by verifying that every $\bar{H}_i$ is indeed a hammock. By Theorem 166 $\bar{H}_i$ is a hammock when we initialize it to $\hat{H}_i$. Next, we claim that adding connected components of $H_{\text{HJ}} \setminus \hat{H}$ to $\bar{H}_i$ does not violate its hammockness. First, notice that $\bar{H}$ does not contain any fundamental cycles of T_{BFS} since $\bar{H}$ is connected below so such a cycle would be a connected below subgraph violating Theorem 162. Thus, since each of the connected components of $H_{\text{HJ}} \setminus \hat{H}$ we add to $\bar{H}_i$ intersect with $\hat{H}_i$ on some vertex, we know $\bar{H}_i$ continues to be spanned by two subtrees of T_{BFS}. To verify that $\bar{H}_i$ is a hammock we must also show that the edges between these two trees in E_c are exactly C_i; since $C_i \subseteq \hat{H}_i$, it suffices to show that no additional edges from E_c become incident to the two hammock trees of $\bar{H}_i$ as we add connected components of $H_{\text{HJ}} \setminus \hat{H}$ to $\bar{H}_i$. However, notice that the existence of such an edge would give us a cycle in $\bar{H}$ incident to two LCA-equivalence classes, contradicting Theorem 163. Lastly, it remains to argue that the roots of $\bar{H}_i$s hammock trees are unrelated. This is immediate from our assumption that π is valid—i.e. if $\pi(T) = i$ then $l(C_i) \notin T$ for every component $T \subseteq H_{\text{HJ}} \setminus \hat{H}$—since the roots of $\bar{H}_i$'s hammock trees would only become related if $l(C_i)$ were included in some T assigned to i.

Next, notice that the $\bar{H}_i$s are edge-disjoint by construction. In particular, the $\hat{H}_i$s are disjoint by Theorem 167. Moreover, each connected component of $H_{\text{HJ}} \setminus \hat{H}$ is pair-wise edge-disjoint by definition and also disjoint from any $\hat{H}_i$ by construction. Thus, in constructing $\bar{H}_i$ by adding connected components from $H_{\text{HJ}} \setminus \hat{H}$ to $\hat{H}_i$, our resulting $\bar{H}_i$s must be edge-disjoint.

Lastly, we verify our cycle property: we must show that any cycle C in $\bar{H}$ satisfies $|\bar{H}_i : E(C) \cap \bar{H}_i \neq \emptyset| = 1$. We use a "shortcutting" argument—which we illustrate in Figure 6.12—and Theorem 163. In particular, suppose that $\bar{H}$ contains a cycle C where $|\bar{H}_i : E(C) \cap \bar{H}_i \neq \emptyset| \geq 2$. Then, we claim that $\bar{H}$ also contains a cycle C' where $|\bar{H}_i : E(C') \cap \bar{H}_i \neq \emptyset| \geq 2$ *and* $|C_i \cap E(C)| \leq 1$ for all i. To see this, notice that, since our $\bar{H}_i$s are edge-disjoint we may "shortcut" C through the trees of $\bar{H}_i$. In particular, let $\bar{H}_i$ be one of our hammocks where $|C_i \cap E(C)| \geq 2$ and let x and y be the first and last vertices of $\bar{H}_i$ visited by C (for some arbitrary cyclic ordering of the edges in C). Then, if x and y are in the same hammock tree of $\bar{H}_i$ then we replace the portion of C between x and y with $T_{\text{BFS}}(x, y)$. If x and y are in different hammock trees, we replace the portion of C between x and y with an arbitrary path in $\bar{H}_i$ which uses at most one edge of C_i. Doing these replacements does not affect $|\bar{H}_i : E(C) \cap \bar{H}_i \neq \emptyset|$ and reduces



(a) C in $\bar{H}_i$. **(b)** Shortcutting C in $\bar{H}_i$.

Figure 6.12: An illustration of how in the proof of Theorem 173 we may assume that $|C \cap C_i| \leq 1$ by shortcutting C in $\bar{H}_i$. We highlight C in yellow both before and after shortcutting.

the number of $\bar{H}_i$ with $|C_i \cap E(C)| \geq 2$ by at least 1; thus, after iterating a finite number of times we produce our desired C'.

The existence of C' allows us to arrive at a contradiction. In particular, since $\bar{H}$ is connected below we know that C' is connected below where $|C_i \cap C'| \leq 1$; but C' is a cycle, contradicting Theorem 163. □

Henceforth we will assume that the root hammock of each tree of hammocks $\mathcal{T}$ in $\bar{\mathcal{H}}$ is $\max_{i:\bar{H}_i \in \mathcal{T}} h(C_i)$ where we break ties arbitrarily. As a reminder, $h(C_i)$ is the height of $l(C_i)$ in T_{BFS}.

Lemma 174. *For any connected component T of $H_{HJ} \setminus \hat{H}$ there is some $i \in I(T)$ such that for any valid assignment π of the connected components of $H_{HJ} \setminus \hat{H}$ with $\bar{\mathcal{H}}$ as the resulting forest of hammocks we have $\bar{H}_j \preceq_{\bar{\mathcal{H}}} \bar{H}_i$ for all $j \in I(T)$.*

Proof. Let $\bar{H} := \hat{H} \cup H_{\text{HJ}}$ where $\hat{H} = G[\hat{\mathcal{H}}]$. Fix a connected component of $\bar{H}$; notice that we always choose the same root hammock for this component regardless of π; we let $\bar{H}_k$ be said root hammock. Fix a component T of $H_{\text{HJ}} \setminus \hat{H}$ and let $I := I(T)$ for the rest of this proof.

If $\bar{H}_k$ is incident to T then the claim trivially holds since $\bar{H}_j \preceq_H \bar{H}_k$ for any assignment and $j \in I \setminus \{k\}$ and so we will assume that $k \notin I$.

Fix an arbitrary valid assignment π of the connected components of $H_{\text{HJ}} \setminus \hat{H}$, let $\bar{\mathcal{H}}$ be the resulting forest of hammocks as per Theorem 173 and say that i is a local max with respect to π if there is no $j \in I \setminus \{i\}$ where $\bar{H}_i$ is a descendant of $\bar{H}_j$ in $\bar{\mathcal{H}}$. Notice that there is at least 1 local max since, by definition of H_{HJ}, there are at least two distinct hammocks among $\bar{\mathcal{H}}$ with a vertex in T. To prove our claim it suffices to show that there is 1 local max under each assignment and this local max is always the same.

We claim that the number of local maxes with respect to π is at most 1. To see this, assume for the sake of contradiction that there are 2 local maxes $\bar{H}_i$ and $\bar{H}_j$ where $i, j \in I$. Since $\bar{H}_i$ and $\bar{H}_j$ are both local maxes neither of which is the root hammock $\bar{H}_k$ then by virtue of $\bar{\mathcal{H}}$ being a forest of hammocks we know that any path from $\bar{H}_i$ to H_j in $G[\bar{\mathcal{H}}]$ must contain at least one vertex of the parent of $\bar{H}_i$ as per Theorem 154. On the other hand, since $i, j \in I$, we know there is a path $P \subseteq T$ between $\bar{H}_i$ and $\bar{H}_j$. Letting $\bar{H}_{i'}$ be the parent of $\bar{H}_i$ in $\bar{\mathcal{H}}$, it follows that $V(\bar{H}_{i'}) \cap P \neq \emptyset$ and so $i' \in I$, contradicting the fact that $\bar{H}_i$ is a local max.

Thus, for any assignment we know that the number of local maxes is 1. We proceed to show that this is always the same local max. In particular, we show that if $\bar{H}_i$ is the local max under some assignment π, then $\bar{H}_i$ is the local max under any other assignment π'. Suppose for the sake of contradiction that i is the local max under π but j for $j \neq i$ is the local max under some other π'. Let $\bar{\mathcal{H}}$ and $\bar{\mathcal{H}}'$ be the forest of hammocks resulting from π and π' as per Theorem 173. Let $\bar{H}_i$ and $\bar{H}'_i$ be i's hammock in $\bar{\mathcal{H}}$ and $\bar{\mathcal{H}}'$ and let $\bar{H}_j$ and $\bar{H}'_j$ be j's hammock in $\bar{\mathcal{H}}$ and $\bar{\mathcal{H}}'$. We emphasize that $\bar{H}_i$ and $\bar{H}'_j$ need not be edge-disjoint since they are hammocks in two different forests of hammocks.

Now by virtue of the fact that i is a local max in $\bar{\mathcal{H}}$ and $i \neq k$, we know by Theorem 154 that every path between $\bar{H}_k$ and $\bar{H}_j$ contains an edge of $\bar{H}_i$; it follows that every path between $\hat{H}_k$ and $\hat{H}_j$ contains an edge of $\bar{H}_i$. We additionally make the stronger claim that every path between $\hat{H}_k$ and $\hat{H}_j$ contains an edge of $\hat{H}_i$. To see this, suppose for the sake of contradiction that there was a path P between $\hat{H}_k$ and $\hat{H}_j$ whose only edges in $\bar{H}_i$ are contained in $\bar{H}_i \setminus \hat{H}_i \subseteq H_{\text{HJ}} \setminus \hat{H}$. Let $\bar{H}_{i'}$ be the parent of $\bar{H}_i$ in $\bar{\mathcal{H}}$ and let P' be the subpath of P restricted to $\bar{H}_i$ where P' is between vertices u and v where $u \in \bar{H}_{i'}$. Since $u \in \bar{H}_{i'}$ and $P' \subseteq H_{\text{HJ}} \setminus \hat{H}$ it follows that $i' \in I$, contradicting our assumption that i is a local max under π. Thus, indeed, every path between $\hat{H}_k$ and $\hat{H}_j$ contains an edge of $\hat{H}_i$.

On the other hand, by virtue of the fact that j is a local max in $\bar{\mathcal{H}}'$, we know by Theorem 154 that in $G[\bar{\mathcal{H}}']$ there is a path from $\bar{H}'_k$ to $\bar{H}'_j$ which does not have an edge in $\bar{H}'_i$. Extending this path through $\bar{H}'_k$ and $\bar{H}'_j$ on either end, we have that there is a path from $\hat{H}_k$ to $\hat{H}_j$ which does not have an edge in $\bar{H}'_i$ and therefore no edge in $\hat{H}_i$. However, this contradicts the above claim that every path between $\hat{H}_k$ and $\hat{H}_j$ contains an edge of $\hat{H}_i$. □

Lemma 175. *There is at least one valid assignment of the components of $H_{HJ} \setminus \hat{H}$.*

Proof. Fix a connected component T of $H_{\text{HJ}} \setminus \hat{H}$. Recall that T is a connected subtree of T_{BFS}. Let x be the highest vertex in T_{BFS} in T. Since x is included in H_{HJ} it must lie on a hammock-joining path P_{ij} between some $\hat{H}_i$ and $\hat{H}_j$. We know that neither $l(C_i)$ nor $l(C_j)$ lie on P_{ij} and so neither $l(C_i)$ nor $l(C_j)$ are in T. Thus, assign T to i. Doing so for each such T results in a valid assignment. □

By Theorem 175 there is at least one valid assignment of the components of $H_{\text{HJ}} \setminus \hat{H}$. It follows that by Theorem 174 there is some valid assignment $\bar{\pi}$ which by Theorem 173 results in a forest of hammocks $\bar{\mathcal{H}}$ where we have $\bar{H}_j \preceq_{\bar{\mathcal{H}}} \bar{H}_i$ for all $j \in I(T)$. We use this assignment in our construction. In particular, henceforth we let $\bar{\mathcal{H}}$ be the forest of hammocks which results from $\bar{\pi}$ and let $\bar{H} := G[\bar{\mathcal{H}}]$ be its induced subgraph. We will let $\bar{T}_i$, $\bar{T}'_i$, $\bar{r}_i$ and $\bar{r}'_i$ refer to the two hammock trees and hammock roots of $\bar{H}_i$ henceforth.

We proceed to argue that $\bar{\mathcal{H}}$ is a rooted forest of hammocks. To do so, we first prove two simple technical lemmas. Recall from Theorem 153 that two hammocks share at most one vertex in a rooted tree of hammocks.

Lemma 176. *Let $\mathcal{T}$ be a rooted tree of hammocks and let $\bar{H}_i$ be the parent of $\bar{H}_j$ in $\mathcal{T}$ where v_{ij} is the one vertex in $V(\bar{H}_i) \cap V(\bar{H}_j)$. Then there is a path P from r to v_{ij} where $P \cap \bar{H}_j = \{v_{ij}\}$.*

Proof. Let $\bar{H}_k$ be the root hammock of $\mathcal{T}$. Notice that it suffices to prove that there is a path from r to $\bar{H}_k$ which is vertex disjoint from $\bar{H}_j$ since we can continue such a path through $G[\mathcal{T}]$ to v_{ij} and such a path will only intersect $\bar{H}_j$ at v_{ij} by Theorem 154.

Let $P_0 := T_{\text{BFS}}(r, l(C_k))$. We claim that $V(\bar{H}_j) \cap P_0 = \emptyset$; to see this notice that if $\bar{H}_j$ had a vertex in P_0 then we would have $l(C_k) \prec l(C_j)$, contradicting our choice of $\bar{H}_k$.

Next, let $\bar{r}_k$ and $\bar{r}'_k$ be the roots of $\bar{H}_k$'s trees and let $P := T_{\text{BFS}}(l(C_k), \bar{r}_k)$ and $P' := T_{\text{BFS}}(l(C_k), \bar{r}'_k)$ be the paths from the LCA of $\bar{H}_k$ to its roots. We claim that either $P \cap V(\bar{H}_j) = \emptyset$ or $P' \cap V(\bar{H}_j) = \emptyset$; this is sufficient to show our claim since P_0 concatenated with the non-intersecting path will give us our required path to $\bar{H}_k$.

Suppose for the sake of contradiction that $P \cap V(\bar{H}_j) \neq \emptyset$ and $P' \cap V(\bar{H}_j) \neq \emptyset$. Let T_j and T'_j be the hammock trees of $\bar{H}_j$ with respective roots $\bar{r}_j$ and $\bar{r}'_j$. We cannot have that both P and P' have a vertex in T_j since then it would follow that $l(C_k) \in V(T_j)$ and so $l(C_k) \prec l(C_j)$ (since $u \prec l(C_j)$ for every $u \in T_j$), again contradicting our choice of $\bar{H}_j$. Thus, it must be the case that (WLOG) P has a vertex in T_j and P' has a vertex in T'_j. It follows that $l(C_k) = l(C_j)$. However, we claim that it also follows that $C_k = C_j$. In particular, any edges $e_k = \{u_k, v_k\} \in C_k$ and $e_j = \{u_j, v_j\} \in C_j$ then satisfy (WLOG) $l(u_k, u_j) \preceq \bar{r}_j$ and $l(v_k, v_j) \preceq \bar{r}'_j$. Since $l(C_j) = l(C_k)$, we know that $\bar{r}_j \neq l(C_j)$ and $\bar{r}'_j \neq l(C_j)$ and so, indeed, e_j and e_k are LCA-equivalent. This is a contradiction since we have assumed that $j \neq k$ in assuming that $\bar{H}_j$ has a parent $\bar{H}_i$ in $\mathcal{T}$. □

Lemma 177. *Fix i. Suppose there is some $x \in \hat{H}_i$ in WLOG $\hat{T}_i$ where $x \neq \hat{r}_i$. Then $\hat{H}_i$ has a hammock-fundamental cycle C containing x where $x \neq$ high$(V(C) \cap V(\hat{T}_i))$.*

Proof. By construction of $\hat{H}_i$ we know that every leaf of $\hat{T}_i$ is incident to an edge of $E_c(\hat{T}_i, \hat{T}'_i)$ and that $\hat{r}_i$ has at least two children. Thus, there is a path which contains x from $\hat{r}_i$ to some vertex u where $\{u, u'\} \in E_c(\hat{T}_i, \hat{T}'_i)$ and another edge-disjoint path in $\hat{T}_i$ from $\hat{r}_i$ to some v where $\{v, v'\} \in E_c(\hat{T}_i, \hat{T}'_i)$. Connecting these paths in $\hat{T}'_i$ gives the stated fundamental cycle. □

Concluding, we have that $\bar{\mathcal{H}}$ is indeed a rooted forest of hammocks containing all shortest cross edge paths.

Lemma 178. *$\bar{\mathcal{H}}$ is a rooted forest of hammocks where if $\bar{H}_i$ is a parent of $\bar{H}_j$ then $\bar{H}_i \cap \bar{H}_j = \{\bar{r}_j\}$. Furthermore, $G[\bar{\mathcal{H}}]$ contains all shortest cross edge paths.*

Proof. Since by Theorem 173 $\bar{\mathcal{H}}$ is a forest of hammocks, it remains to show that $v_{ij} = \bar{r}_j$ for any i, j pair where $\bar{H}_i$ is a parent of $\bar{H}_j$ in $\bar{\mathcal{H}}$ and, as before, $v_{ij} \in V(\bar{H}_i) \cap V(\bar{H}_j)$ is the one vertex in both $\bar{H}_i$ and $\bar{H}_j$. Let T_j and T'_j be the two hammock trees of $\bar{H}_j$ with roots $\bar{r}_j$ and $\bar{r}'_j$ and let $\hat{T}_j \subseteq T_j$ and $\hat{T}'_j \subseteq T'_j$ be the corresponding hammock trees of $\hat{H}_j$ with roots (i.e. highest vertices in T_{BFS}) $\hat{r}_j$ and $\hat{r}'_j$. We assume WLOG that $v_{ij} \in T_j$ and so it suffices to show that $v_{ij} = \bar{r}_j$.

Assume for the sake of contradiction that $v_{ij} \neq \bar{r}_j$. We claim that it follows that $v_{ij} \neq \hat{r}_j$: if $v_{ij} = \hat{r}_j$ then since $v_{ij} \in \bar{H}_i$, by how we construct $\bar{\mathcal{H}}$ we know that no component of $H_{\text{HJ}} \setminus \hat{H}$ incident to v_{ij} would be assigned to j (because it could have been assigned to i) and so if v_{ij} were $\hat{r}_j$ (which is to say it is the highest vertex in $\hat{T}_j$), then it would also be the highest vertex in T_j as no component of $H_{\text{HJ}} \setminus \hat{H}$ assigned to $\bar{H}_j$ could have a vertex higher than v_{ij}, contradicting our assumption that $v_{ij} \neq \bar{r}_j$. On the other hand it must be the case that $v_{ij} \in \hat{H}_j$: no component of $H_{\text{HJ}} \setminus \hat{H}$ which v_{ij} is incident to will be assigned to $\bar{H}_i$ and so it cannot be the case that $v_{ij} \in \bar{H}_j \setminus \hat{H}_j$.

Applying Theorem 177, it follows that $\bar{H}_j$ contains a fundamental cycle C with highest vertices v_j and v'_j in T_j and T'_j respectively where $v_j, v'_j \neq v_{ij}$. Such a cycle gives a contradiction since

it follows that C along with $T_{\text{BFS}}(r, v_j)$, $T_{\text{BFS}}(r, v'_j)$ and the path from r to v_{ij} as guaranteed by Theorem 176 gives a clawed cycle.

It remains to verify that $G[\bar{\mathcal{H}}]$ indeed contains every shortest cross edge path. It suffices to show that $G[\bar{\mathcal{H}}]$ contains all LCA-free cross edge paths (recall that an LCA-free cross edge path is one between two cross edges which contains neither of the cross edges' LCAs) since every shortest cross edge path is LCA-free. Let P be an arbitrary LCA-free cross edge path.

1. If P is between two edges e, e' where $e, e' \in C_i$ for some i then we know that $E(P) \subseteq \hat{H}_i$ by definition of $\hat{H}_i$.
2. On the other hand, suppose P is between $e \in C_i$ and $e' \in C_j$ for $i \neq j$. Let P' be the edges of P which are not in $\hat{H}_i$ or $\hat{H}_j$. P' must be a connected subpath by definition of $\hat{H}_i$ and $\hat{H}_j$ and so P' is a hammock-joining path between $\hat{H}_i$ and $\hat{H}_j$; thus every edge of P will be included in H.

It follows that $G[\bar{\mathcal{H}}]$ contains all shortest cross edge paths. □

6.8.3 Extending $\bar{\mathcal{H}}$ to $\tilde{\mathcal{H}}$ by LCA Paths

The second to last step in the construction of our hammock decomposition is to extend $\bar{H}_i$ along the path between $\bar{r}'_i$ and $\text{LCA}(\bar{r}_i, \bar{r}'_i)$ to the child of $\text{LCA}(\bar{r}_i, \bar{r}'_i)$ in T_{BFS} which contains $\bar{r}'_i$ in its T_{BFS} subtree. This will allow us to argue that our final hammock decomposition is LCA-respecting.

Hammock Ancestry $\leftrightarrow$ LCA Ancestry in $\bar{\mathcal{H}}$

The key to extending along LCA paths will be to show that the LCA structure of our LCA equivalence classes reflects the ancestry structure of the hammocks in our forest of hammocks. As with the proofs of the previous section, we will leverage the connected belowness of certain edges, as summarized in the following lemma.

Lemma 179. *Suppose $V(\bar{H}_i) \cap V(\bar{H}_j) = \{v_{ij}\}$ where $C_j \prec C_i$ and let $e = \{l(C_j), v\}$ be the child edge of $l(C_j)$ in T_{BFS} satisfying $v_{ij} \in T_{BFS}(v)$. Then e is connected below.*

Proof. Our aim is to construct a path from r to $T_{\text{BFS}}(v)$ whose only vertex in $T_{\text{BFS}}(l(C_j))$ is its endpoint in $T_{\text{BFS}}(v)$. We assume WLOG that $v_{ij} \in \bar{T}'_i$.

If $e \in H$ then we know that e is connected below since by Theorem 168 and Theorem 170 H is connected below. Thus, we may assume that $e \notin H$. It follows that $l(C_j) \notin \bar{H}_i$: if $l(C_j)$ were in $\bar{H}_i$ then it would have to be in $\bar{T}'_i$ since $v_{ij} \in \bar{T}'_i$ and no vertex in $\bar{T}_i$ is related to a vertex in $\bar{T}'_i$ but v_{ij} and $l(C_j)$ are related; however if $l(C_j)$ were in $\bar{T}'_i$ then since v_{ij} is also in $\bar{T}'_i$ and $\bar{T}'_i$ is a connected subtree of T_{BFS}, it would follow that $e \in \bar{H}_i$ and therefore $e \in H$ and connected below.

Given that $l(C_j) \notin \bar{H}_i$, we can construct our path P from r to $T_{\text{BFS}}(v)$ as follows. First, take the path $P_1 := T_{\text{BFS}}(r, \bar{r}_i)$ from r to $\bar{r}_i$. Continue this through an arbitrary path $P_2 \subseteq \bar{H}_i$ to any vertex in $T_{\text{BFS}}(v)$, using a single edge of E_c; P_2 is well-defined since $v_{ij} \in T_{\text{BFS}}(v) \cap \bar{T}'_i$. Let $P = P_1 \oplus P_2$ be the resulting path. Since $v_{ij} \in T_{\text{BFS}}(v)$ this path is indeed from r to $T_{\text{BFS}}(v)$.

It remains to verify that $P \cap T_{\text{BFS}}(l(C_j)) \subseteq T_{\text{BFS}}(v)$ and in particular we show that $P \cap T_{\text{BFS}}(l(C_j)) = \{v_{ij}\}$. It cannot be the case that P_1 intersects $T_{\text{BFS}}(l(C_j))$ since it would then follow that $l(C_j)$ is an ancestor of $\bar{r}_j$ and since $l(C_j)$ is an ancestor of v_{ij} and also therefore an ancestor of $\bar{r}'_i$ so $l(C_i) \preceq l(C_j)$, contradicting our assumption that $C_j \prec C_i$. Similarly, we know that $\bar{T}_i$

does not contain any vertices of $T_{\text{BFS}}(l(C_j))$ since it would then follow that $l(C_i) \preceq l(C_j)$. Thus, edges of P_2 in $\bar{T}_i$ have no vertices from $T_{\text{BFS}}(l(C_j))$; similarly, edges of P_2 in $\bar{T}'_i$ cannot contain vertices of $T_{\text{BFS}}(l(C_j)) \setminus T_{\text{BFS}}(v)$ since $\bar{T}'_i$ is a connected subtree of $T_{\text{BFS}}(v)$ by our assumption that $l(C_j) \notin \bar{H}_i$. Concluding, it follows that $P \cap T_{\text{BFS}}(l(C_j)) \subseteq T_{\text{BFS}}(v)$ as desired. □

In arguing that the LCA structure reflects the hammock ancestry structure, we will distinguish between hammocks based on which of their parent's trees they connect to.

Definition 180 (Left, right child)**.** *Let $\bar{H}_j$ be a child of $\bar{H}_i$ in $\bar{\mathcal{H}}$. Then $\bar{H}_j$ is a left child of $\bar{H}_i$ if $\bar{r}_j \in \bar{T}_i$ and a right child of $\bar{H}_i$ if $\bar{r}_j \in \bar{T}'_i$.*

The following will allow us to argue that the paths we construct for left children in our forest of hammocks interact with their parents in an appropriate way.

Lemma 181. *If $\bar{H}_j$ is a left child of $\bar{H}_i$ and $\bar{H}_i$ is not a root hammock then $l(C_j) \in T_{BFS}(\bar{r}_i, \bar{r}_j) \subseteq \bar{T}_i$.*

Proof. First, notice that since $\bar{r}_j \in \bar{T}_i$ we know that both $l(C_j)$ and $l(C_i)$ are ancestors of $\bar{r}_j$ and so $l(C_j)$ and $l(C_i)$ are related. Thus, if $l(C_j) \notin T_{\text{BFS}}(\bar{r}_i, \bar{r}_j)$ then it must be the case that $\bar{r}_i \prec l(C_j)$; suppose for the sake of contradiction that indeed $\bar{r}_i \prec l(C_j)$.

We first claim that there is a path P_1 with edges contained in $\bar{T}_i$ from $\bar{r}_j$ to a vertex v which is in a cycle C in $\hat{H}_i$. Further, if $u = \text{high}(V(\bar{T}_i) \cap V(C))$ and $u' = \text{high}(V(\bar{T}'_i) \cap V(C))$ then we will have $v \neq u, u'$. Let us argue why such a P_1 exists. It suffices to argue that there is some $v \in \hat{T}_i$ where $v \neq \hat{r}_i$ and a path between $\bar{r}_j$ and $\hat{T}_i$ ending in v since by Theorem 177 the existence of such an v would give us our required cycle. By noting that $\hat{r}_i = \bar{r}_i$ we can further simplify what we must prove. In particular, let $\bar{H}_{i'}$ be the parent of $\bar{H}_i$ in $\bar{\mathcal{H}}$ which we know exists by our assumption that $\bar{H}_i$ is not a root hammock. Additionally, notice that we always know that $\bar{r}_i = \hat{r}_i$ since if $\bar{r}_i$ were strictly higher than $\hat{r}_i$ it would be because it lies in a connected component of $H_{\text{HJ}} \setminus \hat{H}$ incident to both $\bar{r}_i$ and $\hat{r}_i$. Such a component is incident to $\bar{H}_{i'}$ by definition of a forest of hammocks. Even stronger, such a component must be incident to $\hat{H}_{i'}$ since every component of $H_{\text{HJ}} \setminus \hat{H}$ incident to $\bar{H}_{i'}$ is also incident to $\hat{H}_{i'}$ which means that any such component would always be assigned to i'.

Thus, it suffices to argue that that there is some $v \in \hat{T}_i$ where $v \neq \bar{r}_i$ and a path between $\bar{r}_j$ and $\hat{T}_i$ ending in v. Note that $\bar{r}_j$ was added to $\bar{T}_i$ either because a component of $H_{\text{HJ}} \setminus \hat{H}$ was assigned to i or because it was already $\hat{T}_i$. In particular, either $\bar{r}_j \in \hat{T}_i$ or $\bar{r}_j \in \bar{T}_i \setminus \hat{T}_i$.

1. Suppose $\bar{r}_j \in \hat{T}_i$. In this case we can let our path from $\bar{r}_j$ to $\hat{T}_i$ be the trivial path consisting only of $\bar{r}_j$; it remains only to show that $\bar{r}_j \neq \bar{r}_i$. However, it cannot be the case that $\bar{r}_j = \bar{r}_i$ since otherwise we would have that H_j is a child of $\bar{H}_{i'}$ not H_i in $\bar{\mathcal{H}}$, a contradiction.
2. Suppose $\bar{r}_j \in \bar{T}_i \setminus \hat{T}_i$. It follows that $\bar{r}_j$ is incident to a component $F \subseteq T_{\text{BFS}}$ of $H_{\text{HJ}} \setminus \hat{H}$ which is assigned to i with exactly one vertex v in $V(\hat{T}_i)$. In this case we can let our path from $\bar{r}_j$ to $\hat{T}_i$ be the path in F from $\bar{r}_j$ to v; it remains only to argue that $v \neq \bar{r}_i$. However, if v were equal to $\bar{r}_i$ then we would have that F shares a vertex with $\bar{H}_{i'}$ and, even stronger, F shares a vertex with $\hat{H}_{i'}$. It follows that F would be assigned to $\bar{H}_{i'}$, not $\bar{H}_i$, a contradiction.

Next, we claim that there is a path P_2 from r to $\bar{r}_j$ such that $P_2 \cap V(\bar{H}_i) = \{\bar{r}_j\}$. Let P'_2 an arbitrary path contained in $\bar{H}_j$ from $\bar{r}'_j$ to $\bar{r}_j$. Since $\bar{H}_i$ and $\bar{H}_j$'s vertex sets only intersect on $\bar{r}_j$, it follows that $P'_2 \cap V(\bar{H}_i) = \{\bar{r}_j\}$. Next we form P_2 by concatenating P'_2 with $T_{\text{BFS}}(l(C_j), \bar{r}'_j)$. We claim that $T_{\text{BFS}}(l(C_j), \bar{r}'_j) \cap V(\bar{H}_i) = \emptyset$. Suppose for the sake of contradiction that there is

some $x \in T_{\text{BFS}}(l(C_j), \bar{r}'_j) \cap V(\bar{H}_i)$. If $x \in \bar{T}_i$ then $\bar{r}_i$ is an ancestor of both $\bar{r}_j$ and $\bar{r}'_j$ and so $l(C_j) \prec l(C_i)$, contradicting our assumption that $\bar{r}_i \prec l(C_j)$. On the other hand, if $x \in \bar{T}'_i$ then we have that $\bar{r}'_i$ is an ancestor of $\bar{r}'_j$; since $\bar{r}_j \in \bar{T}_i$, it follows that $C_i = C_j$, contradicting our assumption that C_i and C_j are distinct LCA-equivalence classes. Thus, P_2 is indeed from r to $\bar{r}_j$ and satisfies $P_2 \cap V(\bar{H}_i) = \{\bar{r}_j\}$.

We conclude by observing that the above gives us a clawed cycle with C as the cycle with paths $T_{\text{BFS}}(r, u)$, $T_{\text{BFS}}(r, u')$ and $P_1 \oplus P_2$, a contradiction. □

We now demonstrate that the tree structure of $\bar{\mathcal{H}}$ reflects the structure of our LCA-equivalence classes. As the previous lemma handled the left child case, most of this proof will be focused on the right child case.

Lemma 182. *$\bar{H}_j \preceq_{\bar{\mathcal{H}}} \bar{H}_i$ implies $C_j \preceq C_i$ for every i, j.*

Proof. It suffices to show that if $\bar{H}_j$ is a child of $\bar{H}_i$ in $\bar{\mathcal{H}}$ then $C_j \preceq C_i$; thus, let $\bar{H}_i$ and $\bar{H}_j$ be an arbitrary parent-child pair in $\bar{\mathcal{H}}$. Since $\bar{\mathcal{H}}$ is a rooted forest of hammocks by Theorem 178 we know that $V(\bar{H}_i) \cap V(\bar{H}_j) = \bar{r}_j$ and since $l(C_i)$ and $l(C_j)$ are ancestors of every vertex in $\bar{H}_i$ and $\bar{H}_j$ respectively, we know that $l(C_i)$ and $l(C_j)$ are both ancestors of $\bar{r}_j$ and therefore either $C_i \prec C_j$ or $C_j \preceq C_i$.

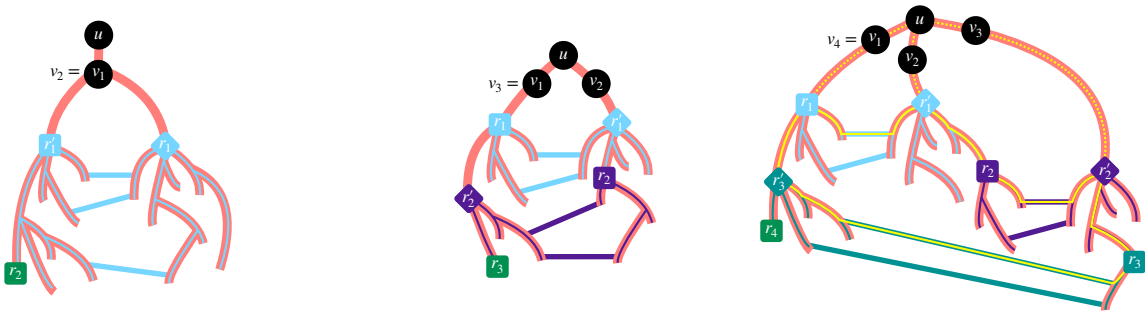
Suppose for the sake of contradiction that $C_i \prec C_j$ and suppose minimality of this counter-example; in particular, suppose that there are no ancestors $\bar{H}_{i'}$ and $\bar{H}_{j'}$ of $\bar{H}_j$ in $\bar{\mathcal{H}}$ where $\bar{H}_{i'}$ is the parent of $\bar{H}_{j'}$ but $l(C_{i'}) \prec l(C_{j'})$. It follows that for every pair of parent-child pairs $\bar{H}_{i'}$ and $\bar{H}_{j'}$ which are ancestors of $\bar{H}_j$ in $\bar{\mathcal{H}}$ we know that $C_{j'} \preceq C_{i'}$. Moreover, letting $\bar{H}_k$ be the root hammock of the tree of hammocks containing $\bar{H}_i$ and $\bar{H}_j$ in $\bar{\mathcal{H}}$, it follows that $C_{i'} \preceq C_k$ for any ancestor $\bar{H}_{i'}$ of $\bar{H}_j$. Additionally, since we chose $\bar{H}_k$ to maximize $h(C_k)$ and $C_j \prec C_i$, there must be some pair of ancestors $\bar{H}_{j'}$ and $\bar{H}_{i'}$ of $\bar{H}_j$ where $C_{j'} \prec C_{i'}$ (since otherwise we would have $C_k \prec C_j$).

It follows that there is a sequence of hammocks $\bar{H}_0, \bar{H}_1, \bar{H}_2, \bar{H}_3, \ldots, \bar{H}_\alpha, \bar{H}_{\alpha+1}$ where $l(C_l) = l(C_{l'})$ for $l, l' \in [\alpha]$ and $\bar{H}_{l+1}$ is the child of $\bar{H}_l$ in $\bar{\mathcal{H}}$ for $0 \leq l \leq \alpha$ where $\alpha \geq 1$ but $l(C_1) = \ldots = l(C_\alpha) \prec l(C_0), l(C_{\alpha+1})$; here, we have slightly abused notation and relabeled $\bar{H}_{i'}$, $\bar{H}_{j'}$, $\bar{H}_i$ and $\bar{H}_j$ to $\bar{H}_0$, $\bar{H}_1$, $\bar{H}_\alpha$ and $\bar{H}_{\alpha+1}$ respectively. We let $u := l(C_l)$ for $l \in [\alpha]$ and let v_l be the child of u in T_{BFS} whose subtree contains r_l Moreover, we claim that by Theorem 181 we know that for each $l \in [\alpha]$ we must have that $\bar{H}_{l+1}$ is a *right* child of $\bar{H}_l$: to see why, notice that if $\bar{H}_{l+1}$ is a left child of $\bar{H}_l$ then Theorem 181 tells us that $l(C_{l+1}) \in T_l$ and so $l(C_{l+1}) \preceq r_l \prec u$; if $l < \alpha$ this contradicts our assumption that $l(C_{l+1}) = u$ and if $l = \alpha$ this contradicts our assumption that $l(C_\alpha) = u \prec l(C_{\alpha+1})$.

We will use the existence of such a sequence of right children to contradict Theorem 162.

First, we claim that v_1 and $v_{\alpha+1}$ are connected in $T_{\text{BFS}}(u) \setminus \{u\}$. To see why, first notice that the graph induced by the union of $\bar{H}_1, \bar{H}_2, \ldots, \bar{H}_\alpha$ is connected by virtue of each hammock in this sequence being the parent of the next hammock in this sequence. v_1 and $v_{\alpha+1}$ are therefore connected in $T_{\text{BFS}}(u) \setminus \{u\}$ since we also know that r_1 and $r_{\alpha+1}$ are both contained in this graph and descendants of v_1 and $v_{\alpha+1}$ respectively.

Notice that $\bar{H}_0$ and $\bar{H}_1$ intersect at $r_1 \preceq v_1$ and $\bar{H}_\alpha$ and $\bar{H}_{\alpha+1}$ intersect at $r_{\alpha+1} \preceq v_{\alpha+1}$. Thus, since $\bar{H}_0 \prec \bar{H}_1$ and $\bar{H}_{\alpha+1} \prec \bar{H}_\alpha$, it follows by Theorem 179 that $\{u, v_1\}$ and $\{u, v_{\alpha+1}\}$ are connected below.



(a) $\gamma = 1$: $l(C_1) \prec u$ **(b)** $\gamma = 2$: $C_1 = C_2$ **(c)** $\gamma \geq 3$: a clawed cycle

Figure 6.13: Each of the three contradictions we arrive at in the proof of Theorem 182 based on the value of γ. In (c) we highlight the cycle of our clawed cycle in solid yellow and each of its paths in dotted yellow.

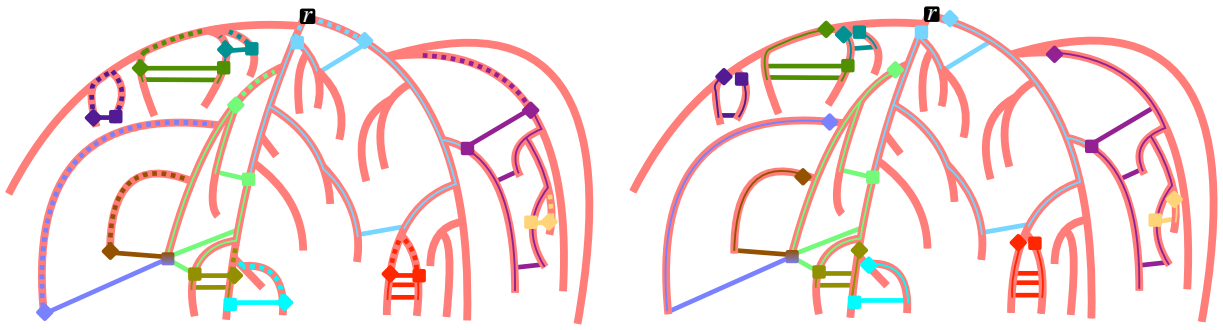
Thus, if we can show that $v_1 \neq v_{\alpha+1}$ then we will have contradicted Theorem 162. Suppose for the sake of contradiction that $v_1 = v_{\alpha+1}$. Then, there must be some contiguous subsequence $(v_\beta, v_{\beta+1}, \ldots v_{\beta+\gamma})$ of $(v_1, \ldots, v_{\alpha+1})$ for $\gamma \geq 1$ where $v_\beta = v_{\beta+\gamma}$ but $v_{\beta+x} \neq v_{\beta+y}$ for all $x, y < \gamma$ where $y \neq x$. To simplify notation we assume that $\beta = 1$. We will show that all choices of γ contradict the assumption that $v_1 = v_{\gamma+1}$ and so it must be the case that $v_1 \neq v_{\alpha+1}$. We illustrate each of the following contradictions in Figures 6.13a, 6.13b and 6.13c.

1. Suppose $\gamma = 1$ (i.e. there is a single hammock $\bar{H}_1$ we are considering with an LCA of u). Then we would have that r_1 and r'_1 are both descendants of v_1 and $v_{\gamma+1}$, meaning $l(C_1) \preceq v_1 = v_{\gamma+1} \prec u$, a contradiction to the fact that $l(C_1) = u$.
2. Suppose $\gamma = 2$. Since $\bar{H}_2$ is a right child of $\bar{H}_1$, we know that $r_2 \preceq v_2$ *and* $r'_1 \preceq v_2$. Moreover, we know that $r_3 \preceq v_3$ and $r_3 \in T'_2$ and so $r'_2 \preceq v_3 = v_1$. Summarizing, we have $r_1, r'_2 \preceq v_1$ and $r'_1, r_2 \preceq v_2$; however since $v_1, v_2 \neq u$, it follows that $C_1 = C_2$, contradicting our assumption that they are distinct LCA-equivalence classes.
3. Suppose $\gamma \geq 3$. Let P be an arbitrary path connecting r_1 and $r_{\gamma+1}$ in the graph induced by $\bar{H}_1, \bar{H}_2, \ldots, \bar{H}_\gamma$. By our assumption that $\bar{H}_{l+1}$ is a right child of $\bar{H}_l$ for $l \in [\gamma]$, we know that such a path uses at least one edge in C_l for $l \in [\gamma]$. Moreover, we may assume WLOG that P uses exactly one edge from C_l for $l \in [\gamma]$: if x_l and y_l are the first and last vertices that P visits in C_l then by definition of a hammock there is a path from x_l to y_l which uses exactly one edge of C_l; we assume that P's subpath restricted to C_l is this path. Next, since we have assumed that v_1 and $v_{\gamma+1}$ are equal, we know that both r_1 and $r_{\gamma+1}$ lie in $T_{\text{BFS}}(v_1)$. By our choice of γ, we have that the graph induced by $T_{\text{BFS}}(r_1, r_{\gamma+1}) \cup P$ contains a cycle C with exactly one edge from at least three C_l. However, we can easily form a clawed cycle from such a cycle. In particular, let P_1, P_2 and P_3 be paths to C from u in T_{BFS} via v_1, v_2 and v_3 respectively. Then C with these paths is a clawed cycle, a contradiction. □

Extending Along LCA Paths

In this section we show that the above mentioned paths are appropriately disjoint from one another as well as from the thus far computed hammocks. We will then use this disjointness to extend $\bar{\mathcal{H}}$ to $\tilde{\mathcal{H}}$. We formally define these paths—which we illustrate in illustrate in Figure 6.14a—as follows.

Definition 183 (LCA Paths P_i, P'_i, $\mathcal{P}_i$ $\mathcal{P}$)**.** *For each i we let $P_i := internal(T_{BFS}(\bar{r}_i, l(C_i)))$ and*



(a) LCA paths. **(b)** Extending $\bar{\mathcal{H}}$ to $\tilde{\mathcal{H}}$ via LCA paths.

Figure 6.14: An illustration of the paths of our hammock decomposition. Each such path dotted and given in a color corresponding to its constituent hammock. On the left we give the paths and on the right we give the result of adding these paths to $\bar{\mathcal{H}}$, resulting in $\tilde{\mathcal{H}}$. Notice that each root hammock has two paths in $\mathcal{P}$.

$P'_i := \text{internal}(T_{BFS}(\bar{r}'_i, l(C_i)))$ be the LCA paths from $\bar{r}_i$ and $\bar{r}'_i$ respectively to $l(C_i)$, excluding the endpoints. We let $\mathcal{P}_i$ be the set which always contains P'_i and which additionally contains P_i if $\bar{H}_i$ is a root hammock in $\bar{\mathcal{H}}$. Lastly, we let $\mathcal{P} := \bigcup_i \mathcal{P}_i$ be the collection of all relevant LCA paths.

Given the above paths we now describe how we construct $\tilde{\mathcal{H}}$ from $\bar{\mathcal{H}}$ be extending our hammocks along these paths. Formally, we let $\tilde{H}_i$ be the graph induced in G by $V(\bar{H}_i)$ along with all vertices in paths in $\mathcal{P}_i$. We also let the root hammocks of $\tilde{\mathcal{H}}$ be the same as the root hammocks of $\bar{\mathcal{H}}$. We illustrate this in Figure 6.14b.

To show that this yields an LCA-respecting tree of hammocks we first observe that all of the above paths are appropriately disjoint.

Lemma 184. *For each $P \in \mathcal{P}$ and i we have $V(P) \cap V(\bar{H}_i) = \emptyset$. Similarly, for distinct $P, P' \in \mathcal{P}$ we have $V(P) \cap V(P') = \emptyset$.*

Proof. First, let us show $V(P) \cap V(\bar{H}_i) = \emptyset$. Let $P \in \mathcal{P}_j$; we assume WLOG that $P = P'_j$. Suppose that there is some vertex $v \in V(P'_j)$ which is contained in $V(\bar{H}_i)$ and let v be the lowest such vertex in P'_j. It follows that the path between $\bar{r}'_j$ and v must be a hammock joining path between $\bar{H}_i$ and $\bar{H}_j$.

Continuing, let e be the parent edge of $\bar{r}'_j$ in T_{BFS}; e is in the above hammock-joining path and so is contained in some hammock $\bar{H}_l$ which intersects $\bar{H}_j$ on $\bar{r}'_j$. Since $\bar{H}_l$ and $\bar{H}_j$ intersect on $\bar{r}'_j$, $\bar{H}_l$ must be either the parent, sibling or child of $\bar{H}_l$ in $\bar{\mathcal{H}}$ as per Theorem 154. However, $\bar{H}_l$ cannot be the parent or sibling of $\bar{H}_j$ since it intersects $\bar{H}_j$ on $\bar{T}'_j$; thus $\bar{H}_l$ is the child of $\bar{H}_j$. We claim that $e \in \hat{H}_l$; indeed this is immediate from the fact that e is incident to $\bar{r}'_j$ and since $\bar{H}_j$ is the parent of $\bar{H}_l$, e would have been included in $\bar{H}_j$, not $\bar{H}_l$, if it weren't in $\hat{H}_l$. However, since v is the parent of $\bar{r}'_j$ and $\bar{r}'_j \in \bar{H}_l$, we know that $\bar{r}'_j \neq \hat{r}_l, \hat{r}'_l$.

Thus, applying Theorem 177 we know that there is a hammock-fundamental cycle $C \subseteq \hat{H}_l$ containing $\bar{r}'_j$ where $\bar{r}'_j$ is not the highest vertex in either $V(C) \cap \bar{T}_l$ or $V(C) \cap \bar{T}'_l$; let these highest vertices be v_l and v'_l and notice that to arrive at a contradiction it suffices to argue that there is a path P from r to $\bar{r}'_j$ which is internally vertex disjoint from $V(\bar{H}_l)$ since C along with paths $T_{\text{BFS}}(r, v_l)$, $T_{\text{BFS}}(r, v'_l)$ and P would then form a clawed cycle. Let P be the concatenation of $T_{\text{BFS}}(r, \bar{r}_j)$ with an arbitrary path in $\bar{H}_j$ connecting $\bar{r}_j$ and $\bar{r}'_j$ and e. $\bar{r}'_j$ is the only vertex

which the latter shares with $\bar{H}_l$ so it suffices to show that $T_{\text{BFS}}(r, \bar{r}_j)$ shares no vertices with $\bar{H}_l$. Suppose for the sake of contradiction that $x \in T_{\text{BFS}}(r, \bar{r}_j) \cap V(\bar{H}_l)$. Then, if x and $\bar{r}'_j$ were in the same hammock trees of $\bar{H}_l$ then we would have $l(C_j) \prec l(C_l)$, contradicting $l(C_l) \preceq l(C_j)$ by Theorem 182 and the fact that $\bar{H}_l$ is a child of $\bar{H}_j$. On the other hand, if x and $\bar{r}'_j$ were in different hammock trees of $\bar{H}_l$ then we would have that $C_j = C_l$, contradicting the fact that C_j and C_l are distinct. Thus, we conclude that $V(P) \cap V(\bar{H}_i) = \emptyset$.

Next, let us show $V(P) \cap V(P') = \emptyset$ for distinct $P, P' \in \mathcal{P}$. We may assume that it is not the case that $P, P' \in \mathcal{P}_i$ for some i since then P and P' are vertex-disjoint by construction. Thus, WLOG we assume $P = P'_i$ and $P' = P'_j$ for $i \neq j$. Suppose for the sake of contradiction that P'_i and P'_j share a vertex x. By definition of P'_i and P'_j we know that $x \neq \bar{r}'_i, \bar{r}'_j$. Moreover, since in the first part of this proof we showed that $V(P_i), V(P_j) \cap V(\bar{H}) = \emptyset$, it follows that $T_{\text{BFS}}(\hat{r}'_i, \bar{r}'_i) \oplus T_{\text{BFS}}(\bar{r}'_i, x) \oplus T_{\text{BFS}}(x, \bar{r}'_j) \oplus T_{\text{BFS}}(\bar{r}'_j, \hat{r}'_j)$ is a hammock-joining path (Theorem 169) and so would have been included in $\bar{H}$. However, then x would have been included in $\bar{H}$ (by definition of $\bar{H}$). But, this contradicts the fact that $V(P'_i), V(P'_j) \cap V(\bar{H}) = \emptyset$. □

We conclude that our forest of hammocks is indeed LCA-respecting (Theorem 155).

Lemma 185. *$\tilde{\mathcal{H}}$ is an LCA-respecting rooted forest of hammocks where for each root hammock $\tilde{H}_k \in \tilde{\mathcal{H}}$ we have LCA$(\tilde{r}_k, \tilde{r}'_k)$ is the parent of $\tilde{r}_k$ and $\tilde{r}'_k$ in T_{BFS}.*

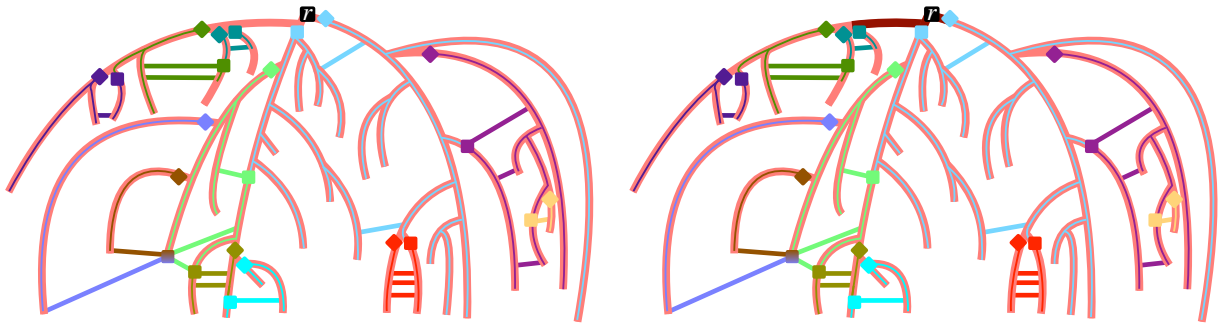
Proof. It is easy to see by the disjointness of the paths in $\mathcal{P}$ that adding all vertices in $\mathcal{P}_i$ (along with all induced edges) to $\bar{H}_i$ indeed results in a hammock. To see that the result is a rooted forest of hammocks we need only verify that any cycle C continues to only be incident to at most 1 hammock. However, notice that the disjointness properties of our paths guarantee that no cycle can use an edge incident to a vertex in a path in $\mathcal{P}$; thus, since $\bar{\mathcal{H}}$ was a forest by Theorem 178, so too is $\tilde{\mathcal{H}}$.

Lastly, to see that the resulting rooted forest of hammocks is LCA-respecting notice that $\tilde{H}_j$ is a child of $\tilde{H}_i$ in $\tilde{\mathcal{H}}$ iff $\bar{H}_i$ is a parent of $\bar{H}_j$ in $\bar{\mathcal{H}}$ and so the first condition of LCA-respecting—$V(\tilde{H}_i) \cap V(\tilde{H}_j) = \{\tilde{r}_j\}$ whenever $\tilde{H}_i$ is a parent of $\tilde{H}_j$ in $\tilde{\mathcal{H}}$—still holds by Theorem 178. The second condition—the parent of $\tilde{r}'_j$ in T_{BFS} is LCA$(\tilde{r}_j, \tilde{r}'_j)$ and LCA$(\tilde{r}_j, \tilde{r}'_j) \in \tilde{H}_i \cup \{\text{LCA}(r_i, r'_i)\}$ where $\tilde{H}_i$ is the parent of $\tilde{H}_j$—holds by definition of the paths in $\mathcal{P}$ and Theorem 182. The condition on each hammock root similarly follows. □

6.8.4 Extending $\tilde{H}$ to $\mathcal{H}$ by Adding Dangling Subtrees

The last step of our hammock decomposition involves adding subtrees of T_{BFS} which consist of unassigned nodes to hammocks. We do this to ensure that our hammock decompositions indeed partitions all edges of the input series-parallel graph.

Formally, we construct $\mathcal{H}$ from $\tilde{\mathcal{H}}$ as follows. Let E_p be the collection of the parent edges in T_{BFS} of $\tilde{r}'_i$ for every hammock H_i (where as usual we use $\tilde{r}_i$ and $\tilde{r}'_i$ to denote the hammock roots of $\tilde{H}_i$). We let T_0 be the connected component of $T_{\text{BFS}} \setminus (E_p \cup E(G[\tilde{\mathcal{H}}]))$ which contains r. For each connected component (i.e. dangling tree) $T \neq T_0$ of $T_{\text{BFS}} \setminus (E_p \cup E(G[\tilde{\mathcal{H}}]))$ we add T to an arbitrary hammock which contains high(T). As a reminder, high(T) gives the vertex in T which is highest in T_{BFS}. Lastly we designate each hammock H_k in $\mathcal{H}$ such that $r_k \in V(T_0)$ and $\tilde{H}_k$ was designated a root hammock in $\tilde{\mathcal{H}}$ as a root of $\mathcal{H}$. We illustrate the result of adding the dangling trees in Figure 6.15a and the final hammock decomposition in Figure 6.15b.



(a) Adding dangling trees.

(b) Final hammock decomposition.

Figure 6.15: The result of adding the dangling trees to each of our hammocks and the final hammock decomposition. On the right we give T_0 in dark red.

We proceed to show that this indeed results in a hammock decomposition (Theorem 156). We begin by observing that $\mathcal{H}$ is an appropriate forest of hammocks.

Lemma 186. *$\mathcal{H}$ is an LCA-respecting rooted forest of hammocks with base tree T_0.*

Proof. We first observe that each H_i is indeed a hammock: since we only add a dangling tree T to $\tilde{H}_i$ if high(T) is contained in $\tilde{H}_i$ and so the roots of H_i are unrelated. Moreover all the H_i must be edge disjoint by the edge disjointness of the $\tilde{\mathcal{H}}$ and how we construct our H_i.

Next, we will argue the cycle property of forests of hammocks; namely that any cycle's edges are fully contained within a single hammock, thereby showing that $\mathcal{H}$ is a forest of hammocks.

First, we observe that if $\{u, v\}$ is an edge in a dangling tree T where $v \prec u$ then there does not exist a cross edge $e_c = \{u_c, v_c\} \in E_c$ such that $u_c \in T_{\text{BFS}}(v)$ but $v_c \notin T_{\text{BFS}}(u)$. To see why this is the case suppose that such an e_c existed and belonged to hammock $\tilde{H}_i \in \tilde{\mathcal{H}}$ and recall that by Theorem 185 $\tilde{\mathcal{H}}$ is an LCA-respecting rooted forest of hammocks. By definition of e_c we know that $u \prec \text{LCA}(\tilde{r}_i, \tilde{r}'_i)$ which is to say that $\text{LCA}(\tilde{r}_i, \tilde{r}'_i)$ lies strictly higher in T_{BFS} than u. But, since $\tilde{\mathcal{H}}$ is LCA-respecting and $\text{LCA}(\tilde{r}_k, \tilde{r}'_k)$ is the parent of $\tilde{r}'_k$ for any root hammock $\tilde{H}_k$ by Theorem 185, it follows that $T_{\text{BFS}}(u_c, \text{LCA}(\tilde{r}_i, \tilde{r}'_i)) \setminus \text{LCA}(\tilde{r}_i, \tilde{r}'_i)$ is contained in $\tilde{H}$ and so $\{u, v\}$ is contained in $\tilde{H}$, contradicting our assumption that it is in a dangling tree.

It follows that if a cycle uses two edges $\{w, u\}$ and $\{u, v\}$ and $\{u, v\}$ lies in a dangling tree then it must be the case that both edges are child edges of u, i.e. $w, v \prec u$.

On the other hand, notice that if we have a cycle which uses $\{w, u\}$ and $\{u, v\}$ then there must be some cross edge between $T_{\text{BFS}}(w)$ and $T_{\text{BFS}}(u)$. Such a cross edge will belong to some hammock $\tilde{H}_i$ for which either w or v is $\tilde{r}'_i$ and so either $\{w, u\}$ or $\{u, v\}$ will be in E_p, contradicting the fact that our cycle cannot use any edges of E_p. Thus, no cycle uses an edge of a dangling tree and so $\mathcal{H}$ is a forest of hammocks.

Next, observe that by construction every connected component of $G[\mathcal{H}]$ will contain exactly one hammock $\tilde{H}_k$ of $\tilde{\mathcal{H}}$ which was designated a root in $\tilde{\mathcal{H}}$ where $\tilde{r}_k = r_k \in T_0$ and so $\mathcal{H}$ is a *rooted* forest of hammocks (i.e. it assigns exactly one root per connected component of $G[\mathcal{H}]$).

It remains to argue that $\mathcal{H}$ is LCA-respecting with base tree T_0. In particular, we must show if H_j is the child of H_i then $V(H_i) \cap V(H_j) = \{r_j\}$ and the parent of r'_j in T_{BFS} is $\text{LCA}(r_j, r'_j)$ and $\text{LCA}(r_j, r'_j) \in V(H_i)$ or $\text{LCA}(r_j, r'_j) = \text{LCA}(r_i, r'_i)$. Notice that if $\tilde{H}_j$ is the child of $\tilde{H}_i$ in $\tilde{\mathcal{H}}$

then we know that H_j will be the child of H_i in $\mathcal{H}$ and so this is immediate from Theorem 185. On the other hand, if H_j is the child of H_i but $\tilde{H}_j$ was not the child of $\tilde{H}_i$ then it must be the case that there is some dangling tree T which connects $\tilde{r}_j$ to $\tilde{H}_i$ which was added to $\tilde{H}_i$. It follows that $\tilde{H}_j$ was a root hammock in $\tilde{\mathcal{H}}$ and so by Theorem 185 we know that $V(H_i) \cap V(H_j) = \{r_j\}$ and the parent of $r'_j = \tilde{r}'_j$ in T_{BFS} is $\text{LCA}(r_j, r'_j)$ and $\text{LCA}(r_j, r'_j) \in V(H_i)$. Additionally we must show that for each root hammock H_k we have $r_k \in V(T_0)$ and the parent of r_k and r'_k in T_{BFS} is $\text{LCA}(r_k, r'_k)$ and $\text{LCA}(r_k, r'_k) \in V(T_0)$. The former is immediate by how we choose our roots and the latter follows from Theorem 185. □

Concluding, we have our hammock decomposition.

Lemma 187. *$(T_0, \mathcal{H}, E_p)$ is a hammock decomposition which can be computed in deterministic poly-time.*

Proof. First, notice that $(T_0, \mathcal{H}, E_p)$ indeed partitions all edges by construction. Moreover, by Theorem 186 we know that $\mathcal{H}$ is an LCA-respecting rooted forest of hammocks with base tree T_0. $G[\mathcal{H}]$ contains all shortest cross edge paths since $G[\bar{\mathcal{H}}]$ contains all shortest cross edge paths by Theorem 178 and $G[\bar{\mathcal{H}}]$ is a subgraph of $G[\mathcal{H}]$.

Lastly, we note that the above hammock decomposition is easily computable in deterministic poly-time. In particular $\hat{\mathcal{H}}$ is trivial to compute, $\bar{\mathcal{H}}$ can be computed by applying one valid assignment, seeing which is the hammock guaranteed to exist by Theorem 174 in $I(T)$ for each T in $H_{\text{HJ}} \setminus \hat{H}$ and then assigning T to this hammock. $\tilde{\mathcal{H}}$ is trivially computable from $\bar{\mathcal{H}}$ and $\mathcal{H}$ is trivially computable from $\tilde{\mathcal{H}}$. □

The above lemma immediately gives Theorem 157.

6.9 Scattering Chops via Hammock Decompositions

In this section, we prove Theorem 134. In particular, we show that every series-parallel graph is $O(1)$-scatter-choppable (Theorem 142) which by Theorem 143 demonstrates that every series-parallel graph is $O(1)$-scatterable (in polynomial time). Combining this fact with Theorem 133 will give our SPR solution. We will demonstrate that every series-parallel graph is $O(1)$-scatter-choppable by using our hammock decompositions and, in particular, Theorem 157.

The concrete lemma we will prove in this section is as follows. Clearly, the lemma below combined with Theorem 143 proves Theorem 134.

Lemma 188. *Consider a graph G with a hammock decomposition $\{H_j\}$. Then, there exists a c-fuzzy, $O(1)$-scattering, Δ-chop of G with respect ot an arbitrary root $r \in V$.*

Our c-fuzzy, $O(1)$-scattering, Δ-chop construction is as follows; see also Figure 6.16.

1. Compute the Δ-chops $A_i = \{v \in V : (i-1)\Delta \leq d(r, v) < i\Delta\}$, and initialize $A'_i = A_i$ for all i
2. For each hammock H_j, independently in parallel:
   (a) For each A_i with $V(H_j) \cap A_i \neq \emptyset$, independently in parallel:
      i. If $r_j \in A_i$ and $d(r, r_j) \geq i\Delta - c\Delta$, then move all of $\{v \in V(H'_j) \cap A_i : i\Delta - c\Delta \leq d(r, v) < i\Delta\}$ from A'_i to A'_{i+1}

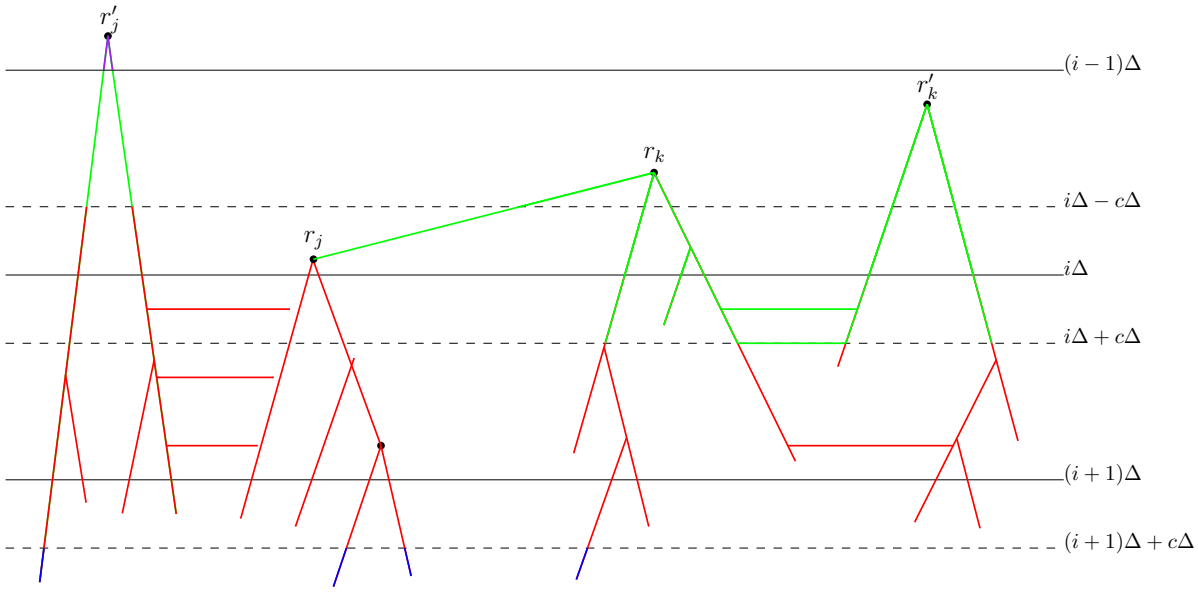


Figure 6.16: Construction of a c-fuzzy, $O(1)$-scattering, Δ-chop. There are two hammocks in the picture, H_j and H_k.

ii. If $d(r, r_j) < i\Delta - c\Delta$, then move all $\{v \in V(H'_j) \cap A_i : d(r, v) \leq (i-1)\Delta + c\Delta\}$ from A'_i to A'_{i-1}

In other words, $\mathcal{A} = \{A_i\}$ are the original Δ-chops and $\mathcal{A}' = \{A'_i\}$ are the sets after the modifications above. All steps can be performed independently in parallel because $c = 1/3$ means that there is no interference between different A_i.

3. For the base tree T_0, treat it as a subgraph H_j with root $r_j = r$ (the root of T_{BFS}), and apply step 2(a) to it.

We begin with the following auxiliary lemmas:

Lemma 189. *Let G be a series-parallel graph, and consider a (unique) shortest u–v path in G that is contained in hammock H_i. Then, P contains at most 2 cross edges in H_i.*

Proof. Assume for contradiction that P contains at least three cross edges. Then, there exist u_1, u_2, u_3 on one tree of the hammock and v_1, v_2, v_3 on the other tree such that the path P contains as a subpath the following edges/paths in order (see Figure 6.17): the edge (u, v), the v_1–v_2 path in T_{BFS}, the edge (v_2, u_2), the u_2–u_3 path in T_{BFS}, and the edge (u_3, v_3). We first claim that it is impossible for the following two conditions to hold simultaneously, since they would imply that G contains a K_4 minor:

1. $\text{LCA}(v_1, v_2)$ is a descendant of $\text{LCA}(v_1, v_2, v_3)$ (the least common ancestor of v_1, v_2, v_3), and
2. $\text{LCA}(u_2, u_3)$ is a descendant of $\text{LCA}(u_1, u_2, u_3)$.

Suppose for contradiction that both of these conditions hold. For ease of notation, define $u_{23} = \text{LCA}(u_2, u_3)$, $u_{123} = \text{LCA}(u_1, u_2, u_3)$, $v_{12} = \text{LCA}(v_1, v_2)$, $v_{123} = \text{LCA}(v_1, v_2, v_3)$. Then, we can find a clawed cycle as follows (see Figure 6.17). Take the cycle composed of edge (u_1, v_1), the v_1–v_2 path in T_{BFS}, the edge (v_2, u_2), and the u_2–u_1 path in T_{BFS}, which contains u_{23} and u_{123} as distinct vertices. Take the claw rooted at v_{123} with following branches: the v_{123}–u_{123} path in T_{BFS},

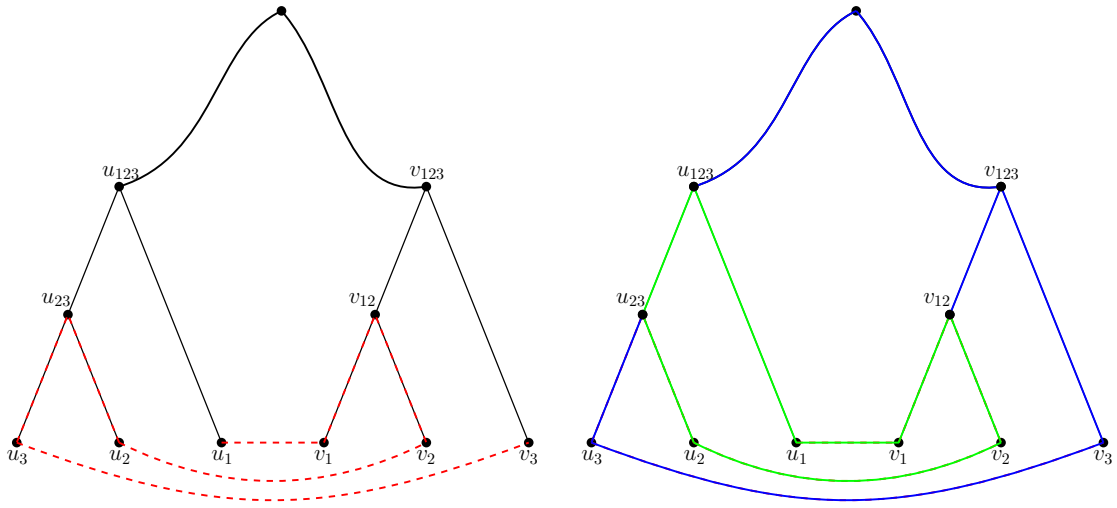


Figure 6.17: Setting for the proof of Theorem 189. The path P is the red, dotted path in the left. The green cycle and the blue claw on the right together form a clawed cycle.

the v_{123}–v_{12} path in T_{BFS}, and the path composed of the v_{123}–v_3 path in T_{BFS}, the edge (v_3, u_3), and the u_3–u_{23} path in T_{BFS}. This clawed cycle identifies a K_4, contradicting the assumption that G is series-parallel.

Therefore, one of the two conditions is false. Suppose first that (1) is false, which means that $v_{12} = v_{123}$. Replace the segment of P from v_{12} to v_3 with the v_{123}–v_3 path in T_{BFS}, which is a shortest v_{123}–v_3 path, so the length of the path cannot increase after the replacement. It follows that the replacement path is also a shortest u–v path, contradicting the assumption that the shortest path is unique.

If (2) is false, then we can similarly replace the segment of P from u_{23} to u_3 with the u_{123}–u_3 path in T_{BFS}, □

Lemma 190. *For each monotone path $P \in T_{BFS}$ of length at most $c\Delta$, the path has $O(1)$ edges cut by $\mathcal{A}'$.*

Proof. First, suppose that the path P is disjoint from the base tree T_0. We handle the base tree T_0 at the end of the proof.

Suppose that for some i, the path P has an edge (u, v) cut by $\mathcal{A}'$ that satisfies $d(r, u), d(r, v) \in [i\Delta, i\Delta + c\Delta]$. The only way for this to happen is if u and v belong to two different hammocks H_j, H_ℓ, and step (a)ii. was applied to one hammock (say H_j) but not the other. In that case, we have $r_j \notin A_j$, so $d(r, r_j) < i\Delta$. Let v' be the first vertex on P after exiting H_j. Then, we must have $d(r, v') < d(r, r_j)$ regardless of whether P exits H_j via r_j or via $\text{LCA}(r_j, r'_j)$. It follows that $d(r, v') < d(r, r_j) < i\Delta$, and since P has length at most $c\Delta$ for $c = 1/3$, it will never again visit a vertex whose distance from r is in $[i'\Delta, i'\Delta + c\Delta]$ for some integer i'.

Suppose next that P has an edge (u, v) cut by $\mathcal{A}'$ that satisfies $d(r, u), d(r, v) \in [i\Delta - c\Delta, i\Delta)$. The only way for this to happen is if u and v belong to two different hammocks H_j, H_ℓ, and step (a)i. was applied to one hammock (say H_j) but not the other. In that case, we have $d(r, r_\ell) < i\Delta - c\Delta$. By the same argument from before, P must exit H_ℓ at some vertex v' satisfying $d(r, v') < d(r, r_\ell) < i\Delta - c\Delta$, and it will never again visit a vertex whose distance from r is in $[i'\Delta - c\Delta, i'\Delta)$ for some integer i'.

Finally, it is easy to see by steps (a)i. and (a)ii. that P cannot have any edge (u, v) cut by $\mathcal{A}'$ that satisfy $d(r, u), d(r, v) \in (i\Delta + c\Delta, (i+1)\Delta - c\Delta)$.

It follows that P cuts at most one edge in each of the ranges $\bigcup_{i\in\mathbb{Z}}[i\Delta, i\Delta + c\Delta]$, $\bigcup_{i\in\mathbb{Z}}[i\Delta - c\Delta, i\Delta)$, and $\bigcup_{i\in\mathbb{Z}}(i\Delta + c\Delta, (i+1)\Delta - c\Delta)$, and at most $O(1)$ edges in between the ranges.

Finally, we consider the case when P intersects the base tree T_0. Let $P_0 = P \cap T_0$ be the subpath of P inside T_0, and let $P' = P - P_0$ be the remainder of P_0, the latter of which has $O(1)$ edges cut by $\mathcal{A}'$ from before. Since T_0 is a tree, P_0 cuts at most one edge in T_0 by construction, so in total, $P = P_0 \cup P'$ has $O(1)$ edges cut by $\mathcal{A}'$. □

Lemma 191. *For each shortest cross edge path P of length less than $c\Delta$, the path has at most $O(1)$ edges cut by $\mathcal{A}'$.*

Proof. By property (1) of Theorem 156, the shortest cross edge path P is contained in $G[\mathcal{H}]$, where $\mathcal{H}$ is the set of hammocks. Let H_m be the hammock of lowest depth (in $\mathcal{T}$) that shares an edge with P_m. Remove the subpath $P_m \cap H_m$ from P_m, which splits it further into two paths P_1, P_2, each of which is monotone with respect to the tree structure $\mathcal{T}$ in the following sense: the set of hammocks that share edges with P_1 form a monotone path in the tree $\mathcal{T}$ of hammocks, and the same for P_2. By Theorem 189, $P_m \cap H_m$ is cut at most twice by $\mathcal{A}'$. It suffices to argue that P_1 is cut $O(1)$ times by $\mathcal{A}'$; the argument for P_2 will be symmetric.

Let (x, y) be the first edge on P_1 that is cut by $\mathcal{A}'$. (If none exist, then we are done.) Suppose that $x \in A'_{i+1}$ and $y \in A'_i$, and let H_j be a hammock containing y. Our goal is to show that all edges on P_1 that are cut by $\mathcal{A}'$ must belong to H_j. Assuming this, we are done by Theorem 189.

We have two cases:

1. $x = r_\ell$ is an endpoint for a child hammock H_ℓ of H_j. In this case, regardless of whether step (a)i. is applied, we must have $i\Delta - c\Delta \le d(r, r_\ell) \le i\Delta$. We first claim that P_1 cannot cross between A'_i and A'_{i-1}. Suppose otherwise that it contains an edge (u, v) with $u \in A'_i$ and $v \in A'_{i-1}$. Then, we must have $d(r, v) \le (i-1)\Delta + c\Delta$ (regardless if step (a)ii. was applied). So the path P_1 has length at least $d(r, r_\ell) - d(r, v) \ge \Delta - 2c\Delta = c\Delta$, contradicting the assumption that P_1 has length less than $c\Delta$.

   We now claim that P_1 cannot cross between A'_i and A'_{i+1} in hammock H_j or beyond. In this case, we must have $d(r, r_j) < i\Delta - c\Delta$, since otherwise step (a)i. would add all vertices in $V(H_j) \cap A_i$ to A'_{i+1}. Therefore, in order to cross between A'_i and A'_{i+1} in H_j, the path P_1 must reach a vertex v with $d(r, v) > i\Delta + c\Delta$ since step (a)ii. was applied to A_i. But then the path P_1 has length at least $d(r, v) - d(r, r_\ell) \ge c\Delta$, contradicting the assumption that P_1 has length less than $c\Delta$.

2. Hammock H_j contains both x, y. In this case, we must have $d(r, r_j) < i\Delta - c\Delta$, since otherwise step (a)i. would add all vertices in $V(H_j) \cap A_i$ to A'_{i+1}. The argument that P_1 cannot cross between A'_i and A'_{i-1} is identical. We now claim that P_1 cannot cross between A'_i and A'_{i+1} in any hammock after H_j. We have $d(r, x) \ge i\Delta$ (regardless if step (a)ii. was applied), and in order for P_1 to travel to any hammock after H_j, it must reach either r_j or $\mathrm{LCA}(r_j, r'_j)$. But then it has length at least $d(r, x) - d(r, r_j) \ge \Delta$, a contradiction.

□

With the above lemmas in hand, we finally prove Theorem 188. Consider two vertices u, v of distance at most Δ in G. We want to show that the (unique) shortest u–v path P_0 has $O(1)$ edges

cut by $\mathcal{A}'$. Partition P_0 into $O(1)$ many subpaths of length less than $c\Delta$ each. We will transform each subpath into a different shortest path between the two endpoints, and which has $O(1)$ edges cut by $\mathcal{A}'$.

We further subdivide path P as follows. First, if P contains no cross edges, then it consists of at most two monotone paths, so by Theorem 190, it is cut $O(1)$ times by $\mathcal{A}'$. Therefore, for the rest of the proof, we assume that P has at least one cross edge. Let P_l be the subpath of P from the beginning to just before the first cross edge of P, and let P_r be the subpath that begins just after the last cross edge of P and continues until the end. Let P_m be the remaining subpath after removing P_l, P_r, so that P_l, P_m, P_r partition the edges of P. The subpaths P_l, P_r consist of at most two monotone paths in T_{BFS}, so by Theorem 190, they are each cut $O(1)$ times by $\mathcal{A}'$. The subpath P_m is a cross edge path, so by Theorem 191, it is cut $O(1)$ times.

6.10 Future Work

We conclude with directions for future work. First, we reiterate the main open question in this area as previously stated by many prior works:

1. For fixed h does every K_h-minor-free graph admit an $O(1)$-SPR solution?

The most exciting next step in settling this long-standing open question would be to tackle the planar case. In particular, if one could demonstrate the existence of a forest-like structure similar to hammock decompositions for planar graphs then the techniques introduced in this chapter would solve the planar case. Notably, it is not too hard to see that the reduction of Filtser [83] of $O(1)$-SPR to $O(1)$-scattering partitions works even if every shortest path is only approximately preserved by the scattering partition. In particular, the reduction of Filtser [83] still works if one can only provide a low-diameter partition where for any vertices u and v there is *some* path between u and v with length at most $O(1) \cdot d_G(u, v)$ which is incident to at most $O(1)$ parts. Consequently, the arguments presented in this chapter show that one need only demonstrate the existence of a forest-like structure which *approximates* the distances between cross edges up to a constant to make use of the scattering chops introduced in this work. Thus, an extremely promising next avenue would be to show that every planar graph has a forest-like subgraph which approximately preserves the distances between all cross edges.

6.11 Ear Decompositions from Hammock Decompositions

A classic result of Eppstein [78] shows that a graph is a 2-vertex-connected series-parallel graph if it has a "nested ear decomposition." These nested ear decompositions need not be unique and, in general, may have little to do with the metric induced by the series-parallel graph. In this section we show how to apply our hammock decompositions to find, among all possible nested ear decompositions, a nested ear decomposition with strong properties regarding how the metric and the nested ear decomposition interact.

We now give a series of definitions and theorems which formalize the nested ear decompositions of Khuller [127] and Eppstein [78].

Definition 192 (Open Ear Decompositions)**.** *An ear is a path whose endpoints may coincide. An ear decomposition is a partition of the edges of a graph into (the edges of) ears $P_1, P_2, \ldots$ where*

for each i internal(P_i) is disjoint from all vertices in $P_1, \ldots, P_{i-1}$ and the two endpoints of P_i are contained among the vertices of $P_1, \ldots, P_{i-1}$. An ear decomposition is open if the two endpoints of each P_i for $i \geq 2$ are distinct.

Khuller [127] introduced the notion of a tree ear decomposition and Eppstein [78] strengthened this to the idea of a nested ear decomposition.

Definition 193 (Tree, Nested Ear Decomposition)**.** *A tree ear decomposition is an open ear decomposition where for $i > 1$ we have that P_i has both endpoints in the same P_j. A nested ear decomposition is a tree ear decomposition where for each P_j the collection of ears with both endpoints in P_i form a collection of nested intervals.*

Recall that a graph is 2-vertex connected if the deletion of any one vertex still leaves the graph connected.

Theorem 194 (Eppstein [78])**.** *A 2-vertex-connected graph is series-parallel if and only if it has a nested ear decomposition.*

Concluding, we apply our hammock decompositions to strengthen the result of Eppstein [78] to respect the input metric in the following way. Recall that a shortest cross edge path is a path that starts and ends with a cross edge and is also a shortest path.

Theorem 195. *Let $G = (V, E)$ be a 2-vertex-connected series-parallel graph with unit weights and unique shortest path lengths. Fix a root $r \in V$ and a BFS tree T_{BFS} with cross edges $E_c := E \setminus E(T_{BFS})$. Then there is a collection of edges $E_p \subseteq E(T_{BFS})$ such that G has a nested ear decomposition $P_1, P_2, \ldots$ where*

1. *Each P_i is of the form $P_u \oplus \{u, v\} \oplus P_v$ where P_u and P_v are monotone paths from u and v towards r and $\{u, v\} \in E_c$.*
2. *$|E(P_i) \cap E_p| \leq 1$ and $(V, E \setminus E_p)$ contains all shortest cross edge paths. Furthermore after deleting all ears which contain an edge of E_p any two ears in the same connected component contain cross edges with the same LCA.*

Proof. Let $(T_0, \mathcal{H}, E_p)$ be the hammock decomposition (Theorem 156) as guaranteed by Theorem 157.

We claim that since G is 2-vertex-connected it must be the case that T_0 is a star with center r and $\mathcal{H}$ consists of a single tree of hammocks where r_k and r'_k are children of r. To see why, notice that since G is 2-vertex connected, for any children u and v of r there must be some cross edge with one endpoint in $T_{\text{BFS}}(u)$ and another in $T_{\text{BFS}}(v)$ (since otherwise the deletion of r would separate the graph into 2 connected components). It follows that such a cross edge will belong to a tree of hammocks which contains both u and v as hammock roots. Applying this to all pairs of children of r shows that all children of r must belong in the same hammock, thereby showing T_0 is a star and $\mathcal{H}$ consists of a single tree of hammocks.

Continuing, we can construct the claimed nested ear decomposition as follows. We let D be our nested ear decomposition so far. We will process hammocks in $\mathcal{H}$ in a BFS order according to the parent-child relationships induced between hammocks in $\mathcal{H}$. To process a single hammock H_i we simply find a candidate cross edge $\{u, v\} = e \in D \setminus E(H_i) \cap E_c$ and add to our nested ear decomposition D the ear consisting of the concatenation of P_u, P_v and e where P_u is the path gotten by going from u towards v until hitting a vertex in D and P_v is defined symmetrically. We

always choose as our cross edge $\{u, v\}$ an edge such that there does not exist another cross edge $\{u', v'\}$ in $E(H_i)$ with either $u \prec u'$ or $v \prec v'$.

It is easy to verify that if no candidate cross edge with the above properties exists then there exists a clawed cycle.

The above construction satisfies properties (1) and (2) in the above theorem statement by the properties of our hammock decompositions and so it remains only to argue that the result is a nested ear decomposition.

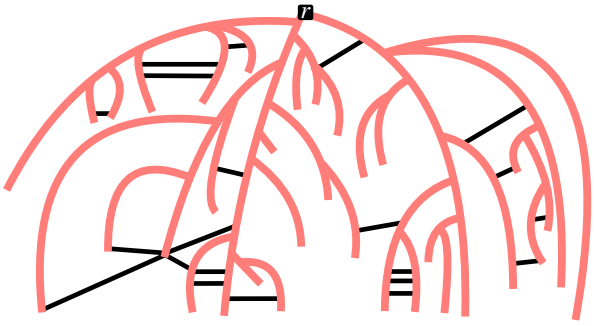
First, we observe that the result of the above process will indeed partition all edges. Every cross edge is included by construction. An edge in T_{BFS} will be added to D the first time any cross edge with an endpoint below it is processed. Since our graph is 2-vertex connected every edge in T_{BFS} has some cross edge with an endpoint below it and so every edge in T_{BFS} will end up in D

Our ear decomposition will be open by definition of our hammock decompositions and, in particular, by the fact that the roots of a hammock are unrelated and by the fact that $\mathcal{H}$ is LCA-respecting.

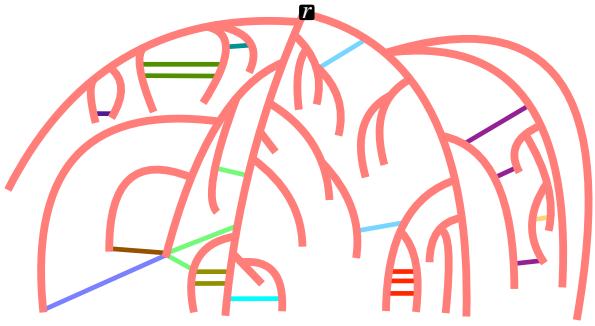
Next, we verify that the result of this process is a tree ear decomposition. Suppose for the sake of contradiction that the result of our construction is not a tree ear decomposition. In particular, suppose P_l is an ear with one endpoint in P_i and another endpoint in P_j. By definition of our construction it must be the case that P_i and P_j are derived from cross edges, say e_i and e_j, which belong to the same hammock. However, it then follows that the hammock fundamental cycle formed by e_i and e_j can be used to construct a clawed cycle, a contradiction. A similar argument shows that if our ear decomposition fails to be nested then we can find a clawed cycle. □

6.12 Hammock Decomposition Construction Figures

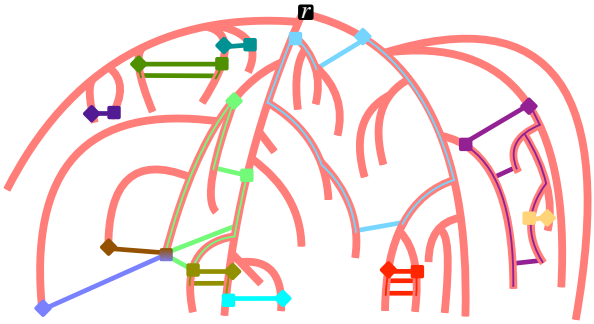
In this section we give all of the illustrations of the construction of our hammock decompositions on a single graph in Figure 6.18.



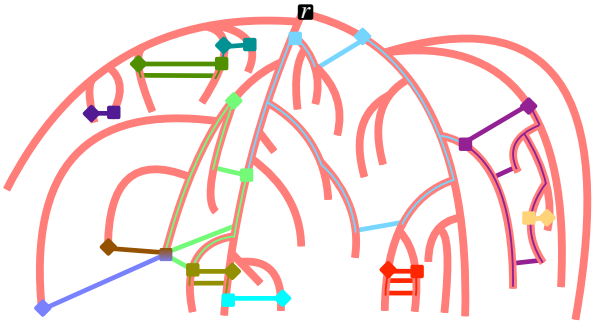
(a) Series parallel graph G.



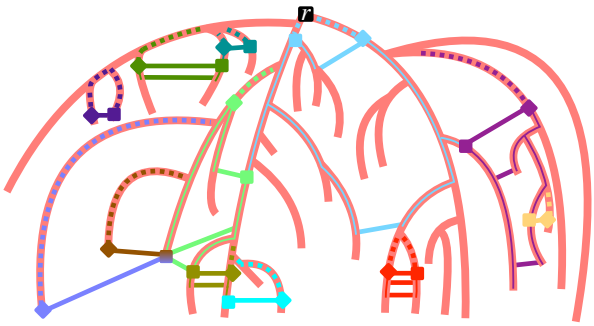
(b) LCA-equivalence classes.



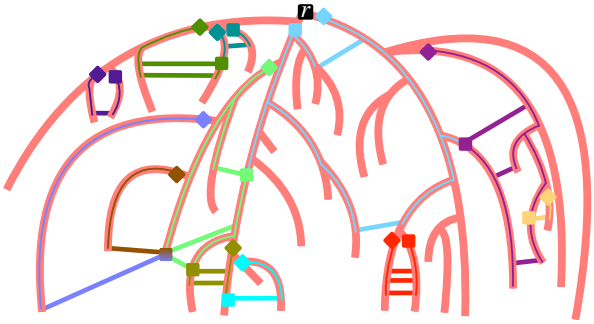
(c) Initial hammocks $\hat{\mathcal{H}}$.



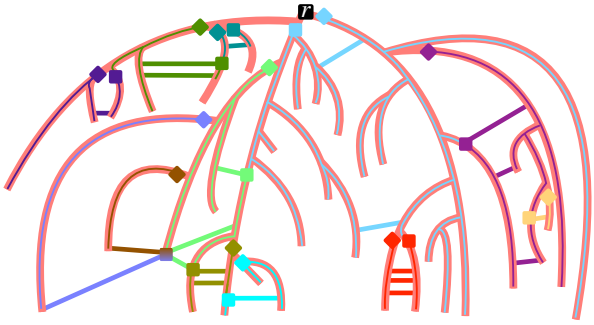
(d) $\bar{\mathcal{H}}$ via hammock-joining paths.



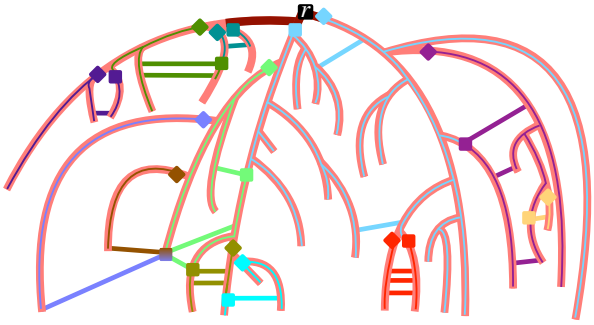
(e) LCA paths $\mathcal{P}$.



(f) $\tilde{\mathcal{H}}$ via LCA paths.



(g) $\mathcal{H}$ via dangling trees.



(h) Final hammock decomposition with T_0.

Figure 6.18: An illustration of the construction of our hammock decomposition on a series-parallel graph.

Part IV

Conclusion

Chapter 7

Conclusion

This thesis presents a variety of new results in how graphs and metrics can be made small or simple while preserving or approximately preserving important properties.

In the first part we introduced two new ways a graph can be represented by tree-like objects—namely copy tree embeddings and hop-constrained tree embeddings—the former approximately preserving the shortest path metric and the latter approximately preserving the hop-constrained distances between nodes. We illustrated the algorithmic utility of these tools by using them as the basis of many new (polynomial-time) approximation algorithms for a variety of network design problems.

In the subsequent part we gave new algorithms in a variety of computational models for computing hop-constrained flows, a crucial primitive in various graph decompositions. We illustrated the use of these algorithms by, among other things, giving the first efficient $(1 - \epsilon)$-approximate distributed algorithms for the bipartite b-matching problem.

In the last past of this thesis we provided new results for vertex sparsification. Specifically, we showed that all series-parallel graphs admit so-called Steiner point removal solutions with only constant distortion.

This thesis is a sampling of a variety of ways in which structure can be extracted from otherwise unstructured objects with a provably reasonable faithfulness to the original object. By any other description it is a study in abstraction and its limits as they pertain to graphs and metrics. The generality of this topic of course means that innumerable exciting future directions abound; throughout this thesis we have tried to enumerate some of them nevertheless.

Bibliography

[1] Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 395–406, 2012. 12

[2] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *Symposium on Foundations of Computer Science (FOCS)*, pages 781–790. IEEE, 2008. 12

[3] Ittai Abraham, Shiri Chechik, Michael Elkin, Arnold Filtser, and Ofer Neiman. Ramsey spanning trees and their applications. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1650–1664. SIAM, 2018. 12

[4] 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*, 48(3):1120–1145, 2019. 48, 49

[5] Ajit Agrawal, Philip Klein, and Ramamoorthi Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995. 41, 61

[6] İbrahim Akgün and Barbaros Ç Tansel. New formulations of the hop-constrained minimum spanning tree problem via miller–tucker–zemlin constraints. *European Journal of Operational Research*, 212(2):263–276, 2011. 41, 44

[7] Noga Alon, Richard M Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995. 2, 11, 12

[8] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms (TALG)*, 2(4):640–660, 2006. 12, 13, 15, 18, 19, 36, 39, 72, 73

[9] Ernst Althaus, Stefan Funke, Sariel Har-Peled, Jochen Könemann, Edgar A Ramos, and Martin Skutella. Approximating k-hop minimum-spanning trees. *Operations Research Letters*, 33(2):115–120, 2005. 41, 44

[10] Kateřina Altmanová, Petr Kolman, and Jan Voborník. On polynomial-time combinatorial algorithms for maximum l-bounded flow. In *Algorithms and Data Structures Symposium (WADS)*, pages 14–27. Springer, 2019. 81, 86

[11] Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected

shortest paths via low hop emulators. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 322–335, 2020. 45, 80, 88

[12] Sanjeev Arora, Michelangelo Grigni, David R Karger, Philip N Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph tsp. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 98, pages 33–41, 1998. 143

[13] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012. 19, 29

[14] Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *Journal of the ACM (JACM)*, 62(5):1–25, 2015. 82

[15] Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *Symposium on Foundations of Computer Science (FOCS)*, pages 542–547. IEEE, 1997. 12, 43, 68

[16] Baruch Awerbuch and David Peleg. Sparse partitions. In *Symposium on Foundations of Computer Science (FOCS)*, pages 503–513. IEEE, 1990. 91

[17] Amy Babay, Michael Dinitz, and Zeyu Zhang. Characterizing demand graphs for (fixed-parameter) shallow-light steiner network. *arXiv preprint arXiv:1802.10566*, 2018. 61

[18] Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Köhler, Petr Kolman, Ondřej Pangrác, Heiko Schilling, and Martin Skutella. Length-bounded cuts and flows. *ACM Transactions on Algorithms (TALG)*, 7(1):1–27, 2010. 80, 86

[19] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *Journal of the ACM (JACM)*, 68(5):1–30, 2021. 82

[20] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Symposium on Foundations of Computer Science (FOCS)*, pages 267–276. IEEE, 2011. 12, 74

[21] Judit Bar-Ilan, Guy Kortsarz, and David Peleg. Generalized submodular cover problems and applications. *Theoretical Computer Science*, 250(1-2):179–200, 2001. 41

[22] Reuven Bar-Yehuda, Keren Censor-Hillel, Mohsen Ghaffari, and Gregory Schwartzman. Distributed approximation of maximum independent set and maximum matching. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 165–174, 2017. 90

[23] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Symposium on Foundations of Computer Science (FOCS)*, pages 184–193. IEEE, 1996. 2, 11, 15

[24] Yair Bartal. Graph decomposition lemmas and their role in metric embedding methods. In *Annual European Symposium on Algorithms (ESA)*, pages 89–97. Springer, 2004. 15

[25] Yair Bartal and Manor Mendel. Multi-embedding and path approximation of metric spaces. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 3, pages 424–433, 2003. 17, 18

[26] Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog (n)-competitive algorithm for metrical task systems. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 711–719, 1997. 12, 15, 74

[27] Yair Bartal, Nova Fandina, and Seeun William Umboh. Online probabilistic metric embedding: a general framework for bypassing inherent bounds. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1538–1557. SIAM, 2020. 12, 13, 36

[28] Amitabh Basu and Anupam Gupta. Steiner point removal in graph metrics. *Unpublished Manuscript, available from http://www. math. ucdavis. edu/˜ abasu/papers/SPR. pdf*, 1:25, 2008. 144, 145, 146

[29] Alok Baveja and Aravind Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25(2):255–280, 2000. 80

[30] Piotr Berman and Chris Coulston. On-line algorithms for steiner tree problems. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 344–353, 1997. 61

[31] Marcin Bienkowski, Bjorn Feldkord, and Pawel Schmidt. A nearly optimal deterministic online algorithm for non-metric facility location. *arXiv preprint*, 2020. 13, 39

[32] Guy E. Blelloch, Yan Gu, and Yihan Sun. Efficient construction of probabilistic tree embeddings. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 80, pages 26:1–26:14, 2017. 12

[33] Andreas Bley, S Mehdi Hashemi, and Mohsen Rezapour. Ip modeling of the survivable hop constrained connected facility location problem. *Electronic Notes in Discrete Mathematics*, 41:463–470, 2013. 44

[34] Hans L Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1-2):1–45, 1998. 148

[35] Quentin Botton, Bernard Fortz, Luis Gouveia, and Michael Poss. Benders decomposition for the hop-constrained survivable network design problem. *INFORMS journal on computing*, 25(1):13–26, 2013. 44

[36] Quentin Botton, Bernard Fortz, and Luis Gouveia. On the hop-constrained survivable network design problem with reliable edges. *Computers & Operations Research*, 64: 159–167, 2015. 44

[37] Sebastian Brandt and Dennis Olivetti. Truly tight-in-δ bounds for bipartite maximal matching and variants. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 69–78, 2020. 82

[38] Bo Brinkman, Adriana Karagiozova, and James R Lee. Vertex cuts, random walks, and dimension reduction in series-parallel graphs. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 621–630, 2007. 145

[39] Andrei Z Broder, Alan M Frieze, and Eli Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM Journal on Computing*, 23(5):976–989, 1994. 80

[40] Niv Buchbinder and Joseph Naor. *The design of competitive online algorithms via a primal-dual approach*. Now Publishers Inc, 2009. 13

[41] Gruia Calinescu, Howard Karloff, and Yuval Rabani. Approximation algorithms for the 0-extension problem. *SIAM Journal on Computing*, 34(2):358–372, 2005. 25

[42] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *Distributed Computing*, 33(3):349–366, 2020. 90

[43] T-H Hubert Chan, Donglin Xia, Goran Konjevod, and Andrea Richa. A tight lower bound for the steiner point removal problem on trees. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 70–81. Springer, 2006. 144, 146

[44] Yi-Jun Chang and Mohsen Ghaffari. Strong-diameter network decomposition. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 273–281, 2021. 91, 92

[45] Yi-Jun Chang and Thatchaphol Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In *Symposium on Foundations of Computer Science (FOCS)*, pages 377–388. IEEE, 2020. 5, 79, 81, 82, 89, 90, 93, 94, 123, 128

[46] Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 114–123, 1998. 12

[47] Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin. Approximating a finite metric by a small number of tree metrics. In *Symposium on Foundations of Computer Science (FOCS)*, pages 379–388. IEEE, 1998. 15, 18, 30

[48] Shiri Chechik and Tianyi Zhang. Dynamic low-stretch spanning trees in subpolynomial time. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 463–475. SIAM, 2020. 12

[49] Chandra Chekuri, Guy Even, and Guy Kortsarz. A greedy approximation algorithm for the group steiner problem. *Discrete Applied Mathematics*, 154(1):15–34, 2006. 12

[50] 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. 145

[51] Chandra Chekuri, Mohammad Taghi Hajiaghayi, Guy Kortsarz, and Mohammad R Salavatipour. Approximation algorithms for non-uniform buy-at-bulk network design. *Symposium on Foundations of Computer Science (FOCS)*, pages 677–686, 2006. 24

[52] Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed steiner network problem. *ACM Transactions on Algorithms (TALG)*, 7(2):1–17, 2011. 13

[53] James Cheng, Yiping Ke, Shumo Chu, and M Tamer Özsu. Efficient core decomposition

in massive networks. In *2011 IEEE 27th International Conference on Data Engineering*, pages 51–62. IEEE, 2011. 1

[54] Yun Kuen Cheung. Steiner point removal—distant terminals don't (really) bother. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1353–1360. SIAM, 2018. 147

[55] Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. Graph minors for preserving terminal distances approximately-lower and upper bounds. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2016. 144, 146

[56] Eden Chlamtáč and Michael Dinitz. Lowest-degree k-spanner: Approximation and hardness. *Theory of Computing*, 12(1):1–29, 2016. 62

[57] Eden Chlamtáč and Petr Kolman. How to cut a ball without separating: Improved approximations for length bounded cut. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. 81

[58] Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation via short cycle decompositions. In *Symposium on Foundations of Computer Science (FOCS)*. SIAM, 202. 92

[59] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167. IEEE, 2020. 79

[60] Edith Cohen. Approximate max-flow on small depth networks. *SIAM Journal on Computing*, 24(3):579–597, 1995. 89, 93, 102, 103, 104, 106

[61] Jérôme De Boeck and Bernard Fortz. Extended formulation for hop constrained distribution network configuration problems. *European Journal of Operational Research*, 265(2):488–502, 2018. 41, 44

[62] Erik D Demaine, MohammadTaghi Hajiaghayi, and Philip N Klein. Node-weighted steiner tree and group steiner tree in planar graphs. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 328–340. Springer, 2009. 12

[63] Laxman Dhulipala. Provably efficient and scalable shared-memory graph processing. *Ph. D. dissertation*, 2020. 1

[64] Ibrahima Diarrassouba, Virginie Gabrel, Ali Ridha Mahjoub, Luís Gouveia, and Pierre Pesneau. Integer programming formulations for the k-edge-connected 3-hop-constrained network design problem. *Networks*, 67(2):148–169, 2016. 44

[65] Ibrahima Diarrassouba, Meriem Mahjoub, A Ridha Mahjoub, and Hande Yaman. k-node-disjoint hop-constrained survivable networks: polyhedral analysis and branch and cut. *Annals of Telecommunications*, 73(1-2):5–28, 2018. 44

[66] Michael Dinitz and Zeyu Zhang. Approximating low-stretch spanners. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 821–840. SIAM, 2016. 61

[67] Michael Dinitz, Guy Kortsarz, and Ran Raz. Min-rep instances with large supergirth and the hardness of approximating basic spanners. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2012. 61

[68] Michael Dinitz, Guy Kortsarz, and Ran Raz. Label cover instances with large girth and the hardness of approximating basic k-spanner. *ACM Transactions on Algorithms (TALG)*, 12 (2):1–16, 2015. 4, 41

[69] Richard J Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965. 148

[70] Michael Elkin and David Peleg. The client-server 2-spanner problem and applications to network design. In *International Colloquium on Structural Information and Communication Complexity*, 1999. 62

[71] Michael Elkin and David Peleg. Strong inapproximability of the basic k-spanner problem. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 636–648. Springer, 2000. 61

[72] Michael Elkin and David Peleg. Approximating k-spanner problems for k¿ 2. *Theoretical Computer Science*, 337(1-3):249–277, 2005. 62

[73] Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008. 12

[74] Yuval Emek and David Peleg. A tight upper bound on the probabilistic embedding of series-parallel graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1827–1841, 2010. 145

[75] Matthias Englert and Harald Räcke. Reordering buffers with logarithmic diameter dependency for trees. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1224–1234. SIAM, 2017. 15

[76] Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. In *Annual ACM Symposium on Theory of Computing (STOC)*, 2007. 15

[77] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Racke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM Journal on Computing*, 43(4):1239–1262, 2014. 146

[78] David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992. 146, 147, 186, 187

[79] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004. 11, 15, 24, 25, 30

[80] Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM Journal on Computing*, 32(6):1403–1422, 2003. 15

[81] Arnold Filtser. Steiner point removal with distortion o(log k), using the noisy-voronoi algorithm. *arXiv preprint arXiv:1808.02800*, 2018. 147

[82] Arnold Filtser. Steiner point removal with distortion o (log k). In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1361–1373. SIAM, 2018. 147

[83] Arnold Filtser. Scattering and sparse partitions, and their applications. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2020. 144, 145, 147, 148, 149, 186

[84] Arnold Filtser. Clan embeddings into trees, and low treewidth graphs. *arXiv preprint arXiv:2101.01146*, 2021. 17, 18, 19

[85] Arnold Filtser. Hop-constrained metric embeddings and their applications. In *Symposium on Foundations of Computer Science (FOCS)*, pages 492–503. IEEE, 2022. 74

[86] Arnold Filtser, Robert Krauthgamer, and Ohad Trabelsi. Relaxed voronoi: A simple framework for terminal-clustering problems. In *SIAM Symposium on Simplicity in Algorithms (SOSA)*, 2018. 144, 146

[87] Manuela Fischer. Improved deterministic distributed matching via rounding. *Distributed Computing*, 33(3):279–291, 2020. 82

[88] Manuela Fischer. *Local Algorithms for Classic Graph Problems*. PhD thesis, ETH Zurich, 2021. 82

[89] Sebastian Forster, Gramoz Goranci, and Monika Henzinger. Dynamic maintanance of low-stretch probabilistic tree embeddings with applications. *arXiv preprint arXiv:2004.10319*, 2020. 12

[90] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. 29

[91] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007. 86

[92] Naveen Garg, Goran Konjevod, and R Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000. 12, 36, 63

[93] Mohsen Ghaffari and Fabian Kuhn. Derandomizing distributed algorithms with small messages: Spanners and dominating set. In *International Symposium on Distributed Computing (DISC)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 92

[94] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 81–90, 2015. 81

[95] Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. *SIAM Journal on Discrete Mathematics*, 34(1):130–162, 2020. 146

[96] Luis Gouveia. Using the miller-tucker-zemlin constraints to formulate a minimal spanning tree problem with hop constraints. *Computers & Operations Research*, 22(9):959–970, 1995. 44

[97] Luis Gouveia. Multicommodity flow models for spanning trees with hop constraints. *European Journal of Operational Research*, 95(1):178–190, 1996. 44

[98] Luis Gouveia and Thomas L Magnanti. Network flow models for designing diameter-constrained minimum-spanning and steiner trees. *Networks: An International Journal*, 41 (3):159–173, 2003. 44

[99] Luis Gouveia and Cristina Requejo. A new lagrangean relaxation approach for the hop-constrained minimum spanning tree problem. *European Journal of Operational Research*, 132(3):539–552, 2001. 44

[100] Xiangyu Guo, Janardhan Kulkarni, Shi Li, and Jiayi Xian. On the facility location problem in online and dynamic models. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. 15

[101] Anupam Gupta. Steiner points in tree metrics don't (really) help. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 682–690, 2001. 143, 144, 146

[102] Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Cuts, trees and l/sub 1/-embeddings of graphs. In *Symposium on Foundations of Computer Science (FOCS)*, pages 399–408. IEEE, 1999. 145

[103] Anupam Gupta, Robert Krauthgamer, and James R Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Symposium on Foundations of Computer Science (FOCS)*, pages 534–543. IEEE, 2003. 48, 49, 145

[104] Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Cuts, trees and l1-embeddings of graphs. *Combinatorica*, 24(2):233–269, 2004. 145, 146

[105] Anupam Gupta, Mohammad T Hajiaghayi, and Harald Räcke. Oblivious network design. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 970–979, 2006. 17, 19, 20, 21, 24, 26, 61, 68

[106] Varun Gupta, Ravishankar Krishnaswamy, and Sai Sandeep. Permutation strikes back: The power of recourse in online metric matching. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2020. 15

[107] Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences*, 67(3): 473–496, 2003. 80, 86, 127

[108] Frank Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975. 143

[109] Bernhard Haeupler, David Wajc, and Goran Zuzic. Network coding gaps for completion

times of multiple unicasts. In *Symposium on Foundations of Computer Science (FOCS)*, pages 494–505. IEEE, 2020. 4, 80

[110] Bernhard Haeupler, D Ellis Hershkowitz, and Thatchaphol Saranurak. Fast algorithms for hop-constrained flows and moving cuts. *arXiv preprint arXiv:2111.01422*, 2021. 5

[111] Bernhard Haeupler, D Ellis Hershkowitz, and Goran Zuzic. Tree embeddings for hop-constrained network design. *Annual ACM Symposium on Theory of Computing (STOC)*, 2021. 17

[112] Bernhard Haeupler, D Ellis Hershkowitz, and Goran Zuzic. Deterministic tree embeddings with copies for algorithms against adaptive adversaries. In *arXiv Preprint*, 2021. 2

[113] Bernhard Haeupler, D Ellis Hershkowitz, and Goran Zuzic. Tree embeddings for hop-constrained network design. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 356–369, 2021. 4, 80, 88

[114] Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 1166–1179, 2021. 80, 81

[115] Bernhard Haeupler, Harald Raecke, and Mohsen Ghaffari. Hop-constrained expander decompositions; oblivious routing, and distributed universal optimality. In *Annual ACM Symposium on Theory of Computing (STOC)*, 2022. 82, 130, 131

[116] Mohammad Taghi Hajiaghayi, Guy Kortsarz, and Mohammad R Salavatipour. Approximating buy-at-bulk and shallow-light k-steiner trees. *Algorithmica*, 53(1):89–103, 2009. 41, 44, 66

[117] D Ellis Hershkowitz and Jason Li. O(1) steiner point removal in series-parallel graphs. *arXiv preprint arXiv:2104.00750*, 2021. 5

[118] Yael Hitron and Merav Parter. General congest compilers against adversarial edges. In *International Symposium on Distributed Computing (DISC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021. 86, 92, 93

[119] Makoto Imase and Bernard M Waxman. Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991. 19, 31

[120] Lior Kamma, Robert Krauthgamer, and Huy L Nguyen. Cutting corners cheaply, or how to remove steiner points. *SIAM Journal on Computing*, 44(4):975–995, 2015. 147

[121] Erez Kantor and David Peleg. Approximate hierarchical facility location and applications to the bounded depth steiner tree and range assignment problems. *Journal of Discrete Algorithms*, 7(3):341–362, 2009. 41, 44

[122] Richard M Karp. A 2k-competitive algorithm for the circle. *Manuscript, August*, 5, 1989. 11

[123] Richard M Karp and Vijaya Ramachandran. A survey of parallel algorithms for shared-memory machines. *survey*, 1989. 84, 101

[124] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *Journal of the ACM (JACM)*, 66(1):1–50, 2018. 82

[125] Maleq Khan, Fabian Kuhn, Dahlia Malkhi, Gopal Pandurangan, and Kunal Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Computing*, 25(3):189–205, 2012. 12

[126] M Reza Khani and Mohammad R Salavatipour. Improved approximations for buy-at-bulk and shallow-light k-steiner trees and (k, 2)-subgraph. In *Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 20–29. Springer, 2011. xix, 41, 44, 66

[127] Samir Khuller. Ear decompositions. *SigAct News*, 20(1):128, 1989. 146, 186, 187

[128] Philip Klein, Serge A Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993. 145, 149

[129] Jon M Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, Massachusetts Institute of Technology, 1996. 80

[130] Jochen Könemann, Asaf Levin, and Amitabh Sinha. Approximating the degree-bounded minimum diameter spanning tree problem. *Algorithmica*, 41(2):117–129, 2005. 41, 44

[131] Goran Konjevod, R Ravi, and F Sibel Salman. On approximating planar metrics by tree metrics. *Information Processing Letters*, 80(4):213–219, 2001. 12, 22, 30

[132] Guy Kortsarz and David Peleg. Approximating shallow-light trees. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 103–110, 1997. 41, 44

[133] Christos Koufogiannakis and Neal E Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. In *International Symposium on Distributed Computing*, pages 221–238. Springer, 2009. 82

[134] Ioannis Koutis, Gary L Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *Symposium on Foundations of Computer Science (FOCS)*, pages 590–598. IEEE, 2011. 12

[135] Robert Krauthgamer and Havana Inbal Rika. Refined vertex sparsifiers of planar graphs. *SIAM Journal on Discrete Mathematics*, 34(1):101–129, 2020. 146

[136] Robert Krauthgamer, Huy L Nguyen, and Tamar Zondiner. Preserving terminal distances using minors. *SIAM Journal on Discrete Mathematics*, 28(1):127–141, 2014. 144, 146

[137] James R. Lee and Cyrus Rashtchian. A simpler proof of the kpr theorem. https://tcsmath.wordpress.com/tag/klein-plotkin-rao/, January 2012. 154

[138] Markus Leitner. Layered graph models and exact algorithms for the generalized hop-constrained minimum spanning tree problem. *Computers & Operations Research*, 65:1–18, 2016. 44

[139] Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13 (4):441–454, 1993. 90

[140] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 129–136, 2008. 79, 89, 93, 94

[141] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM journal on computing*, 15(4):1036–1053, 1986. 94

[142] Madhav V Marathe, Ramamoorthi Ravi, Ravi Sundaram, SS Ravi, Daniel J Rosenkrantz, and Harry B Hunt III. Bicriteria network design problems. *Journal of algorithms*, 28(1): 142–171, 1998. 41, 44

[143] Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. In *Symposium on Foundations of Computer Science (FOCS)*, pages 109–118. IEEE, 2006. 12, 19

[144] Gary L Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 196–203, 2013. 90

[145] Viswanath Nagarajan and Ramamoorthi Ravi. Approximation algorithms for requirement cut on graphs. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 209–220. Springer, 2005. 74

[146] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Symposium on Foundations of Computer Science (FOCS)*, pages 950–961. IEEE, 2017. 82

[147] Assaf Naor and Terence Tao. Scale-oblivious metric fragmentation and the nonlinear dvoretzky theorem. *Israel Journal of Mathematics*, 192(1):489–504, 2012. 12

[148] Joseph Naor and Baruch Schieber. Improved approximations for shallow-light spanning trees. In *Symposium on Foundations of Computer Science (FOCS)*, pages 536–541. IEEE, 1997. 44

[149] Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted steiner tree and related problems. In *Symposium on Foundations of Computer Science (FOCS)*, pages 210–219. IEEE, 2011. 12, 13, 15, 72, 73

[150] Haruko Okamura and Paul D Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory, Series B*, 31(1):75–81, 1981. 145

[151] Merav Parter and Eylon Yogev. Optimal short cycle decomposition in almost linear time. In *International Colloquium on Automata, Languages and Programming (ICALP)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. 86, 92, 93

[152] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000. 84

[153] Harald Racke. Minimizing congestion in general networks. In *Symposium on Foundations of Computer Science (FOCS)*, pages 43–52. IEEE, 2002. 12

[154] Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 755–764, 2010. 82

[155] R Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In *Symposium on Foundations of Computer Science (FOCS)*, pages 202–213. IEEE, 1994. 41, 44

[156] Gabriele Reich and Peter Widmayer. Beyond steiner's problem: A vlsi oriented generalization. In *International Workshop on Graph-theoretic Concepts in Computer Science*, pages 196–210. Springer, 1989. 12

[157] Neil Robertson and Paul D Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. 143

[158] André Rossi, Alexis Aubry, and Mireille Jacomino. Connectivity-and-hop-constrained design of electricity distribution networks. *European journal of operational research*, 218 (1):48–57, 2012. 3, 41, 44

[159] Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 350–363, 2020. 92

[160] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2616–2635. SIAM, 2019. 79

[161] 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*, 41(5):1235–1265, 2012. 81

[162] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90, 2004. 82

[163] Babacar Thiongane, Jean-François Cordeau, and Bernard Gendron. Formulations for the nonbifurcated hop-constrained multicommodity capacitated fixed-charge network design problem. *Computers & Operations Research*, 53:1–8, 2015. 44

[164] Luca Trevisan. Approximation algorithms for unique games. In *Symposium on Foundations of Computer Science (FOCS)*, pages 197–205. IEEE, 2005. 82

[165] Stefan Voß. The steiner tree problem with hop constraints. *Annals of Operations Research*, 86:321–345, 1999. 44

[166] Thomas Victor Wimer. Linear algorithms on k-terminal graphs. *PhD Thesis*, 1987. AAI8803914. 143

[167] Kathleen A Woolston and Susan L Albin. The design of centralized networks with reliability and availability constraints. *Computers & Operations Research*, 15(3):207–217, 1988. 3, 41, 44