Optimization Problems in Network Connectivity

by

Debmalya Panigrahi

B.E., Jadavpur University (2004)
M.E., Indian Institute of Science (2006)

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

Author .....
Department of Electrical Engineering and Computer Science
June 29, 2012

Certified by .....
David R. Karger
Professor
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Students

Optimization Problems in Network Connectivity

by

Debmalya Panigrahi

Submitted to the Department of Electrical Engineering and Computer Science
on June 29, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Besides being one of the principal driving forces behind research in algorithmic theory for more than five decades, network optimization has assumed increased significance in recent times with the advent and widespread use of a variety of large-scale real-life networks. The primary goal of such networks is to connect vertices (representing a variety of real-life entities) in a robust and inexpensive manner, and to store and retrieve such connectivity information efficiently. In this thesis, we present efficient algorithms aimed at achieving these broad goals.

The main results presented in this thesis are as follows.

- **Cactus Construction.** We give a near-linear time Monte Carlo algorithm for constructing a cactus representation of all the minimum cuts in an undirected graph.
- **Cut Sparsification.** A cut sparsifier of an undirected graph is a sparse graph on the same set of vertices that preserves its cut values up to small errors. We give new combinatorial and algorithmic results for constructing cut sparsifiers.
- **Online Steiner Tree.** Given an undirected graph as input, the goal of the Steiner tree problem is to select its minimum cost subgraph that connects a designated subset of vertices. We give the first online algorithm for the Steiner tree problem that has a poly-logarithmic competitive ratio when the input graph has both node and edge costs.
- **Network Activation Problems.** In the design of real-life wireless networks, a typical objective is to select one among a possible set of parameter values at each node such that the set of activated links satisfy some desired connectivity properties. We formalize this as the network activation model, and give approximation algorithms for various fundamental network design problems in this model.

Thesis Supervisor: David R. Karger
Title: Professor

Acknowledgments

This thesis represents the culmination of a journey, both academic and personal, and as with any journey in life, it owes its fruition to a large number of people who have enriched it in more ways than I can possibly recount now. To all of them, my heartfelt gratitude, and if I fail to acknowledge any of them in person, my most sincere apologies.

I would not have had an academic career today had it not been for the untiring efforts of Ramesh Hariharan, who not only propelled me into the world of research over numerous lunches and late-night meetings at IISc, but also made it a point to meet me in person whenever he was in Boston over the last five years just to enquire about my well-being at MIT. I have cherished every discussion I have had with him, about academic matters and beyond, and would hope our association continues for many years to come.

Over the last five years, David Karger has been an incredible adviser, effortlessly playing multiple simultaneous roles: as my primary point of reference in the department and the university; as the most vocal champion of my work in the research community at large; and as a research supervisor who has ensured that I stay on track while allowing me complete freedom to pursue my own interests. David is also the best exponent of scholarly writing that I have come across, and any parts of the thesis that are readable are entirely due to whatever little I have learnt about writing lucidly from him. I hope I get the opportunity to seek his advice and guidance long after I cease to be his advisee.

The results presented in this thesis have been an outcome of fruitful collaborations with Ramesh Hariharan, David Karger, Seffi Naor, and Mohit Singh. I have also worked with Gagan Aggarwal, Yossi Azar, Susan Davidson, Atish Das Sarma, Sreenivas Gollapudi, Sanjeev Khanna, Aleksander Mądry, Aranyak Mehta, Tova Milo, Thomas Moscibroda, Bhaskaran Raman, Sudeepa Roy, Aravind Srinivasan, and Andrew Tomkins in the last five years on results that do not appear in this thesis, and with Anand Bhalgat, Partha Dutta, Ramesh Hariharan, Sharad Jaiswal, Telikepalli Kavitha, Vivek Mhatre, K.V.M. Naidu, Rajeev Rastogi, and Ajay Todimala on various research projects earlier. These collaborations have been one of the most enjoyable aspects of my academic life, and I would like to express my heartfelt gratitude to all of my collaborators for working with me over the years.

I spent all my four summers on the west coast: in the distributed computing group at Microsoft Research (Redmond), the search labs at Microsoft Research (Mountain View), the theory group at Microsoft Research (Redmond), and the social networking research group at Google Research (Mountain View). Many thanks to Atish Das Sarma, Alex Fabrikant, Sreenivas Gollapudi, Kamal Jain, Thomas Moscibroda, and Andrew Tomkins for being wonderful hosts during these internships. I would also like to sincerely thank Swarup Acharya, Pankaj Agarwal, Nikhil Bansal, Avrim Blum, Sunil Chandran, Richard Cole, Jie Gao, Navin Goyal, Elena Grigorescu, Anupam Gupta, MohammadTaghi Hajiaghayi, Anna Karlin, Sanjeev Khanna, Phil Klein, Mohammad Mahdian, Aranyak Mehta, Kamesh Munagala, Viswanath Nagarajan, Rina Panigrahy, Yuval Peres, Seth Pettie, Vijaya Ramachandran, John Reif, Tim

Roughgarden, Barna Saha, Baruch Schieber, Steven Skiena, Prasad Tetali, and David Woodruff for being wonderful hosts on various academic visits.

Thanks are due to Michel Goemans, Piotr Indyk, David Karger, and Jon Kelner for not only agreeing to serve on my thesis committee but also for helping and advising me in various capacities over the years. I would also like to thank the other theory faculty at CSAIL for making the theory group one of the most productive and lively places in the Stata Center. In particular, I would like to personally thank Costis Daskalakis, Jon Kelner, and Madhu Sudan. Costis and Jon have been inspiring examples of young academicians, who, in spite of their extremely busy schedules, have always been willing to keep every request I have made, be it discussing research, job opportunities, or giving feedback on my practice talks. Madhu has been a tower of strength — someone I could turn to for advice at any time without any reservation. Madhu also deserves a special note of thanks for funding me for a semester even though I had absolutely no right to his funds as a non-advisee.

Many thanks to the staff and students in the theory group and more generally, in CSAIL and EECS. The last five years would not have been half as enjoyable as it turned out were it not for the many hours of passionate debate and idle gossip with the many friends that I made here. In particular, I would like to mention Arnab Bhattacharyya, Yang Cai, Deepti Bhatnagar, Neha Gupta, Bernhard Haeupler, Aleksander Mądry, Adam Marcus, Ankur Moitra, Jelani Nelson, Rotem Oshman, Eric Price, Shubhangi Saraf, and Vineet Sinha for being such wonderful friends over the last five years. I would personally like to thank Adam Marcus and Jelani Nelson for all the fun we have had over the years, and for being always ready to help whenever I needed it. Many thanks to Arnab and Payel for making my last few months at MIT the most enjoyable phase of the last five years from a social perspective, and to my friends from college — Aditi, Anirban, Arindam, Barna, Debarghya, Dipanjan, Joydeep, Sayak, Shaon, Supratim, Sunny, and many others — for their love and friendship.

Finally, it goes without saying that I owe everything that I have done to my parents *Maa* and *Baba*, my siblings *Dada* and *Dondi*, and other members of my family, especially *Jethu* and *Dida*. The loving embrace that they have ensconced me in since my first day cannot be repaid, nor do I wish to. My sister-in-law *Bunu* and my parents-in-law have given me a second family in the last few years, one that has been just as warm as my own family. Last but not the least, my wife, Sudeepa, has been a pillar of strength like no other, offering unstinted support and enjoyable company in times of stress and jubilation alike. As I look forward to life with her, I can only offer my deepest sense of gratitude for standing by me even when everything seemed lost.

This research has been supported by an Akamai MIT presidential fellowship, and NSF contracts CCF-0635286, CCF-1117381, and STC-0939370.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Networks are ubiquitous in nature and show up in various guises in a wide variety of natural, social, and applied sciences such as communication, manufacturing, electronics, biology, evolution, sociology, urban development, and so on. It therefore comes as no surprise that network optimization has been an important component of computer science and operations research since their very early days. The primary objective of a network is to connect entities and a fundamental goal in studying a network is to explore the properties of such connections. In this thesis, we explore and exploit the combinatorial structure of graphs to achieve significant progress in several classical optimization problems in network connectivity.

We focus on two fundamental categories of problems in this domain. The first category, which we call *connectivity data structures*, focuses on efficiently estimating connectivity parameters of existing networks and encapsulating them in succinct data structures that provide efficient access to such information. The second category, called *network design*, comprises problems where the goal is to design networks using priced network elements such as nodes and edges, that satisfy desired connectivity requirements. For example, the problem of finding a minimum cut in a graph belongs to the first category whereas that of designing a minimum weight Steiner tree is in network design. We propose new combinatorial and algorithmic techniques for optimization problems in both categories.

1.1 Preliminaries

In this section, we set up the notation and terminology that we will use throughout the thesis. It is important to mention at the outset that this thesis deals only with undirected graphs and unless otherwise mentioned, all graphs are assumed to be undirected.

The input graph is denoted by $G = (V, E)$ where V is a set of n vertices and E is a set of m edges. Each edge $e = (u, v)$ has two vertices as *endpoints*. We will deal with both *node-weighted* and *edge-weighted* graphs. In node-weighted graphs, each vertex and each edge is allowed to have a non-negative, polynomially bounded weighted. Typically, for most problems on node-weighted graphs, we will first reduce the prob-

lem to one where only the vertices (and not the edges) have weights by replacing each weighted edge by a path of length 2 where the weight of the intermediate vertex equals the weight of the edge. On the other hand, in edge-weighted graphs, only the edges are allowed to have non-negative, polynomially bounded weights. A special case of edge-weighted graphs are *unweighted* graphs, where each edge has a weight of 1. However, unweighted graphs are allowed to have *parallel edges*, i.e. multiple edges between the same pair of vertices.

A *cut* is a bi-partition of vertices $(S, V - S)$, and may either mean one of the sides of the bi-partition, or the set of edges that have exactly one endpoint on each side of the bi-partition. The *value of a cut* is the sum of weights of edges in the cut. A *minimum cut* in a graph is a cut of minimum value. A cut is said to *separate* two vertices s and t is they appear on two sides of the bi-partition. For any two vertices s, t, an $s - t$ *minimum cut* is a cut of minimum value that separates s and t. We will often use the shorthand *min-cut* for minimum cut.

A *flow* between a pair of vertices s, t is a collection of paths between s and t and a non-negative, real *flow value* associated with each such path satisfying the constraint that the sum of flow values on all paths containing any particular edge is at most the weight (often called *capacity* in the context of flows) of the edge. The overall *value* of the flow is the sum of the flow values over all the paths. A *maximum flow* is a flow of maximum value between s and t. It is well-known that if all edge weights are integers, then there always exists at least one maximum flow between any pair of vertices s, t where all the flow values are integers. In unweighted graphs, such a flow corresponds to a maximum-sized collection of edge-disjoint paths between s and t.

We will use the following duality between cuts and flows called Menger's theorem (see e.g. [24] for a proof).

Theorem 1.1 (Menger's Theorem)**.** *The value of a maximum flow between any two vertices s, t in a graph is equal to the value of an $s - t$ minimum cut.*

In network design, a graph is said to *connect* a set of vertices $S \subseteq V$ if it contains at least one path between every pair of vertices in S. The *approximation ratio* of an algorithm for a minimization (resp., maximization) problem is the maximum (resp., minimum), over all input instances, of the ratio of the objective value in the algorithmic solution to that in an optimal solution. In the *online* setting, the entire input is not revealed to the algorithm at the outset; rather it receives constraints in online steps that it must satisfy by augmenting the current solution. The approximation ratio of an online algorithm (where we consider the worst input sequence) is called its *competitive ratio*.

We will present both *deterministic* and *randomized* algorithms. Randomized algorithms are further subdivided into *Las Vegas* and *Monte Carlo* algorithms. A Las Vegas algorithm always produces the desired output, whereas a Monte Carlo algorithm produces the desired output *with high probability* (or whp), which means that the probability of error is $o(1)$. The running time of a randomized algorithm is its expected running time. Similarly, the approximation/competitive ratio of a randomized algorithm is the ratio of the expected value of the objective in the algorithmic solution to the optimal objective value.

An algorithm is said to have a *linear* time complexity if its running time $O(I)$ where I is the size of the input. An algorithm is said to have a *near-linear* time complexity if its have a running time is $O(I \log^c I)$ for some constant c. Typically, we will denote a time complexity of $O(f(I) \log^c(I))$ by $\tilde{O}(f(I))$, where $f(.)$ is a function of the input size I. In particular, a near-linear time algorithm has a running time of $\tilde{O}(I)$. An algorithm is said to have *polynomial* time complexity if its running time is $O(f(I))$, where f is a polynomial function. The approximation/competitive ratio of an algorithm is said to be *poly-logarithmic* if it is $O(\log^c I)$ for some constant c. An algorithm is said to have *quasi-polynomial* time complexity if its running time is $O(I^{\log^c I})$ for some constant c.

The probability of an event $\mathcal{E}$ is denoted $\mathbb{P}[E]$ while the expectation of a random variable X is denoted $\mathbb{E}[X]$. We will use the following elementary tools from probability theory.

Theorem 1.2 (Linearity of Expectation)**.** *For any set of random variables $X_1, X_2, \ldots, X_k$,*

$$\mathbb{E}\left[\sum_{i=1}^{k} X_i\right] = \sum_{i=1}^{k} \mathbb{E}[X_i].$$

Theorem 1.3. *For any set of independent random variables $X_1, X_2, \ldots, X_k$,*

$$\mathbb{P}\left[\cap_{i=1}^{k} X_i = a_i\right] = \prod_{i=1}^{k} \mathbb{P}[X_i = a_i].$$

Theorem 1.4 (Chernoff Bounds)**.** *For any independent set of random variables $X_1, X_2, \ldots, X_k$ and for any $\epsilon \in (0,1)$, where $\mathbb{P}[X_i = 1] = p_i$ and $\mathbb{P}[X_i = 0] = 1 - p_i$, we have*

$$\begin{aligned}
\mathbb{P}\left[\sum_{i=1}^{k} X_i < (1-\epsilon)\sum_{i=1}^{k} p_i\right] &< \exp\left(-\epsilon^2 \sum_{i=1}^{k} p_i/2\right) \\
\mathbb{P}\left[\sum_{i=1}^{k} X_i > (1+\epsilon)\sum_{i=1}^{k} p_i\right] &< \exp\left(-\epsilon^2 \sum_{i=1}^{k} p_i/3\right)
\end{aligned}$$

1.2 Overview of Results

1.2.1 Connectivity Data Structures

We consider two connectivity data structures. In the first problem, we are given an undirected graph with edge weights and the goal is to find *all* min-cuts in the graph. We will represent these min-cuts succinctly using a data structure known as the *cactus representation* [25] of the graph, and call this the *cactus construction* problem. For this problem, we present a randomized Monte Carlo algorithm that has a running time of $\tilde{O}(m)$. This improves upon the previous best time complexity of $\tilde{O}(n^2)$ [60] and is optimal up to poly-logarithmic factors. The best deterministic algorithm for

this problem has a running time of $\tilde{O}(mn)$ [76, 32], which improves to $\tilde{O}(m\sqrt{n})$ [57] for Las Vegas algorithms on unweighted input graphs.

We observe that all previous algorithms for the cactus construction problem relied on an intermediate data structure called the *chain representation* of cuts which requires $\Omega(n^2)$ space even for graphs containing $O(n)$ edges. In fact, there are simple graphs such as a cycle on n vertices that contain $O(n)$ edges but have $\Omega(n^2)$ min-cuts. To circumvent this quadratic barrier, we give an algorithm that identifies a subset of min-cuts called *minimal min-cuts* (introduced originally by Gabow [35]) in $\tilde{O}(m)$ time and show that the minimal min-cuts contain sufficient information to efficiently reconstruct the entire cactus.

The second problem in this category asks the following question: given any graph, does there always exist a sparse graph on the same set of vertices that (approximately) preserves the values of all cuts? It is important to note that the output graph (called a *cut sparsifier*) is allowed to have edge weights irrespective of whether the input graph is weighted or unweighted. This problem, called *cut sparsification*, was introduced by Benczúr and Karger [11], who answered the above question in the affirmative. They proposed a sampling scheme on the edges of the input graph and showed that for a particular choice of sampling probabilities, the resulting sample is a cut sparsifier with high probability. They also conjectured that a much simpler and more natural set of sampling probabilities based on edge connectivities should also yield a cut sparsifier, and opined that such a result would substantially simplify cut sparsification algorithms. We settle this conjecture in the affirmative and extend our proof techniques to develop a sampling-based framework for cut sparsification. This framework unifies, simplifies, and extends previous sparsification results, including that of Benczúr and Karger. We also give a Monte Carlo cut sparsification algorithm that has a running time of $O(m)$ for unweighted graphs and $O(m) + \tilde{O}(n)$ for weighted graphs. The running times are optimal, except if the input graph is very sparse (has $\tilde{O}(n)$ edges) and is weighted. The previous best running times were $O(m \log^3 n)$ for weighted graphs and $O(m \log^2 n)$ for unweighted graphs [11].

1.2.2 Network Design

Our second category of network connectivity problems is network design. We consider two problems in this category. In the Steiner tree problem, we are given a weighted graph G and a subset T of k vertices called *terminals*. The goal is to find a minimum weight subgraph H that connects all terminals. We consider the *online* node-weighted Steiner tree problem where the node-weighted input graph G is given at the outset (i.e. *offline*) but the terminals are identified one at a time. On the arrival of a terminal, it has to be connected to the previous terminals by augmenting the selected subgraph. As usual, the goal is to minimize the weight of the selected subgraph. We explore new combinatorial properties of the Steiner tree problem to obtain an online Las Vegas algorithm that has a poly-logarithmic competitive ratio. The competitive ratio of our algorithm is $O(\log n \log^2 k)$, which is optimal up to a logarithmic factor against a known lower bound of $\Omega(\log n \log k)$. The previous best algorithm for this problem was the naïve greedy algorithm, i.e. we select the cheapest path to connect

the new terminal to the component containing the previous terminals, which has the optimal competitive ratio of $O(\log k)$ if the input graph is edge-weighted, but the competitive ratio degrades to $\Omega(k)$ in the presence of node weights.

We also give the first online algorithms with poly-logarithmic competitive ratio for several generalizations of the Steiner tree problem such as the Steiner forest problem and the group Steiner tree problem. However, unlike the online Steiner tree algorithm which has polynomial running time, these algorithms run in quasi-polynomial time.

In this thesis, we introduce a new suite of network design problems that generalize the node-weighted setting. Abstractly, in a traditional node-weighted network design problem, each vertex has two choices, that of paying a cost equal to the weight of the vertex or paying a cost of 0, and only the edges where both endpoints were paid for are *activated*. The goal is to make choices at the vertices that minimize the overall cost subject to the constraint that the activated set of edges satisfies some desired connectivity property (such as connecting a given a set of terminals in the Steiner tree problem). We generalize this view of network design problems in two ways: first, there is an arbitrary set of cost options at a vertex, and second, an edge is activated according to an arbitrary monotonic function applied to the choices made at its two endpoints. As usual, the goal is to make choices at the vertices that minimizes overall cost while ensuring that the set of activated edges satisfies some desired connectivity property. We call these *network activation* problems. It turns out that this generalization helps unify a variety of network design problems for wireless networks that were previously considered individually in either the theory or the networking literature.

In this thesis, we formally define the network activation framework, and give approximation algorithms and matching lower bounds for a variety of classical network design problems in this framework. In particular, we show that in the network activation framework,

- The shortest path problem has a deterministic polynomial time exact algorithm.
- The Steiner forest problem has a deterministic algorithm with approximation ratio $O(\log k)$, which is NP-hard to improve beyond constant factors. As a corollary, the minimum spanning tree problem, which is a special case of the Steiner tree problem with all vertices being designated as terminals, has an approximation ratio of $O(\log n)$. Perhaps surprisingly, we show that this result is also tight, i.e. it is NP-hard to approximate the minimum spanning tree problem to a factor of $o(\log n)$ in the network activation framework.
- The 2-edge-connectivity (resp., 2-vertex-connectivity or bi-connectivity) problem, where the connectivity constraint is that there must be at least two edge-disjoint (resp., vertex-disjoint) paths in the activated set of edges connecting every pair of vertices, has a deterministic algorithm with approximation ratio $O(\log n)$.
- The ℓ-edge-connectivity problem, where the connectivity constraint is that there must be ℓ edge-disjoint paths in the activated set of edges connecting every pair

of vertices, has a deterministic reduction to a more tractable degree-constrained problem. In fact, we give a bi-criteria approximation algorithm for the ℓ-edge-connectivity problem via the corresponding degree-constrained problem for an interesting application of the network activation framework called installation cost optimization.

Roadmap

The next two chapters are devoted to connectivity data structures. We present our results in cactus construction in Chapter 2 and those in cut sparsification in Chapter 3. Chapters 4 and 5 focus on network design and respectively present results on online Steiner trees and network activation problems.

Part I

Connectivity Data Structures

Chapter 2

Cactus Construction

The cactus is an elegant data structure introduced by Dinitz *et al* [25] that represents all (possibly $\Theta(n^2)$) minimum cuts of an undirected graph using a different undirected graph on $O(n)$ edges. The representing graph is a tree of cycles—a collection of cycles connected to each other by non-cycle edges that form a tree. Each vertex in the original graph is mapped to a vertex of the cactus (though the mapping can be non-injective and non-surjective). In the cactus, removing any tree edge, or any pair of edges from the same cycle, divides the cactus vertices in two. Each such partition induces a corresponding partition of the original graph vertices; these are precisely the min-cuts of the original graph. A cactus makes it easy to enumerate all min-cuts, to find a min-cut separating any two vertices if one exists, and to compute other useful characteristics of the min-cuts of the original graph.

In this chapter, we give an $\tilde{O}(m)$-time Monte Carlo algorithm for constructing the cactus representation of a (weighted/unweighted) graph. This improves on the previous best $\tilde{O}(n^2)$-time algorithm of Karger and Stein [60], and is optimal up to logarithmic factors. It also matches, up to logarithmic factors, the time complexity for finding a *single* min-cut [58].

2.1 Background

Let us first formally define a cactus graph.

Definition 2.1. *An undirected graph is said to be a* cactus graph *(or simply, a* cactus*) if each edge in the graph belongs to at most one cycle. An edge that belongs to exactly one cycle is called a* cycle edge *while one that does not belong to any cycle is called a* tree edge*.*

Note that if the weight of every tree edge is k and that of every cycle edge is $k/2$ in a cactus graph, then every min-cut of the graph corresponds to two cycle edges in the same cycle or a single tree edge. We use this structure to define the cactus representation of a graph.

Definition 2.2. *A* cactus representation *of a graph $G = (V, E)$ is a cactus graph $H = (U, F)$ and a (possibly non-surjective and non-injective) function $\psi : V \to U$*

such that

- *Each tree edge in H has weight λ and every cycle edge has weight $\lambda/2$, where λ is the value of a min-cut in G.*
- *For any min-cut $S \subset U$ in H, the cut defined by $\{v \in V : \psi(v) \in S\}$ is a min-cut in G. Conversely, for any min-cut $S \subset V$ in G, all cuts $S' \subset U$ in H satisfying $S = \{v \in V : \psi(v) \in S'\}$ are min-cuts in H.*

The vertices in the cactus graph that do not have any vertex in V mapping to them are called *empty nodes* in the cactus representation. Note that a graph may have multiple cactus representations. Our goal is to find at least one of them.

The Cactus Construction Problem
The input comprises a (weighted/unweighted) graph $G = (V, E)$ and the goal is to construct a cactus representation of G.

2.1.1 History

Karzanov and Timofeev gave a sequential algorithm for cactus construction [62] that was parallelized by Naor and Vazirani [76]. This algorithm used an explicit listing of all the min-cuts of the graph. An undirected graph potentially has $\Theta(n^2)$ min-cuts, each of which can be described explicitly in $\Theta(n)$ space; thus, the explicit listing uses $\Theta(n^3)$ space. Any cactus construction using such a listing will of course require $\Theta(n^3)$ time just to read its input. This was unimportant when min-cut algorithms were slow, as the time to find the min-cuts dominated the time to construct the cactus. However, as faster min-cut algorithms were developed [50, 74, 60, 58], faster cactus construction algorithms became imaginable. Karger and Stein [60] gave an $\tilde{O}(n^2)$-time cactus construction based on the chain representation of min-cuts.

Definition 2.3. *A* chain *of min-cuts is a set of concentric min-cuts in a graph, i.e. $\emptyset \subset S_1 \subset S_2 \subset \ldots \subset S_k \subset V$. A* chain representation *of min-cuts encodes all min-cuts into n chains.*

Note that each chain of min-cuts can be represented in $O(n)$ space, and therefore, the chain representation takes $O(n^2)$ space. The running time of the Karger-Stein algorithm is thus near-linear in the (maximum) size of the representation it uses.

Subsequently, Karger [58] gave a near-linear time algorithm for finding *a* min-cut in a graph. However, no algorithm with similar time bounds was given for cactus construction, the quadratic-space intermediate representation used by the previous algorithms being the bottleneck.

2.1.2 Our Contributions

To achieve near-linear time, we need to find a better way to examine all the min-cuts of the graph that need to be incorporated in the cactus. Indeed, it is not only the size of these cuts' representation that is at issue: since there can be as many as $\Theta(n^2)$ cuts even in a graph with n edges (consider the cycle), a near-linear time cactus algorithm does not even have time to *encounter* all the min-cuts, let alone build a large representation of them. This is the challenge we surmount in the chapter: to go directly from the size-m graph to the size-n cactus, without ever "unpacking" the set of min-cuts that we need to examine for the construction.

Theorem 2.4. *There is a Monte Carlo algorithm that finds a cactus representation of a (weighted/unweighted) graph with high probability in $O(m \log^4 n)$ time.*

2.1.3 Our Approach

Although there may be $\Theta(n^2)$ min cuts, we show that the cactus can be constructed by finding only $O(m+n)$ *minimal* min-cuts. (The concept of minimal min-cuts was introduced by Gabow in [35].) For intuition, suppose the cactus is a tree (i.e. the cactus has no cycles) and the vertices of the graph are in 1-1 correspondence with those of the cactus. To build the cactus, we need only identify the tree edges in the cactus. To do so, pick some vertex r of the graph, and imagine we root the cactus at this vertex. Then the min-cuts correspond to subtrees of this tree, and the subtree rooted at v is precisely the *minimal* (in number of vertices) min-cut separating v from the root r. Enumerating all such subtrees could generate $\Theta(n^2)$ work. So instead, we aim to identify the *parent* of each vertex. If w is the parent of v, then the subtree rooted at w, besides being a minimal min-cut (for w) is the *second smallest* min-cut separating v from r. Thus, to construct the cactus, we label each vertex w with its minimal min-cut, and we find the parent of v by finding the vertex labeled by the second smallest min-cut for v among these minimal min-cuts.

The first complication arises due to multiple graph vertices mapping to the same cactus vertex. All these vertices however have the same minimal min-cut and second smallest min-cut. These vertices are therefore indistinguishable and are treated as a single vertex in the cactus construction algorithm.

Things get more complicated in the presence of cycles. We need to identify cycle edges of the cactus. We can still speak of rooting the cactus, and of vertices "below" others in the rooted cactus. Vertices on cycles, however, can have two "parents", meaning there is no one second-smallest cut identifying a unique parent for such vertices. Instead, we argue that if (v, w) is a cycle edge, then there is an edge e with one endpoint below v, and another below w. In this case, the smallest min-cut separating *both* endpoints of edge e from the root r is the one that detaches edge (v, w) from the rest of its cactus cycle, and is thus the union of the minimal min-cuts labeled by v and w. In other words, there is a minimal min-cut for some edge e that "certifies" that v and w are neighbors on a cactus cycle. Thus, enumerating the minimal min-cuts for *edges* in the graph lets us identify cycle edges of the cactus.

We similarly handle the last complication, that of empty nodes in the cactus representation. In this case, we show there is an edge e of the graph whose two endpoints have this empty node as a least common ancestor—thus, the minimal min-cut separating (both endpoints of) e from r corresponds to the subtree rooted at u. We go a step further- we show that if a min-cut X represented by an empty cactus node is the smallest min-cut containing another min-cut Y, then X is the minimal min-cut separating (both endpoints of) some edge f with exactly one endpoint in Y from r.

To find the minimal min-cuts, our algorithm builds on Karger's $\tilde{O}(m)$-time min-cut algorithm [58]. Given a graph G, Karger's algorithm uses random sampling to construct a set $\mathcal{T}$ of $O(\log n)$ trees with the property that any min-cut of G *2-respects* some tree of $\mathcal{T}$. That is, there will be some tree $T \in \mathcal{T}$ such that it has at most two edges crossing the given min-cut. Removing this edge or pair of edges will divide the tree into two or three pieces that correspond to the vertex partition of the min-cut (if two pieces, the partition is obvious; if three pieces, then the two non-adjacent pieces form one side of the min-cut). The min-cut algorithm finds a cut by inspecting all pairs of potentially removable tree edges—not explicitly, as that would take $\Omega(n^2)$ time, but by finding a "best match" second edge for each of the n edges of T. While this algorithm will find *some* min-cut, it will not find all.

To find the minimal min-cuts, we augment Karger's algorithm. We know that each min-cut corresponds to a singleton or pair of edges from a tree $T \in \mathcal{T}$, so we seek the edges and pairs corresponding to minimal min-cuts. We piggyback on the part of Karger's algorithm that hunts for min-cuts, checking the values of cuts corresponding to certain singletons or pairs of edges. However, where Karger's algorithm can stop once it finds *some* min-cut, we need to work more exhaustively to enumerate *all* minimal min-cuts. This puts us at risk of spending $\Omega(n^2)$ time encountering too-many *non-minimal* min-cuts; we must therefore prove and exploit structural theorems regarding these minimal min-cuts that let us terminate the exploration early so as to guarantee spending only $\tilde{O}(m)$ time.

Once we have this (implicitly represented) list of all minimal min-cuts, we construct the tree T of vertex minimal min-cuts using the second smallest min-cuts that we described earlier. The following property follows immediately from the fact that a min-cut is contiguous in the cactus.

Lemma 2.5. *Consider any vertex x and its children $y_1, y_2, \ldots, y_k$ in tree T. The vertices in the subtree of x in T (call this set X) are contiguous in a cactus. Further, the vertices in the subtree of each of y_i (call these sets Y_i) are contiguous in a cactus.*

For each node of this tree, our goal then becomes constructing the cactus representation for the vertices in its subtended subtree, assuming that the cactus representation of its children subtrees have been constructed recursively. We show that this can be done in $\tilde{O}(m)$ time for the entire tree using the minimal min-cuts for edges that we described earlier.

Roadmap

We review Karger's min-cut algorithm [58] in Section 2.2. In Section 2.3, we describe our cactus construction algorithm that uses a modified version of Karger's algorithm.

2.2 Near-linear Time Min-cut Algorithm

In this section, we review Karger's $\tilde{O}(m)$ min-cut algorithm from [58]. To describe this algorithm, we need to first define some terms that we will use throughout this chapter.

Definition 2.6. *A cut is said to* k-respect *a spanning tree of a graph if the spanning tree contains at most k edges of the cut. A cut is said to* strictly k-respect *a spanning tree of a graph if the spanning tree contains exactly k edges of the cut.*

The following theorem provides the starting point of the min-cut algorithm.

Theorem 2.7 (Karger [58])**.** *Given any weighted, undirected graph G, in $O(m + n\log^3 n)$ time we can construct a set of $O(\log n)$ spanning trees such that each minimum cut 2-respects 1/3 of them with high probability.*

Throughout this discussion, we will consider these trees to be rooted at the same vertex. Since there are only $O(\log n)$ trees to consider, the problem of finding a minimum cut in the graph reduces to finding, given a spanning tree T, a min-cut that 2-respects T provided such a min-cut exists. This problem is further sub-divided into finding any min-cut that 1-respects T and finding any min-cut that strictly 2-respects T. The first subproblem can be solved easily by a post-order traversal that leads to the following lemma.

Lemma 2.8 (Karger [58])**.** *The values of all cuts that 1-respect a given spanning tree can be determined in $O(m + n)$ time.*

The more involved case is that of finding a minimum cut that strictly 2-respects the spanning tree. We can however restrict the problem further.

Definition 2.9. *A* bough *in a tree is a maximal path on the tree with a leaf at one end and having the property that all the other vertices have degree 2 in the tree.*

If we contract all the boughs into their immediate parent in the tree, then the number of leaves in the new tree is at most half of that in the original tree. This follows from the fact that each leaf in the new tree consumes at least 2 leaves of the original tree. Thus, in $O(\log n)$ iterations, the entire tree will shrink into a single vertex. The problem then becomes one of finding the strictly 2-respecting min-cuts where one of the edges is on a bough in $\tilde{O}(m)$ time.

Now, we further sub-divide the problem.

Definition 2.10. *The set of descendants of a vertex v in a spanning tree T is denoted by $v^{\downarrow}_T$. Similarly, the set of ancestors of v in T is denoted by $v^{\uparrow}_T$.*

If there is no scope of confusion, we will often drop the suffix and denote these sets by $v^{\downarrow}$ and $v^{\uparrow}$ respectively. Now observe that since the spanning tree is rooted, any tree edge can be uniquely represented by its lower endpoint in the tree. Therefore, each strictly 2-respecting cut corresponds to two unique vertices in the tree. We sub-divide the problem based of the relative locations of these vertices.

Definition 2.11. *Consider any two vertices v and w in a spanning tree T. If $v \in w^{\downarrow}$ or $v \in w^{\uparrow}$, then we write $v \parallel w$ (v and w are said to be* comparable*); else, $v \perp w$ (v and w are said to be* incomparable*).*

Let us first show how to handle the case when $v \perp w$. We need to define some notation.

Definition 2.12. *$\mathcal{C}(X, Y)$ is the sum of weights of edges with one endpoint in vertex set X and the other in vertex set Y, where $X \cap Y = \emptyset$. Overloading our notation, $\mathcal{C}(S) = \mathcal{C}(S, V - S)$.*

Definition 2.13. *The* v-precut at w*, denoted $\mathcal{C}_v(w)$, is the value*

$$\mathcal{C}_v(w) = \mathcal{C}(v^{\downarrow} \cup w^{\downarrow}) - \mathcal{C}(v^{\downarrow}) = \mathcal{C}(w^{\downarrow}) - 2\mathcal{C}(v^{\downarrow}, w^{\downarrow})$$

if $v \perp w$ and ∞ otherwise.

In the example in Figure 2-1, the precut

$$\mathcal{C}_v(w) = X - Z = (X+Y) - (Y+Z) = \mathcal{C}(v^{\downarrow} \cup w^{\downarrow}) - \mathcal{C}(v^{\downarrow}) = (X+Z) - 2Z = \mathcal{C}(w^{\downarrow}) - 2\mathcal{C}(v^{\downarrow}, w^{\downarrow}).$$

Definition 2.14. *The* minimum v-precut*, denoted $\mathcal{C}_v$, is the value*

$$\mathcal{C}_v = \min\{\mathcal{C}_v(w) | \exists (v', w') \in E, v' \in v^{\downarrow}, w' \in w^{\downarrow}\}.$$

Correspondingly, $\arg\min\{\mathcal{C}_v(w) | \exists (v', w') \in E, v' \in v^{\downarrow}, w' \in w^{\downarrow}\}$ *is called a* minimum precut of v.

The next lemma appears in [58] and follows from simple arithmetic on edge sets.

Lemma 2.15 (Karger [58])**.** *If it is determined by incomparable vertices, the minimum cut is* $\min_v(\mathcal{C}(v^{\downarrow}) + \mathcal{C}_v)$.

Calculating $\mathcal{C}(v^{\downarrow})$ for each vertex v follows directly from Lemma 2.8- these are the respective cut values. So, we are left to calculate $\mathcal{C}_v$ for each vertex v; the minimum cut can then be found in additional $O(n)$ time.

Recall that we have already assumed that v is on a bough of the spanning tree. Let us restrict ourselves initially only to the case when v is a leaf. We maintain a value $val[w]$ at each vertex w and initialize it to $\mathcal{C}(w^{\downarrow})$. Now, we require to subtract $2\mathcal{C}(v^{\downarrow}, w^{\downarrow})$ from each $val[w]$. Further, once $val[w]$ has been computed for each w, we need to find a vertex which has the minimum value of $val[w]$. The dynamic tree data structure [86] helps solving both problems. Basically, it provides the following primitives:

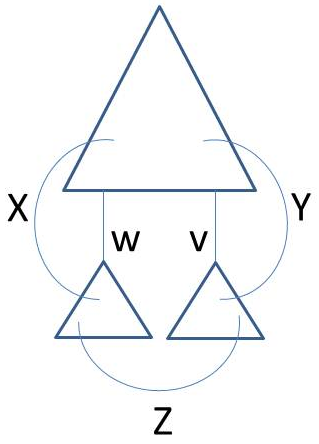

Figure 2-1: Example of a precut

- `Addpath`(v, x): add x to $val[u]$ for every $u \in v^{\uparrow}$.
- `MinPath`(v): return $\min_{u \in v^{\uparrow}} val[u]$ as well as the u achieving this minimum.

Karger shows that for a leaf v with d incident edges in the graph, we can find $\mathcal{C}_v$ via $O(d)$ dynamic tree operations that require $O(d \log n)$ time.

This procedure is now extended from a single leaf to an entire bough.

Lemma 2.16 (Karger [58])**.** *Let v be a vertex with a unique child u. Then either $\mathcal{C}_v = \mathcal{C}_u$, or else $\mathcal{C}_v = \mathcal{C}_v(w)$ for some ancestor w of a neighbor of vertex v. In the first case, all the minimum precuts of u which are not ancestors of neighbors of v continue to be minimum precuts of v.*

This lemma leads to a simple bottom-up walk on the bough, where in each step, the value of $\mathcal{C}_u$ computed inductively has to be compared with the values of $\mathcal{C}_v(w)$ in Lemma 2.16, which can be computed using an additional $O(d)$ dynamic tree operations, where d is the number of edges incident on vertex v in the graph.

We now describe the case of comparable v and w, i.e. $v \parallel w$. Without loss of generality, let us assume that $v \in w^{\downarrow}$ and v is on a bough. We need to compute $\mathcal{C}(w^{\downarrow} - v^{\downarrow})$. It is shown in [58] that

$$\mathcal{C}(w^{\downarrow} - v^{\downarrow}) = \mathcal{C}(w^{\downarrow}) - \mathcal{C}(v^{\downarrow}) + 2(\mathcal{C}(v^{\downarrow}, w^{\downarrow}) - 2\mathcal{C}(v^{\downarrow}, v^{\downarrow})).$$

For a given v, $\mathcal{C}(v^{\downarrow}, v^{\downarrow})$ and $\mathcal{C}(v^{\downarrow})$ are fixed and can be computed in $\tilde{O}(m)$ time for all the vertices by a minor extension of Lemma 2.8. Thus, it is sufficient to compute, for each vertex $w \in v^{\uparrow}$, the quantity $\mathcal{C}(w^{\downarrow}) + 2\mathcal{C}(v^{\downarrow}, w^{\downarrow})$. $val[w]$ is initialized to $\mathcal{C}(w^{\downarrow})$ for each vertex using Lemma 2.8. Now, using a post-order traversal as earlier and the dynamic tree operations mentioned above, $2\mathcal{C}(v^{\downarrow}, w^{\downarrow})$ is added to $val[w]$ for each $w \in v^{\uparrow}$. Once $val[w]$ has been computed for each w, we need to find a vertex which have the minimum value of $val[w]$. This can also be done using dynamic tree operations.

In summary, a strictly 2-respecting min-cut can be found in $O(D \log n)$ time for a bough which has a total of D edges incident on the vertices of the bough. After running the above procedure, $val[w]$ is reset to its original value by undoing all the operations (subtracting instead of adding in `AddPath`) in $O(D \log n)$ time. A different bough can now start running its procedure. Since an edge is incident on at most 2 boughs, the algorithm takes $\tilde{O}(m)$ time for processing all the boughs. As discussed earlier, all the boughs are now folded up and a new phase begins.

This algorithm yields the following theorem.

Theorem 2.17 (Karger [58])**.** *There is a Monte Carlo algorithm that finds a minimum cut in a (weighted/unweighted) graph with high probability in $O(m \log^3 n)$ time.*

2.3 Cactus Construction Algorithm

Unlike the min-cut algorithm presented above, we are not interested in finding only a single min-cut in the graph. To specify our goals, we need some more definitions.

Definition 2.18. *Let r be the vertex that was selected to be the root of the $O(\log n)$ trees in the tree packing. Then, the* size of a cut *is the number of vertices* not *on the side of r in the cut.*

Note that the sum of weights of edges in a cut is its value; it is important to keep the distinction between size and value of a cut in mind.

Definition 2.19. *A* minimal min-cut of a vertex *v is a min-cut of least size which separates v from r. If v is not separated from r by any min-cut, then its minimal min-cut is undefined. Overloading the definition, a* minimal min-cut of an edge *(u, v) is a min-cut of least size that separates r from both u and v. As earlier, if no min-cut separates both u and v from r, then its minimal min-cut is undefined.*

In the following discussion, we will often refer to a cut by a subset of vertices; such a subset would be the side of the cut not containing the root vertex r.

Definition 2.20. *Two cuts X and Y are said to be* crossing *if each of $X \cap Y, X - Y, Y - X$ and $X^C \cap Y^C$ is non-empty.*

Lemma 2.21 (see eg, Karger [58])**.** *If X and Y are crossing min-cuts, then $X \cap Y, X - Y, Y - X$ and $X \cup Y$ are min-cuts. Further,*

$$\mathcal{C}(X \cap Y, X^C \cap Y^C) = \mathcal{C}(X - Y, Y - X) = 0.$$

Lemma 2.22. *The minimal min-cut of a vertex or edge is unique. Further, the minimal min-cut of a vertex v does not cross any other min-cut of the graph.*

Proof. If there are two minimal min-cuts of a vertex v (resp., edge (u, v)), then their intersection is a smaller min-cut containing v (resp., u and v), contradicting minimality. Similarly, if the minimal min-cut X of a vertex v crosses another min-cut Y, then either $X \cap Y$ or $X - Y$ is a smaller min-cut containing v, contradicting minimality. □

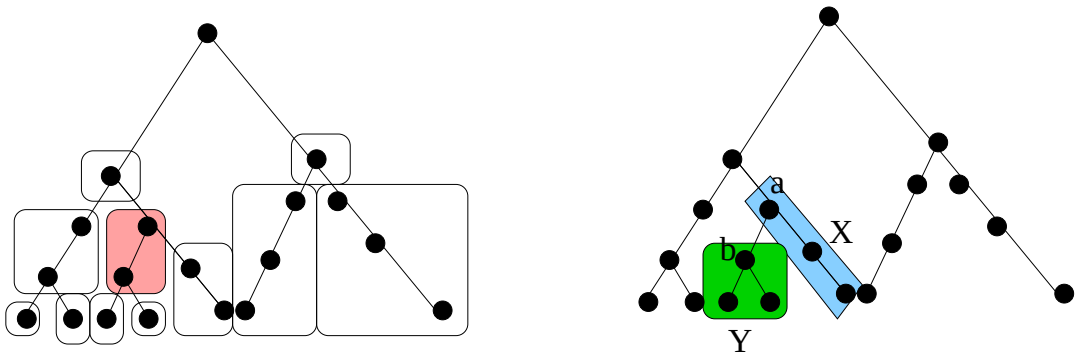

Figure 2-2: The two trees used in the modified min-cut algorithm

We are finally in a position to describe our plan. We first construct a list of $\tilde{O}(m)$ min-cuts containing the minimal min-cut of each vertex, if one exists. Then, we label each vertex with its corresponding minimal min-cut from the list and also, subsequently, find the minimal min-cuts of a *sufficient* set of edges. Finally, this set of minimal min-cuts for vertices and edges are used to construct a cactus representation of the graph.

2.3.1 Listing minimal min-cuts of vertices

We now modify Karger's min-cut algorithm to meet our objective. All our modifications are for the strictly 2-respecting scenario; the part of the algorithm that finds all 1-respecting min-cuts is exactly the same as above. Recall that the original algorithm has $O(\log n)$ phases, where the algorithm is run on progressively smaller trees in each phase. Now, consider a scenario where the minimal min-cut for a vertex u is defined by vertices v and w, where $v \perp w$ and $u \in v^{\downarrow}$. Our goal is to ensure that we identify this min-cut in the phase where we process v. However, due to the recursive process, $w^{\downarrow}$ might be a proper subset of the vertices compressed into a single node at this stage. In this case, we will fail to identify the minimal min-cut for u. So, we need to modify Karger's min-cut algorithm.

We also have $O(\log n)$ phases, but we maintain two trees in each phase. One tree is the shrunk tree S, identical to the earlier algorithm. The other tree T is the original spanning tree without any edge contraction.

Each vertex in the contracted tree S represents a set of vertices in the original tree T. For example, in Figure 2-2, the tree on the left is a spanning tree T where the boughs in the different phases are marked. The shaded bough (on the left) is processed in phase 2 at which stage its two vertices correspond to sets X and Y in T (due to boughs being folded up). These sets are shown on the right. For each such set X, let $\ell(X)$ denote the *leader* of the set, which is the highest vertex in T (eg, in Figure 2-2, $a = \ell(X)$ and $b = \ell(Y)$). Conversely, each vertex v is the leader of a contracted vertex in S (i.e. set of vertices in T) in some phase; denote this set by $\ell^{-1}(v)$.

Now, any set X that gets contracted into a single vertex in S can have two possible

structures in T:

- If X is a leaf in S, then it represents a subtree rooted at $\ell(X)$ in T.
- If X is a vertex with degree 2 in S, then it represents a subtree rooted at $\ell(X)$ in T, where one of the child subtrees of $\ell(X)$ has been removed. This child subtree is rooted at $\ell(Y)$, where Y is the only child of X in S.

Our goal is to ensure that if the minimal min-cut of vertex u is defined by vertices v and w, where $v \perp w$ and $u \in v^{\downarrow}$, then this min-cut is identified when $\ell^{-1}(v)$ is processed.

We need another definition.

Definition 2.23. *Consider any vertex v on a bough and let w be a minimum precut of v. If there exists no descendant x of w such that x is also a minimum precut of v, then w is said to be a* minimal minprecut *of v. If w is the only such vertex, it is said to be the* unique minimal minprecut *of v.*

Lemma 2.24. *Let the minimal min-cut of vertex u 2-respect a tree T, where it is represented by vertices v and w, $v \perp w$ and $u \in v^{\downarrow}$. Then, w is the unique minimal minprecut of v.*

Proof. Clearly, w is a minimal minprecut of v, else there is a smaller min-cut separating u from root r. If there are multiple minimal minprecuts of v, then these min-cuts cross, violating Lemma 2.22. □

To identify minimal minprecuts, we need to strengthen the `MinPath` primitive provided by the dynamic tree data structure. Recall that `MinPath`(v) for a variable *val* returns the minimum value of *val* among ancestors of v and a vertex which achieves this minimum value. If there are multiple ancestors of v achieving this minimum value, then the vertex returned by `MinPath` is ambiguous. However, we would like `MinPath` to return the closest ancestor of v achieving this minimum value. To achieve this, we run `AddPath`(v, ϵ) (for some $\epsilon > 0$) for each vertex v as a pre-processing step. Clearly, $val[v]$ now has a value $\epsilon|v^{\downarrow}|$. We choose a small enough ϵ so that this pre-processing step does not tamper with the ability of the algorithm to distinguish min-cuts from other cuts. Now, if we use our usual `MinPath` operations, we will always find the closest ancestor in case of a tie in the original graph.

We need to impose some additional structure on the minimal minprecuts of a vertex.

Definition 2.25. *Consider a vertex v and let its minimal minprecuts be $w_1, w_2, \ldots, w_k$, where each $w_i \perp v$. Let ℓ_i be the lca of w_i and v in tree T. Clearly, each ℓ_i lies on the path connecting v to root r; let ℓ_o be the vertex of least depth among the ℓ_is. Correspondingly, let w_o be a minimal minprecut of v such that lca of w_o and v is ℓ_o ($w_o = w_i$ for some i). We call w_o an* outermost minimal minprecut *of v. If w_o is unique, it is called the* unique outermost minimal minprecut *of v.*

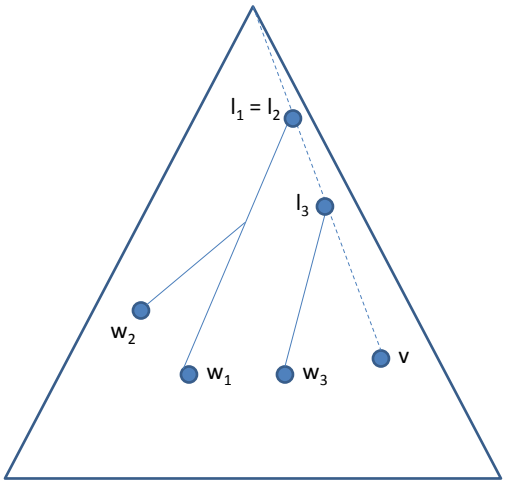

Figure 2-3: Outermost Minimal Minprecut

As an example, consider the tree in Figure 2-3. Here, $\ell_o = \ell_1 = \ell_2$ and w_o can be defined as either w_1 or w_2.

The following lemma is an extension of Lemma 2.16.

Lemma 2.26. *Let v be a vertex in T, $X = \ell^{-1}(v)$ and w be the unique outermost minimal minprecut of v. If X is a leaf in S, then there exists at least one edge between X and $w^{\downarrow}$. On the other hand, if X is a non-leaf in S, let Y be the only child of X in S and $u = \ell(Y)$. Then, either there exists at least one edge between X and $w^{\downarrow}$, or w is the unique outermost minimal minprecut of u.*

Proof. If X is a leaf in S, then by definition of a minimum precut, X and $w^{\downarrow}$ must be connected. Otherwise, let there be no edge between X and $w^{\downarrow}$. We need to prove that w is the unique outermost minimal minprecut of u. First, note that by definition of minimum precut, $u^{\downarrow}$ and $w^{\downarrow}$ must be connected since $v^{\downarrow} = X \cup u^{\downarrow}$. Further, by Lemma 2.16, $\mathcal{C}_u(w) = \mathcal{C}_v(w) = \mathcal{C}_v = \mathcal{C}_u$ and if any descendant of w is a minimal minprecut of u, then it is also a minimal minprecut of v, contradicting the minimality of w. Thus, w is a minimal minprecut of u. If z is a minimal minprecut of u such that the lca of z and u is either an ancestor of or the same as the lca of w and u, then $z \perp v$ and is a minimal minprecut of v. Thus, w is the unique outermost minimal minprecut of u. □

If w and v represent a minimal min-cut for a vertex $u \in v^{\downarrow}$, where $w \perp v$, then w is the unique outermost minimal minprecut of v. Thus, our goal is to identify the unique outermost minimal minprecut of a vertex, if it exists. Processing a leaf X in a bough in S is easy; we simply run `AddPath` followed by `MinPath` queries for the other endpoint of each edge with one endpoint in X. To process a vertex X of degree 2 in S, assume inductively that we have already found the unique outermost minimal minprecut w corresponding to its child Y in S, provided such a vertex exists. We now run `AddPath` followed by `MinPath` queries for the other endpoint of each edge with one

endpoint in X. Also, we check if w is a minprecut of $v = \ell(X)$ by inspecting the value of $val[w]$. The set of minimal minprecuts identified contains the unique outermost minimal minprecut of v, if it exists. We now run lca queries to determine if v has a unique outermost minimal minprecut among the minimal minprecuts identified. The total time consumed by this procedure is $O(d \log n)$, where d edges are incident on vertices in X. Overall, in any round, the time complexity is $O(m \log n)$ for this procedure.

We now describe the algorithm used to identify all strictly 2-respecting minimal min-cuts which are represented by comparable vertices. When we process vertex v, we would like to find all such min-cuts represented by v and w, where $v \in w^{\downarrow}$. Recall that Karger's min-cut algorithm allows us to compute $val[w]$ for each w such that inspecting $val[w]$ for each vertex w reveals all the min-cuts we want to identify. However, whereas in the min-cut algorithm, only one `MinPath` query needs to be run at vertex v, a single query would only reveal the deepest w which forms such a min-cut with v, but would not reveal additional vertices satisfying the property further up the spanning tree. To overcome this challenge, we maintain a list at each vertex w, denoted by $desc[w]$, which contains its descendants v with which it has been found to form a min-cut that is potentially minimal for some vertex. This list is initially empty for each vertex and populated by the following procedure (which is run, for each vertex being processed on a bough in S, after the `AddPath` calls): *Run a* `MinPath` *query at v. Let w be returned by this query. If v, w do not form a min-cut, then stop; else, if $desc[w]$ is non-empty, then add v to $desc[w]$ and stop; otherwise, add v to $desc[w]$ and recurse at w (ie, run a* `MinPath` *query at w and so on).* The correctness of the procedure is established by the following lemma.

Lemma 2.27. *Let $u \in v^{\downarrow}$ or $u \perp v$ in T. If both u and v represent min-cuts with some vertex $w \in v^{\uparrow} \cap u^{\uparrow}$, then any min-cut represented by v and any z such that $z \in w^{\uparrow}$ is not a minimal min-cut for any vertex.*

Proof. This follows directly from the observation that any min-cut represented by v and z must necessarily cross the min-cut represented by w and u. If such a min-cut is minimal for a vertex, then Lemma 2.22 is violated. □

The `AddPath` queries for a bough take $O(D \log n)$ time in the above procedure, where D is the number of edges incident on the bough. Overall, in any round, the time complexity is $O(m \log n)$ for these queries.

All the `MinPath` queries, except possibly the last one during the processing of each vertex v, result in populating the previously empty *desc* list of some vertex; thus, there are at most n such queries. The last query can be charged to the vertex being processed; thus there are at most n such queries as well. Overall, in any round, $O(n)$ `MinPath` queries are made, and they take $O(n \log n)$ time.

The next lemma follows from the above analysis by the fact that there are $O(\log n)$ rounds and $O(\log n)$ spanning trees in the algorithm.

Lemma 2.28. *The time complexity of producing the list of min-cuts that contains all the minimal min-cuts of vertices is $O(m \log^3 n)$.*

2.3.2 Labeling minimal min-cuts of vertices

We will now label each vertex with the smallest min-cut containing it among those that 2-respect a fixed spanning tree T. As discussed earlier, the vertices representing a min-cut can be used to classify the min-cuts into 3 categories: 1-respecting min-cuts (category 1) and strictly 2-respecting min-cuts where the vertices are incomparable (category 2) or comparable (category 3). We label each vertex with the smallest min-cut containing it in each category. Finally, for each vertex, we find the minimum among its $O(\log n)$ labels corresponding to the 3 categories of edges in the $O(\log n)$ trees.

Category 1 (1-respecting). Lemma 2.8 states that in $O(n)$ time, we can find the weights of all 1-respecting cuts. Assuming that we know the value of a min-cut using the algorithm in [58], it immediately follows that we can identify all the 1-respecting min-cuts in $O(n)$ time. To label each vertex with the minimal min-cut containing it, we contract all the edges in T which do not represent min-cuts. Then, the smallest min-cut containing vertex v is the edge connecting the contracted vertex containing v to its parent. Thus, we simply label all vertices in a contracted set by the root of the set. This takes $O(n)$ time.

Category 2 (strictly 2-respecting, incomparable). The smallest min-cut containing a vertex is the smallest among the min-cuts represented by its ancestors in the bough containing it in S. We trace the path along the bough downward maintaining the smallest encountered min-cut C. The label given to a vertex v is the min-cut stored as C when v is encountered. Clearly, this takes $O(1)$ time for each vertex along the walk, and therefore $O(n)$ time overall.

Category 3 (strictly 2-respecting, comparable). We perform a post-order tree traversal using a mergeable minheap (see e.g. [24]) to hold all the minimal min-cuts that contain the current vertex u. These min-cuts are exactly the min-cuts whose lower vertex has been encountered but the upper vertex has not been encountered yet. Labeling u with the smallest min-cut in the heap takes $O(1)$ time. Now, all the cuts whose upper vertex is u are removed from the heap and the heap is passed on to the parent of u, say v. All the heaps passed up from its children are now merged at v in amortized $O(\log n)$ time. This takes $O(n \log n)$ time overall.

The next lemma follows from the above analysis by the fact that there $O(\log n)$ spanning trees in the algorithm.

Lemma 2.29. *The time complexity of labeling each vertex with its minimal min-cut is $O(n \log^2 n)$.*

We now form a partition of the vertices according to their minimal min-cut and contract subsets of vertices having the same minimal min-cut. Clearly, this does not change the set of min-cuts in the graph, and hence does not affect any cactus representation of the graph. In the remaining discussion, a vertex will denote such

a contracted vertex. For simplicity, we will continue to denote the number of (contracted) vertices in the graph by n, which is an upper bound on this number.

2.3.3 Labeling second-smallest min-cuts of vertices

As mentioned in the introduction, we will now construct a tree of min-cuts by labeling each vertex with the second-smallest min-cut separating it from r among all the minimal min-cuts of vertices. First, we show that this second-smallest min-cut of a vertex is unique.

Lemma 2.30. *The second smallest min-cut separating a vertex from the root r among the minimal min-cuts of vertices is unique.*

Proof. Suppose not. Let X, Y, Z, W be the partition of the vertices formed by the crossing second smallest minimal min-cuts containing v, the crossing min-cuts being $X \cup Y$ and $X \cup Z$. Then, for any vertex $u \in X \cup Y$, either X or Y is a smaller min-cut than $X \cup Y$ containing u. Thus, $X \cup Y$ is not a minimal min-cut, which is a contradiction. □

As earlier, we sub-divide the problem based on the spanning tree where a minimal min-cut is 2-respecting and the structure of the min-cut in the spanning tree. So, given a spanning tree T, we aim to label each vertex with the second-smallest min-cut (in each of the three categories) containing the vertex. First, consider the 1-respecting cuts. After contracting edges which do not correspond to minimal min-cuts, let vertex v be in vertex set X. If Y is the parent of X in the tree, then the second-smallest 1-respecting min-cut containing v is represented by the root of Y. This takes $O(n)$ time for all the vertices in each spanning tree.

Now, consider the 2-respecting minimal min-cuts where the representative vertices are incomparable. Recall that in this case we used a top-down walk on the vertices to be labeled, maintaining the smallest min-cut containing the vertex. To label a vertex with the second-smallest cut, we need to maintain a list of two smallest minimal min-cuts rather than one in the top-down walk. So, this also takes $O(n)$ time in each spanning tree.

Next, consider the case where the vertices representing the min-cut in the tree are comparable. Here, recall that in the bottom up pass, we maintain a mergeable minheap of the minimal min-cuts containing the vertex. The min-cut at the top of the heap is the smallest mincut and one of its children is the second-smallest min-cut. This takes $O(n \log n)$ additional time in each spanning tree.

Finally, we consider all the labels (for both smallest and second-smallest min-cut) from the $O(\log n)$ spanning trees and retain the labels corresponding to the two smallest min-cuts for each vertex.

We now form a tree T by connecting vertex v with vertex u if the minimal mincut of u is the second-smallest minimal min-cut of v. This tree can be constructed in $O(n)$ additional time.

The next lemma follows from the analysis above along with Lemmas 2.28 and 2.29.

Lemma 2.31. *The time complexity of constructing a tree of minimal min-cuts of vertices where for every vertex v,*

- *the subtree subtended at v is the minimal min-cut separating v from r, and*
- *the subtree subtended at the parent of v in the tree is the second-smallest min-cut separating v from r*

is $O(m \log^3 n)$.

2.3.4 Minimal min-cuts of edges

As described earlier, we need to find the minimal min-cuts of edges. We need the following definitions.

Definition 2.32. *A* certificate *of a min-cut X is an edge e such that X is the minimal min-cut of e.*

Definition 2.33. *A min-cut Y is a* maximal min-cut *contained in a min-cut X if $Y \subset X$ and there does not exist any min-cut Z such that $Y \subset Z \subset X$.*

The following lemma lower bounds the sum of weights of certificates of a min-cut.

Lemma 2.34. *Consider a min-cut X. Let $Y \subset X$ be a min-cut satisfying the following properties:*

- *Y is a maximal min-cut contained in X.*
- *Y does not cross any other min-cut.*

Then, the total weight of certificates of X with one endpoint in Y is at least $\lambda/2$, where λ is the weight of a min-cut.

Proof. The properties satisfied by Y ensure that any edge between Y and $X - Y$ is a certificate of X. Now, if the total weight of edges between Y and $X - Y$ is less than $\lambda/2$, then $X - Y$ is a cut of weight less than λ. □

We now show that the above lemma ensures that the min-cuts of interest to us have sufficiently large number of certificates. Recall from the introduction that we are interested in min-cuts which are represented either by empty cactus nodes or by consecutive nodes on a cactus cycle.

Let us consider the first category of min-cuts, i.e. those represented by empty cactus nodes. Recall that we are interested not only in finding one certificate of such a min-cut (say X), but in finding a certificate for each maximal min-cut contained in it (i.e. min-cut Y such that there is no min-cut Z with $Y \subset Z \subset X$), where the certificate needs to have exactly one endpoint in Y. The following lemma, combined with Lemma 2.34 shows that the total weight of such certificates is large for each such Y.

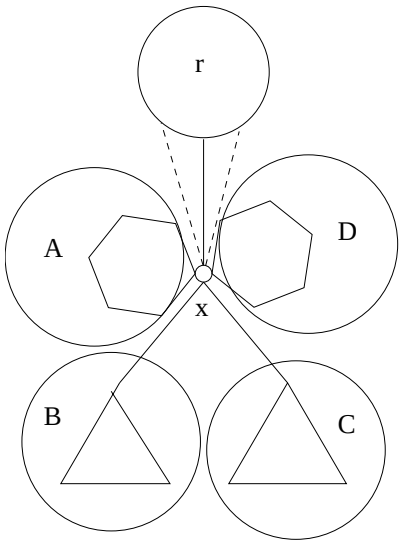

Figure 2-4: Maximal min-cuts contained in a min-cut represented by an empty cactus node

Lemma 2.35. *For any min-cut represented by an empty node in the cactus, each maximal min-cut contained in it satisfies the properties of Y in Lemma 2.34.*

Proof. The maximal min-cuts contained in a min-cut X represented by an empty node are the subtrees and cycles below the node. (For an example, see Figure 2-4, where A, B, C, and D are the maximal min-cuts contained in the min-cut represented by the empty node x.). Any min-cut represented by a subtree or a cycle in the cactus does not cross any other min-cut. □

Now, we consider the second category of min-cuts, those represented by a pair of contiguous nodes on a cycle in the cactus. The cactus representation itself shows that the total weight of edges between the min-cuts represented by the cycle nodes is $\lambda/2$.

We now show that we find the minimal min-cuts of edges using the algorithm for constructing minimal min-cuts of vertices. We construct a set of $C \lg n$ graphs for a large enough constant C from the input graph G, where each edge (of weight, say w) in G is *contracted* with probability $\min(w/2\lambda, 1)$ independently in each new graph, where λ is the value of a min-cut in the input graph.

Lemma 2.36. *Let G' be any new graph produced by random contraction of edges of G. If X and Y satisfy the condition in Lemma 2.34, then with constant probability, there exists a certificate of X with exactly one endpoint in Y which is contracted in G',* and *no edge in the cut X is contracted.*

Proof. Let the certificates of X with one endpoint in Y have weights $w_1, w_2, \ldots, w_k$, where $\sum_{i=1}^{k} w_i \geq \lambda/2$ by Lemma 2.34. If there is an edge of weight $\geq 2\lambda$ in this set, then it is necessarily contracted. Thus, let us assume that $w_i < 2\lambda,\ \forall i$. Then, the

probability that none of the certificates is contracted is

$$\prod_{i=1}^{k}\left(1-\frac{w_i}{2\lambda}\right) \leq \prod_{i=1}^{k}\left(1-\frac{1}{2\lambda}\right)^{w_i} \leq \left(1-\frac{1}{2\lambda}\right)^{\lambda/2} \leq e^{-1/4}.$$

On the other hand, let the edges in cut X have weight $W_1, W_2, \ldots, W_l$, where $\sum_{i=1}^{l} W_i = \lambda$. Then, the probability that none of these edges is contracted is

$$\prod_{i=1}^{l}\left(1-\frac{W_i}{2\lambda}\right) \geq \left(1-\frac{\sum_{i=1}^{l} W_i}{2\lambda}\right) = 1/2. \qquad \square$$

The next corollary follows from the above lemma by the fact that $C \lg n$ independent instantiations of contracted graphs are used by the algorithm.

Corollary 2.37. *For each min-cut X that is represented by either an empty cactus node or a pair of adjacent nodes on a cycle in the cactus, and for each maximal min-cut Y contained in X, at least one certificate of X with exactly one endpoint in Y is contracted* and *no edge in cut X is contracted, in at least one of the $C \lg n$ contracted graphs, with high probability.*

Note that a vertex in a contracted graph represents a set of vertices and edges on them in the original graph. A disjoint-set data structure (see e.g. [24]) is used to construct each contracted graph and keep track of the composition of a vertex. Constructing each contracted graph takes $O(m\alpha(m))$ time where $\alpha()$ is the inverse Ackermann function. Further, we also keep track of the size of a contracted vertex, i.e. number of vertices contracted into the vertex. This serves to compute the sizes of min-cuts in the above algorithm. The next lemma follows from the above analysis since there are $O(\log n)$ contracted graphs.

Lemma 2.38. *The time complexity of labeling each edge with the smallest min-cut containing it in any of the contracted graphs is $O(m \log^4 n)$.*

Note that while each vertex will necessarily be given its correct label, some edges might have wrong labels (which correspond to larger min-cuts containing it) or have no label at all. We show later that the set of edges correctly labeled is sufficient.

2.3.5 Cactus construction from minimal min-cuts

Recall that T is the tree of minimal min-cuts of vertices. First, we discard all edges that are not between incomparable vertices in T in $O(m)$ time using lca queries in tree T.

Lemma 2.39. *The minimal min-cut of an edge whose endpoints are in comparable vertices in T is also the minimal min-cut of some vertex.*

Proof. Consider an edge (u, v), where $u \in v^{\downarrow}$ in T. Then, the minimal min-cut of v also contains u, which implies that all min-cuts containing v also contain u. Thus, the minimal min-cut of (u, v) is also the minimal min-cut of v. $\square$

We now describe the construction of the cactus representation from tree T. In the following discussion, a contiguous part of the cactus representing min-cuts that are subsets of a set of vertices S is referred to as the *cactus of* S. Recall from the introduction that Lemma 2.5 reduces our task to constructing the cactus for X with each Y_i contracted into a single vertex, where X is a subtree in T and Y_i are its children subtrees. This cactus will represent all the min-cuts which are contained in X but not in any Y_i. For this purpose, we need to identify the set of edges whose minimal min-cuts will be represented in the cactus for X. Let (u, v) be an edge. Let $x = lca(u, v)$ in tree T and y and z be children of x containing u and v respectively. Then, we define a function $f : E \to V \times V$ such that $f(u, v) = (y, z)$. The following lemma states that for the purpose of the construction, we can consider (u, v) to be an edge between y and z.

Lemma 2.40. *Consider an edge e between vertices u and v, where $u \perp v$ in T. The minimal min-cut of e must be contained in $x^{\downarrow}$, where $x = lca(u, v)$, and must contain $y^{\downarrow}$ and $z^{\downarrow}$, where y and z are children of x such that $u \in y^{\downarrow}$ and $v \in z^{\downarrow}$. Further, x, y, z can be identified for all edges in $O(m\alpha(m))$ time.*

Proof. The first proposition follows from the observation that $u, v \in x^{\downarrow}$ and $x^{\downarrow}$, being a minimal min-cut of a vertex, does not cross any min-cut according to Lemma 2.22. The second proposition follows from the non-crossing property of the min-cuts $y^{\downarrow}$ and $z^{\downarrow}$.

The value of x, y and z for each edge can be found using a post-order traversal maintaining a disjoint-set data structure that keeps track of all edges whose one endpoint has been encountered but the other has not. □

Thus, we can now concentrate on the following problem. We are given a set of vertices X comprising subsets $Y_1, Y_2, \ldots, Y_k$, where $\cup_{i=1}^{k} Y_i \subset X$ and $Y_i \cap Y_j = \emptyset$ for all $i \neq j$. For convenience, we assume that $Y_{k+1} = X - \cup_{i=1}^{k} Y_i$. We assume, for the purpose of this construction, that each Y_i is contracted into a single vertex. Further, we are given all edges between any pair of (contracted) vertices Y_i and Y_j and their corresponding minimal min-cut labels. Our goal is to construct a cactus on the vertices $Y_1, \ldots, Y_{k+1}$ such that all the min-cuts contained in X but not in any of the Y_is are represented by the cactus.

One further complication arises from the fact that while some edges are labeled with the minimal min-cuts containing them, other edges may have no label or incorrect labels. First, we remove all edges without a label since Corollary 2.37 ensures that each minimal min-cut has a (correctly) labeled certificate. Now, we need to distinguish between min-cuts having the correct label and those that have erroneous labels. The following property helps us make this distinction.

Lemma 2.41. *If $X_1, X_2, \ldots, X_k$ be min-cuts such that $\cup_{i=1}^{k} X_i$ is also a min-cut, then among all edges with their two endpoints in different X_is, the label corresponding to the smallest min-cut is a correct label.*

Proof. This follows from Lemma 2.36, coupled with the fact that each edge label is either correct (i.e. gives the minimal min-cut of the edge) or gives a strictly larger min-cut containing both endpoints of the edge. □

During the construction, we will *mark* any node for which the cactus has not been constructed yet. Initially, all the Y_is are unmarked. Now, let (Y_i, Y_j) be the edge with the smallest label. We introduce a new node a as a parent of Y_i and Y_j and mark a. We move all edges with exactly one endpoint in $Y_i \cup Y_j$ to a and remove all edges between Y_i and Y_j. We then move on to the next smallest label among the surviving edges. The previous lemma ensures that the label we process at any stage is correct on account of being the minimum surviving label.

We now describe the construction for all the possible cases. In general, at any stage of the construction, suppose the smallest label corresponds to an edge (a, b). If both a and b are unmarked nodes (case (a)), then we have already constructed the cactus for a and b; so we can assume that a and b are singleton vertices. In this case, we simply introduce a new node c which is the parent of a and b, and mark the new node. Now, suppose a is a marked node but b is not. Then, we introduce a new node c and make it the parent of b. However, the relationship between c and a is not clear at this stage. There are three possibilities: either c overlaps a, or it is the same cut as a, or it is the parent of a. To distinguish between these possibilities, we run a set of containment queries, each of which can be answered in $O(1)$ time using lca values in the spanning trees. First, we check if $b \in a$; if so, then $c = a$ (case (b)). In that case, we add b as a child of a and remove c; a remains marked. Suppose the above check fails. Then, a and c are not the same min-cut. Now, if a has more than 2 children, then a has no siblings (case (c)); so, c must be the parent of a. We unmark a and mark c. On the other hand, if a has exactly 2 children x and y, then we check if $x \in c$ and $y \in c$. If both $x, y \in c$, then c is the parent of a (case (d)). In this case, if a has no sibling, we unmark a, mark c and add c as the parent of a. Otherwise, if a has siblings (case (e)), we mark c, remove a and its siblings, connect the children of the siblings of a (including the children of a) in a chain and close the chain using c to form a cycle; also, c is marked. The final possibility is that x is contained in c, but y is not (case (f)). In this case, c is marked and added as a sibling of a containing x and b. If both a and b are marked, then we need to run the above checks for both a and b.

In Figure 2-5, we show the various possibilities. The marked nodes are dark. The cases are (a) a and b are both unmarked, (b-f) a is marked and b is unmarked; (b) $b \in a$, (c) a has more than 2 children, (d) a has no sibling, (e) a has siblings but both of a's children are contained in c, and (f) a has siblings and $x \in c$ while $y \notin c$.

When we are left with no edges, we have found all the min-cuts contained in X. At this stage, we can have two situations. If the node representing X in the cactus has more than one cycle/subtree below it, then X is the minimal min-cut for all the edges between these cycles/subtrees. In this case, there will be a single empty node at the highest level of the cactus formed. We replace this empty node with the node representing X. The other possibility is that the node representing X has exactly one cycle/subtree below it in the cactus. If it has exactly one subtree below it, then there is no edge with endpoints that are incomparable in T and have X as their lca. If it has exactly one cycle below it, then there will be a cycle at the highest level of the cactus, where the last node added to the cycle is an empty node. In this case, we replace this empty node by the node representing X. This completes the construction

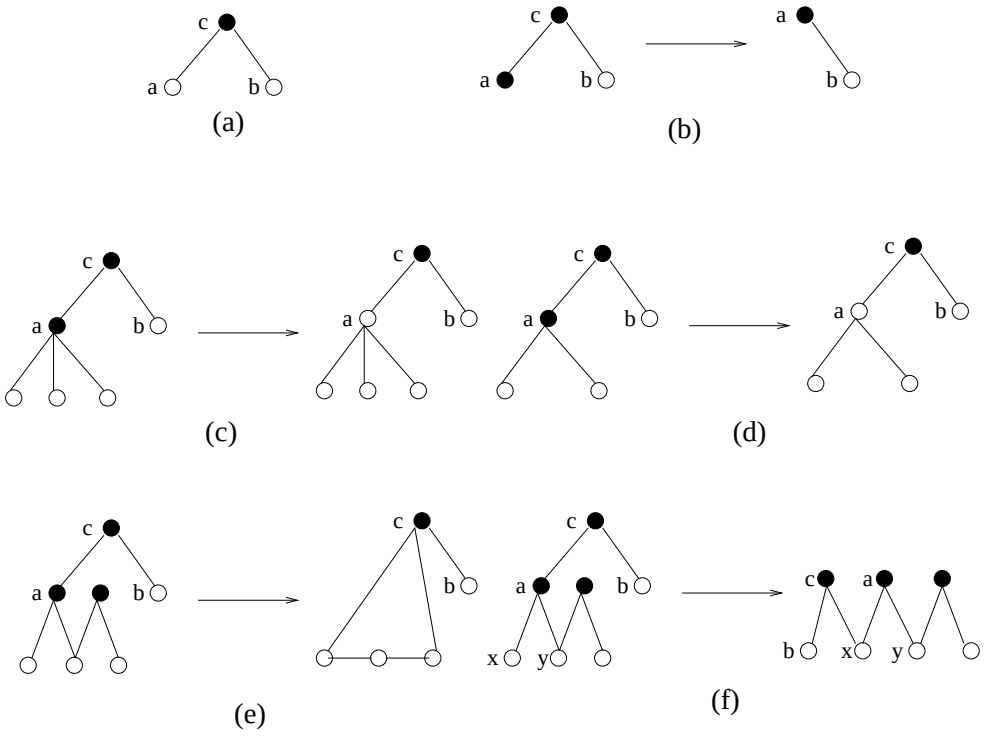

Figure 2-5: Various cases in the cactus construction algorithm

of the cactus of X.

Lemma 2.42. *The time complexity of constructing the cactus from the tree of minimal min-cuts of vertices and the minimal min-cut labels on edges is $O(m\alpha(m))$.*

Proof. The lemma follows from the observation that we use a constant number of disjoint-set operations per edge. □

Combining Lemmas 2.31, 2.38, and 2.42 yields Theorem 2.4.

2.4 Concluding Remarks

We have presented an $\tilde{O}(m)$ algorithm for constructing a cactus representation of a graph. An important open question is whether there exists an $\tilde{O}(m)$ time computable certificate for this problem. Even the potentially simpler question of whether there exists an $\tilde{O}(m)$ time computable certificate for a single min-cut (which would lead to a near-linear time Las Vegas/deterministic algorithm for finding a min-cut) is also open.

2.5 Notes

This chapter is based on joint work with David R. Karger. A preliminary version of this work appeared in the *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms*, 2009 [59].

Chapter 3

Cut Sparsification

Can *any* dense graph be approximated by a sparse graph? Surprisingly, the answer is a resounding "yes", under a variety of notions of approximation. For example, given any undirected graph, there are sparse subgraphs that approximate *all* pairwise distances up to a multiplicative and/or additive error, *every* cut to an arbitrarily small multiplicative error, *every* eigenvalue to an arbitrarily small multiplicative error and so on. Such approximations are a cornerstone of numerous important results in theoretical computer science.

In this chapter, we consider the problem of *cut sparsification*, i.e. approximating every cut arbitrarily well. This problem was originally studied by Karger [57] and Benczúr and Karger [11], who proved the existence of cut sparsifiers for any graph and gave an efficient algorithm for their construction. Since then, other cut sparsification schemes have been proposed and stronger notions of sparsification called spectral sparsification have emerged. In this chapter, we give a general sampling framework for cut sparsification that simplifies, unifies, and improves upon previous cut sparsification results, resolves a conjecture of Benczúr and Karger on sampling using edge connectivities, and leads to faster cut sparsification algorithms.

3.1 Background

Cut sparsification was introduced by Benczúr and Karger [11] as a means to accelerate connectivity algorithms.

The Cut Sparsification Problem

The input comprises a (weighted/unweighted) graph $G = (V, E)$ and an error parameter ϵ. The goal is to output a weighted graph $G_\epsilon = (V, F)$ such that the value of every cut in H is within a multiplicative factor $(1 \pm \epsilon)$ of the value of the corresponding cut in G.

The graph G_ϵ is called a *cut sparsifier* of G, which will often be abbreviated as $G_\epsilon \in (1 \pm \epsilon)G$. Cut sparsification is frequently used a pre-processing step in connec-

tivity algorithms so that the algorithms run on graphs containing fewer edges and are therefore faster.

Spielman and Teng [88] realized that a stronger notion of sparsification (that they called *spectral sparsification*) would be useful for efficiently solving systems of linear equations defined by Laplacian matrices. To describe spectral sparsification, we need to define the Laplacian matrix of a graph.

Definition 3.1. *Let $G = (V, E)$ be an undirected graph where w_e is the weight of edge e. Then, the Laplacian matrix of G is defined as*

$$L(G)_{u,v} = \begin{cases} \sum_{e=(u,t)\in E} w_e & \text{if } u = v \\ -w_e & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

The Spectral Sparsification Problem

The input comprises a (weighted/unweighted) graph $G = (V, E)$ and an error parameter ϵ. The goal is to output a weighted graph $G_\epsilon = (V, F)$ such that for every n-dimensional vector $\mathbf{x}$,

$$\mathbf{x}^T L(G_\epsilon)\mathbf{x} \in (1 \pm \epsilon)\mathbf{x}^T L(G)\mathbf{x}.$$

The graph G_ϵ is called a *spectral sparsifier* of G. Note that a spectral sparsifier is also a cut sparsifier, since spectral sparsification yields cut sparsification when $\mathbf{x}$ is constrained to be a boolean vector.

3.1.1 Connectivity Parameters

In this chapter, we will use several connectivity parameters, of which edge connectivity is perhaps the most natural.

Definition 3.2. *For any pair of vertices u and v, the* edge connectivity *between u and v, denoted k_{uv}, is defined as the minimum value of a cut that separates u and v. The* connectivity *of edge $e = (u, v)$, denoted k_e, is defined as k_{uv}.*

Benczúr and Karger introduced a new connectivity parameter called edge strength and used it in their sparsification scheme.

Definition 3.3. *A k-*strong component *of G is a maximal k-edge-connected, vertex-induced subgraph of G. The* strength *of edge $e = (u, v)$, denoted s_e or s_{uv}, is the maximum value of k such that a k-strong component of G contains both u and v.*

Next, we define electrical resistances and conductances of edges which is useful for spectral sparsification.

Definition 3.4. *The* effective conductance *of edge $e = (u, v)$, denoted c_e or c_{st}, is the amount of current that flows when each edge e of weight w_e is viewed as a resistor of resistance $1/w_e$ and a unit voltage difference is applied between u and v. The* effective resistance *of an edge e is the reciprocal of its effective conductance.*

The following well-known property of effective resistances will be used later.

Lemma 3.5. *The effective resistance of an edge is equal to the probability that is appears in a spanning tree drawn uniformly at random from the set of spanning trees of the graph.*

Nagamochi and Ibaraki [75, 74] introduced a simple graph partitioning scheme for estimating connectivities that leads to a new connectivity parameter called Nagamochi-Ibaraki (NI) indices.

Definition 3.6. *A set of edge-disjoint spanning forests $T_1, T_2, \ldots, T_k$ of a graph G is said to be a* NI forest *packing if T_i is a spanning forest on the edges left in G after removing those in $T_1, T_2, \ldots, T_{i-1}$. For weighted graphs, an edge with weight w_e must appear in w_e contiguous forests. The* NI index *of edge e, denoted ℓ_e, is the index of the (last, if weighted) NI forest in which e appears.*

The parameters s_e, c_e, and ℓ_e are mutually incomparable; however, $k_e \geq \max(c_e, s_e, \ell_e)$ always holds.

Lemma 3.7. *Suppose edge e in an undirected graph G has edge connectivity k_e, effective conductance c_e, edge strength s_e, and NI index ℓ_e. Then, $k_e \geq \max(c_e, s_e, \ell_e)$.*

Proof. $k_e \geq s_e$ follows from the fact that the strength of an edge is equal to its connectivity in a subgraph.

Consider a cut C of weight k_e separating the terminals of edge e. We contract each side of this cut into a single vertex. In other words, we increase the conductance of each edge, other than those in C, to ∞. By Rayleigh's monotonicity principle (see e.g. [26]), the effective conductance of e does not decrease due to this transformation. Since the effective conductance of e after the transformation is k_e, $c_e \leq k_e$ in the original graph.

Note that there are ℓ_e edge-disjoint paths connecting the end-points of edge e in the first ℓ_e NI forests. It follows, by Menger's theorem (see e.g. [24]), that $k_e \geq \ell_e$. □

Further, there are known bounds on the sum of reciprocals of these connectivity parameters. The bound on edge strengths is given in [11].

Lemma 3.8 (Benczúr-Karger [11])**.** *Suppose G is an undirected graph where edge e has weight w_e and strength s_e. Then, $\sum_e \frac{w_e}{s_e} \leq n - 1$.*

The bound on edge connectivities now follows from Lemma 3.7.

Corollary 3.9. *Suppose G is an undirected graph where edge e has weight w_e and connectivity k_e. Then, $\sum_e \frac{w_e}{k_e} \leq n - 1$.*

We now show similar bounds for conductances and NI indices. The bound for conductance follows directly from Lemma 3.5.

Lemma 3.10. *Suppose G is an undirected graph where edge e has weight w_e and conductance c_e. Then, $\sum_e \frac{w_e}{c_e} = n - 1$.*

On the other hand, the bound for NI indices is slighter weaker and follows from a counting argument.

Lemma 3.11. *Suppose G is an undirected graph and let $T_1, T_2, \ldots, T_k$ be a NI forest packing where edge e has weight w_e and NI index ℓ_e. Then, $\sum_e \frac{w_e}{\ell_e} = O(n \log n)$.*

Proof. Since ℓ_e is the *last* index of a forest that contains a copy of e, we can upper bound $\sum_e \frac{w_e}{\ell_e}$ by treating edge e as a set of w_e distinct parallel edges, each having an NI index equal to the NI forest it belongs to. Then, NI forest T_i contributes at most $(n-1)/i$ to the sum, and the overall bound follows by summing over all i. (Since all edge weights are polynomial in n, the number of NI forests in any NI forest packing is also polynomial in n.) □

3.1.2 Edge Compression

A key idea in cut sparsification is that of edge compression.

Definition 3.12. *An edge e is said to be* compressed *with probability p_e if the edge is sampled with probability p_e and if selected, it is given a weight of $1/p_e$ in the output.*

Note that the expected weight of an edge after compression is equal to its weight of 1 before compression. However, the variance of the edge weight after compression depends on the probability p_e.

3.1.3 History

We will describe all the previous non-algorithmic results in sparsification for unweighted input graphs. This is wlog since an edge of (integer) weight w is equivalent to w parallel edges. However, such a substitution affects algorithmic performance; hence, for algorithmic results we will distinguish between unweighted and weighted input graphs.

The first result in cut sparsification was obtained by Karger [57] who proposed a uniform compression of edges.

Theorem 3.13 (Karger [57])**.** *Let G_ϵ be obtained from an unweighted graph G by independently compressing edge e with probability $p = \min(\rho/\lambda, 1)$, where $\rho = 3(d+2)\ln n/\epsilon^2$ and λ is the value of a minimum cut in G. Then, G_ϵ contains $O\left(\frac{m \log n}{\lambda \epsilon^2}\right)$ edges in expectation and, $G_\epsilon \in (1 \pm \epsilon)G$ with probability at least $1 - n^{-d}$.*

Karger also gave an $O(m)$-time implementation of this compression scheme for weighted graphs. Clearly, this theorem is weak if the minimum cut in G is small. Benczúr and Karger [11] improved this theorem by using a non-uniform compression of edges to show that that for every graph G, there exists a cut sparsifier H containing only $O(n \log n/\epsilon^2)$ edges.

Theorem 3.14 (Benczúr-Karger [11]). *Let G_ϵ be obtained from an unweighted graph G by independently compressing edge e with probability $p_e = \min(\rho/s_e, 1)$, where $\rho = 16(d+2)\ln n/\epsilon^2$. Then, G_ϵ contains $O(n \log n/\epsilon^2)$ edges in expectation, and $G_\epsilon \in (1 \pm \epsilon)G$ with probability at least $1 - n^{-d}$.*

Benczúr and Karger also gave an efficient randomized algorithm to construct a cut sparsifier containing $O(n \log n/\epsilon^2)$ edges in expectation. This algorithm runs in $O(m \log^2 n)$ time if G is unweighted and $O(m \log^3 n)$ time if G is weighted. They also conjectured that replacing edge strengths by edge connectivity in their compression scheme will lead to significant simplification.

As noted earlier, Spielman and Teng [88] introduced spectral sparsification as a generalization of cut sparsification and proved the existence of spectral sparsifiers containing $O(n \log^{O(1)} n/\epsilon^2)$ edges for every graph. This was improved by Spielman and Srivastava [87] who obtained spectral sparsifiers containing $O(n \log n/\epsilon^2)$ edges.

Theorem 3.15 (Spielman-Srivastava [87]). *Let G_ϵ be obtained from an unweighted graph G by independently compressing edge e with probability $p_e = \min(\rho/c_e, 1)$, where $\rho = C \ln n/\epsilon^2$ for a large enough constant C. Then, G_ϵ contains $O(n \log n/\epsilon^2)$ edges in expectation, and G_ϵ is a spectral sparsifier of G with constant probability.*

Spielman and Srivastava [87] also gave an efficient algorithm to construct a spectral sparsifier with $O(n \log n/\epsilon^2)$ edges in expectation; using later improvements to linear system solvers [69, 70, 68], the best algorithm for producing a spectral sparsifier containing $O(n \log n/\epsilon^2)$ edges now runs in $O(\min(m \log^2 n, m \log n + n \log^5 n))$ time (ignoring $\log \log n$ factors).

Further improvement in spectral sparsification was achieved by Batson *et al* who showed the existence of spectral sparsifiers containing (the optimal) $O(n/\epsilon^2)$ edges for every graph. They also gave a deterministic algorithm for constructing such spectral sparsifiers in $O(n^3 m)$ time [10].

Recently, connections between spectral sparsifiers and graph spanners [55], and variants of spectral sparsification where specific subgraphs need to be retained [65] have been studied. Both cut sparsification [2, 37, 3, 4, 38] and spectral sparsification [63] have also been studied recently in the semi-streaming model.

3.2 Our Contributions

3.2.1 A General Sparsification Framework

We propose a general sparsification framework and set out sufficient conditions for a sampling scheme to result in cut sparsifiers. In describing the framework, we will assume that the input graph G is unweighted (allowing for parallel edges). Let G_ϵ be obtained from G by independently compressing edge e with probability $p_e = \min\left(\frac{96\alpha \ln n}{0.38\lambda_e \epsilon^2}, 1\right)$, where α is independent of e and $\lambda_e \leq 2^n - 1$ for all edges. We describe below a sufficient condition on the values of α and λ_e's for G_ϵ to be a cut sparsifier.

To describe this sufficient condition, we partition the edges in G according to the value of λ_e into sets $F_0, F_1, \ldots, F_k$ where $k = \lfloor \lg \max_{e \in E}\{\lambda_e\} \rfloor \leq n-1$ and $F_i = \{e : 2^i \leq \lambda_e \leq 2^{i+1} - 1\}$. We will obtain concentration bounds for each F_i separately since edges in any F_i have roughly the same sampling probability in the compression scheme. Ideally, we would like to bound the error due to compression of edges in F_i by a multiplicative factor of the size of F_i. Then, summing over all F_i's would immediately yield a concentration bound on the entire graph since the F_i's are disjoint. However, it might so happen that the number of edges in a particular F_i is small, yet these edges have a low sampling probability. This is inconvenient since we cannot hope to bound the error due to such an F_i by a multiplicative factor of the size of F_i. To overcome this problem, we define a subgraph G_i of G (with edges replicated, if required) for each F_i such that edges in F_i are well-connected in G_i and therefore the error due to F_i can be bounded by a multiplicative factor of the size of G_i. The goal then becomes one of choosing G_i's such that no edge in G is replicated a large number of times across all the G_i's. This ensures that the sum of the individual error bounds on the F_i's in terms of the G_i's can be expressed as a multiplicative error on the entire graph G.

Formally, let $\mathcal{G} = \{G_i = (V, E_i) : 1 \leq j \leq k\}$ be a set of subgraphs of G such that $F_i \subseteq E_i$ for every i. For a set of parameters $\Pi = (\pi_0, \pi_1, \ldots, \pi_k)$, $\mathcal{G}$ is said to be a (Π, α)-*certificate* corresponding to the above choice of α and λ_e's if the following properties are satisfied for all $i \geq 0$:

- **(Π-connectivity.)** The connectivity of any edge $e \in F_i$ in graph G_i is at least π_i.
- **(α-overlap.)** For any cut C, $\sum_{i=0}^{k} \frac{e_i^{(C)} 2^{i-1}}{\pi_i} \leq \alpha w_C$, where w_C and $e_i^{(C)}$ denote the value of cut C in graphs G and G_i respectively.

Theorem 3.16 describes the properties of such a sampling scheme.

Theorem 3.16. *If there exists a (π, α)-certificate for a particular choice of α and λ_e's, then $G_\epsilon \in (1 \pm \epsilon)G$ with probability at least $1 - 4/n$. Furthermore G_ϵ has $O\left(\frac{\alpha \log n}{\epsilon^2} \sum_{e \in E} \frac{1}{\lambda_e}\right)$ edges in expectation.*

3.2.2 Applications of the Sparsification Framework

Our first application of the sparsification framework is to show that compressing by edge connectivities yields cut sparsifiers, thereby resolving the conjecture of Benczúr and Karger.

Theorem 3.17. *Let G_ϵ be obtained from an unweighted graph G by independently compressing edge e with probability $p_e = \min(\rho/k_e, 1)$, where $\rho = Cd \ln^2 n/\epsilon^2$ for a large enough constant C. Then, G_ϵ contains $O(n \log^2 n/\epsilon^2)$ edges in expectation, and $G_\epsilon \in (1 \pm \epsilon)G$ with probability at least $1 - n^{-d}$.*

The next three corollaries of this theorem follow from Lemma 3.7 and Lemmas 3.8, 3.10, and 3.11.

Corollary 3.18. *Let G_ϵ be obtained from an unweighted graph G by independently compressing edge e with probability $p_e = \min(\rho/s_e, 1)$, where $\rho = Cd\ln^2 n/\epsilon^2$ for a large enough constant C. Then, G_ϵ contains $O(n\log^2 n/\epsilon^2)$ edges in expectation, and $G_\epsilon \in (1\pm\epsilon)G$ with probability at least $1 - n^{-d}$.*

Recall that the corresponding result of Benczúr and Karger [11] (Theorem 3.14) is stronger than this result since it produces sparsifiers containing $O(n\log n/\epsilon^2)$ edges in expectation. We show later that we can match the Benczúr-Karger bound by using our sparsification framework directly.

Corollary 3.19. *Let G_ϵ be obtained from an unweighted graph G by independently compressing edge e with probability $p_e = \min(\rho/c_e, 1)$, where $\rho = Cd\ln^2 n/\epsilon^2$ for a large enough constant C. Then, G_ϵ contains $O(n\log^2 n/\epsilon^2)$ edges in expectation, and $G_\epsilon \in (1\pm\epsilon)G$ with probability at least $1 - n^{-d}$.*

As we noted earlier, the main caveat is that Spielman and Srivastava (Theorem 3.15) prove spectral sparsification whereas we do not.

Corollary 3.20. *Let G_ϵ be obtained from an unweighted graph G by independently compressing edge e with probability $p_e = \min(\rho/\ell_e, 1)$, where $\rho = Cd\ln^2 n/\epsilon^2$ for a large enough constant C. Then, G_ϵ contains $O(n\log^3 n/\epsilon^2)$ edges in expectation, and $G_\epsilon \in (1\pm\epsilon)G$ with probability at least $1 - n^{-d}$.*

As in the case of edge strengths, we will show later that this result can be improved by applying the sparsification framework directly to obtain the following theorem. We state this theorem for weighted graphs since we will use this theorem for algorithmic applications.

Theorem 3.21. *Let G_ϵ be obtained from a weighted graph G by independently compressing edge e with probability $p_e = \min(\rho/\ell_e, 1)$, where $\rho = Cd\ln n/\epsilon^2$ for a large enough constant C. Then, G_ϵ contains $O(n\log^2 n/\epsilon^2)$ edges in expectation, and $G_\epsilon \in (1\pm\epsilon)G$ with probability at least $1 - n^{-d}$.*

3.2.3 Sparsification Algorithms

Our framework yields sparsification algorithms that are not only simpler, but also faster. Nagamochi and Ibaraki showed that a NI forest packing can be constructed in $O(m)$-time for unweighted graphs [75], and $O(m + n\log n)$-time for weighted graphs [74]. For weighted graphs, note that sampling an edge e involves the generation of a binomial random variable with parameters w_e and p_e. This can be done in $O(w_e p_e)$ time (see e.g. [54]), and therefore $O(\sum_{e\in E} w_e p_e)$ time overall for all edges. We can now use Theorem 3.21 to claim that the time complexity of sampling all edges is $O(n\log^2 n) = O(m)$. (Note that if $m = O(n\log^2 n)$, then we can retain all edges; therefore, we assume wlog that $n\log^2 n = O(m)$.) Coupled with Theorem 3.21, we get the following theorem.

Theorem 3.22. *For any input graph G and any constants $\epsilon \in (0,1), d > 0$, there is a randomized algorithm that runs in $O(m)$-time and produces a graph G_ϵ containing $O(n \log^2 n/\epsilon^2)$ edges in expectation, where $G_\epsilon \in (1 \pm \epsilon)G$ with probability at least $1 - n^{-d}$.*

The cut sparsifier produced by this algorithm contains $O(n \log^2 n/\epsilon^2)$ edges in expectation, which is a factor of $\log n$ greater than that produced by previous algorithms of Benczúr and Karger. However, the output of this algorithm can be post-processed using the previous algorithm for weighted graphs to obtain a cut sparsifier containing $O(n \log n/\epsilon^2)$ edges. Recall that the best previously known algorithm runs in $O(m \log^3 n)$ time for weighted graphs.

Corollary 3.23. *For any input graph G and any constants $\epsilon \in (0,1), d > 0$, there is a randomized algorithm that runs in $O(m + n \log^5 n)$-time, and produces a graph G_ϵ containing $O(n \log n/\epsilon^2)$ edges in expectation, where $G_\epsilon \in (1 \pm \epsilon)G$ with probability at least $1 - n^{-d}$.*

We give a new algorithm for unweighted graphs that reduces the running time to the optimal $O(m)$ without increasing the number of edges in the cut sparsifier.

Theorem 3.24. *For any unweighted input graph G and any constants $\epsilon \in (0,1), d > 0$, there is a randomized algorithm that runs in $O(m)$-time and produces a graph G_ϵ containing $O(n \log n/\epsilon^2)$ edges in expectation, where $G_\epsilon \in (1 \pm \epsilon)G$ with probability at least $1 - n^{-d}$.*

Roadmap

This chapter is organized as follows. Section 3.3 and Section 3.4 give proofs of a modified Chernoff bound and a cut counting theorem respectively, which are the two main technical tools that we use to prove properties of the sparsification framework (Theorem 3.16) in Section 3.5. Applications of the framework to various sampling schemes appears in Section 3.6. Finally, we present sparsification algorithms in Section 3.7.

3.3 Modified Chernoff Bounds

We will use the following form of Chernoff bounds for a set of random variables with non-uniform sampling probabilities but uniform expectation.

Theorem 3.25. *Let $X_1, X_2, \ldots, X_n$ be n independent random variables such that X_i takes value $1/p_i$ with probability p_i and 0 otherwise. Then, for any p such that $p \leq p_i$ for each i, any $\epsilon \in (0,1)$, and any $N \geq n$,*

$$\mathbb{P}\left[\left|\sum_{i=1}^{n} X_i - n\right| > \epsilon N\right] < 2e^{-0.38\epsilon^2 pN}.$$

To prove the theorem, we need the following facts.

Fact 3.26. *Let $f(x) = x - (1+x)\ln(1+x)$ for $x > 0$. If $\alpha = 1 - 2\ln 2$, then*

$$f(x) \leq \begin{cases} \alpha x^2 & \text{if } x \in (0,1) \\ \alpha x & \text{if } x \geq 1. \end{cases}$$

Proof. First, consider $x \in (0,1)$. Define

$$g(x) = \frac{f(x)}{x^2} = \frac{1}{x} - \left(\frac{1}{x} + \frac{1}{x^2}\right)\ln(1+x).$$

We note that $g(x)$ is an increasing function of x for $x \in (0,1)$. Further, at $x = 1$, $g(x) = \alpha$. Thus, $f(x) < \alpha x^2$ for $x \in (0,1)$.

Now, consider $x \geq 1$. Define

$$h(x) = \frac{f(x)}{x} = 1 - \left(1 + \frac{1}{x}\right)\ln(1+x).$$

We can verify that $h(x)$ is a decreasing function of x for $x \geq 1$. Further, at $x = 1$, $h(x) = \alpha$. Thus, $f(x) \leq \alpha x$ for $x \geq 1$. □

Fact 3.27. *Let $f(x) = -x - (1-x)\ln(1-x)$ for $x \in (0,1)$. Then,*

$$f(x) \leq -x^2/2$$

Proof. Using the Taylor expansion of $\ln(1-x)$, we have

$$-x - (1-x)\ln(1-x) = -x + (1-x)(x + \frac{x^2}{2} + \frac{x^3}{3} + \ldots) \leq -x^2/2.$$

□

Fact 3.28. *The functions $f(x) = x(e^{t/x} - 1)$ and $g(x) = -x(1 - e^{-t/x})$ are non-increasing in the range $x \in (0,1]$ for any $t > 0$.*

Proof. We have

$$\frac{df}{dx} = e^{t/x}\left(1 - \frac{t}{x}\right) - 1 \leq e^{t/x}e^{-t/x} - 1 = 0.$$

Similarly

$$\frac{dg}{dx} = e^{-t/x}\left(1 + \frac{t}{x}\right) - 1 \leq e^{-t/x}e^{t/x} - 1 = 0.$$ □

We use the facts that we derived to prove the following lemmas.

Lemma 3.29. *Let $X_1, X_2, \ldots, X_n$ be a set of n independent random variables such that X_i takes value $1/p_i$ with probability p_i and 0 otherwise. Then, for any $p \leq p_i$ for each i, and for any $\epsilon > 0$,*

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i > (1+\epsilon)n\right] < \begin{cases} e^{-0.38\epsilon^2 pn} & \text{if } 0 < \epsilon < 1 \\ e^{-0.38\epsilon pn} & \text{if } \epsilon \geq 1. \end{cases}$$

Proof. For any $t > 0$,

$$\begin{aligned}
\mathbb{P}\left[\sum_{i=1}^{n} X_i > (1+\epsilon)n\right] &= \mathbb{P}\left[e^{t\sum_i X_i} > e^{t(1+\epsilon)n}\right] \\
&< \frac{\mathbb{E}\left[e^{t\sum_i X_i}\right]}{e^{t(1+\epsilon)n}} \quad \text{(by Markov bound (see e.g. [73]))} \\
&= \prod_{i=1}^{n} \frac{\mathbb{E}\left[e^{tX_i}\right]}{e^{t(1+\epsilon)n}} \quad \text{(by independence of } X_1, X_2, \ldots, X_n) \\
&= \prod_{i=1}^{n} \frac{p_i e^{t/p_i} + 1 - p_i}{e^{t(1+\epsilon)n}} \\
&= \prod_{i=1}^{n} \frac{1 + p_i(e^{t/p_i} - 1)}{e^{t(1+\epsilon)n}} \\
&\leq \exp\left(\sum_{i=1}^{n} p_i(e^{t/p_i} - 1) - t(1+\epsilon)n\right) \quad \text{(since } 1 + x \leq e^x,\ \forall x \geq 0).
\end{aligned}$$

Since $p_i \leq p$ for each i, we use Fact 3.28 to get

$$\sum_{i=1}^{n}(p_i(e^{t/p_i} - 1)) \quad \leq \quad \sum_{i=1}^{n}(p(e^{t/p} - 1)) \quad = \quad np(e^{t/p} - 1).$$

Thus,

$$\mathbb{P}\left[\sum_{i} X_i > (1+\epsilon)n\right] < \exp(np(e^{t/p} - 1) - t(1+\epsilon)n).$$

Setting $t = p\ln(1+\epsilon)$, we get

$$\mathbb{P}\left[\sum_{i} X_i > (1+\epsilon)n\right] < \left(\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right)^{pn}.$$

Since $1 - 2\ln 2 < -0.38$, we use Fact 3.26 to conclude that

$$\mathbb{P}\left[\sum_{i} X_i > (1+\epsilon)n\right] < \begin{cases} e^{-0.38\epsilon^2 pn} & \text{if } 0 < \epsilon < 1 \\ e^{-0.38\epsilon pn} & \text{if } \epsilon \geq 1. \end{cases}$$ □

Lemma 3.30. *Let $X_1, X_2, \ldots, X_n$ be a set of n independent random variables such that X_i takes value $1/p_i$ with probability p_i and 0 otherwise. Then, for any $p \leq p_i$ for each i, and for any $\epsilon > 0$,*

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i < (1-\epsilon)n\right] \begin{cases} < e^{-0.5\epsilon^2 pn} & \text{if } 0 < \epsilon < 1 \\ = 0 & \text{if } \epsilon \geq 1. \end{cases}$$

Proof. For $\epsilon \geq 1$,

$$\mathbb{P}\left[\sum_i X_i < (1-\epsilon)n\right] \leq \mathbb{P}\left[\sum_i X_i < 0\right] = 0.$$

Now, suppose $\epsilon \in (0,1)$. For any $t > 0$,

$$\begin{aligned}
\mathbb{P}\left[\sum_i X_i < (1-\epsilon)n\right] &= \mathbb{P}\left[e^{-t\sum_i X_i} > e^{-t(1-\epsilon)n}\right] \\
&< \frac{\mathbb{E}\left[e^{-t\sum_i X_i}\right]}{e^{-t(1-\epsilon)n}} \quad \text{(by Markov bound)} \\
&= \prod_{i=1}^{n} \frac{\mathbb{E}\left[e^{-tX_i}\right]}{e^{-t(1-\epsilon)n}} \quad \text{(by independence of } X_1, X_2, \ldots, X_n) \\
&= \prod_{i=1}^{n} \frac{p_i e^{-t/p_i} + 1 - p_i}{e^{-t(1-\epsilon)n}} \\
&= \prod_{i=1}^{n} \frac{1 - p_i(1 - e^{-t/p_i})}{e^{-t(1-\epsilon)n}} \\
&\leq \exp(\sum_{i=1}^{n} -p_i(e^{-t/p_i} - 1) - t(1-\epsilon)n) \quad (\text{since } 1 - x \leq e^{-x},\ \forall x \geq 0).
\end{aligned}$$

Since $p_i \geq p$ for each i, we use Fact 3.28 to get

$$\sum_{i=1}^{n}(-p_i(1 - e^{-t/p_i})) \leq \sum_{i=1}^{n}(-p(1 - e^{-t/p})) = -np(1 - e^{-t/p}).$$

Thus,

$$\mathbb{P}\left[\sum_i X_i < (1-\epsilon)n\right] < \exp(-np(1 - e^{-t/p}) - t(1-\epsilon)n).$$

Setting $t = -p\ln(1-\epsilon)$, we get

$$\mathbb{P}\left[\sum_i X_i < (1-\epsilon)n\right] < \left(\frac{e^{\epsilon}}{(1-\epsilon)^{1-\epsilon}}\right)^{-pn} \leq e^{-0.5\epsilon^2 pn}.$$

The last inequality follows from Fact 3.27. □

We now prove Theorem 3.25 using the above lemmas.

Proof of Theorem 3.25. Let $\delta = \frac{\epsilon N}{n}$. First, consider $\delta \in (0,1)$. From Lemmas 3.29

and 3.30, we conclude that

$$\begin{aligned}\mathbb{P}\left[\left|\sum_{i=1}^{n} X_i - n\right| > \epsilon N\right] &= \mathbb{P}\left[\left|\sum_{i=1}^{n} X_i - n\right| > \delta n\right] \\ &< 2e^{-0.38\delta^2 pn} \\ &= 2e^{-0.38\epsilon^2 pN(N/n)} \\ &\leq 2e^{-0.38\epsilon^2 pN} \quad (\text{since } N \geq n).\end{aligned}$$

Now, consider $\delta \geq 1$. From Lemmas 3.29 and 3.30, we conclude that

$$\begin{aligned}\mathbb{P}\left[\left|\sum_{i=1}^{n} X_i - n\right| > \epsilon N\right] &= \mathbb{P}\left[\left|\sum_{i=1}^{n} X_i - n\right| > \delta n\right] \\ &< e^{-0.38\delta pn} \\ &= e^{-0.38\epsilon pN} \\ &\leq e^{-0.38\epsilon^2 pN}.\end{aligned}$$

The last inequality follows from the fact that $\epsilon < 1$. □

3.4 Counting Cut Projections

Our next key ingredient is a natural generalization of the following cut counting theorem due to Karger [56, 60].

Theorem 3.31 (Karger [56, 60])**.** *For any $\alpha \geq 1$, the number of cuts of value at most $\alpha\lambda$ in a graph is at most $n^{2\alpha}$, where λ is the minimum value of a cut in the graph.*

To state our generalization, we need some definitions.

Definition 3.32. *An edge is said to be* k-heavy *if its connectivity is at least k; otherwise, it is said to be* k-light*. The* k-projection *of a cut is the set of k-heavy edges in it.*

Intuitively, we show that for a larger value of α, the large number of cuts of size $\alpha\lambda$ predicted by Karger's theorem arises from *many* distinct k-projections of these cuts for small values of k, whereas there are *few* distinct k-projections of these cuts for large values of k.

Theorem 3.33. *For any $k \geq \lambda$ and any $\alpha \geq 1$, the number of distinct k-projections in cuts of of value at most αk in a graph is at most $n^{2\alpha}$, where λ is the minimum value of a cut in the graph.*

Before proceeding further, we need to introduce the splitting-off operation.

Definition 3.34. *The* splitting-off *operation replaces a pair of edges (u, v) and (v, w) with the edge (u, w), and is said to be* admissible *if it does not change the edge connectivity between any two vertices $s, t \neq v$.*

A key technical tool in our proof of Theorem 3.33 is a theorem of Mader [72] on the feasibility of the splitting-off operation, whose statement requires us to define cut edges first.

Definition 3.35. *A* cut edge *is an edge whose removal separates a connected graph into two disconnected components.*

Theorem 3.36 (Mader [72])**.** *Let $G = (V, E)$ be a connected graph where $v \in V$ is a vertex that has degree $\neq 3$ and is not incident to any cut edge. Then, there is a pair of edges (u, v) and (v, w) such that their splitting-off is admissible.*

Since uniformly scaling edge weights does not affect the conditions of Theorem 3.33, we may assume that G is Eulerian and does not have any cut edge. Therefore, Theorem 3.36 applies to our graph. The operation of splitting-off of edges can also be extended to vertices.

Definition 3.37. *The* splitting-off *operation on an even-degree vertex v repeatedly performs admissible splitting-off operations on the edges incident on v until v becomes an isolated vertex.*

Note that Theorem 3.36 implies that we can split-off any vertex in a Eulerian graph with no cut edge.

Our proof strategy for Theorem 3.33 is to give an algorithm (Algorithm 1) with the following property, which immediately implies Theorem 3.33. Here, $q(F)$ denotes the minimum value of a cut whose k-projection is F.

Lemma 3.38. *For any k-projection F with $q(F) \leq \alpha k$, Algorithm 1 outputs F with probability at least $n^{-2\alpha}$.*

To describe Algorithm 1, we need some an additional definition.

Definition 3.39. *A vertex is said to be* k-heavy *if it is incident to a k-heavy edge; otherwise, it is* k-light*.*

As a pre-processing step, Algorithm 1 splits-off all k-light vertices in G. Since Algorithm 1 preserves Eulerianness in G and does not introduce any cut edge, this step (and subsequent splitting-off operations) is feasible. Next, it performs a set of iterations, where in each iteration, it contracts an edge selected uniformly at random (where an edge e of weight w_e is replaced by w_e parallel edges), removes all self-loops, and splits-off any vertices that may have become k-light as a result of the contraction. The iterations terminate when at most $\lceil 2\alpha \rceil$ vertices are left in the graph. At this point, the algorithm outputs the k-projection of a cut selected uniformly at random. Note that the algorithm adds new edges to G via the splitting-off process. All new edges are treated as k-light irrespective of their connectivity. Therefore, the k-projection of a cut that is output by the algorithm does not include any new edge.

When $k = \lambda$, there is no k-light vertex and Algorithm 1 reduces to a random contraction algorithm which was used by Karger to prove Theorem 3.31. Our main

Algorithm 1 An algorithm for proving bound on cut projections

- **procedure** Contract($G,\ k,\ \alpha$)
- **input:** A graph $G = (V, E)$, a parameter $k \geq K$ where K is the weight of a minimum cut in G, and an approximation factor α
- **output:** a k-projection
- While there exists a k-light vertex v
  - Perform admissible splitting-off at v until v becomes an isolated vertex
  - Remove v
- While there are more than $\lceil 2\alpha \rceil$ vertices remaining
  - Pick an edge e uniformly at random
  - Contract e and remove any self loops
  - While there exists a k-light vertex v
    - Perform admissible splitting-off at v until v becomes an isolated vertex
    - Remove v
- Output the k-projection of a cut selected uniformly at random

idea is that we can remove the k-light vertices while preserving the connectivities of all k-heavy edges by using the splitting-off operation.

To prove Lemma 3.38, we fix a k-projection F with $q(F) \leq \alpha k$. Note that it is sufficient to show that the following invariants hold over all iterations with probability at least $n^{-2\alpha}$ in Algorithm 1:

- **(I1)** F is a k-projection in the remaining graph,
- **(I2)** $q(F) \leq \alpha k$ (where $q(F)$ now minimizes over cuts in the remaining graph), and
- **(I3)** Every remaining k-heavy edge e has connectivity at least k.

In Algorithm 1, modifications to the graph are due to admissible splitting-offs, contraction of edges, and removal of self-loops. Clearly removing self-loops does not affect the invariants. For the splitting-off operation, we note that

- **(I1)** is preserved because we only split-off k-light edges.
- **(I2)** is preserved because splitting-off never increases the size of any cut.
- **(I3)** is preserved because we only split-off at a light vertex and the splitting-offs are admissible.

Finally, we consider edge contraction.

Lemma 3.40. *Let the number of remaining vertices be r. Assuming that the invariants hold, they will continue to hold after the contraction operation with probability at least $1 - 2\alpha/r$.*

Proof. For **(I3)**, note that since contraction does not create new cuts, the edge connectivity of an uncontracted edge cannot decrease. Now consider the graph before the contraction. Since every remaining vertex v is k-heavy, the degree of each vertex is at least k; thus the number of remaining edges is at least $kr/2$. Let C be a cut such that F is the k-projection of C and $q(F)$ is the value of C. Note that **(I1)** and **(I2)** are preserved if the contracted edge e is not in cut C. Since e is picked uniformly at random, the probability that e is in C is at most $\frac{q(F)}{kr/2} \leq 2\alpha/r$. □

Let r_i be the number of remaining vertices after the splitting-off operations in iteration i of Algorithm 1. Then, the probability that all the invariants hold throughout the execution of Algorithm 1, and F is the output is at least

$$\left(1 - \frac{2\alpha}{r_0}\right)\left(1 - \frac{2\alpha}{r_1}\right)\ldots\left(1 - \frac{2\alpha}{\lceil 2\alpha \rceil + 1}\right) 2^{-(\lceil 2\alpha \rceil - 1)} \geq n^{-2\alpha}.$$

This completes the proof of Lemma 3.38, which implies Theorem 3.33. Note that this theorem reduces to Karger's cut counting theorem by setting $k = \lambda$. Given the numerous applications of Karger's theorem, e.g. [8, 39, 57, 83], we suspect our generalization may be of further interest.

3.5 The General Sparsification Framework

In this section, we will prove Theorem 3.16. We re-use the notation defined in Section 3.2.1. Recall that the Π-connectivity property ensures that every edge in F_i has connectivity at least π_i in the subgraph $G_i = (V, E_i)$. Also, the α-overlap property ensures that for any cut C, $\sum_{i=0}^{k} \frac{e_i^{(C)} 2^{i-1}}{\pi_i} \leq \alpha w_C$, where w_C and $e_i^{(C)}$ denote the value of cut C in graphs G and G_i respectively.

We also introduce some additional notation. For any cut C, let $F_i^{(C)} = F_i \cap C$ and $E_i^{(C)} = E_i \cap C$; correspondingly, let $f_i^{(C)} = |F_i^{(C)}|$ and $e_i^{(C)} = |E_i^{(C)}|$. Also, let $\widehat{f_i^{(C)}}$ be the sum of weights of all edges in $F_i^{(C)}$ that appear in G_ϵ. Note that $\mathbb{E}[\widehat{f_i^{(C)}}] = f_i^{(C)}$. We first prove a key lemma.

Lemma 3.41. *For any fixed i, with probability at least $1 - 4/n^2$, every cut C in G satisfies*

$$\left| f_i^{(C)} - \widehat{f_i^{(C)}} \right| \leq \frac{\epsilon}{2} \max\left(\frac{e_i^{(C)} \cdot 2^{i-1}}{\pi_i \cdot \alpha}, f_i^{(C)} \right)$$

Proof. By the Π-connectivity property, any edge $e \in F_i$ is π_i-heavy in G_i for any $i \geq 0$. Therefore, $e_i^{(C)} \geq \pi_i$. Let $\mathcal{C}_{ij}$ be the set of all cuts C such that $\pi_i \cdot 2^j \leq e_i^{(C)} \leq \pi_i \cdot 2^{j+1} - 1$, $j \geq 0$. We will prove that with probability at least $1 - 2n^{-2^{j+1}}$, all cuts in $\mathcal{C}_{ij}$ satisfy the property of the lemma. The lemma then follows by using the union bound over j (keeping i fixed) since $2n^{-2} + 2n^{-4} + \ldots + 2n^{-2j} + \ldots \leq 4n^{-2}$.

Suppose $C \in \mathcal{C}_{ij}$. Let $X_i^{(C)}$ denote the set of edges in $F_i^{(C)}$ that are sampled with probability strictly less than one; correspondingly, let $x_i^{(C)} = |X_i^{(C)}|$ and let $\widehat{x_i^{(C)}}$ be the total weight of edges in $X_i^{(C)}$ in the sampled graph G_ϵ. Since edges in $F_i^{(C)} - X_i^{(C)}$ retain their weight exactly in G_ϵ, it is sufficient to show that with probability at least $1 - 2n^{-2^{j+1}}$,

$$|x_i^{(C)} - \widehat{x_i^{(C)}}| \leq \left(\frac{\epsilon}{2}\right) \max\left(\frac{e_i^{(C)} \cdot 2^{i-1}}{\pi_i \cdot \alpha}, x_i^{(C)} \right)$$

for all cuts $C \in \mathcal{C}_{ij}$. Since each edge $e \in X_i^{(C)}$ has $\lambda_e < 2^{i+1}$, we can use Theorem 3.25 with the lower bound on probabilities $p = \frac{96\alpha \ln n}{0.38 \cdot 2^{i+1} \epsilon^2}$. There are two cases. In the first case, suppose $x_i^{(C)} \leq \frac{e_i^{(C)} \cdot 2^{i-1}}{\pi_i \cdot \alpha}$. Then, for any $X_i^{(C)}$ where $C \in \mathcal{C}_{ij}$, by Theorem 3.25, we have

$$\begin{aligned}
\mathbb{P}\left[\left| x_i^{(C)} - \widehat{x_i^{(C)}} \right| > \left(\frac{\epsilon}{2}\right) \frac{e_i^{(C)} \cdot 2^{i-1}}{\pi_i \cdot \alpha} \right] &< 2e^{-0.38 \frac{\epsilon^2}{4} \left(\frac{96\alpha \ln n}{0.38 \cdot 2^{i+1} \epsilon^2}\right) \frac{e_i^{(C)} \cdot 2^{i-1}}{\pi_i \cdot \alpha}} \\
&\leq 2e^{-\frac{6 e_i^{(C)} \ln n}{\pi_i}} \\
&\leq 2e^{-6 \cdot 2^j \ln n},
\end{aligned}$$

since $e_i^{(C)} \geq \pi_i \cdot 2^j$ for any $C \in \mathcal{C}_{ij}$. In the second case, $x_i^{(C)} > \frac{e_i^{(C)} \cdot 2^{i-1}}{\pi_i \cdot \alpha}$. Then, for any

$X_i^{(C)}$ where $C \in \mathcal{C}_{ij}$, by Theorem 3.25, we have

$$\begin{aligned}\mathbb{P}\left[\left|x_i^{(C)} - \widehat{x_i^{(C)}}\right| > \left(\frac{\epsilon}{2}\right) x_i^{(C)}\right] &< 2e^{-0.38\frac{\epsilon^2}{4}\left(\frac{96\alpha \ln n}{0.38\cdot 2^{i+1}\epsilon^2}\right)x_i^{(C)}} \\ &< 2e^{-\frac{6e_i^{(C)}\ln n}{\pi_i}} \\ &\leq 2e^{-6\cdot 2^j \ln n},\end{aligned}$$

since $x_i^{(C)} > \frac{e_i^{(C)}\cdot 2^{i-1}}{\pi_i \cdot \alpha} \geq \frac{2^{i+j-1}}{\alpha}$ for any $C \in \mathcal{C}_{ij}$. Thus, we have proved that

$$\begin{aligned}\mathbb{P}\left[\left|x_i^{(C)} - \widehat{x_i^{(C)}}\right| > \left(\frac{\epsilon}{2}\right)\max\left(\frac{e_i^{(C)}\cdot 2^{i-1}}{\pi_i\cdot\alpha}, x_i^{(C)}\right)\right] &< 2e^{-6\cdot 2^j \ln n} \\ &= 2n^{-6\cdot 2^j}\end{aligned}$$

for any cut $C \in \mathcal{C}_{ij}$. Now, by the Π-connectivity property, we know that edges in $F_i^{(C)}$, and therefore those in $X_i^{(C)}$, are π_i-heavy in G_i. Therefore, by Theorem 3.33, the number of distinct $X_i^{(C)}$ sets for cuts $C \in \mathcal{C}_{ij}$ is at most $n^{2\left(\frac{\pi_i\cdot 2^{j+1}}{\pi_i}\right)} = n^{4\cdot 2^j}$. Using the union bound over these distinct $X_i^{(C)}$ edge sets, we conclude that with probability at least $1 - 2n^{-2^{j+1}}$, all cuts in $\mathcal{C}_{ij}$ satisfy the property of the lemma. □

We now use the above lemma to prove Theorem 3.16. Lemma 3.41 bounds the sampling error for a fixed i. In this theorem we bound the total error by summing over $i = 0, \ldots, k$. (Recall that $k \leq n-1$.)

Let w_C and $\widehat{w_C}$ be the weight of edges crossing a cut C in G and G_ϵ respectively. By a union bound, the conclusion of Lemma 3.41 holds for every value of i with probability at least $1 - 4/n$. Therefore

$$\sum_{i=0}^{k} |\widehat{f_i^{(C)}} - f_i^{(C)}| \leq \sum_{i=0}^{k}\left(\frac{\epsilon}{2}\right)\max\left(\frac{e_i^{(C)}\cdot 2^{i-1}}{\pi_i\cdot\alpha}, f_i^{(C)}\right)$$

for all cuts C. Then, with probability at least $1 - 4/n$,

$$\begin{aligned}
|\widehat{w_C} - w_C| &= |\sum_{i=0}^{k} \widehat{f_i^{(C)}} - \sum_{i=0}^{k} f_i^{(C)}| \\
&\leq \sum_{i=0}^{k} |\widehat{f_i^{(C)}} - f_i^{(C)}| \\
&\leq \frac{\epsilon}{2} \sum_{i=0}^{k} \max\left(\frac{e_i^{(C)} \cdot 2^{i-1}}{\pi_i \cdot \alpha}, f_i^{(C)}\right) \\
&\leq \frac{\epsilon}{2}\left(\sum_{i=0}^{k} \frac{e_i^{(C)} \cdot 2^{i-1}}{\pi_i \cdot \alpha} + \sum_{i=0}^{k} f_i^{(C)}\right) \\
&\leq \epsilon w_C,
\end{aligned}$$

since $\sum_{i=0}^{k} \frac{e_i^{(C)} \cdot 2^{i-1}}{\pi_i \cdot \alpha} \leq w_C$ by the α-overlap property and $\sum_{i=0}^{k} f_i^{(C)} \leq w_C$ since $F_i^{(C)}$'s form a partition of the edges in C.

We now prove the bound on the expected number of edges in G_ϵ. The expected number of distinct edges in G_ϵ is

$$\sum_{e \in E} (1 - (1 - p_e)^{w_e}) \leq \sum_{e} w_e p_e.$$

The bound follows by substituting the value of p_e.

This completes the proof of Theorem 3.16.

3.6 Sparsification by Edge Compression

In this section, we present edge compression schemes using various connectivity parameters and apply the sparsification framework to show that they yield cut sparsifiers.

3.6.1 Compression using Edge Connectivities

First, we use the sparsification framework to show Theorem 3.17.

For any edge $e = (u, v)$, set λ_e to its connectivity k_e. Let $\alpha = 3 + \lg n$ and $\pi_i = 2^{i-1}$. Recall that F_i is defined as the set of all edges e with $2^i \leq \lambda_e \leq 2^{i+1} - 1$. For any $i \geq 1 + \lg n$, let G_i contain all edges in NI forests $T_{2^{i-1-\lg n}}, T_{2^{i-1-\log n}+1}, \ldots, T_{2^{i+1}-1}$ and all edges in F_i. For $i \leq \lg n$, G_i contains all edges in $T_1, T_2, \ldots, T_i$ and all edges in F_i.

Lemma 3.42. *The α-overlap property is satisfied by the above definitions.*

Proof. Let Y_i denote the set of edges in G_i but not in F_i. For any $i \neq j$, $F_i \cap F_j = \emptyset$ and each edge appears in Y_i for at most $2 + \log n$ different values of i. This proves the α-overlap property. □

Lemma 3.43. *The* Π*-connectivity property is satisfied by the above definitions.*

Proof. Note that for any pair of vertices u, v and for any $i \geq 1$, u, v are at least $\min(k_{uv}, i)$-connected in the first i NI forests, i.e. in $T_1 \cup T_2 \cup \ldots \cup T_i$. Thus, any edge $e \in F_i$ is at least 2^i-heavy in the union of NI forests $T_1, T_2, \ldots, T_{2^{i+1}-1}$. Since there are at most 2^{i-1} edges overall in $T_1, T_2, \ldots, T_{2^{i-1-\lg n}-1}$, any edge $e \in F_i$ is 2^{i-1}-heavy in G_i. This proves the Π-connectivity property. □

Theorem 3.17 now follows from the above lemmas and Corollary 3.9 applied to Theorem 3.16. Note that Corollary 3.19 follows immediately from Lemma 3.7 and Theorem 3.17; hence, we will not consider sampling by edge conductances separately.

3.6.2 Compression using Edge Strengths

Now, we use the sparsification framework to show the result of Benczúr and Karger on compression using edge strengths (Theorem 3.14).

For any edge e, set λ_e to its strength s_e. Let $\alpha = 1$ and $\pi_i = 2^k$ for all i, where $k = \lfloor \lg \max_{e \in E}\{\lambda_e\} \rfloor$. Let G_i contain all edges in F_r for all $r \geq i$, where each edge in F_r is replicated 2^{k-r} times. (Recall that replication of edges is allowed in G_i, which are only used in the analysis and not in the actual compression algorithm)

Lemma 3.44. *The* α*-overlap property is satisfied by the above definitions.*

Proof. Consider any cut C with an edge $e \in F_i$. Let the corresponding cut (i.e. with the same bi-partition of vertices) in G_i be C_i. Recall that $f_i^{(C)}$ and $e_i^{(C)}$ respectively denote the number of edges in $F_i \cap C$ and in C_i respectively. Then,

$$\begin{aligned}
\sum_{i=0}^{k} \frac{e_i^{(C)} 2^{i-1}}{\pi_i} &= \sum_{i=0}^{k} \sum_{r=i}^{k} \frac{f_r^{(C)} 2^{k-r} 2^{i-1}}{2^k} \\
&= \sum_{i=0}^{k} \sum_{r=i}^{k} \frac{f_r^{(C)}}{2^{r-i+1}} \\
&= \sum_{r=0}^{k} \sum_{i=0}^{r} \frac{f_r^{(C)}}{2^{r-i+1}} \\
&= \sum_{r=0}^{k} f_r^{(C)} \sum_{i=0}^{r} \frac{1}{2^{r-i+1}} \\
&< \sum_{r=0}^{k} f_r^{(C)}.
\end{aligned}$$

□

Lemma 3.45. *The* Π*-connectivity property is satisfied by the above definitions.*

To prove this lemma, we use the following property of edge strengths [11].

Lemma 3.46. *The strength of an edge does not decrease even if all edges with lower strength are removed from the graph.*

We need to show that the number of edges in C_i is at least 2^k to prove Lemma 3.45. Let the maximum edge strength in C be k_C, where $2^j \leq k_C \leq 2^{j+1} - 1$ for some $j \geq i$. By Lemma 3.46, C_i contains at least 2^j distinct edges of G, each of which is replicated at least 2^{k-j} times. Thus, C_i contains at least 2^k edges. This completes the proof of Lemma 3.45.

Theorem 3.14 now follows from the above lemmas and Lemma 3.8 applied to Theorem 3.16.

3.6.3 Compression using NI indices

Now, we use the sparsification framework to show Theorem 3.21.

For any edge $e = (u, v)$, set λ_e to its NI index ℓ_e. Let $\alpha = 2$ and $\pi = 2^{i-1}$. For any $i \geq 1$, define G_i to be the union of the set of edges in NI forests $T_{2^{i-1}}, T_{2^{i-1}+1}, \ldots, T_{2^i-1}$ (call this set of edges Y_i) and all edges in F_i. (Note that Y_i may contain parallel edges.) Let G_0 only contain edges in F_0.

Lemma 3.47. *The α-overlap property is satisfied by the above definitions.*

Proof. For any $i \neq j$, $F_i \cap F_j = Y_i \cap Y_j = \emptyset$. Thus, each edge appears in G_i for at most two different values of i, proving the α-overlap property. □

Lemma 3.48. *The Π-connectivity property is satisfied by the above definitions.*

Proof. For any edge $e \in F_i$, the endpoints of e are connected in each of $T_{2^{i-1}}, T_{2^{i-1}+1}, \ldots, T_{2^i-1}$ by definition of a NI forest packing. It follows that e is 2^{i-1}-heavy in G_i, thereby proving the Π-connectivity property. □

Theorem 3.21 now follows from the above lemmas and Lemma 3.11 applied to Theorem 3.16.

3.7 Cut Sparsification Algorithm

Recall that an implementation of edge compression using NI indices has a running time of $O(m)$ and produces a cut sparsifier containing $O(n \log^2 n)$ edges in expectation. In this section, we give a more refined algorithm for unweighted input graphs that will have the same time complexity, but will produce cut sparsifiers containing $O(n \log n)$ edges in expectation. This algorithm proves Theorem 3.24.

Before formally describing the algorithm, let us give some intuition about it. Let us abstractly view compression using NI indices as an iterative algorithm that finds a set of edges F_i in iteration i (these are the edges in NI forests $T_{2^i}, T_{2^i+1}, \ldots, T_{2^{i+1}-1}$ and are sampled with probability $\Theta(\log n/2^i)$) with the following properties:

- **(P1)** Each edge in F_i has connectivity of $\Theta(2^i)$ in F_{i-1}.

- **(P2)** The number of edges in F_i is $\Theta(n \cdot 2^i)$.

Our first observation is that property **(P1)** can be weakened — using the general framework, we show it is sufficient for each edge in F_i to have connectivity of $\Theta(2^i)$ in $H_{i-1} = (V, E_{i-1})$ where $E_{i-1} = F_{i-1} \cup F_i \cup \ldots$. Since we are aiming for a sparser sample than in the previous algorithm, we also need to make **(P2)** stricter. Our new requirement is that the number of edges in F_{i-1} from any connected component $\mathbf{C}$ of H_{i-1} is $O(2^i)$ times the number of components into which $\mathbf{C}$ decomposes in H_i. This stricter condition ensures that the expected number of edges in G_ϵ decreases to $\Theta(n \log n/\epsilon^2)$.

We also need to give a linear-time construction of F_i's satisfying the above properties. Iteration i runs on each component of H_i separately; we describe the algorithm for any one component $\mathbf{C}$. First, $(2^i + 1)$ NI forests $T_1, T_2, \ldots, T_{2^i+1}$ are constructed in $\mathbf{C}$ and all edges in T_{2^i+1} are contracted; let the resulting graph be $G_{\mathbf{C}} = (V_{\mathbf{C}}, E_{\mathbf{C}})$. If $|E_{\mathbf{C}}| = O(|V_{\mathbf{C}}| \cdot 2^i)$, we add the edges in $E_{\mathbf{C}}$ to F_i and retain the remaining edges for iteration $i + 1$. Otherwise, we construct $(2^i + 1)$ NI forests on $G_{\mathbf{C}}$, contract the edges in the $(2^i + 1)$st NI forest, and update $G_{\mathbf{C}}$ to this contracted graph. We repeat these steps until $|E_{\mathbf{C}}| = O(|V_{\mathbf{C}}| \cdot 2^i)$; then, we add the edges in $E_{\mathbf{C}}$ to F_i and retain the remaining edges for iteration $i + 1$. One may verify that properties **(P1)** and **(P2)** are satisfied by the F_i's constructed by this algorithm.

This algorithm, with a pre-processing step where the number of edges is reduced to $\tilde{O}(n)$ by sampling using NI indices, runs in $O(m) + \tilde{O}(n)$ time, and yields a sparsifier of expected size $O(n \log n/\epsilon^2)$. We need one additional idea to turn this into a strictly linear-time algorithm for unweighted graphs. Observe that we would ideally like to place as many edges as we can in subsets F_i for large values of i so as to obtain a sparse G_ϵ. On the other hand, the fact that these edges are retained till the later iterative stages implies that we pay for them in our time complexity repeatedly. To overcome this dilemma, we use the following trick: instead of sampling these edges with probability $1/2^i$ in iteration i, we sample them with probability $1/2$ in each iteration $j < i$, and retain them in the set of edges for the next iteration only if selected in the sample. Now, we are able to reduce the size of our edge set by a factor of 2 (in expectation) in each iteration; therefore, implementing a single iteration in linear time immediately yields a linear-time algorithm overall. However, this iterative sampling scheme creates several technical hurdles since it introduces dependencies between the sampling processes for different edges. Our key technical contribution is in showing that these dependencies are mild enough for us to continue to use the sparsification framework that we developed independent compression of edges.

Now, we will formally describe our sparsification algorithm. The algorithm (Algorithm 2) has three phases.

The first phase has the following steps:

- If $m \leq 2\rho n$, where $\rho = \frac{1014 \ln n}{0.38\epsilon^2}$, then $G_\epsilon = G$.
- Otherwise, we construct a NI forest packing of G and all edges in the first 2ρ NI forests are included in G_ϵ with weight one. We call these edges F_0. The edge set Y_0 is then defined as $E - F_0$.

The second phase is iterative. The input to iteration i is a graph (V, Y_{i-1}), which is a subgraph of the input graph to iteration $i-1$ (i.e. $Y_{i-1} \subseteq Y_{i-2}$). Iteration i comprises the following steps:

- If the number of edges in Y_{i-1} is at most $2\rho n$, we take all those edges in G_ϵ with weight 2^{i-1} each, and terminate the algorithm.
- Otherwise, all edges in Y_{i-1} are sampled with probability $1/2$; call the sample X_i and let $G_i = (V, X_i)$.
- We identify a set of edges $F_i \subseteq X_i$ with the following properties:
  - The number of edges in F_i is at most $2k_i|V_c|$, where $k_i = \rho \cdot 2^{i+1}$, and V_c is the set of components in (V, Y_i), where $Y_i = X_i - F_i$.
  - Each edge in Y_i is k_i-heavy in G_i.
- We give a sampling probability $p_i = \min\left(\frac{3}{169 \cdot 2^{2i-9}}, 1\right)$ to all edges in F_i.

The final phase consists of replacing each edge in F_i with 2^i parallel edges, and then compressing each edge independently with probability p_i. (Recall that in the interest of time complexity of the compression procedure, we will generate a Binomial random variable to represent the weight of the edge in the sparsifier.) The weighted graph formed by this compression procedure is the sparsifier G_ϵ.

We now give a short description of the sub-routine that constructs the set F_i in the second phase of the algorithm. This sub-routine is iterative itself. We start with $V_c = V$ and $E_c = X_i$, and let $G_c = (V_c, E_c)$. We repeatedly construct an NI forest packing for G_c and contract all edges in the (k_i+1)st forest, where $k_i = \rho \cdot 2^{i+1}$, to obtain a new G_c. We terminate this iterative process when $|E_c| \leq 2k_i|V_c|$. The set of edges E_c that finally achieves this property forms F_i.

3.7.1 Cut Preservation

We use the following notation throughout: for any set of unweighted edges Z, cZ denotes these edges with a weight of c given to each edge. Our goal is to prove the following theorem.

Theorem 3.49. *$G_\epsilon \in (1 \pm \epsilon)G$ with probability at least $1 - 8/n$.*

Let K be the maximum value of i for which $F_i \neq \emptyset$; let $S = \left(\cup_{i=0}^{K} 2^i F_i\right) \cup 2^K Y_K$ and $G_S = (V, S)$. Then, we prove the following two theorems, which together yield Theorem 3.49 using the union bound. (Observe that since $\epsilon < 1$, $(1+\epsilon/3)^2 \leq 1+\epsilon$ and $(1-\epsilon/3)^2 \geq 1-\epsilon$.)

Theorem 3.50. *$G_S \in (1 \pm \epsilon/3)G$ with probability at least $1 - 4/n$.*

Theorem 3.51. *$G_\epsilon \in (1 \pm \epsilon/3)G_S$ with probability at least $1 - 4/n$.*

The following property is key to proving both theorems.

Algorithm 2 The cut sparsification algorithm

- **procedure** Sparsify(G)
- **input:** An undirected unweighted graph $G = (V, E)$, a parameter $\epsilon \in (0, 1)$
- **output:** An undirected weighted graph $G_\epsilon = (V, E_\epsilon)$
- Set $\rho = \frac{1014 \ln n}{0.38\epsilon^2}$.
- If $m \leq 2\rho n$, then $G_\epsilon = G$ and terminate; else, continue.
- Construct NI forests $T_1, T_2, \ldots$ for G.
- Set $i = 0$; $X_0 = E$; $F_0 = \cup_{1 \leq j \leq 2\rho} T_j$; $Y_0 = X_0 - F_0$.
- Add each edge in F_0 to G_ϵ with weight 1.
- **OuterLoop:** If $|Y_i| \leq 2\rho n$, then add each edge in Y_i to G_ϵ with weight 2^{i-1} and terminate; else, continue.
- Sample each edge in Y_i with probability 1/2 to construct X_{i+1}.
- Increment i by 1; set $E_c = X_i$; $V_c = V$; $k_i = \rho \cdot 2^{i+1}$.
- **InnerLoop:** If $|E_c| \leq 2k_i|V_c|$, then
  - Set $F_i = E_c$; $Y_i = X_i - E_c$.
  - For each edge $e \in F_i$, set $\lambda_e = \rho \cdot 4^i$.
  - Go to **OuterLoop**.

  Else,
  - Construct NI forests $T_1, T_2, \ldots, T_{k_i+1}$ for graph $G_c = (V_c, E_c)$.
  - Update G_c by contracting all edges in T_{k_i+1}.
  - Go to **InnerLoop**.
- For each i, for each edge $e \in F_i$,
  - Set $p_e = \min\left(\frac{9216 \ln n}{0.38\lambda_e\epsilon^2}, 1\right) = \min\left(\frac{3}{169 \cdot 2^{2i-9}}, 1\right)$.
  - Generate r_e from **Binomial**$(2^i, p_e)$.
  - If $r_e > 0$, add edge e to G_ϵ with weight r_e/p_e.

Lemma 3.52. *For any $i \geq 0$, any edge $e \in Y_i$ is k_i-heavy in $G_i = (V, X_i)$, where $k_i = \rho \cdot 2^{i+1}$.*

Proof. Since all edges in Y_0 are in NI forests $T_{2\rho+1}, T_{2\rho+2}, \ldots$ of $G_0 = G$, the lemma holds for $i = 0$.

We now prove the lemma for $i \geq 1$. Let $G_e = (V_e, E_e)$ be the component of G_i containing e. We will show that e is k_i-heavy in G_e; since G_e is a subgraph of G_i, the lemma follows. In the execution of the else block of **InnerLoop** on G_e, there are multiple contraction operations, each comprising the contraction of a set of edges. We show that any such contracted edge is k_i-heavy in G_e; it follows that e is k_i-heavy in G_e.

Let G_e have t contraction phases and let the graph produced after contraction phase r be $G_{e,r}$. We now prove that all edges contracted in phase r must be k_i-heavy in G_e by induction on r. For $r = 1$, since e appears in the $(k_i + 1)$st NI forest of phase 1, e is k_i-heavy in G_e. For the inductive step, assume that the property holds for phases $1, 2, \ldots, r$. Any edge that is contracted in phase $r + 1$ appears in the $(k_i + 1)$st NI forest of phase $r + 1$; therefore, e is k_i-connected in $G_{e,r}$. By the inductive hypothesis, all edges of G_e contracted in previous phases are k_i-heavy in G_e; therefore, an edge that is k_i-heavy in $G_{e,r}$ must have been k_i-heavy in G_e. □

We will now prove Theorem 3.50. First, we state a property of edge sampling. Let $R \subseteq Q$ be subsets of edges such that R is π-heavy in (V, Q). Suppose each edge $e \in R$ is sampled with probability p, and if selected, given a weight of $1/p$ to form a set of weighted edges $\widehat{R}$. Now, for any cut C in G, let $R^{(C)} = R \cap C$, $Q^{(C)} = Q \cap C$, and $\widehat{R^{(C)}} = \widehat{R} \cap C$ respectively; also let the total weight of edges in $R^{(C)}$, $Q^{(C)}$ and $\widehat{R^{(C)}}$ be $r^{(C)}$, $q^{(C)}$ and $\widehat{r^{(C)}}$ respectively. Then the following lemma holds.

Lemma 3.53. *For any $\delta \in (0, 1]$ satisfying $\delta^2 p\pi \geq \frac{6 \ln n}{0.38}$,*

$$|r^{(C)} - \widehat{r^{(C)}}| \leq \delta q^{(C)}$$

for all cuts C, with probability at least $1 - 4/n^2$.

Proof. Let $\mathcal{C}_j$ be the set of all cuts C such that

$$2^j \cdot \pi \leq r^{(C)} \leq 2^{j+1} \cdot \pi - 1$$

for each $j \geq 0$. We will prove that with probability at least $1 - 2n^{-2^{j+1}}$, all cuts in $\mathcal{C}_j$ satisfy the property of the lemma. Then, the lemma follows by using the union bound over j since

$$2n^{-2} + 2n^{-4} + \ldots + 2n^{-2j} + \ldots \leq 4n^{-2}.$$

We now prove the property for cuts $C \in \mathcal{C}_j$. Since each edge $e \in R^{(C)}$ is sampled with probability p in obtaining $\widehat{R^{(C)}}$, we can use Theorem 3.25 with sampling

probability p. Then, for any $R^{(C)}$ where $C \in \mathcal{C}_j$, by Theorem 3.25, we have

$$\begin{aligned}\mathbb{P}\left[\left|\widehat{r^{(C)}} - r^{(C)}\right| > \delta q^{(C)}\right] &< 2e^{-0.38\cdot\delta^2\cdot p\cdot q^{(C)}} \\ &\leq 2e^{-0.38\cdot\delta^2\cdot p\cdot\pi\cdot 2^j} \\ &\leq 2e^{-6\cdot 2^j \ln n} \\ &= 2n^{-6\cdot 2^j},\end{aligned}$$

since $q^{(C)} \geq \pi \cdot 2^j$ for any $C \in \mathcal{C}_j$. Since each edge in $R^{(C)}$ is π-heavy in (V, Q), Theorem 3.33 ensures that the number of distinct $R^{(C)}$ sets for cuts $C \in \mathcal{C}_j$ is at most $n^{2\left(\frac{\pi\cdot 2^{j+1}}{\pi}\right)} = n^{4\cdot 2^j}$. Using the union bound over these distinct $R^{(C)}$ edge sets, we conclude that with probability at least $1 - 2n^{-2^{j+1}}$, all cuts in $\mathcal{C}_j$ satisfy the property of the lemma. □

Setting $R = Y_i, Q = X_i, \widehat{R} = 2X_{i+1}, \delta = \frac{\epsilon/13}{2^{i/2}}, p = 1/2$, and $\pi = \rho \cdot 2^{i+1}$ in Lemma 3.53 gives Corollary 3.54.

Corollary 3.54. *With probability at least $1 - 4/n^2$, for every cut C in G_i, $|2x_{i+1}^{(C)} + f_i^{(C)} - x_i^{(C)}| \leq \frac{\epsilon/13}{2^{i/2}} \cdot x_i^{(C)}$, where $x_i^{(C)}, x_{i+1}^{(C)}$ and $f_i^{(C)}$ respectively denote the weight of $X_i \cap C, X_{i+1} \cap C$ and $F_i \cap C$.*

Next, we show the following fact.

Fact 3.55. *Let $x \in (0, 1]$ and $r_i = 13 \cdot 2^{i/2}$. Then, for any $k \geq 0$,*

$$\begin{aligned}\prod_{i=0}^{k}(1 + x/r_i) &\leq 1 + x/3 \\ \prod_{i=0}^{k}(1 - x/r_i) &\geq 1 - x/3.\end{aligned}$$

Proof. We prove by induction on k. For $k = 0$, the property trivially holds. Suppose

the property holds for $k-1$. Then,

$$\begin{aligned}
\prod_{i=0}^{k}(1+x/r_i) &= \prod_{i=0}^{k}(1+\frac{x}{13\cdot 2^{i/2}}) \\
&= (1+x/13)\cdot\prod_{i=1}^{k}\left(1+\frac{x/\sqrt{2}}{13\cdot 2^{(i-1)/2}}\right) \\
&\leq (1+x/13)\cdot(1+x/(3\sqrt{2})) \\
&\leq 1+x/3 \\
\prod_{i=0}^{k}(1-x/r_i) &= \prod_{i=0}^{k}(1-\frac{x}{13\cdot 2^{i/2}}) \\
&= (1-x/13)\cdot\prod_{i=1}^{k}\left(1-\frac{x/\sqrt{2}}{13\cdot 2^{(i-1)/2}}\right) \\
&\geq (1-x/13)\cdot(1-x/(3\sqrt{2})) \\
&\geq 1-x/3. \quad \square
\end{aligned}$$

We now use Fact 3.55 and Corollary 3.54 to prove the following lemma.

Lemma 3.56. *Let* $S_j = \left(\cup_{i=j}^{K} 2^{i-j}F_i\right) \cup 2^{K-j}Y_K$ *for any* $j \geq 0$. *Then,* $S_j \in (1 \pm (\epsilon/3)2^{-j/2})G_j$ *with probability at least* $1-4/n$, *where* $G_j = (V, X_j)$.

Proof. For any cut C in G, let the edges crossing C in S_j be $S_j^{(C)}$, and let their total weight be $s_j^{(C)}$. Also, let $X_i^{(C)} = X_i \cap C$, $Y_i^{(C)} = Y_i \cap C$, and $F_i^{(C)} = F_i \cap C$ respectively; Let their respective sum of weights be $x_i^{(C)}$, $y_i^{(C)}$ and $f_i^{(C)}$.

Since $K \leq n-1$, we can use the union bound on Corollary 3.54 to conclude that with probability at least $1-4/n$, for every $0 \leq i \leq K$ and for all cuts C,

$$\begin{aligned}
2x_{i+1}^{(C)} + f_i^{(C)} &\leq (1+\epsilon/r_i)x_i^{(C)} \\
2x_{i+1}^{(C)} + f_i^{(C)} &\geq (1-\epsilon/r_i)x_i^{(C)},
\end{aligned}$$

where $r_i = 13 \cdot 2^{i/2}$. Then,

$$\begin{aligned}
s_j^C &= 2^{K-j} y_K^{(C)} + 2^{K-j} f_K^{(C)} + 2^{K-1-j} f_{K-1}^{(C)} + \ldots + f_j^{(C)} \\
&= 2^{K-j} x_K^{(C)} + 2^{K-1-j} f_{K-1}^{(C)} + \ldots + f_j^{(C)} \\
&\quad \text{since } y_K^{(C)} + f_K^{(C)} = x_K^{(C)} \\
&= 2^{K-1-j}(2x_K^{(C)} + f_{K-1}^{(C)}) + (2^{K-2-j} f_{K-2}^{(C)} + \ldots) \\
&\leq (1+\epsilon/r_{K-1}) 2^{K-1-j} x_{K-1}^{(C)} + (2^{K-2-j} f_{K-2}^{(C)} + \ldots) \\
&\leq (1+\epsilon/r_{K-1})(2^{K-1-j} x_{K-1}^{(C)} + 2^{K-2-j} f_{K-2}^{(C)} + \ldots) \\
&\ldots \\
&\leq (1+\epsilon/r_{K-1})(1+\epsilon/r_{K-2}) \ldots (1+\epsilon/r_j) x_j^{(C)} \\
&\leq (1+(\epsilon 2^{-j/2})/r_{K-1-j})(1+(\epsilon 2^{-j/2})/r_{K-2-j}) \ldots \\
&\quad \ldots (1+(\epsilon 2^{-j/2})/r_0) x_j^{(C)} \quad \text{since } r_{j+i} = r_i \cdot 2^{j/2} \\
&\leq (1+(\epsilon/3)2^{-j/2}) x_j^{(C)} \quad \text{by Fact 3.55.}
\end{aligned}$$

Similarly, we can show that $s_j^C \geq (1-(\epsilon/3)2^{-j/2}) x_j^{(C)}$. □

Finally, we observe that Theorem 3.50 follows from Lemma 3.56 if we set $j = 0$.

Now, we will prove Theorem 3.51. First, observe that edges $F_0 \cup 2^K Y_K$ are identical in G_S and G_ϵ. Therefore, we do not consider these edges in the analysis below. For any $i \geq 1$, let $\psi(i)$ be such that $2^{\psi(i)} \leq \rho \cdot 4^i \leq 2^{\psi(i)+1} - 1$. Note that for any j, $\psi(i) = j$ for at most one value of i. Then, for any $j \geq 1$, $R_j = F_i$ if $j = \psi(i)$ and $R_j = \emptyset$ if there is no i such that $j = \psi(i)$. We set $\alpha = 32/3$; $\pi_j = \rho \cdot 4^K$; for any $j \geq 1$, $Q_j = (V, W_j)$ where $W_j = \cup_{i-1 \leq r \leq K} 4^{K-r+1} 2^r F_r$ if $R_j \neq \emptyset$ and $j = \psi(i)$, and $W_j = \emptyset$ if $R_j = \emptyset$.

The following lemma ensures Π-connectivity.

Lemma 3.57. *With probability at least $1 - 4/n$, every edge $e \in F_i = R_{\psi(i)}$ for each $i \geq 1$ is $\rho \cdot 4^K$-heavy in $Q_{\psi(i)}$.*

Proof. Consider any edge $e \in F_i$. Since $F_i \subseteq Y_{i-1}$, Lemma 3.52 ensures that e is $\rho \cdot 2^i$-heavy in $G_{i-1} = (V, X_{i-1})$, and therefore $\rho \cdot 2^{2i-1}$-heavy in $(V, 2^{i-1} X_{i-1})$. Since $\epsilon \leq 1$, Lemma 3.56 ensures that with probability at least $1 - 4/n$, the weight of each cut in $(V, 2^{i-1} X_{i-1})$ is preserved up to a factor of 2 in $Z_i = (V, \cup_{i-1 \leq r \leq K} 2^r F_r)$. Thus, e is $\rho \cdot 4^{i-1}$-heavy in Z_i.

Consider any cut C containing $e \in F_i$. We need to show that the weight of this cut in $Q_{\psi(i)}$ is at least 4^K. Let the maximum λ_a of an edge a in C be $\rho \cdot 4^{k_C}$, for some $k_C \geq i$. By the above proof, a is $\rho \cdot 4^{k_C - 1}$-heavy in Z_{k_C}. Then, the total weight of edges crossing cut C in $Q_{\psi(k_C)}$ is at least $\rho \cdot 4^{k_C - 1} \cdot 4^{K - k_C + 1} = \rho \cdot 4^K$. Since $k_c \geq i$, $\psi(k_C) \geq \psi(i)$ and $Q_{\psi(k_C)}$ is a subgraph of $Q_{\psi(i)}$. Therefore, the the total weight of edges crossing cut C in $Q_{\psi(i)}$ is at least $\rho \cdot 4^K$. □

We now prove the α-overlap property. For any cut C, let $f_i^{(C)}$ and $w_i^{(C)}$ respectively denote the total weight of edges in $F_i \cap C$ and $W_{\psi(i)} \cap C$ respectively. Further, let the

number of edges in $\cup_{i=0}^{K} 2^i F_i \cap C$ be $f^{(C)}$. Then, we have the following bound:

$$\begin{aligned}
\sum_{i=1}^{K} \frac{w_i^{(C)} 2^{\psi(i)-1}}{\pi} &\leq \sum_{i=1}^{K} \frac{w_i^{(C)} \rho \cdot 4^i}{2\rho \cdot 4^K} \\
&= \sum_{i=1}^{K} \frac{w_i^{(C)}}{2 \cdot 4^{K-i}} \\
&= \sum_{i=1}^{K} \sum_{r=i-1}^{K} \frac{f_r^{(C)} \cdot 2^r \cdot 4^{K-r+1}}{2 \cdot 4^{K-i}} \\
&= \sum_{i=1}^{K} \sum_{r=i-1}^{K} \frac{f_r^{(C)}}{2^{r-2i-1}} \\
&= \sum_{r=0}^{K} \sum_{i=1}^{r+1} \frac{f_r^{(C)}}{2^{r-2i-1}} \\
&= \sum_{r=0}^{K} \frac{f_r^{(C)}}{2^r} \sum_{i=1}^{r+1} 2^{2i+1} \\
&= \frac{32}{3} \sum_{r=0}^{K} 2^r f_r^{(C)} \\
&= \frac{32}{3} f^{(C)}.
\end{aligned}$$

Using Theorem 3.16, we conclude the proof of Theorem 3.51.

3.7.2 Size of the sparsifier

We now prove that the expected number of edges in G_ϵ is $O(n \log n/\epsilon^2)$. For $i \geq 1$, define D_i to be the set of connected components in the graph $G_i = (V, X_i)$; let D_0 be the single connected component in G. For any $i \geq 1$, if any connected component in D_i remains intact in D_{i+1}, then there is no edge from that connected component in F_i. On the other hand, if a component in D_i splits into η components in D_{i+1}, then the algorithm explicitly ensures that $\sum_{e \in F_i} \frac{w_e}{\lambda_e}$ (where w_e is the number of parallel copies of e in the Binomial sampling step) from that connected component is

$$\sum_{e \in F_i} \frac{2^i}{\rho \cdot 4^i} \leq \left(\frac{\rho \cdot 2^{i+2} \cdot 2^i}{\rho \cdot 4^i} \right) \eta = 4\eta \leq 8(\eta - 1).$$

Therefore, if $d_i = |D_i|$, then

$$\sum_{i=1}^{K} \sum_{e \in F_i} \frac{w_e}{\lambda_e} \leq \sum_{i=1}^{K} 8(d_{i+1} - d_i) \leq 8n,$$

since we can have at most n singleton components. It follows from Theorem 3.16 that the expected number of edges in G_ϵ is $O(n \log n/\epsilon^2)$.

3.7.3 Time complexity

If $m \leq 2\rho n$, the algorithm terminates after the first step which takes $O(m)$ time. Otherwise, we prove that the expected running time of the algorithm is $O(m + n \log n/\epsilon^2) = O(m)$ since $\rho = \Theta(\log n/\epsilon^2)$. First, observe that phase 1 takes $O(m + n \log n)$ time. In iteration i of phase 2, the first step takes $|Y_{i-1}|$ time. We will show that all the remaining steps take $O(|X_i| + n \log n)$ time. Since $X_i \subseteq Y_{i-1}$ and the steps are executed only if $Y_{i-1} = \Omega(n \log n/\epsilon^2)$, it follows that the total time complexity of iteration i of phase 2 is $O(|Y_{i-1}|)$. Since $Y_i \subset X_i$ and $\mathbb{E}[|X_i|] = \mathbb{E}[|X_{i-1}|]/2$, and $|Y_0| \leq m$, it follows that the expected overall time complexity of phase 2 is $O(m)$. Finally, the time complexity of phase 3 is $O(m + n \log n/\epsilon^2)$ (see e.g. [54]).

We are now left to prove that all, except the first step, of iteration i in phase 2 takes $O(|X_i| + n \log n)$ time. Each iteration of the else block takes $O(|V_c| \log n + |E_c|)$ time for the current $G_c = (V_c, E_c)$. So, the last invocation of the else block takes at most $O(|X_i| + n \log n)$ time. In any other invocation, $|E_c| = \Omega(|V_c| \log n)$ and hence the time spent is $O(|E_c|)$. Now, consider an iteration that begins with $|E_c| > 2k_i \cdot |V_c|$. Note that E_c for the next iteration (denoted by E'_c) comprises only edges in the first k_i NI forests constructed in the current iteration. Hence, $|E'_c| \leq k_i \cdot |V_c| < |E_c|/2$. Since $|E_c|$ decreases by a factor of 2 from one invocation of the else block to the next, the total time over all invocations of the else block is $O(|X_i| + n \log n)$.

3.8 Concluding Remarks

In this chapter, we gave a general sampling framework for cut sparsification and used it to show that various sampling schemes produce cut sparsifiers. In addition, we gave two algorithms for cut sparsification both of which run in $O(m)$ time and produce cut sparsifiers containing an expected $O(n \log n/\epsilon^2)$ edges for unweighted graphs and $O(n \log^2 n/\epsilon^2)$ edges for weighted graphs. For weighted graphs, using previously known algorithms for post-processing, we can obtain cut sparsifiers with an expected $O(n \log n/\epsilon^2)$ edges in $O(m) + O(n \log^5 n)$ time. Several problems are left open by our work. For example, can we improve the running time of the sparsification algorithm to $O(m)$ even for weighted graphs? Also, can we obtain a near-linear time algorithm for cut sparsification that produces a sparsifier containing $o(n \log n)$ edges in expectation? Another interesting combinatorial question is to remove the additional factor of $\log n$ in the sampling probability used in Theorem 3.17, i.e. sampling using edge connectivities.

3.9 Notes

This chapter is based on joint work with Ramesh Hariharan. Some of the results in this chapter were independently and concurrently obtained by Wai Shing Fung

and Nicholas J. A. Harvey [34]. A preliminary version of this work appeared in the *Proceedings of the 43rd ACM Symposium on Theory of Computing*, 2011 [33].

Part II

Network Design

Chapter 4

Online Steiner Tree and Related Problems

The *minimum Steiner tree* (or simply, *Steiner tree*) problem, where the goal is to find the cheapest network connecting a designated set of vertices, is one of the most fundamental and well-studied network design problems. Traditionally, network design problems have been studied under two cost models—the edge-weighted (EW) model and the more general node-weighted (NW) model. The Steiner tree problem has been shown to be NP-hard in the EW model (and therefore in the NW model as well), and algorithms with near-optimal approximation ratios have previously been proposed for both models.

In many optimization problems arising in real-world applications, part of the input is not known in advance but is revealed during the execution of the algorithm. The *online model* has been widely used in algorithmic theory to model such instances. In particular, online optimization for network design problems has received significant attention. While a near-optimal online algorithm for the EW Steiner tree problem was known previously, no non-trivial algorithm was known for the corresponding NW problem. In this chapter, we propose the first online algorithm with a poly-logarithmic competitive ratio for the online NW Steiner tree problem and several related problems.

4.1 Background

The Steiner tree problem is defined as follows.

The Steiner Tree Problem

The input is comprised of an undirected graph $G = (V, E)$ and a set of k terminals $T \subseteq V$. Each edge and/or vertex in G has a given cost. The goal is to output the minimum cost subgraph $H = (V, F)$ of G such that every pair of vertices $t, t' \in T$ is connected in H.

The Steiner tree problem has been extended to various other network design problems. Of particular relevance to our work are the *Steiner forest* and the *group Steiner tree* problems.

The Steiner Forest Problem

The input is comprised of an undirected graph $G = (V, E)$ and a set of k terminal pairs $T \subseteq V^{(2)}$. Each edge and/or vertex in G has a given cost. The goal is to output the minimum cost subgraph $H = (V, F)$ of G such that every terminal pair $(s, t) \in T$ is connected in H.

Note that an instance of the Steiner tree problem can be expressed as an instance of the Steiner forest problem where the terminal pairs are $\{(r, t) : t \in T - \{r\}\}$ for any fixed vertex $r \in T$.

The Group Steiner Tree Problem

The input is comprised of an undirected graph $G = (V, E)$, a root vertex $r \in V$, and a collection of sets of terminals $T \subseteq 2^V$. Each edge and/or vertex in G has a given cost. The goal is to output the minimum cost subgraph $H = (V, F)$ of G such that the root vertex r is connected to at least one vertex in every terminal set.

Note that an instance of the Steiner tree problem can be expressed as an instance of the group Steiner tree problem where the root vertex r is an arbitrary fixed vertex in T and each vertex $t \in T - \{r\}$ is in a separate singleton terminal set.

Even though both the above problems generalize the Steiner tree problem, they are mutually incomparable. Instead of treating these two problems separately, we will often consider the group Steiner forest problem which generalizes the Steiner forest and the group Steiner tree problems.

The Group Steiner Forest Problem

The input is comprised of an undirected graph $G = (V, E)$, a root vertex $r \in V$, and a collection of pairs of sets of terminals $T \subseteq (2^V)^{(2)}$. Each edge and/or vertex in G has a given cost. The goal is to output the minimum cost subgraph $H = (V, F)$ of G such that for every pair of terminal sets $(S, S') \in T$, at least one vertex pair $s \in S, s' \in S'$ is connected in H.

We also consider another generalization of the Steiner tree problem called the *single-source ℓ-vertex connectivity* problem.

The Single-source ℓ-vertex Connectivity problem

The input is comprised of an undirected graph $G = (V, E)$, a root vertex $r \in V$, a set of k terminals $T \subseteq V - \{r\}$, and a connectivity requirement ℓ. Each edge and/or vertex in G has a given cost. The goal is to output the minimum cost subgraph $H = (V, F)$ of G such that for every terminal $t \in T$, there are at least ℓ vertex-disjoint paths between t and r in H.

Note that an instance of the Steiner tree problem can be expressed as an instance of the group Steiner forest problem where $\ell = 1$, the root vertex r is an arbitrary fixed vertex in T, and the terminal set is $T - \{r\}$.

4.1.1 Edge-weighted and Node-weighted Problems

The edge-weighted (EW) versions of the above problems allow only edge costs, which are often used to model costs of network links. On the other hand, the corresponding node-weighted (NW) versions allow both edge and vertex costs. Node costs are often used to model equipment cost at network nodes, load on network switches and routers [46], latency and cost of recovery from power outages in electrical networks [44], etc. For any NW instance, we will replace every edge of cost c with two edges of cost 0 connected by a vertex of cost c as a pre-processing step. It is easy to see that solutions for the transformed instance have a one-to-one correspondence to solutions for the original instance. The number of vertices increases to $n + m$ and the number of edges increases to $2m$ because of this transformation. This increase in the size of the instance does not affect (quasi-)polynomiality of the running time of algorithms. Therefore, we will henceforth assume that only vertices have associated costs in NW versions of these problems.

4.1.2 The Online Model

In many optimization problems, part of the input is not known apriori and is periodically revealed during the execution of the algorithm. Such scenarios have typically been modeled in the algorithmic literature using the *online optimization* (see e.g. [13]) paradigm. In online network design problems, the entire input graph G and the associated costs are given offline, i.e. before the execution of the algorithm. Initially, the algorithmic solution H_0 is the empty subgraph. In each online step, a new constraint appears online and must be satisfied by the algorithmic solution. (The online constraint for each of the above problems is given in Table 4.1.) In other words, the algorithm must maintain the invariant that the algorithmic solution H_i at the end of the i^{th} online step satisfies the constraint that appeared in that step. Further, the algorithm is only allowed to *add* edges and vertices to its solution, i.e. H_{i-1} is a subgraph of H_i for each i. The goal is to minimize the cost of the algorithmic solution after all the terminals have been revealed.

Problem	Constraint that arrives online in the i^{th} step
Steiner tree	A new terminal t_i that has to be connected to the previous terminals in H_i.
Steiner forest	A new terminal pair (s_i, t_i) where s_i must be connected to t_i in H_i.
Group Steiner tree	A new terminal set T_i where some vertex $t_i \in T_i$ must be connected to the root r (which is given offline) in H_i.
Group Steiner forest	A new pair of terminal sets (S_i, T_i) where some pair of vertices $s_i \in S_i, t_i \in T_i$ must be connected in H_i.

Table 4.1: The online constraints for Steiner tree and its generalizations.

4.1.3 Bi-criteria Approximation for Network Design Problems

We will use the notion of bi-criteria approximation for the single-source ℓ-vertex connectivity problem. A bi-criteria approximation/competitive ratio of (a, b) for an ℓ connectivity problem implies that the solution produced by the offline/online algorithm achieves a connectivity of ℓ/b while its cost is at most a times that of an optimal offline solution that achieves a connectivity of ℓ.

4.1.4 History

The EW Steiner tree problem is one of the most well-studied problems in combinatorial optimization. It is easy to show that a minimum spanning tree over the terminals obtains an approximation factor of 2. This approximation factor was improved in a sequence of works [90, 61, 82, 84] ultimately leading to an approximation factor of 1.39 obtained by Byrka *et al* [15]. The last algorithm differs from the previous ones in that it uses an LP-rounding approach as against the combinatorial techniques used earlier. It is also known that the EW Steiner tree problem is inapproximable to within a factor of $\frac{96}{95}$, unless $\mathbf{NP} = \mathbf{P}$ [22]. In the online model, Imase and Waxman [52] observed that the natural greedy algorithm, which adds the minimum cost path from the new terminal to any previous terminal in every online step, has a competitive ratio of $O(\log k)$ for the EW Steiner tree. It is easy to show that this competitive ratio is the best possible up to constant factors.

Agrawal *et al* [1] and Goemans and Williamson [40] introduced the notion of primal-dual algorithms to achieve an approximation factor of 2 for the EW Steiner forest problem. For the online version of this problem, Awerbuch *et al* [9] showed that the greedy algorithm has a competitive ratio of $O(\log n \log k)$. Berman and Coulston added a small but critical modification to the greedy algorithm to improve this competitive ratio to $O(\log k)$.

For the EW group Steiner tree problem, Charikar *et al* gave an $O(k^{\epsilon})$-approximation polynomial-time algorithm for any constant $\epsilon > 0$ [17]. They also obtained an approximation ratio of $O(\log^2 k)$ in quasi-polynomial time. Garg *et al* [36] used probabilistic embedding in trees to obtain an $O(\log^3 n \log k)$ approximation randomized algorithm

that runs in polynomial time. This algorithm was derandomized by Charikar *et al* who obtained an approximation ratio of $O(\log^2 n \log k \log\log n)$ [18]. For other results on the EW group Steiner tree problem, see [91] and [20]. An approximation hardness of $\Omega(\log^{2-\epsilon} k)$ for any $\epsilon > 0$ under the complexity theoretic assumption that **NP** does not have quasi-polynomial Las Vegas algorithms is also known [49]. For the online version of this problem, Alon *et al* [5] used the online primal-dual schema to give an $O(\log^3 n \log k)$ competitive randomized algorithm.

For the EW group Steiner forest problem, Chekuri *et al* [19] gave the first non-trivial algorithm, which achieves an approximation ratio of $O(\log^2 n \log^2 k)$. They posed the design of an online algorithm with poly-logarithmic competitive ratio for this problem as an open question that we resolve in this chapter.

Let us now move on to node-weighted problems. For the NW Steiner tree problem, Klein and Ravi [64] gave a greedy algorithm based on *spider decompositions* and showed that it achieves an approximation factor of $2 \ln k$. Their algorithm also generalizes to the NW Steiner forest problem. Subsequently, a primal-dual interpretation of this algorithm was given by Guha *et al* [44], and the approximation factor was improved to $(1.35 + \epsilon) \ln k$ using a different combinatorial approach by Guha and Khuller [43]. It is also known (due to Berman) that the NW Steiner tree problem generalizes the *minimum set cover* problem [64], which implies that it is inapproximable to within a factor of $(1 - o(1)) \ln k$, unless $\mathbf{NP} = \mathbf{P}$ [29]. Our work gives the first online algorithm for this problem with a poly-logarithmic competitive ratio.

4.2 Our Contributions

Our main result for the online NW Steiner tree problem is the following theorem.

Theorem 4.1. *There is a polynomial-time randomized online algorithm for the node-weighted Steiner tree problem with a competitive ratio of* $O(\log n \log^2 k)$.

We note that there is a lower bound of $\Omega(\log n \log k)$ on the competitive ratio of any polynomial-time online algorithm for this problem, even if the algorithm is randomized. This follows from a corresponding lower bound for the online set cover problem, under the $BPP \neq NP$ assumption [30].

We also obtain an online algorithm for the NW group Steiner forest problem (and therefore for the NW Steiner forest and NW group Steiner tree problems as well) with poly-logarithmic competitive ratio.

Theorem 4.2. *There is a quasi-polynomial time randomized online algorithm for the node-weighted group Steiner forest problem with a competitive ratio of* $O(\log^4 n \log^5 k)$.

Our techniques also lead to new results in online EW network design. First, we give a polynomial-time online algorithm for the EW group Steiner forest problem, thus resolving an open question of Chekuri *et al* [19].

Theorem 4.3. *There is polynomial-time randomized online algorithm for the edge-weighted group Steiner forest problem with a competitive ratio of* $O(\log^6 n \log k)$.

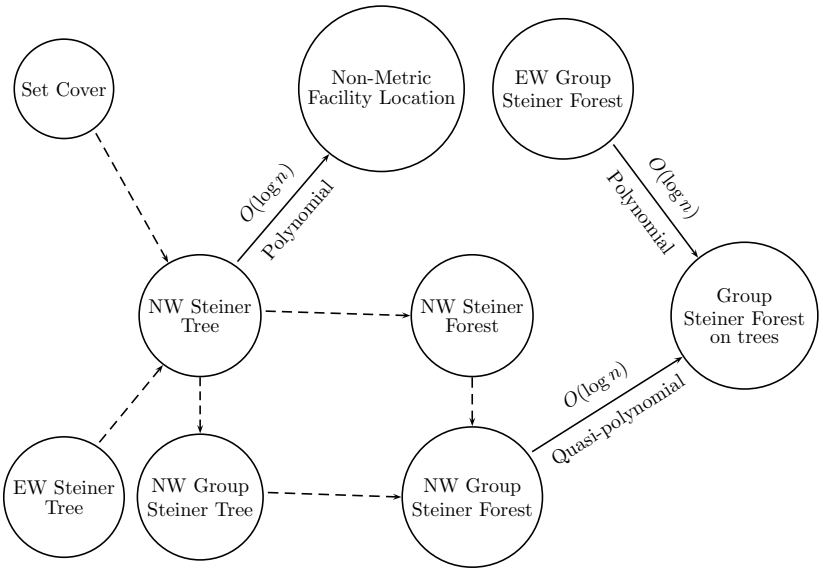

Figure 4-1: Relationships between network design problems.

For the EW *single-source ℓ-vertex connectivity* problem, we obtain the following theorem.

Theorem 4.4. *There is a polynomial-time deterministic online algorithm for the edge-weighted single-source ℓ-vertex connectivity problem with a (bi-criteria) competitive ratio of $(O(\ell \log k/\epsilon), 2+\epsilon)$ for any $\epsilon > 0$.*

This theorem complements the results of Gupta *et al* [45] for the corresponding online edge-connectivity problem.

Roadmap

The set of problems we consider and various reductions we perform are given in Figure 4-1. The arrows show the reductions from one problem to the other. Dashed lines represent the reductions via generalization. The labels on the reductions represent the approximation factor lost in the reduction and the size of the reduction.

This chapter is organized as follows. The online NW Steiner tree algorithm is presented in Section 4.3. We give the group Steiner forest algorithm on a tree and then use it to solve the EW and NW group Steiner forest problems in general graphs in Section 4.4. The online algorithm for the EW single-source ℓ-vertex connectivity problem is given in Section 4.5.

4.3 Online Node-weighted Steiner Tree

As noted earlier, the online NW Steiner tree problem generalizes the online EW Steiner tree problem, for which there are mainly two algorithmic approaches. Imase

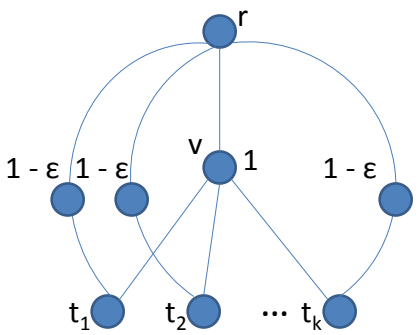

Figure 4-2: A lower bound for the greedy algorithm for the online NW Steiner tree problem

and Waxman [52] showed that the natural greedy algorithm which chooses, in each online step, the cheapest path to connect the new terminal to any previous terminal, has a competitive ratio of $O(\log k)$ for the online EW Steiner tree problem. Unfortunately, the greedy algorithm has a polynomial competitive ratio for the NW version. Consider the example in Figure 4-2. Here, the vertices $r, t_1, t_2, \ldots, t_k$ appear as terminals, all of which have cost 0. The costs of non-terminal vertices are shown in the figure. Since each terminal has a *private* path of cost $1 - \epsilon$ to r, the greedy algorithm selects these private paths with total cost $(1 - \epsilon)k$, whereas the optimal solution chooses the paths through vertex v and has cost 1. Choosing ϵ to be an arbitrarily small positive constant leads to a lower bound of k on the competitive ratio of this algorithm.

The second approach for the online EW Steiner tree problem is based on probabilistic tree embeddings [28], which have been successfully used by Gupta *et al* [45], even for higher connectivity requirements in EW online settings. We will discuss such embeddings in more detail later when we consider edge-weighted problems, but such embeddings do not exist in the presence of vertex costs. Therefore, this approach is also ruled out.

One of the reasons for the above two techniques failing for the NW Steiner tree problem is that it also generalizes the *set cover* problem.

The Set Cover Problem

The input comprises a collection of m subsets $\mathcal{S}$ of a universe X containing n elements. Each subset has an associated cost. The goal is to find a sub-collection of subsets $\mathcal{T} \subseteq \mathcal{S}$ of minimum cost such that every element in U is covered by some subset in $\mathcal{T}$, i.e. $\cup_{T \in \mathcal{T}} T = X$.

Note that an instance of the set cover problem can be expressed as a NW Steiner tree problem on a graph $G = (V, E)$, where the vertices $V = X \cup \mathcal{S} \cup \{r\}$ and the

$$\begin{array}{rcl} \min & & \sum_{v \in V} \sum_{v \in V} c_v x_v \quad \text{such that} \\ \sum_{v \in W} x_v & \geq & 1 \quad \forall \text{ vertex cuts } W \text{ separating some } t, t' \in T \\ x_v & \in & \{0,1\} \quad \forall\, v \in V. \end{array}$$

Figure 4-3: The standard ILP for the NW Steiner tree problem

edges $E = E_1 \cup E_2$ with $E_1 = \{(x, S) : x \in X \text{ is in } S\}$ and $E_2 = \{(r, S) : S \in \mathcal{S}\}$. The terminal set $T = X \cup \{r\}$. The costs of the non-terminal vertices are identical to the costs of the corresponding sets. The terminal vertices have cost 0.

In the online set cover problem, the collection of sets $\mathcal{S}$ and the universe X is given offline, but all elements need not be covered. In each online step, a new element x that has to be covered is revealed and the selected subsets $\mathcal{T}$ has to be augmented to cover x. The goal is to minimize the overall cost of $\mathcal{T}$ after all the elements to be covered have been revealed.

The first non-trivial algorithm for the online set cover problem was given by Alon *et al* [6], who introduced the online primal-dual paradigm (for a comprehensive survey on this technique, see [14]) to obtain a competitive ratio of $O(\log m \log n)$. This algorithm works by first obtaining online a fractional solution to the standard LP relaxation of the set cover problem to within an $O(\log m)$ factor, and then adapting the randomized rounding method for set cover to work online, losing another factor of $O(\log n)$ in the competitive ratio. Let us try to apply this technique to the NW Steiner tree problem. The standard LP for this problem is given in Figure 4-3.

Using the methods of Alon *et al* [5] for online covering of cuts in a graph, we can compute a fractional solution to this LP (i.e. when $x_v \in [0, 1]$) online which has a competitive ratio of $O(\log n)$. However, this LP appears to be too weak to allow for online rounding without losing a polynomial factor in the competitive ratio. Consider the example in Figure 4-4. If each edge and vertex has a value of $1/\sqrt{n}$ in the fractional solution, then an independent rounding of the edges and vertices does not produce a feasible solution. On the other hand, since the value on an edge or vertex accumulates over multiple rounds, dependent rounding may produce an integer solution that is polynomially more expensive than the fractional solution. In fact, even for the EW Steiner tree problem, an online rounding technique for the standard LP is not known.

Observe that one can view a solution to the online Steiner tree problem as a collection of paths, one from each terminal to another terminal that appeared earlier in the online sequence. If each terminal could afford to pay for its entire path (to some previous terminal), then a greedy algorithm suffices. For the EW version, this is indeed the case, and this property is crucial for the analysis of the greedy algorithm in the online setting. However, as indicated earlier, the example in Figure 4-2 asserts that for the NW version this property is not true, i.e. terminals must *necessarily* share the cost of these paths in order to obtain a poly-logarithmic competitive ratio.

A natural next step is bounding the extent of the cost sharing among the terminals.

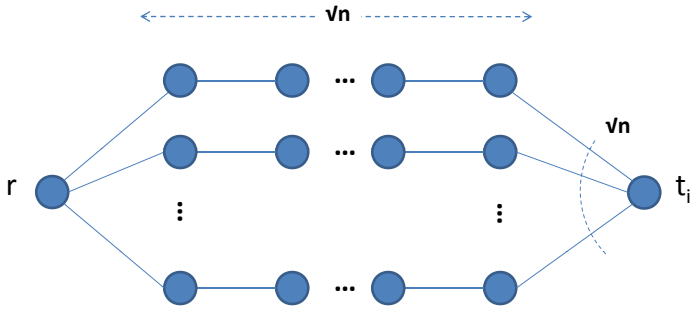

Figure 4-4: Online rounding of the standard LP relaxation of NW Steiner tree

For example, in Figure 4-2, terminals $t_1, t_2, \ldots, t_k$ only need to share the cost on the solitary vertex v on their paths to terminal r. Our key lemma, somewhat surprisingly, generalizes this to show that if we are ready to sacrifice a factor of $O(\log k)$ in the cost, then the cost sharing among terminals can be restricted to a single vertex on every path.

Lemma 4.5. *Let $G = (V, E)$ be an undirected graph with vertex and edge costs c_v, c_e for vertices $v \in V$ and edges $e \in E$ respectively. Suppose $T \subseteq V$ is a set of k terminal vertices. Then, for any ordering of the terminals $t_1, t_2, \ldots, t_k$, and for any subgraph G_T of G connecting all the terminals, there exists a set of paths $P_2, P_3, \ldots, P_k$ and a corresponding set of vertices $v_2, v_3, \ldots, v_k$ such that*

- *P_i is a path from terminal t_i to another terminal t_j which is earlier in the order, i.e., $j < i$, in G_T*
- *v_i is on path P_i, and*
- *$\sum_{i=2}^{k} (c(P_i) - c_{v_i}) \leq \lceil 2 \lg k \rceil \cdot c(G_T)$,*

where $c(P_i)$ is the sum of costs of vertices and edges on P_i, and $c(G_T)$ is the sum of costs of vertices and edges in G_T.

To prove this lemma, we need to introduce the technique of spider decomposition of trees due to Klein and Ravi [64].

Definition 4.6. *A* spider *is a connected graph containing at least three vertices, where at most one vertex has degree greater than two. Each vertex that has degree equal to one is called a* foot*, while the unique vertex that has degree greater than two is called the* head*. If no vertex has degree greater than two, then any of the vertices with degree equal to two can be called the head. A head-to-foot path is called a* leg *of the spider.*

An example of a spider is given in Figure 4-5.

Klein and Ravi [64] defined the notion of a *spider decomposition* of a tree and proved its existence.

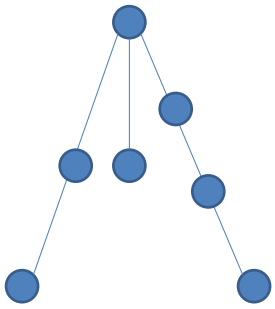

Figure 4-5: An example of a spider

Lemma 4.7 (Klein-Ravi [64])**.** *Any tree R contains a set of vertex-disjoint spiders such that the feet of the spiders are exactly the leaves of the tree. This set is called a spider decomposition.*

We extend this lemma to produce a recursive spider decomposition $\mathcal{S}$ of any tree R. Suppose $\mathcal{L} = \ell_1, \ell_2, \ldots, \ell_k$ is an arbitrary ordering of the leaves of tree R. A *covering spider decomposition* of R with respect to the ordering $\mathcal{L}$ is a sequence of sets of spiders $\mathcal{S}_1, \mathcal{S}_2, \ldots$ with the following properties:

- The spiders in any set $\mathcal{S}_i$ are node-disjoint.
- $\mathcal{S}_1$ is a spider decomposition of R, i.e. the feet of the spiders in $\mathcal{S}_1$ are the leaves of R.
- Let $\mathcal{S}_i = \{s_{i1}, s_{i2}, \ldots, s_{ir_i}\}$. Now, let the leaves of R that are feet of spider s_{ij} be L_{ij}; further let ℓ_{ij} be the first among these leaves in the ordering $\mathcal{L}$. Then, the feet of the spiders in $\mathcal{S}_{i+1}$ are exactly the leaves $\{\ell_{ij} : 1 \leq j \leq r_i\}$.

An example of a covering spider decomposition is given in Figure 4-6. The leaves are ordered as $\{t_1, t_2, t_3, t_4, t_5\}$. The spiders in the first recursive level (the top right corner) form a spider decomposition of the tree. In the second recursive level (the bottom right corner), there are only two terminals t_1 and t_4, which were the two earliest terminals in their respective spiders. Thus, there is a single spider connecting them. In general, instead of two recursive levels, we might have $\lceil \lg k \rceil$ recursive levels.

Before showing that such a recursive decomposition of spiders exists for any tree, let us show that its existence implies Lemma 4.5. Recall that in Lemma 4.5, G_T is a connected subgraph of G containing all the terminals in T. Let R be a spanning tree of G_T. We also assume that all terminals in T are leaves of R. (Otherwise, we introduce a dummy terminal of cost 0 and connect it to the original terminal using an edge of cost 0). Now, for each terminal t_i, the path P_i and the vertex v_i on it (as in Lemma 4.5) are defined as follows.

Let j_i be the maximum index j such that terminal t_i is a foot in a spider $s \in \mathcal{S}_j$. Let T_s be the ordering of the terminals that are feet of spider s with respect to arrival

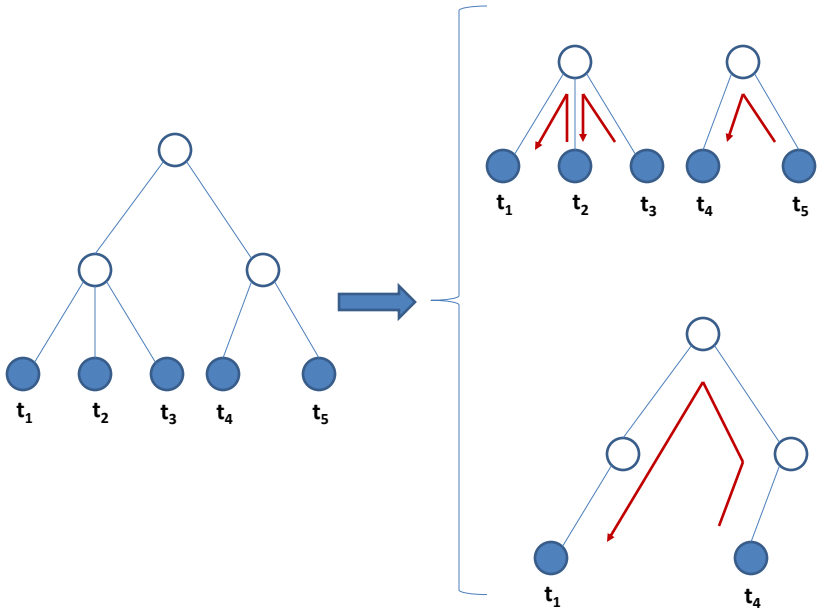

Figure 4-6: A covering spider decomposition of a tree

order. Then, we define the path p_i as the path from t_i to the terminal immediately before t_i in the sequence T_s. Also, v_i is defined as the head of spider s. In Figure 4-6, the red arrows indicate the paths used by t_2 to connect to t_1 (path p_2), t_3 to t_2 (path p_3), t_5 to t_4 (path p_5), and t_4 to t_1 (path p_4). The following property is a direct consequence of this definition.

Lemma 4.8. *The sum of costs $c(P_i) - c_{v_i}$ for all terminals t_i having $j_i = j$ for a fixed j is at most $2c(R)$.*

Proof. The proof follows by observing that each leg of a spider $s \in \mathcal{S}_j$ appears in path p_i for at most two terminals t_i having $j_i = j$. □

The next lemma follows from the fact that each spider must contain at least two feet.

Lemma 4.9. *The number of sets of spiders in a covering spider decomposition of a tree containing k leaves is $\lceil \lg k \rceil$, irrespective of the ordering of the leaves.*

The above two lemmas immediately imply Lemma 4.5.

Finally, we need to show that any tree has a covering spider decomposition with respect to any ordering of the leaves. We give a recursive procedure for constructing such a decomposition. First, we use Lemma 4.7 to produce a spider decomposition $\mathcal{S}_1$. Then, we delete all the legs of each spider in $\mathcal{S}_1$, except one leg from each spider $s \in \mathcal{S}_1$ that ends at the leaf that appears earliest in the ordering among the leaves in s. We now recursively construct the spider decompositions $\mathcal{S}_2, \mathcal{S}_3, \ldots$ in the remaining tree. This completes the proof of Lemma 4.5.

$$\begin{aligned}
\min \quad & \textstyle\sum_{i=2}^{k} \sum_{v \in V} c_i^{(v)} x_i^{(v)} \; + \; \sum_{v \in V} c_v y_v \qquad \text{s.t.:} \\
\sum_{v \in V} x_i^{(v)} \; & \geq \; 1 \quad \forall\, 2 \leq i \leq k \\
x_i^{(v)} \; & \leq \; y_v \quad \forall\, v \in V,\; 2 \leq i \leq k \\
x_i^{(v)} \; & \in \; \{0,1\} \quad \forall\, v \in V,\; 2 \leq i \leq k \\
y_v \; & \in \; \{0,1\} \quad \forall\, v \in V.
\end{aligned}$$

Figure 4-7: A new ILP for the online NW Steiner tree problem

It is interesting to note that Lemma 4.5 implies that no cost sharing is necessary in the edge-weighted case. The Corollary below formalizes this claim.

Corollary 4.10. *Let $G = (V, E)$ be an undirected graph with edge costs only. Suppose $T \subseteq V$ is a set of k terminal vertices. Then, for any ordering of the terminals $t_1, t_2, \ldots, t_k$, and for any subgraph G_T of G connecting all the terminals, there exists a set of paths $P_2, P_3, \ldots, P_k$ such that:*

- *P_i is a path from terminal t_i to some t_j which is earlier in the order, i.e. $j < i$,*
- *$\sum_{i=2}^{k} c(P_i) \leq 2\lceil \lg k \rceil c(G_T)$,*

where $c(P_i)$ is the sum of costs of edges on P_i, and $c(G_T)$ is the sum of costs of edges in G_T.

It follows from Corollary 4.10 that the greedy algorithm for the online edge-weighted Steiner tree problem is $O(\log k)$-competitive, providing an alternative proof for the Imase-Waxman result.

Our goal now is to select vertex v_i and path P_i for each terminal t_i; in fact, selecting v_i immediately selects the path P_i as the cheapest path from t_i to v_i, and then from v_i to any $t_j, j < i$. This observation allows us to encode the Steiner tree problem as a new ILP in Figure 4-7.

In this ILP, $c_i^{(v)}$ is the sum of the costs of the cheapest path from terminal t_i to vertex v and the cheapest path from v to any of previous terminals, i.e. any t_j, $j < i$. Both of these costs do not include the cost of v, and can be computed on the arrival of terminal t_i. The variable $x_i^{(v)}$ is an indicator variable for the event $v = v_i$, and y_v is an indicator variable for selecting vertex v in the solution. The first constraint guarantees that for each terminal t_i, we choose at least one vertex as v_i; the second constraint guarantees that if a vertex v is chosen as v_i by at least one terminal t_i, then we pay c_v in the objective value.

Now, we claim that the ILP in Figure 4-7 is equivalent to the non-metric *facility location* problem.

The Non-metric Facility Location Problem

The input comprises a set of facilities F and a set of clients C. Each facility $f \in F$ has an *opening cost* c_f and each client-facility pair ($i \in C, f \in F$) has a *connection cost* c_{if}. The goal is to open a subset of facilities and connect each client to an open facility such that the sum of the opening costs and connection costs is minimized.

In the online version of the problem, the facilities and opening costs are given offline, but a new client along with its connection costs arrives in each online step. On the arrival of a client, we can either open a new facility and connect the client to it or connect the client to a previously opened facility. Alon *et al* [5] gave an algorithm using the online primal-dual schema for the non-metric facility location problem and showed the following theorem.

Theorem 4.11 (Alon *et al* [5])**.** *There is a randomized online algorithm for the non-metric facility location problem that has a competitive ratio of* $O(\log |C| \log |F|)$.

To model the ILP in Figure 4-7 as a facility location problem, we give the following reduction. Consider the set of terminals t_i, $2 \leq i \leq k$, as clients and the vertices $v \in V$ as facilities. The cost of opening a facility v is c_v, while the connection cost of serving a client t_i using facility v is $c_i^{(v)}$. Then, the ILP in Figure 4-7 asks for the cheapest assignment of clients to facilities. Using Lemma 4.5, we can conclude that the algorithm of Alon *et al* (Theorem 4.11), applied to our facility location instance, yields an $O(\log n \log^2 k)$-competitive algorithm for the online NW Steiner tree problem, thereby proving Theorem 4.1.

4.4 Online Group Steiner Forest

We now turn our attention to the *online group Steiner forest* problem, both for the node-weighted and edge-weighted case. As with the Steiner tree problem, the methods of [5] for online covering of cuts in a graph can be used to obtain a fractional solution with a logarithmic competitive ratio for the standard LP formulations of both problems, but all known online rounding techniques lose a polynomial factor in the competitive ratio. Instead, we show that by losing a poly-logarithmic factor in the competitive ratio, we can reduce both problems to instances where the input graph is a tree. First, we give an algorithm for solving the online group Steiner forest on trees.

4.4.1 Online Group Steiner Forest on Trees

We show the following theorem for the online group Steiner forest problem on trees.

Theorem 4.12. *There is a randomized online algorithm for the group Steiner forest problem on trees of depth* h *that has a competitive ratio of* $O(h \log^4 n \log k)$.

$$\begin{array}{rcll} & & \min & \sum_{e \in E} c_e x_e \quad \text{such that} \\ \sum_{e \in W} x_e & \geq & 1 & \forall i,\ \forall \text{ cuts } W \text{ separating } (S_i, T_i) \\ x_e & \in & \{0,1\} & \forall\ e \in E. \end{array}$$

Figure 4-8: An ILP for the online edge-weighted group Steiner forest problem

Note that for a tree, the EW and NW versions are identical since there is a one-to-one correspondence between the edges and non-root vertices of a tree. For convenience, we will consider edge-weighted instances when showing the above theorem.

We describe our algorithm in two stages. In the first stage, we give an online algorithm that obtains a fractional solution to the problem, and in the second stage, we give an algorithm for rounding a fractional solution online. Note that these two algorithms are interleaved in their actual execution on an online instance of the problem.

We will assume throughout that we know the value of the optimal solution α. This is without loss of generality because we can guess the value of α, doubling our guess every time the ratio of the cost of the algorithmic solution to the guessed value of α exceeds the competitive ratio we are going to prove. We will eventually obtain a guess whose value is at most twice α. For simplicity, we will assume that we know the value of α exactly (rather than up to a factor of 2); the entire analysis is valid (up to constants) even if the guess were a constant factor more than the optimum. At the outset, we divide all edge costs by α so that the cost of an optimal solution is 1.

In the description of the algorithm, $R = (V, E)$ will denote the input tree.

First Stage: Fractional Algorithm

In the first stage, we use an online primal-dual algorithm for generalized cut problems due to Alon *et al* [5] to obtain a fractional solution for our problem.

The standard LP for the EW group Steiner forest problem is given in Figure 4-8. A cut is said to separate a terminal set pair (S_i, T_i) if removal of edges in the cut disconnects all vertices in S_i from all vertices in T_i. The constraint is that at least one edge must be selected from each such cut, which by Menger's theorem (see e.g. [24]) implies that there is a path connecting some vertex in S_i to some vertex in T_i in the solution. In the fractional version, we are allowed to choose edges fractionally, i.e. edges $e \in E$ are chosen to fractions $0 \leq x_e \leq 1$.

Initialization. We categorize edges $e \in E$ into three groups:

- If $c_e > 1$, we initialize x_e to 0.
- If $c_e \leq 1/n$, we initialize x_e to 1.

- If $1/n < c_e \leq 1$, we initialize x_e to $1/n$.

Online Algorithm. In each online step, the following augmentation is repeatedly performed until the current fractional solution is feasible for the LP in Figure 4-8: find the minimum cut separating (S_i, T_i) and augment x_e to $x_e(1 + 1/c_e)$ on every edge of this cut.

Analysis We will show that the fractional solution produced by the above algorithm has a logarithmic competitive ratio.

Lemma 4.13. *The fractional algorithm for the online EW group Steiner forest problem has competitive ratio of* $O(\log n)$.

Recall that the optimal cost is 1; hence, we need to show that the cost of the algorithmic solution is $O(\log n)$. The proof will follow from the following properties that we show.

Lemma 4.14. *The total cost of the initialization is at most 2.*

Proof. Setting $x_e = 1$ for each edge e with $c_e \leq 1/n$ has a total cost of at most 1. Setting $x_e = 1/n$ for each edge e with $1/n < c_e \leq 1$ also has a total cost of at most 1. □

Lemma 4.15. *In each online step, the additional cost accrued is at most 1.*

Proof. In each online step, the algorithm picks a cut W with $\sum_{e \in W} x_e < 1$ and augments each x_e to $x_e(1+1/c_e)$ thereby incurring an overall cost of $\sum_{e \in W} x_e < 1$. □

Lemma 4.16. *The total number of iterative steps of the online algorithm is* $O(\log n)$.

Proof. In each online step, at least one edge that is part of the optimal solution suffers an augmentation. The proof follows from the fact that the maximum number of augmentations on an edge is $O(c_e \log n)$, and that the cost of the optimal solution is 1. □

This completes the proof of Lemma 4.13. Note that the above analysis did not use the fact that the input graph is a tree. Indeed, as we mentioned earlier, this algorithm produces a fractional solution with a logarithmic competitive ratio for the online EW (and also for NW where augmentations are on vertex cuts) group Steiner forest problem. The difficulty lies in the online rounding, and we will use the fact that the input instance is a tree when we describe the online rounding algorithm.

Second Stage: Online Rounding Algorithm

Our rounding algorithm has close resemblance to a rounding technique for the group Steiner tree problem on a tree due to Garg *et al* [36] (whose online version was given by Alon *et al* [5]). However, there are some differences, e.g. we have to apply the rounding algorithm on each subtree of the input tree R rather than only on R.

We will show that the integer solution produced by the rounding algorithm connects at least one pair of vertices from each terminal group pair (S_i, T_i) with probability $\Omega(1/\log^2 n)$. Moreover, the expected cost of the integer solution is $O(h \log n)$ times the cost of the fractional solution, where h is the height of R. We run $O(\log^2 n \log k)$ parallel instantiations of this rounding technique; using standard analysis, we then conclude that any terminal group pairs is connected with probability at least $1 - 1/k$. This allows us to add the cheapest path from a vertex in S_i to a vertex in T_i for a terminal group pair (S_i, T_i) that did not get connected; the expected overhead because of this step is at most 1 since the cheapest path has cost at most that of an optimal solution.

We will need the following definition.

Definition 4.17. *The* least common ancestor *of a pair of vertices in a rooted tree is their deepest common ancestor.*

Suppose the new group pair in an online step is (S_i, T_i). Our first step is to identify a collective flow of 1 between vertices in S_i and T_i that can be supported by the fractional solution; such a flow is guaranteed by the fact the fractional solution is feasible in conjunction with Menger's theorem (see e.g. [24]). We decompose this flow into flow paths characterized by their endpoints $(s_{i1}, t_{i1}), (s_{i2}, t_{i2}), \ldots$ where $s_{ij} \in S_i, t_{ij} \in T_i$. Let $f_i^{(v)}(e)$ denote the total flow routed through edge e on flow paths such that the least common ancestor of s_{ij}, t_{ij} in R is vertex v. Here, e is an edge in the subtree (denoted R_v) of R rooted at v. We view the flow from S_i to T_i through v in R_v as a flow from S_i to v *and* a separate flow from T_i to v of the same value; let $f_i^{(v)}$ denote the value of these two flows. For tree R_v, we select a scaling factor γ_v from the distribution (assuming wlog that n is a power of 2)

$$\mathbb{P}[\gamma_v = 2^{i+1}] = \frac{1}{2^i}, \text{ for } 1 \leq i \leq \lg n,$$

and $\gamma_v = 1$ with the remaining probability. Now, let $x_i^{(v)}(e) = \max_{j \leq i}\{f_j^{(v)}(e)\}$. and $y_i^{(v)}(e) = \gamma_v x_i^{(v)}(e)$.

Our rounding algorithm works on each subtree R_v separately as follows. We round the edges e in R_v in topological order away from root v using the following rule. Here, $e(p)$ is the parent edge of e. The rules are identical to that of Alon *et al* in [6]; the only difference is that we use the scaling procedure before applying the rules.

- If $y_i^{(v)}(e) \geq 1$, then we select edge e.
- If e is incident on v or $y_i^{(v)}(e(p)) \geq 1$, then we select edge e with probability $\min\left(\frac{y_i^{(v)}(e) - y_{i-1}^{(v)}(e)}{1 - y_{i-1}^{(v)}(e)}, 1\right)$.
- If $e(p)$ is already in the integer solution or has been selected, then we select edge e with probability $\min\left(\frac{y_i^{(v)}(e) - y_{i-1}^{(v)}(e)}{y_i^{(v)}(e(p)) - y_{i-1}^{(v)}(e)}, 1\right)$.

We perform the above selection twice with independent randomness and include all edges selected in either round to the integer solution.

Analysis First, we show the following property of the scaling procedure.

Lemma 4.18. *For any terminal set pair (S_i, T_i), with constant probability, there is at least one tree R_v where the variables $y_i^{(v)}$ can support a flow of at least 1.*

Proof. For any tree R_v, the probability that $y_i^{(v)} < 1$ is at most $1 - x_i^{(v)} \leq 1 - f_i^{(v)}$. The lemma now follows from the fact that $\sum_{v \in R} f_i^{(v)} \geq 1$. □

The next lemma now follows directly from the analysis of Alon *et al.*

Lemma 4.19. *For any R_v, a path is selected in the integer solution from v to some vertex in S_i and some vertex in T_i with probability $\Omega\left(\frac{1}{\log^2 n}\right)$.*

Proof. Lemma 4.18 asserts that with constant probability, the variables $y_i^{(v)}$ are feasible in some subtree R_v. Then, by the analysis of Alon *et al*, with probability $\Omega\left(\frac{1}{\log^2 n}\right)$, the integer solution on that subtree contains a path from some vertex in S_i to v (resp., from some vertex in T_i to v). The two independent rounding iterations ensure that this property holds simultaneously for S_i and T_i with probability $\Omega\left(\frac{1}{\log^2 n}\right)$.

□

Lemma 4.20. *The cost of the integer solution is at most $O(h \log n \sum_{e \in E} x_e)$.*

Proof. First, we note that each edge in R_v is selected with probability at most $2y_i^{(v)}(e)$, given γ_v. This follows from the analysis of Alon *et al* [5], the factor of 2 arising because of the two rounding iterations. Now, we observe that edge e appears in at most h subtrees R_v, and the integer solution obtained from each tree R_v has cost at most

$$\sum_{e \in R_v} y_k^{(v)}(e) \leq \sum_{e \in R_v} x_e \mathbb{E}[\gamma_v] \leq O(\log n \sum_{e \in R_v} x_e).$$

□

Lemmas 4.13, 4.19, and 4.20 imply Theorem 4.12.

4.4.2 Online Node-weighted Group Steiner Forest

We first consider the NW version of this problem on general graphs. The following structural lemma about an offline optimal solution (which generalizes a similar lemma for the EW case due to Robins and Zelikovsky [84]) is key to our reduction of this problem to the corresponding problem on trees.

Lemma 4.21. *Given any instance of the NW group Steiner forest problem on a graph $G = (V, E)$ with terminal group pairs $\mathcal{T} = ((S_1, T_1), (S_2, T_2), \ldots, (S_k, T_k))$ ($S_i, T_i \subseteq V$ for $1 \leq i \leq k$), there exists another instance of the NW group Steiner forest problem on a graph $G' = (V', E')$ where $V \subseteq V'$ with the same terminal group pairs $\mathcal{T} = ((S_1, T_1), (S_2, T_2), \ldots, (S_k, T_k))$ such that*

- *For any feasible solution H for the the instance on graph G, there exists a feasible solution H' for the instance on graph G' such that $c(H') \leq 3\lceil \lg k \rceil c(H)$.*
- *For any feasible solution H' for the instance on graph G', there exists a feasible solution H for the instance on graph G such that $c(H) \leq c(H')$.*
- *There is an optimal solution H' for the instance on G' such that every tree in H' has depth at most $\lceil \lg k \rceil$.*

Proof. For simplicity, we will allow edge costs in G'; each edge can be replaced by a path of length 2 losing a factor of 2 in the height of the tree if we are restricted to node costs only.

Let $V' = V$ and for every pair of vertices in $u, v \in V$, we have an edge in G' whose cost is the minimum cost path between u and v in G (excluding c_u, c_v).

Let H be any group Steiner forest in G and let T be any tree in H. We assume without loss of generality that all terminals are leaves in T by adding dummy nodes of cost 0. Now, we construct a tree T' of logarithmic depth in G' as follows. The leaves of T' (at level 1) are the leaves of T. Root tree T and pair the leaves from left to right. Call the least common ancestor of each pair a level 2 vertex in T'. Further, connect each level 2 vertex v to the two level 1 vertices whose least common ancestor is v. In general, suppose we have constructed T' up to level d. Then we sort the vertices of level d of tree T' from left to right in tree T and pair them up. For each pair, we add the least common ancestor in level $d + 1$ and join it to the two vertices in level d for which it is the least common ancestor. Moreover, we merge all copies of a vertex in a single level into a single vertex. We terminate after we get the root of T' at level $\lceil \lg k \rceil$.

Now, we bound the cost of tree T'. Each vertex in T' is also in T and appears at most once in a level; hence node costs in T' add to at most $\lceil \lg k \rceil$ times that in T. Now we bound the cost of edges in T'. For each edge $e \in E(T')$, replace it by the cheapest path in T between its two endpoints. Now, consider any vertex $v \in T$. A path between two vertices includes v as an internal vertex if one endpoint of the path is in the subtree rooted at v and the other outside v. Since the vertices are paired left to right in each level, this can happen at most twice in each level. Hence, vertex v is included at most $2\lceil \lg k \rceil$ times as an internal vertex among all the paths.

The converse direction simply follows by replacing each edge in T' by the corresponding path in G and taking a spanning tree over the resulting graph. □

Using Lemma 4.21, we construct a reduction from NW group Steiner forest on general graphs to trees, and show the next lemma.

Lemma 4.22. *Given a instance of the group Steiner forest problem on a graph $G = (V, E)$ such that each tree of the optimal forest has depth at most $\lceil \lg k \rceil$, there exists an instance of the group Steiner forest problem on a tree R of size $O(n^{\lceil \lg k \rceil})$ such that every feasible solution on G corresponds to a feasible solution on R of the same cost and vice versa.*

Proof. The tree $R = (V_R, E_R)$ has $\lg k$ levels indexed by $0, 1, 2, \ldots, \lceil \lg k \rceil - 1$. Level i contains n^i copies of each vertex in V, the cost of each copy of a vertex being equal to its cost in G. To index these vertices, let us first arbitrarily index the vertices in V as $v_1, v_2, \ldots, v_n$. Then, the vertices in level i are denoted by $(v_p, j_1, j_2, \ldots, j_i)$ where each $1 \leq j_s \leq n$ and $1 \leq p \leq n$. For each $i \geq 0$, edges between $(v_p, j_1, j_2, \ldots, j_i)$ and $(v_q, j_1, j_2, \ldots, j_i, p)$ for each $1 \leq p, q \leq n$, of cost $d_{v_p v_q}$, i.e. the distance between vertices v_p and v_q in graph G (excluding c_{v_p}, c_{v_q}) are added to R. For a terminal group pair (S_i, T_i) in the NW group Steiner forest problem, we introduce a terminal set pair (S'_i, T'_i) in R where S'_i contains all copies of vertices in S_i and T'_i contains all copies of vertices in T_i.

Consider any feasible solution H in graph G. Root every tree T of H at any vertex, say v_T. Then, there is a copy of this tree rooted at the unique copy of vertex v_T at level 0.

Conversely, consider any feasible solution H' in tree R. For any tree T' in H', there exists a subgraph T connecting exactly the same set of nodes connected by T' and of cheaper cost. This follows from the fact that every edge between a copy of node v_p and v_q in R corresponds to a path between v_p and v_q in G. □

Lemma 4.22 and Theorem 4.12 implies Theorem 4.2.

4.4.3 Online Edge-weighted Group Steiner Forest

We now give a polynomial-time algorithm for the online EW group Steiner forest problem and prove Theorem 4.3. The algorithm follows from a reduction to the EW group Steiner forest problem on a tree using small-depth low-distortion probabilistic tree embeddings [28].

Theorem 4.23 (Fakcharoenphol *et al* [28])**.** *There exists a polynomial-time algorithm that finds, for any metric space on n points, a distribution on tree metrics on the same set of points such that the average distortion is $O(\log n)$. Moreover,*

- *each tree in the support of the distribution has depth $O(\log \Delta)$ where Δ is the diameter of the metric space, and*
- *the length of an edge at level i is 2^i and the minimum distance between any two points is 1.*

Now, we give our algorithm for the online EW group Steiner forest problem. Consider the shortest path metric d on $G = (V, E)$ with weights given by c_e on edge e. Sample one of the trees from the distribution given by Theorem 4.23 and solve the

online EW group Steiner forest problem on the tree using the algorithm given earlier for the group Steiner forest on trees.

Using Theorem 4.23 and Theorem 4.12, we conclude that the competitive ratio of this algorithm is $O(\log^5 n \log k \log \Delta)$. To remove the dependence on Δ, recall that the cost of an optimal solution is 1 and we never use an edge of cost greater than 1. Therefore, the effective diameter of the graph is at most n. This completes proof of Theorem 4.3.

4.5 Online Edge-weighted Single-Source Vertex Connectivity

We give an algorithm for the online single source ℓ-vertex connectivity problem on EW graphs with competitive ratio $(O(\ell \log k/\epsilon), 2+\epsilon)$ for any fixed $\epsilon > 0$. Let the set of terminals in arrival order be $\{t_1, t_2, \ldots, t_k\}$ and let r be the root that is known offline. In round i, we find the minimum cost (i.e. greedy) vertex-disjoint collection of $\ell/(2+\epsilon)$ paths from terminal t_i to $T_i \cup \{r\}$, where $T_i = \{t_j : j < i\}$, such that each $t_j \in T_i$ is the endpoint of at most one of these paths. This can be done in polynomial time by using a standard min-cost flow subroutine. The selected paths are added to the solution.

Analysis

Our analysis will combine tools from this chapter with those developed by Chuzhoy and Khanna for the offline version of this problem [23]. In fact, we will use the techniques used by Chekuri and Korula [21] to prove the theorems of Chuzhoy and Khanna. Our starting point is the the following theorem (Theorem 4.4[1] in [21]).

Theorem 4.24. *Suppose T is a set of vertices each of which is ℓ-vertex connected to a fixed vertex r in an undirected EW graph $G = (V, E)$. Let $T \cup \{r\}$ be called* black *vertices, while all other vertices in V are* white*. Then, there exists a subgraph H of G whose edges can be decomposed into a set of spiders such that:*

- *For each spider, its feet are distinct black vertices and all the intermediate vertices are white.*
- *Each black vertex is the foot of exactly ℓ spiders, and each white vertex appears in at most one spider.*
- *If a white vertex is the head of a spider, then the spider has at least two feet.*

We use this theorem to prove our key structural lemma.

Lemma 4.25. *Suppose we are given an ℓ vertex-connected EW graph $G = (V, E)$ with terminals T and root r, where the cost of edge e is c_e. For an arbitrary ordering of*

[1] All references are to the full version of the paper at `http://arxiv.org/abs/0902.2795`.

terminals $t_1, t_2, \ldots, t_k$, let $ElemCost(t_i)$ denote the minimum cost internally *vertex-disjoint collection of $\ell/(2+\epsilon)$ paths from terminal t_i to $T_i \cup \{r\}$, where $T_i = \{t_j : j < i\}$. Then, $\sum_{i=1}^{k} ElemCost(t_i) = O(\log k/\epsilon) \sum_{e \in E} c_e$.*

Proof. We give an algorithm that constructs collections of paths with the given cost bound. First, we obtain the spiders in Theorem 4.24 for graph G. Now, for each spider with s terminals, we can find one path each for $s-1$ terminals to a previous terminal (or to r) in the spider such that each edge is used at most twice in these paths (see our proof of Lemma 4.5 for a construction of these paths). We say that a terminal t_i is *satisfied* if these paths contain at least $\ell/(2+\epsilon)$ paths from t_i to a vertex in $T_i \cup \{r\}$. (Note that these paths are internally vertex disjoint since the spiders are disjoint for white vertices.) By averaging, at least k/ϵ terminals are satisfied by these paths. We remove these terminals from our set of terminals and recurse. Note that for any terminal, all the paths come from a single recursive subproblem; hence, they must be internally vertex disjoint. We conclude by noting that the cost bound follows from the fact that the number of recursive calls is $\log k/\epsilon$. □

Comparing this lemma to Lemma 4.3 in [21], we note that while the previous lemma gets a better cost bound and guarantees ℓ internally vertex-disjoint paths (rather than $\ell/2$) for every vertex, our lemma is robust to the online ordering of terminals.

Our final step is to use Lemma 4.25 to deduce the competitive ratio of the algorithm. This step closely mirrors the proof of Theorem 4.2 in [21].

Theorem 4.26. *Suppose $AugCost(t_i)$ is the cost of the greedy algorithm for terminal t_i. Then, $\sum_{i=1}^{k} AugCost(t_i)$ is $O(\ell \log k/\epsilon)$ times the cost of an optimal solution.*

Proof. Let $\lambda = \ell/(2+\epsilon)$. Following Chekuri and Korula [21], we give an iterative algorithm that runs for $4\lambda^2$ iterations and gives, in iteration j, a set of λ internally vertex disjoint paths $P_j(t_i)$ from each terminal t_i to $T_i \cup \{r\}$ such that:

- For each terminal t_i, every other terminal is an end-point in fewer than $4\lambda^2 + 2\lambda$ paths in $\cup_{j=1}^{4\lambda^2} P_j(t_i)$.
- $\sum_{i=1}^{k} c(P_j(t_i))$ is $O(\ell \log k/\epsilon)$ times the cost of an optimal solution, where $c(P_j(t_i))$ is the sum of costs of paths in $P_j(t_i)$.

The theorem follows from these properties using standard arguments (see [21] for details).

We now describe the iterative algorithm and show that it satisfies the above two properties. The first property is proved inductively, and will also define our algorithm. Specifically, we will show that after j rounds, for each terminal t_i, every other terminal is an end-point in fewer than $4\lambda^2 + 2\lambda$ paths in $\cup_{j'=1}^{j} P_{j'}(t_i)$. In iteration j, let $Blocked(t_i)$ be the terminals in T_i such that at least $j - 1 + \lambda$ paths in $\cup_{j'=1}^{j-1} P_{j'}(t_i)$ terminate on each of them. We construct a meta-graph on the terminals where we have an edge between $(t_i, t_{i'})$ if $t_{i'} \in Blocked(t_i)$. In any induced subgraph of this meta-graph, the minimum degree of a vertex is at most $2(\lambda - 1)$; hence, this meta-graph is $2\lambda - 1$ colorable. For each such color class C, we apply Lemma 4.25 (with

the ordering on the terminals in the color class induced by the overall ordering) to obtain λ internally vertex-disjoint paths $P_j(t_i)$ from each terminal t_i to terminals in $(T_i \cap C) \cup \{r\}$. If a path in $P_j(T_i)$ contains another terminal in T_i internally, then we terminate the path at the first such terminal. The second property follows immediately from Lemma 4.25.

We now prove the first property inductively. For any t_i, if $t_{i'} \in \mathit{Blocked}(t_i)$, then there is at most one path in $P_j(t_i)$ that terminates at $t_{i'}$ since the paths in $P_j(t_i)$ are internally vertex disjoint. Thus, the first property holds after iteration j. On the other hand, if $t_{i'} \notin \mathit{Blocked}(t_i)$, then the first property holds after iteration j even if all λ paths of t_i terminate at $t_{i'}$. □

4.6 Concluding Remarks

In this chapter, we gave online algorithms with poly-logarithmic competitive ratios for a set of fundamental network design problems. Several questions are left open by our work. While we gave a polynomial time algorithm for the online NW Steiner tree problem, our algorithm for the generalizations of this problem to the Steiner forest/group Steiner tree/group Steiner forest settings have a quasi-polynomial running time. It would be desirable to design online algorithms for these problems that have a poly-logarithmic competitive ratio and polynomial running time. While we gave a bi-criteria approximation algorithm for the online ℓ vertex connectivity problem for EW graphs, obtaining an algorithm for this problem without relaxing the constraint is an open question.

4.7 Notes

This chapter is based on joint work with Joseph (Seffi) Naor and Mohit Singh. A preliminary version of this work appeared in the *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, 2011.

Chapter 5

Network Activation Problems

Network design problems have traditionally assumed that individual edges (resp., vertices) have fixed costs and can be bought independent of other edges (resp., vertices). While being reasonably accurate for wired networks, this model does not faithfully represent wireless networks, where the activation of an edge is dependent on the selection of parameter values (such as transmitter power level) at its endpoints, and the cost incurred is a function of these values. In this chapter, we present a realistic optimization model for the design of survivable wireless networks called the *network activation* model that generalizes various suites of connectivity problems studied in the theory literature, e.g. node-weighted Steiner connectivity problems and power optimization, as well as problems studied in the networking literature, e.g. installation cost optimization. We obtain algorithmic results for several fundamental network design problems in the network activation model.

5.1 Background

In wireless networks, one typically needs to choose the value of a parameter x_v from a domain of possible values D at each vertex $v \in V$ (V being the set of vertices) and the *activation* of an edge (u, v) depends on the parameters chosen at its endpoints. Formally, there is an *activation function* $f_{uv} : D \times D \to \{0, 1\}$ that takes as input the values of x_u and x_v and returns 1 iff the edge (u, v) is activated for the chosen values of x_u and x_v. For example, one might need to set power levels at the nodes, and depending on the power levels of the endpoints, an edge is either active or inactive; or, one may need to fix the heights of towers for mounting antennas at nodes, and whether an edge is active depends on whether there is line-of-sight between the antennas at its endpoints. The objective is to minimize the total cost $\sum_{v \in V} x_v$, while ensuring that the activated set of edges satisfies some connectivity requirement $\mathcal{C}$. We call this the network activation model.

The Network Activation Problem

The input comprises a graph $G = (V, E)$, a domain of parameter values D, activation functions $f_{uv} : D \times D \to \{0, 1\}$ for every edge $(u, v) \in E$, and a connectivity requirement $\mathcal{C}$. The goal is to select a parameter $x_v \in D$ at each vertex $v \in V$ such that $\sum_{v \in V} x_v$ is minimized subject to the constraint that the set of activated edges (i.e. (u, v) with $f_{uv}(x_u, x_v) = 1$) satisfies connectivity requirement $\mathcal{C}$.

Before proceeding further, let us give some examples of activation functions and the corresponding suite of problems defined by them:

- **Node-weighted Network Design.** The activation functions are defined by

$$f_{uv}(x_u, x_v) = \begin{cases} 1, & \text{if } x_u \geq c_u \text{ and } x_v \geq c_v \\ 0, & \text{otherwise.} \end{cases}$$

- **Power Optimization.** (see e.g. [47, 48, 71, 66, 67, 77]) In this problem, each edge (u, v) has a *threshold power requirement* θ_{uv} and edge (u, v) is activated if the power at each of its endpoints is at least this threshold. Thus,

$$f_{uv}(x_u, x_v) = \begin{cases} 1, & \text{if } \min(x_u, x_v) \geq \theta_{uv} \\ 0, & \text{otherwise.} \end{cases}$$

- **Installation Cost Optimization.** (see e.g. [27, 85, 81]) The installation cost of a wireless network is dominated by the cost of building towers at the nodes for mounting antennas, which in turn is proportional to the height of the towers. An edge (u, v) is activated if the towers at its endpoints u and v are tall enough to overcome obstructions in the middle and establish line-of-sight between the antennas mounted on the towers; this is modeled as each edge (u, v) having a *threshold height requirement* τ_{uv}, and edge (u, v) is activated if the scaled height of the towers at its endpoints sum to at least τ_{uv}. Thus,

$$f_{uv}(x_u, x_v) = \begin{cases} 1, & \text{if } \alpha_{u,uv} x_u + \alpha_{v,uv} x_v \geq \tau_{uv} \\ 0, & \text{otherwise,} \end{cases}$$

  where $\alpha_{u,uv} x_u, \alpha_{v,uv}$ are constants.

Finite domain. In practice, the set of possible parameter values is often a small, discrete finite set. Therefore, we consider finite, discrete domains D, and allow our algorithms to run in time polynomial in $|D|$ (so e.g., activation functions are lookup tables). For the problems described above, this is without loss of generality. In node-weighted Steiner network, we can restrict D to the vertex costs and 0. In power optimization, restricting D to the thresholds on the edges does not change the optimal

solution. In installation cost optimization, it can be shown that the same restriction increases the optimal cost by at most a factor of two; as we show later, this problem is NP-hard to approximate to a factor of $o(\log n)$ and hence this transformation is without significant loss in the approximation factor.

Monotonicity. We will assume throughout that the activation functions are *monotonic*, i.e. $f_{uv}(a_1, b_1) = 1$ implies $f_{uv}(a_2, b_2) = 1$ for any $a_2 \geq a_1, b_2 \geq b_1$. This is indeed the case in all the above applications, and in any other realistic scenario.

In this chapter, we consider several connectivity requirements $\mathcal{C}$ which are summarized in Table 5.1. Besides being theoretically important, these connectivity requirements are also practically relevant since they ensure robustness against edge and node failures.

5.2 Our Contributions

We now outline the main results presented in this chapter.

We show the following hardness of approximation result for the MSAT problem.

Theorem 5.1. *It is NP-hard to approximate the MSAT problem to a factor of $o(\log n)$, even for the installation cost setting.*

We give the following result for the MSAF problem.

Theorem 5.2. *There is a deterministic $O(\log k)$-approximation algorithm for the MSAF problem.*

The approximation factor in this theorem is optimal up to constant factors since the MSAF problem is known to be NP-hard to approximate to a factor of $o(\log k)$ even in the node-weighted setting [64]. In the power optimization setting, there is a 4-approximation algorithm for the MSAF problem [67]; no previous result is known for installation cost optimization.

As a corollary of Theorem 5.2, we obtain the following result for the MSAT problem.

Corollary 5.3. *There is a deterministic $O(\log n)$-approximation algorithm for the MSAT problem.*

The approximation factor in this theorem is optimal up to constant factors by Theorem 5.1. Corollary 5.3 generalizes an $O(\log n)$-approximation algorithm for the installation cost setting [81]; for power optimization, the problem is much more tractable, and a 5/3-approximation algorithm is known [7].

We give the following results for the MVAN and MEAN problems with $R = 2$.

Theorem 5.4. *There is a deterministic $O(\log n)$-approximation algorithm for the MVAN problem with $R = 2$.*

Theorem 5.5. *There is a deterministic $O(\log n)$-approximation algorithm for the MEAN problem with $R = 2$.*

Minimum Edge-connected Activation Network (MEAN)	Each pair of vertices must have R edge-disjoint paths in the activated subgraph, for some given $R > 0$.
Minimum Vertex-connected Activation Network (MVAN)	Each pair of vertices must have R vertex-disjoint paths in the activated subgraph, for some given $R > 0$.
Minimum Steiner Activation Forest (MSAF)	Each of k given pairs of vertices must be connected by a path in the activated subgraph.
Minimum Spanning Activation Tree (MSAT)	The activated set of edges must contain a spanning tree on the vertices. This is a special case of the MEAN and MVAN problems for $R = 1$, and the MSAF problem where the set of terminal pairs is $\{(r, v) : v \in V - \{r\}\}$ for some vertex $r \in V$.
Minimum Degree Activation Network (MDAN)	For every vertex subset U in a given partition of vertices $\mathcal{P}$, there must be at least R_U activated edges with exactly one endpoint in U, where R_U is the (given) *requirement* of U.
Minimum Activation Flow (MAF)	The activated set of edges must contain R edge-disjoint paths between two specified vertices s, t.
Minimum Activation Path (MAP)	The activated set of edges must contain a path between two specified vertices s, t. This is a special case of the MAF problem for $R = 1$, and the MSAF problem with only one terminal pair $\{s, t\}$.

Table 5.1: Network Activation Problems

Previously, no result was known for installation cost optimization, but the problems have 4-approximation [16] and 11/3-approximation [67] algorithms respectively in the power optimization setting.

We show the following connection between the MEAN and MDAN problems.

Lemma 5.6. *For any connectivity requirement R, an α-approximation algorithm for the* MDAN *problem implies an $O(\alpha \log n, 2)$-approximation*[1] *algorithm for the* MEAN *problem.*

As an application, we obtain the following result for the MEAN problem with arbitrary k in the installation cost optimization setting.

Theorem 5.7. *For any $\epsilon > 0$ and any connectivity requirement k, there is a deterministic algorithm for the MEAN algorithm in the installation cost optimization setting that has an approximation ratio of $(O(\log n \log(4 + 8/\epsilon)), 2 + \epsilon)$.*

Note that this theorem implies the next corollary by setting $\epsilon < 4/(k-2)$ and using the integrality of edge connectivity.

Corollary 5.8. *For any connectivity requirement k, there is an $(O(\log n \log k), 2)$-approximation algorithm for the MEAN problem in the installation cost optimization setting.*

We give an exact algorithm for the MAP problem.

Theorem 5.9. *The MAP problem is solvable in polynomial time.*

However, by an observation of Nutov [78] for the node-weighted setting, the MAF problem is at least as hard as the well-known ℓ-densest subgraph problem [31, 12].

Roadmap

This chapter is organized as follows. The proof of the lower bound for the MSAT problem appears in Section 5.3 and the $O(\log k)$-approximation algorithm for the MSAF problem is presented in Section 5.4. In Sections 5.5 and 5.7, we present our algorithms for the MVAN and MEAN problems with $k = 2$ respectively. We establish the connection between the MDAN and MEAN problems, and use it to obtain algorithms for the MEAN problem with arbitrary k for the installation cost optimization setting in Section 5.7. We give an exact algorithm for the MAP problem in Section 5.8.

[1] *Recall that a (β, γ)-approximation for the MEAN problem implies that the cost of the algorithmic solution is β times that of an optimal solution, but the algorithmic solution achieves an edge connectivity of k/γ instead of k.*

5.3 Minimum Spanning Activation Tree

In this section, we prove Theorem 5.1 by showing that the MSAT problem in the installation cost setting generalizes the Minimum Connected Dominating Set (MCDS) problem. To describe the MCDS problem, we need to define a dominating set.

Definition 5.10. *A subset of vertices S is said to be a* dominating set *if every vertex in the graph is either in S or has an incident edge whose other endpoint is in S.*

The Minimum Connected Dominating Set Problem

The input comprises an undirected unweighted graph $G = (V, E)$ and the goal is to find a connected dominating set in G containing the least number of vertices.

It is known that the MCDS problem is NP-hard to approximate to a factor of $o(\log n)$ [42].

We also need to define the notion of a vertex cover for this reduction.

Definition 5.11. *A* vertex cover *is a subset of vertices S such that every edge is incident on at least one vertex in S.*

Given an instance $G = (V, E)$ of the MCDS problem, we define an instance of the MSAT problem on V as follows: *The domain $D = \{0, 1\}$ and for any pair of vertices u, v, if $(u, v) \in E$ then $f_{uv}(a, b) = 1$ iff $a + b \geq 1$, while if $(u, v) \notin E$, then $f_{uv}(a, b)$ is identically 0.* The following theorem establishes the validity of this reduction.

Theorem 5.12. *If there is a connected dominating set S in G that contains c vertices, then there is a solution of cost c to the instance of the MSAT problem. Conversely, if there is a solution to the instance of the MSAT problem of cost c, then there is a connected dominating set S in G that contains at most $2c$ vertices.*

Proof. For the forward direction, observe that setting $x_v = 1$ for all vertices in S activates a spanning subgraph of G. For the converse direction, note that we have chosen the activation function in a way that the cost of activating a set of edges is at least its *minimum vertex cover*. Decompose an activated spanning tree into a set of node disjoint paths in the following manner: Root the tree at an arbitrary vertex and remove an arbitrary root-to-leaf path from the tree; then recurse on each tree of the resulting forest. Note that each path in this decomposition contains at most one leaf vertex. Then, a vertex cover of the tree must include (disjoint) vertex covers of each of these paths, and the size of a vertex cover of any of these paths must be at least half the number of non-leaf vertices in the path. Thus, the number of non-leaf vertices in the optimal spanning tree in the MSAT solution is at most $2c$, and these vertices form a connected dominating set of the graph. □

This completes the proof of Theorem 5.1.

5.4 Minimum Steiner Activation Forest

In this section, we prove Theorem 5.2 by giving an approximation-preserving reduction of the MSAF problem to the node-weighted (NW) Steiner forest problem. Given an instance of the MSAF problem on a set of vertices V with terminal pairs $R \subseteq V^{(2)}$ and activation functions f_{uv}, we construct an instance of the NW Steiner forest problem as follows: *For each vertex $v \in V$, construct a star with $|D| + 1$ vertices having v_0 as its center and $\{v_a : a \in D\}$ as the peripheral vertices. v_0 has cost 0 while v_a has cost a for each $a \in D$. Now, connect u_a to v_b with an edge iff $f_{uv}(a, b) = 1$. The terminal pairs in the constructed graph are $\{(u_0, v_0) : (u, v) \in R\}$.* The following lemma establishes that the validity of the reduction.

Lemma 5.13. *For any solution to the instance of the MSAF problem, there is a solution to the instance of the NW Steiner forest problem constructed by the reduction with at most as much cost, and vice-versa.*

Proof. Suppose that in the MSAF solution, $x_v = a_v$ for vertices $v \in V$ and let the activated edges be T. Then, the edges $\{(v_0, v_{a_v}) : v \in V\}$ and edges $\{(u_{a_u}, v_{a_v}) : (u, v) \in T\}$ connect the terminal pairs and have total node cost equal to $\sum_{v \in V} a_v$ in the NW Steiner forest instance.

Conversely, for any solution to NW Steiner forest instance, if there are multiple v_a vertices in the solution, then we only retain the vertex with the maximum value of a, breaking ties arbitrarily if required. By the monotonicity property of the activation functions, this retained vertex has edges to all vertices that were connected via the vertices that we did not retain. Therefore, after the transformation, we obtain a new Steiner forest with at most as much weight as the original solution. Now, in the MSAF instance, set $x_v = a$ if v_a is selected in the transformed solution to the NW Steiner forest instance. Clearly, all the terminal pairs can be connected using these values of x_v since the solution to the NW Steiner forest instance connects all the terminal pairs. □

Theorem 5.2 now follows from the $O(\log k)$-approximation algorithm of Klein and Ravi [64] for the NW Steiner forest problem.

5.5 Minimum Vertex-connected Activation Network with $R = 2$

In this section, we give an algorithm for the MVAN problem for $R = 2$. Our algorithm has two phases.

- In the first phase, we run the $O(\log n)$-approximation algorithm for the MSAT problem in Corollary 5.3. This produces a connected spanning activation subgraph whose cost is at most $O(\log n)$ times that of an optimal solution of the MVAN problem.

- In the second phase, we give an $O(\log n)$-approximation algorithm for optimally augmenting the solution from the first phase to make the activated subgraph biconnected.

To describe the second phase of the algorithm, we need to define *block-cutpoint* graphs [51].

Definition 5.14. *A* block *of an undirected graph is a maximal biconnected subset of vertices, while a* cutpoint *is a vertex whose removal increases the number of components in the graph. A* block-cutpoint graph *T of an undirected graph G is a graph whose nodes are the blocks and cutpoints of G and edges connect each block to the cutpoints contained in it.*

It turns out that the block-cutpoint graph T of a connected graph G is a tree (hence we will call it a block-cutpoint tree) such that the vertex subsets that T and G split into on removing a cutpoint v correspond to each other. We introduce the notion of *partition number* of a vertex/graph.

Definition 5.15. *The* partition number *of a vertex v (denoted $\rho(v)$) in an undirected graph G is the number of components G splits into when v is removed minus one. Equivalently, if v is a cutpoint, $\rho(v)$ is the degree of v in the block-cutpoint tree T minus one; if v is not a cutpoint, $\rho(v) = 0$. Overloading our notation, the partition number of graph G (denoted by $\rho(G)$) is the sum of the partition numbers of its vertices.*

We show the following property of $\rho(G)$ for connected graphs.

Lemma 5.16. *If G is a connected graph, then its partition number $\rho(G)$ is at most $2n - 4$.*

Proof. The partition number of any spanning tree of G containing ℓ leaves is $2(n - 1) - \ell$ (since each non-leaf vertex v is a cutpoint whose removal splits the graph into a number of components equal to the number of edges incident on v), which is at least $2n - 4$ since any spanning tree has at least two leaves. To complete the proof, note that adding edges does not increase the partition number of a graph. □

Observe that the partition number of a biconnected graph is 0. So, the process of augmenting a connected graph into a biconnected one decreases the partition number from at most $2n - 4$ to 0. This motivates us to describe a greedy algorithm where, in each round, the goal is to obtain the maximum decrease in $\rho(A)$ of the activated subgraph A while minimizing cost. To describe each such round, we need to define a star.

Definition 5.17. *A* star *is a graph where at most one vertex has degree greater than one; this vertex is called the* center *while the other vertices are called* peripheral *vertices. (A single edge is also a star, but either vertex can be called the center.)*

In each round of the algorithm, we add a star s that minimizes the ratio c_s/b_s, where the *cost* c_s is the sum of x_v of the vertices in s required to activate all the edges in s, and the *benefit* b_s is the decrease in the partition number of the activated subgraph as a result of adding s to it. Before giving a polynomial-time implementation of this algorithm, we analyze its approximation ratio.

Analysis

We will now prove Theorem 5.4. The following lemma is crucial.

Lemma 5.18. *Let $G = (V, E)$ be any connected graph and F_1, F_2 be two sets of edges on V. Let b_1, b_2 and $b_{1,2}$ be the decrease in the partition number of G due to the addition of F_1, F_2 and $F_1 \cup F_2$ respectively. Then, $b_{1,2} \leq b_1 + b_2$.*

Proof. For any cutpoint v, let the vertex subsets that graph G decomposes into on removing v be called the *components* of v. Define a star graph G_v where v is the central vertex and each component of v is contracted into a single peripheral vertex. Then, the decrease in the partition number of v due to the addition of a set of edges F is equal to the number of edges in any spanning forest of F on graph G_v. Clearly, the sum of the number of edges in spanning forests of F_1 and F_2 on G_v is at least as much as the number of edges in a spanning forest for $F_1 \cup F_2$. Thus, the decrease in $\rho(v)$ due to F_1 and F_2 separately is at least at much as the decrease in $\rho(v)$ due to $F_1 \cup F_2$. To complete the proof, note that the decrease in $\rho(G)$ is the sum of decreases in $\rho(v)$ for the cutpoints v. □

Let (V, A) be a connected graph, and let F be any *minimal* set of edges such that $(V, F \cup A)$ biconnected. Because of minimality, F must be a forest. We root each tree of this forest arbitrarily and decompose each tree into stars $\mathcal{S}$ centered at each non-leaf vertex and containing its children as peripheral vertices. Then, the previous lemma ensures that $\mathcal{S}$ satisfies the next lemma.

Lemma 5.19. *$\mathcal{S}$ satisfies both the following properties:*

- *Each vertex in V appears in at most two stars in $\mathcal{S}$.*
- *The sum of benefits of the stars in $\mathcal{S}$ is at least the partition number of (V, A).*

Proof. Each vertex can only appear in the stars centered at itself and its parent in the tree containing it in F. Since $A \cup F$ is a biconnected graph, the benefit of F, and therefore the sum of benefits of the individual stars by Lemma 5.18, is at least the partition number of A. □

Theorem 5.4 now follows from the above lemma using standard techniques (cf. the analysis of the greedy approximation algorithm for the set cover problem in e.g. [89]).

5.5.1 Minimum Leaf-weighted Subtree

We now show that we can reduce the problem of finding the optimal star in any round of the algorithm to the **Minimum Leaf-weighted Subtree (MLS)** problem and give an exact algorithm for the MLS problem.

> The Minimum Leaf-weighted Subtree Problem
>
> The input comprises a node-weighted tree T rooted at a vertex r and a parameter $\lambda > 0$. The goal is to choose a subtree rooted at r containing at least λ edges that minimizes the sum of costs of the (non-root) leaves of the subtree.

Finding Optimal Star Using Minimum Leaf-weighted Subtree

Since there are n vertices and $|D|$ possible values of x_v for any vertex v, it is sufficient to give an algorithm to find the optimal star among stars centered at a particular vertex v and having $x_v = a$ for some fixed $a \in D$. Since all edges in T are between a block and a cutpoint, and all the leaves are blocks, the tree T can be decomposed into a set of maximal stars, each of which is centered at a cutpoint and has blocks as its peripheral vertices; the cutpoint at the center is contained in the peripheral blocks in G. We call each such star a *full component*. For the purposes of this reduction, if v is a cutpoint, we will consider the full component containing v as a single block that appears in all the full components that any block containing v appeared in. With this assumption, we root the tree T at the unique block containing v. Each full component C now has a root block r_C. We replace each full component C in T with edges between r_C and the other blocks in C. Let S be the resulting rooted tree defined on the blocks. Now, for each vertex $u \in V$ $(u \neq v)$, define b_u as the unique block containing u if u is not a cutpoint, and as the root block of the full component that was centered at u in T if u is a cutpoint. Then, for every block b (other than the one containing v), we define $c_b = \min_{b_u=b}(x_u^{(v,a)})$ and a unique $e_b = \arg\min_{b_u=b}(x_u^{(v,a)})$ breaking ties arbitrarily if required. The following property ensures that we do not need to consider edges not in $E_b = \{e_b : b \text{ is a block not containing } v\}$ when finding the optimal star.

Lemma 5.20. *There is an optimal star s centered at v and having $x_v = a$ that only contains edges from E_b.*

Proof. In a star, we can replace any edge not in E_b with the edge in E_b having its endpoint other v in the same block; this does not increase the cost or change the benefit of the star. □

For any subset of blocks B that does not include the block containing v, the cost of activating the subset of edges $E_B = \{e_b : b \in B\}$ with $x_v = a$ is $a + \sum_{b \in B} c_b$ and its benefit is the number of edges in the minimal rooted subtree in S containing all blocks in B. This yields the following algorithm for finding the optimal star centered

at v with $x_v = a$ (here, β is the number of blocks in the transformed block-cutpoint tree): *For each $\lambda = 1, 2, \ldots, |S| - 1$, solve the MLS problem to obtain the minimum $\sum_{b \in B} c_b$ given that the benefit has to be at least λ. Now, compare the solutions and take the one that has the best c_s/b_s ratio.*

Exact Algorithm for Minimum Leaf-weighted Subtree

We now give an algorithm for exactly solving the MLS problem for parameter λ on an arbitrary tree S with root r and cost c_v for vertex v. For every vertex, order its children arbitrarily. Then, consider the MLS subproblem for the subtree subtended at a vertex v, but only including the subtrees subtended at the first i children of v, with the constraint that the size of the selected subtree needs to be exactly j. We denote this subproblem by $\text{MLS}(v, i, j)$. We solve the $\text{MLS}(v, i, j)$ problem for all $v \in V$ (in post-order), for all $1 \leq i \leq n_v$ where v has n_v children (in increasing order of i) and for all $1 \leq j \leq \lambda$. Clearly, the overall MLS problem is identical to $\text{MLS}(r, n_r, \lambda)$. Our dynamic program is the following (here $u_1, u_2, \ldots$ are children of v in the arbitrary ordering that we defined):

$$\text{MLS}(v, i, j) = \begin{cases} \min_{\ell=1}^{i}(c_{u_\ell}), & \text{if } j = 1 \\ \text{MLS}(u_1, n_{u_1}, j-1), & \text{if } i = 1 \text{ and } j > 1 \\ \min(\text{MLS}(v, i-1, j), \text{MLS}(v, i-1, j-1) + c_{u_i}, \text{MLS}(u_i, n_{u_i}, j), & \\ \quad \min_{\ell=1}^{j-2}(\text{MLS}(v, i-1, \ell) + \text{MLS}(u_i, n_{u_i}, j-\ell-1))), & \text{otherwise.} \end{cases}$$

To describe this dynamic program, let S_i denote the subtree subtended at u_i plus the edge (v, u_i). In the non-trivial third case above, we compare the following possibilities:

- The entire optimal subtree is contained in $S_1, S_2, \ldots, S_{i-1}$.
- The optimal subtree contains only the (v, u_i) edge from S_i; the remaining edges are from $S_1, S_2, \ldots, S_{i-1}$.
- The entire optimal subtree is contained in S_i.
- Exactly $1 \leq \ell \leq j-2$ of the edges of the optimal subtree are from $S_1, S_2, \ldots, S_{i-1}$ while $j - \ell$ edges are from S_i.

Our base case are the leaf vertices v, for which $\text{MLS}(v, i, j) = \infty$ for all i, j. This dynamic program runs in polynomial time and solves the MLS problem exactly.

5.6 Minimum Edge-connected Activation Network with $R = 2$

In this section, we give an algorithm for the MEAN problem for $R = 2$. As with the corresponding MVAN algorithm, this algorithm has two phases.

- In the first phase, we run the $O(\log n)$-approximation algorithm for the MSAT problem in Corollary 5.3. This produces a connected spanning activation graph whose cost is at most $O(\log n)$ times that of the optimal solution for the MEAN problem.
- In the second phase, we give an $O(\log n)$-approximation algorithm for optimally augmenting the solution from the first phase to make the activated subgraph 2-edge-connected.

To describe the second phase of the algorithm, we need to use the following property of connected graphs.

Lemma 5.21. *A connected graph induces a spanning tree on its 2-edge-connected components.*

Proof. Clearly, the induced graph on the 2-edge-connected components is a connected graph; if it contains cycles, that the maximality of the 2-edge-connected components is violated. □

Observe that this spanning tree is vacuous, i.e. contains no edges, for a 2-edge-connected graph. So, the process of augmenting a connected graph into a 2-edge-connected one decreases the number of edges from at most $n-1$ to 0. Let us denote the number of edges in this spanning tree for a connected graph G as $\eta(G)$. This motivates us to describe a greedy algorithm where, in any round, the goal is to obtain maximum decrease in $\eta(A)$ of the activated subgraph A while minimizing cost. To this end, in each round, we add a star s that minimizes the ratio c_s/b_s, where the cost c_s is the sum of x_v of the vertices in s required to activate all the edges in s, and the benefit b_s is the decrease in the partition number of the activated subgraph as a result of adding s to it. Before giving a polynomial-time implementation of this algorithm, we analyze its approximation ratio.

Analysis

We will now prove Theorem 5.5. The following lemma is crucial.

Lemma 5.22. *Let $G = (V, E)$ be any connected graph and F_1, F_2 be two sets of edges on V. Let b_1, b_2 and $b_{1,2}$ be the decrease in the number of edges in the spanning tree of 2-edge-connected components of G due to the addition of F_1, F_2 and $F_1 \cup F_2$ respectively. Then, $b_{1,2} \leq b_1 + b_2$.*

Proof. The lemma follows from the observation that for any particular edge of the spanning tree, it appears in the benefit if it is in the fundamental cycle of any added edge. □

Let (V, A) be a connected graph, and let Y be any *minimal* set of edges such that $(V, F \cup A)$ 2-edge-connected. Because of minimality, Y must be a forest. We root each tree of this forest arbitrarily and decompose each tree into stars $\mathcal{S}$ centered at each non-leaf vertex and containing its children as peripheral vertices. Then, the previous lemma ensures that $\mathcal{S}$ satisfies the next lemma.

Lemma 5.23. *$\mathcal{S}$ satisfies both the following properties:*

- *Each vertex in V appears in at most two stars in $\mathcal{S}$.*
- *The sum of benefits of the stars in $\mathcal{S}$ is at least the partition number of (V, A).*

Proof. Each vertex appears in at most two stars—one where it is the center and another where its parent in F is the center. The second assertion follows from applying the above lemma multiple times. □

Theorem 5.5 now follows from the above lemma using standard techniques (cf. the analysis of the greedy approximation algorithm for the set cover problem in e.g. [89]).

Finding the Optimal Star Using Minimum Leaf-weighted Subtree

We now show that we can reduce the problem of finding the optimal star in any round of the algorithm to the MLS problem. Similar to the MVAN problem, we focus on finding the optimal star centered at a vertex v with $x_v = a$ for some $a \in D$. For any vertex u, let the 2-edge-connected component containing u be $C(u)$. Then, the benefit of a star s is the number of edges in a minimal subtree of the spanning tree described above that contains all the components $\{C : C = C(u), (u, v) \in s\}$ tree rooted at $C(v)$. For each 2-edge-connected component $C \neq C(v)$, we define $w_C = \min_{C(u)=C} x_u^{(v,a)}$ and a unique $e_C = \arg\min_{C(u)=C} x_u^{(v,a)}$ breaking ties arbitrarily if required. The following properties are crucial.

Lemma 5.24. *There is an optimal star s centered at v and having $x_v = a$ that only contains edges from $E_b = \{e_b : b$ is a block not containing $v\}$.*

Proof. For any star s, first discard all edges with both endpoints in $C(v)$. Then, for any 2-edge-connected component $C \in \{C : \exists (u, v) \in s \text{ s.t. } C(u) = C\}$, we replace the set of edges with the endpoint other than v in C by the edge e_C. The new star formed after the transformation has the same benefit and at most as much cost as s. □

The cost of activating a star $s \subseteq \{e_C : C \neq C(v)\}$ is $a + \sum_{C(u):(u,v)\in s} w_{C(u)}$. This leads to the following algorithm for finding the optimal star centered at v with $x_v = a$ (here, t is the number of edges in the spanning tree): *For each $\lambda = 1, 2, \ldots, t$, solve the MLS problem to obtain the minimum $\sum_C w_C$ given that the benefit has to be at least λ. Now, compare the solutions and take the one that has the best c_s/b_s ratio.*

5.7 Minimum Edge-connected Activation Network for Arbitrary R

In this section, our first goal is to prove Lemma 5.6 which connects the MEAN and MDAN problems. Then, we use this connection to prove Theorem 5.7.

5.7.1 Connection between MEAN and MDAN Problems

Note that by Menger's theorem (see e.g., [24]), the MDAN problem with a degree requirement of R for any vertex partition imposes a strictly weaker constraint that the MEAN problem with connectivity requirement R. Thus, any feasible solution to the MEAN problem is also feasible for the corresponding MDAN problem. We are interested in identifying a converse relationship between these two problems.

Let us first consider the case $R = 1$. Suppose we repeatedly run an MDAN algorithm, where the partition in each iteration is the set of connected components in the subgraph activated by previous iterations, and the requirement $R_U = 1$ for each component. Since the number of components decreases by a factor of at least two in each step, we terminate after at most $\lg n$ iterations yielding an overall approximation factor of $O(\alpha \log n)$ for the MEAN problem if α is the approximation factor for the MDAN problem.

For the same approach to work for arbitrary values of R, we need to quantify the progress we make towards satisfying the R-edge connectivity constraint in each iteration of the MDAN algorithm. For this purpose, we need the following definition:

Definition 5.25. *An* R-edge connected component *is a* maximal *subset of vertices such that every pair of vertices in the subset have* R *edge-disjoint paths between them.*

Edge connectivity is transitive, i.e. if vertex pairs u, v and v, w are R-edge connected, then so too is u, w. This implies that the R-edge connected components form a partition of the vertex set. We can then view our goal as activating a set of edges so that this partition, which has n singleton components before any edge is activated, coalesces into a single R-edge connected component spanning all the vertices. Now, suppose any solution to the MDAN problem were to produce a set of at most cn R-edge connected components, where $c < 1$ is a constant. Let us define these components as the subsets of our partition $\mathcal{P}$ in the next iteration of MDAN. Then, after the next iteration, we get at most $c^2 n$ R-edge connected components. This algorithm, where in each iteration we call the MDAN sub-routine with the subsets of the partition defined as the R-edge-connected components, will terminate in $O(\log n)$ iterations since the number of R-edge-connected components decreases by a constant factor in each iteration. The overall approximation ratio of the algorithm would then be $O(\alpha \log n)$ for the MEAN problem, where α is the approximation factor of the MDAN subroutine.

However, consider a line graph where each edge has $R/2$ parallel copies, except the two edges at the two ends of the line that have R parallel copies each. The minimum degree in this graph is R, but the number of R-edge-connected components is $n - 2$. Therefore, a solution to the MDAN problem need not produce a graph containing at most cn components for some constant c. Instead, we show the following weaker property.

Lemma 5.26. *For any* R*, if* $(c + 1/2)n$ *vertices in an undirected graph* $G = (V, E)$ *have degree at least* R *for some* $0 < c \leq 1/2$*, then the number of* $\lceil R/2 \rceil$*-edge-connected components in* G *is at most* $(1 - c/2)n$*.*

Before proving the lemma, we note that the following corollary follows from the above lemma by setting $c = 1/2$.

Corollary 5.27. *The number of $\lceil R/2 \rceil$-edge-connected components in a graph with minimum degree R is at most $3n/4$.*

For notational brevity, let us replace $\lceil k/2 \rceil$ by $k/2$ in Lemma 5.26; all proofs continue to hold with $\lceil k/2 \rceil$ instead of $k/2$. We need to introduce a data structure called *Gomory-Hu trees* [41]. (We only need the version for unweighted graphs, and that is what we define.)

Definition 5.28. *A* Gomory-Hu tree *is a weighted tree T defined on the vertices of an undirected unweighted graph $G = (V, E)$ satisfying the following properties:*

- *For any pair of vertices $u, v \in V$, the number of edge-disjoint paths between u and v in G is equal to the minimum weight of an edge in the unique path between u and v in T.*
- *For any edge $e \in T$ with weight $w(e)$, the cut in G corresponding to the vertex partition produced by removing e from T has $w(e)$ edges in it.*

We will now prove Lemma 5.26. Let vertices that have degree at least R in G, but do not have R edge-disjoint paths to *any* other vertex in G, be called *dangerous* vertices. We will show that every dangerous vertex must have at least three neighbors in any Gomory-Hu tree T of G. Since any tree has at most $n/2$ vertices with degree three or more, it follows that there are at most $n/2$ dangerous vertices in G. But, note that every vertex that is not dangerous must be in a $R/2$-edge connected component that has at least two vertices. This will immediately yield the lemma.

We now show that a dangerous vertex v has at least three neighbors in a Gomory-Hu tree T. By the second property of Gomory-Hu trees, the sum of weights on edges incident on v in T is at least the total degree of v in G, which is at least R since v is a dangerous vertex. Thus, if there are less than three neighbors of v in T, there is at least one edge (u, v) incident on v in T with weight at least $R/2$. But property 1 then ensures that T has at least $R/2$ edge-disjoint paths in G between u and v, contradicting that v is dangerous.

This completes the proof of Lemma 5.26.

Now, consider an iterative algorithm for the MEAN problem that runs the α-approximation MDAN subroutine in every round with the partition defined by the $R/2$-edge connected components of the subgraph activated in previous iterations. By Corollary 5.27, this algorithm terminates in $O(\log n)$ iterations, thereby proving Lemma 5.6.

5.7.2 Installation Cost Optimization

We will now give an algorithm for the MEAN problem in the installation cost setting that proves Theorem 5.7. Recall that in this setting, every edge (u, v) has a threshold

τ_{uv} and scaling constants $\alpha_{u,uv}, \alpha_{v,uv}$, and an edge is activated iff $\alpha_{u,uv}x_u + \alpha_{v,uv}x_v \geq \tau_{uv}$.

First, we give an algorithm for the MDAN problem in this setting. We define the *cost* of a star s as $c_s = \max_{(u,v)\in s} \tau_{uv}$. Observe that we can activate all the edges of s by increasing the value of x_v to c_s and keeping the value of x_u for all other vertices u unchanged. We define the *benefit* of s as the overall decrease in the degree requirements of the subsets in partition $\mathcal{P}$ when we activate the edges of s. Our algorithm runs in rounds, where in each round, it greedily select a star with the minimum cost-benefit ratio to add to the set of activated edges. We terminate once the set of activated edges satisfies $\frac{8+3\epsilon}{8+4\epsilon}$ fraction of the total degree requirement. Clearly, this algorithm can be implemented in polynomial time.

Analysis

The next lemma analyzes the approximation ratio of the MDAN algorithm.

Lemma 5.29. *The approximation ratio of the MDAN algorithm given above is* $O(\log(4+8/\epsilon))$.

Proof. Let $\delta = \frac{8+3\epsilon}{8+4\epsilon}$. So, we need to show that the approximation ratio is $O\left(\log \frac{1}{1-\delta}\right)$. The total cost of the partial MDAN solution is at most

$$\left(\frac{1}{nR} + \frac{1}{nR-1} + \ldots + \frac{1}{\delta nR}\right) \textsc{opt} = O\left(\log \frac{1}{1-\delta}\right) \textsc{opt},$$

where OPT is the cost of an optimal MDAN solution. □

The next lemma states that this terminating condition implies that a sufficiently large number of vertices have sufficiently high degree.

Lemma 5.30. *If the total degree requirement satisfied by a partial solution to the MDAN problem is at least* $Rn\left(\frac{8+3\epsilon}{8+4\epsilon}\right)$*, then there are at least* $3n/4$ *vertices with degree at least* $\left(\frac{2}{2+\epsilon}\right) R$.

Proof. Let us order the vertices $v_1, v_2, \ldots, v_n$ by decreasing degree in the activated subgraph. Then, the degree of $v_{3n/4}$ is at least

$$\frac{Rn(8+3\epsilon)/(8+4\epsilon) - 3Rn/4}{n/4} = R\left(\frac{2}{2+\epsilon}\right).$$ □

It follows from Lemmas 5.30 and 5.26 that $O(\log n)$ iterations of the MDAN algorithm yields a $R/(2+\epsilon)$-edge-connected activated subgraph. Theorem 5.7 now follows from Lemma 5.29.

5.8 Minimum Activation Path

In this section, we give an exact algorithm for the MAP problem with terminals s, t. The algorithm mimics Bellman-Ford's single-source shortest path algorithm (see

e.g. [24]) with source vertex s, except that the dynamic program has to be additionally parameterized by the value $x_v \in D$ chosen at the destination v. Let $d(v, a)$ and $\pi(v, a)$ be variables respectively representing the length and the predecessor of v in the shortest path from s to v discovered thus far, where $x_v = a \in D$. Initially, $d(v, a) = \infty$ and $\pi(v, a) = \text{NULL}$ for all vertices $v \neq s$, and for all $a \in D$; also, $d(s, a) = a$ and $\pi(s, a) = \text{NULL}$ for all $a \in D$. The algorithm runs in $n - 1$ rounds, where in each round it *relaxes* each edge (u, v) by making the following updates for each $a, b \in D$ such that $f_{uv}(a, b) = 1$:

- if $d(v, b) > d(u, a) + b$, update $d(v, b)$ to $d(u, a) + b$ and $\pi(v, b)$ to (u, a).
- if $d(u, a) > d(v, b) + a$, update $d(u, a)$ to $d(v, b) + a$ and $\pi(u, a)$ to (v, b).

The final output is $\min_{a \in D} d(t, a)$ and the corresponding path given by the predecessors.

The following lemma is crucial to proving correctness of the above algorithm.

Lemma 5.31. *Suppose u is the immediate neighbor of v on a shortest path between s and v with $x_v = a$. Also, let $x_u = b$ on this path. Then, the prefix of this path between s and u is a shortest path between s and u where $x_u = b$.*

Proof. If not, we can replace the s to u segment of the s to v shortest path with the shorter alternative path. Since x_u and x_v remain unchanged, the edge (u, v) remains activated. □

We now use the above lemma to prove the following lemma; setting $i = n - 1$ in the lemma proves correctness of the algorithm.

Lemma 5.32. *If a shortest path from s to v with $x_v = a$ contains i edges, then $d(v, a)$ and $\pi(v, a)$ are correctly set after i rounds of the algorithm.*

Proof. We prove by induction on i. The base case, for $i = 0$, is immediate. For the inductive case, let u be the neighbor of v on the shortest path from s to v with $x_v = a$; let $x_u = b$ on this path. By Lemma 5.31 and the inductive hypothesis, $d(u, b)$ and $\pi(u, b)$ are correctly set at the end of round $i - 1$. The proof follows since edge (u, v) is relaxed with values $x_u = b, x_v = a$ in round i. □

5.9 Concluding Remarks

The activation network model introduces a new set of practically relevant network design problems. One important objective is to obtain similar results in directed networks. The algorithmic techniques presented here are tailored to undirected graphs, and do not, in general, extend to directed graphs. Even for undirected graphs, an interesting direction is to obtain approximation algorithms for higher connectivity requirements. Several interesting results in these directions have been obtained by Nutov [79] since the publication of this work. However, several questions remain:

e.g., can we obtain bi-criteria algorithms for higher connectivity requirements with poly-logarithmic approximation factors for arbitrary activation functions?

Another interesting direction of research is to restrict the activation functions to obtain better approximation ratios. For example, the minimum Steiner forest problem (or even the much richer generalized Steiner forest problem [53]) admits constant factor approximation algorithms in edge-weighted graphs. These problems are special cases of our general framework; so is this a manifestation of some special structural property in their activation functions? Can we identify and exploit these structural properties to obtain good approximation algorithms for broader classes of activation functions?

5.10 Notes

A preliminary version of this work appeared in the *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms*, 2011 [80].

Bibliography

[1] A. Agrawal, P. N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.

[2] K. J. Ahn and S. Guha. Graph sparsification in the semi-streaming model. In *ICALP (2)*, pages 328–338, 2009.

[3] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *SODA*, pages 459–467, 2012.

[4] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, pages 5–14, 2012.

[5] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms*, 2(4):640–660, 2006.

[6] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009.

[7] E. Althaus, G. Calinescu, I. I. Mandoiu, S. K. Prasad, N. Tchervenski, and A. Zelikovsky. Power efficient range assignment for symmetric connectivity in static ad hoc wireless networks. *Wireless Networks*, 12(3):287–299, 2006.

[8] A. Asadpour, M. X. Goemans, A. Madry, S. O. Gharan, and A. Saberi. An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem. In *SODA*, pages 379–389, 2010.

[9] B. Awerbuch, Y. Azar, and Y. Bartal. On-line generalized steiner problem. *Theor. Comput. Sci.*, 324(2-3):313–324, 2004.

[10] J. D. Batson, D. A. Spielman, and N. Srivastava. Twice-Ramanujan sparsifiers. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 255–262, 2009.

[11] A. A. Benczúr and D. R. Karger. Approximating s-t minimum cuts in $\tilde{o}(n^2)$ time. In *STOC*, pages 47–55, 1996.

[12] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, pages 201–210, 2010.

[13] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[14] N. Buchbinder and J. Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009.

[15] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. An improved LP-based approximation for steiner tree. In *STOC*, pages 583–592, 2010.

[16] G. Calinescu and P.-J. Wan. Range assignment for biconnectivity and k-edge connectivity in wireless ad hoc networks. *MONET*, 11(2):121–128, 2006.

[17] M. Charikar, C. Chekuri, T.-Y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999.

[18] M. Charikar, C. Chekuri, A. Goel, and S. Guha. Rounding via trees: Deterministic approximation algorithms for group steiner trees and k-median. In *STOC*, pages 114–123, 1998.

[19] C. Chekuri, G. Even, A. Gupta, and D. Segev. Set connectivity problems in undirected graphs and the directed steiner network problem. *ACM Transactions on Algorithms*, 7(2):18, 2011.

[20] C. Chekuri, G. Even, and G. Kortsarz. A greedy approximation algorithm for the group steiner problem. *Discrete Applied Mathematics*, 154(1):15–34, 2006.

[21] C. Chekuri and N. Korula. A graph reduction step preserving element-connectivity and applications. In *ICALP (1)*, pages 254–265, 2009.

[22] M. Chlebík and J. Chlebíková. The steiner tree problem on graphs: Inapproximability results. *Theor. Comput. Sci.*, 406(3):207–214, 2008.

[23] J. Chuzhoy and S. Khanna. Algorithms for single-source vertex connectivity. In *FOCS*, pages 105–114, 2008.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[25] E. A. Dinic, A. V. Karzanov, and M. V. Lomonosov. The structure of a system of minimal edge cuts of a graph. In A. A. Fridman, editor, *Studies in Discrete Optimization*, pages 290–306. Izdatel'stvo "Nauka", 1976.

[26] P. G. Doyle and L. J. Snell. *Random Walks and Electric Networks*. Carus Mathematical Monographs, 1984.

[27] P. Dutta, S. Jaiswal, D. Panigrahi, K. V. M. Naidu, R. Rastogi, and A. K. Todimala. Villagenet: A low-cost, 802.11-based mesh network for rural regions. In *COMSWARE*, 2007.

[28] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[29] U. Feige. A threshold of ln for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[30] U. Feige and S. Korman. Personal communication. 2010.

[31] U. Feige, D. Peleg, and G. Kortsarz. The dense -subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[32] L. Fleischer. Building chain and cactus representations of all minimum cuts from hao-orlin in the same asymptotic run time. *J. Algorithms*, 33(1):51–72, 1999.

[33] W. S. Fung, R. Hariharan, N. J. A. Harvey, and D. Panigrahi. A general framework for graph sparsification. In *STOC*, pages 71–80, 2011.

[34] W. S. Fung and N. J. A. Harvey. Graph sparsification by edge-connectivity and random spanning trees. *CoRR*, abs/1005.0265, 2010.

[35] H. N. Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *FOCS*, pages 812–821, 1991.

[36] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000.

[37] A. Goel, M. Kapralov, and S. Khanna. Graph sparsification via refinement sampling. *CoRR*, abs/1004.4915, 2010.

[38] A. Goel, M. Kapralov, and I. Post. Single pass sparsification in the streaming model with edge deletions. *CoRR*, abs/1203.4900, 2012.

[39] M. X. Goemans, N. J. A. Harvey, K. Jain, and M. Singh. A randomized rounding algorithm for the asymmetric traveling salesman problem. *CoRR*, abs/0909.0941, 2009.

[40] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[41] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *J. Soc. Indust. Appl. Math.*, 9(4):551–570, 1961.

[42] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

[43] S. Guha and S. Khuller. Improved methods for approximating node weighted steiner trees and connected dominating sets. *Inf. Comput.*, 150(1):57–74, 1999.

[44] S. Guha, A. Moss, J. Naor, and B. Schieber. Efficient recovery from power outage (extended abstract). In *STOC*, pages 574–582, 1999.

[45] A. Gupta, R. Krishnaswamy, and R. Ravi. Online and stochastic survivable network design. In *STOC*, pages 685–694, 2009.

[46] G. R. Gupta, S. Sanghavi, and N. B. Shroff. Node weighted scheduling. In *SIGMETRICS/Performance*, pages 97–108, 2009.

[47] M. T. Hajiaghayi, N. Immorlica, and V. S. Mirrokni. Power optimization in fault-tolerant topology control algorithms for wireless multi-hop networks. *IEEE/ACM Trans. Netw.*, 15(6):1345–1358, 2007.

[48] M. T. Hajiaghayi, G. Kortsarz, V. S. Mirrokni, and Z. Nutov. Power optimization for connectivity problems. *Math. Program.*, 110(1):195–208, 2007.

[49] E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *STOC*, pages 585–594, 2003.

[50] J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, 17(3):424–446, 1994.

[51] F. Harary. *Graph Theory.* Addison-Wesley, 1969.

[52] M. Imase and B. M. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991.

[53] K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[54] V. Kachitvichyanukul and B. W. Schmeiser. Binomial random variate generation. *Commun. ACM*, 31(2):216–222, 1988.

[55] M. Kapralov and R. Panigrahy. Spectral sparsification via random spanners. In *ITCS*, pages 393–398, 2012.

[56] D. R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *SODA*, pages 21–30, 1993.

[57] D. R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, May 1999.

[58] D. R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000.

[59] D. R. Karger and D. Panigrahi. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In *SODA*, pages 246–255, 2009.

[60] D. R. Karger and C. Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996.

[61] M. Karpinski and A. Zelikovsky. New approximation algorithms for the steiner tree problems. *J. Comb. Optim.*, 1(1):47–65, 1997.

[62] A. V. Karzanov and E. A. Timofeev. Efficient algorithm for finding all minimal edge cuts of a non-oriented graph. *Cybernetics*, 22:156–162, 1986.

[63] J. A. Kelner and A. Levin. Spectral sparsification in the semi-streaming setting. In *STACS*, pages 440–451, 2011.

[64] P. N. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *J. Algorithms*, 19(1):104–115, 1995.

[65] A. Kolla, Y. Makarychev, A. Saberi, and S.-H. Teng. Subgraph sparsification and nearly optimal ultrasparsifiers. In *STOC*, pages 57–66, 2010.

[66] G. Kortsarz, V. S. Mirrokni, Z. Nutov, and E. Tsanko. Approximating minimum-power degree and connectivity problems. In *LATIN*, pages 423–435, 2008.

[67] G. Kortsarz and Z. Nutov. Approximating minimum-power edge-covers and 2, 3-connectivity. *Discrete Applied Mathematics*, 157(8):1840–1847, 2009.

[68] I. Koutis, A. Levin, and R. Peng. Improved spectral sparsification and numerical algorithms for SDD matrices. In *STACS*, pages 266–277, 2012.

[69] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving SDD linear systems. In *FOCS*, pages 235–244, 2010.

[70] I. Koutis, G. L. Miller, and R. Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *FOCS*, pages 590–598, 2011.

[71] Y. Lando and Z. Nutov. On minimum power connectivity problems. *J. Discrete Algorithms*, 8(2):164–173, 2010.

[72] W. Mader. A reduction method for edge-connectivity in graphs. *Ann. Discrete Math.*, 3:145–164, 1978.

[73] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

[74] H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discrete Math.*, 5(1):54–66, 1992.

[75] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

[76] D. Naor and V. V. Vazirani. Representing and enumerating edge connectivity cuts in rnc. In *WADS*, pages 273–285, 1991.

[77] Z. Nutov. Approximating minimum-power k-connectivity. *Ad Hoc & Sensor Wireless Networks*, 9(1-2):129–137, 2010.

[78] Z. Nutov. Approximating steiner networks with node-weights. *SIAM J. Comput.*, 39(7):3001–3022, 2010.

[79] Z. Nutov. Survivable network activation problems. In *LATIN*, pages 594–605, 2012.

[80] D. Panigrahi. Survivable network design problems in wireless networks. In *SODA*, pages 1014–1027, 2011.

[81] D. Panigrahi, P. Dutta, S. Jaiswal, K. V. M. Naidu, and R. Rastogi. Minimum cost topology construction for rural wireless mesh networks. In *INFOCOM*, pages 771–779, 2008.

[82] H. J. Prömel and A. Steger. A new approximation algorithm for the steiner tree problem with performance ratio 5/3. *J. Algorithms*, 36(1):89–101, 2000.

[83] S. Rao and S. Zhou. Edge disjoint paths in moderately connected graphs. *SIAM J. Comput.*, 39(5):1856–1887, 2010.

[84] G. Robins and A. Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005.

[85] S. Sen and B. Raman. Long distance wireless mesh network planning: problem formulation and solution. In *WWW*, pages 893–902, 2007.

[86] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, June 1983.

[87] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011.

[88] D. A. Spielman and S.-H. Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.

[89] V. Vazirani. *Approximation algorithms*. Springer-Verlag, Berlin, 2001.

[90] A. Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470, 1993.

[91] L. Zosin and S. Khuller. On directed steiner trees. In *SODA*, pages 59–63, 2002.