

© 2013 by Alina Raluca Ene. All rights reserved.

APPROXIMATION ALGORITHMS FOR
SUBMODULAR OPTIMIZATION AND GRAPH PROBLEMS

BY

ALINA RALUCA ENE

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Doctoral Committee:

Associate Professor Chandra Chekuri, Chair
Assistant Professor Brighten Godfrey
Associate Professor Sarel Har-Peled
Doctor Jan Vondrák, IBM Almaden Research

Abstract

In this thesis, we consider combinatorial optimization problems involving submodular functions and graphs. The problems we study are **NP**-hard and therefore, assuming that $\mathbf{P} \neq \mathbf{NP}$, there do not exist polynomial-time algorithms that always output an optimal solution. In order to cope with the intractability of these problems, we focus on algorithms that construct *approximate* solutions: An approximation algorithm is a polynomial-time algorithm that, for any instance of the problem, it outputs a solution whose value is within a multiplicative factor ρ of the value of the optimal solution for the instance. The quantity ρ is the approximation ratio of the algorithm and we aim to achieve the smallest ratio possible.

Our focus in this thesis is on designing approximation algorithms for several combinatorial optimization problems. In the first part of this thesis, we study a class of constrained submodular minimization problems. We introduce a model that captures *allocation* problems with submodular costs and we give a generic approach for designing approximation algorithms for problems in this model. Our model captures several problems of interest, such as non-metric facility location, multiway cut problems in graphs and hypergraphs, uniform metric labeling and its generalization to hub location. Using a convex relaxation and rounding strategy, we achieve good approximation guarantees for several problems in this model. In particular, we match or improve the known approximation ratios for several problems in a unified fashion.

In the second part of this thesis, we study multicommodity flow problems in both undirected and directed graphs and we make several contributions towards understanding the gap between fractional and integral multicommodity flows. We give a poly-logarithmic approximation with constant congestion for the node-disjoint paths problem and we show a poly-logarithmic upper bound on the gap between the maximum fractional and integral throughput flows in node-capacitated undirected graphs. Prior to our work, the best guarantees were only polynomial. In the process,

we prove a conjecture of Chekuri, Khanna, and Shepherd on the connection between the treewidth of the graph and the existence of a good routing structure. Additionally, we initiate the study of integral throughput flow problems in directed graphs with symmetric demand pairs. We obtain a poly-logarithmic approximation with constant congestion for the all-or-nothing flow problem.

In the third part of this thesis, we study several network design problems. The input to these problems is a graph with costs on the edges or the nodes and the output is a minimum cost subgraph that meets certain connectivity requirements. We study several network design problems in *planar* graphs, including the prize-collecting Steiner tree and forest and the survivable network design problem with node costs. We show that the special structure of planar graphs — and more generally, bounded genus and minor-free graphs — leads to algorithms whose approximation guarantees are a significant improvement over what can be achieved for general graphs.

To my mother, Ioana.

Acknowledgements

First and foremost, I would like to thank my advisor Chandra Chekuri for his inspiring advice and encouragement. I could not have asked for a better role model, collaborator, and teacher. Chandra's faith in me has kept me going through countless moments when my confidence was wavering and his encouragements have helped me through several roadblocks that seemed too great to overcome. His door was always open for me and his tireless feedback have contributed tremendously to my research, my writing, and my talks. I feel very fortunate to have had a collaborator like Chandra and his approach to research will always be an inspiration to me. After every meeting, there was always a promising direction to explore and I always walked out of his office feeling that I learned something new. I have greatly benefited from Chandra's insightful advice and support outside of work as well, and I am thankful for it.

I am very grateful to Sarel Har-Peled for being very much like a second advisor to me. I thank Sarel for introducing me to computational geometry and for our enjoyable collaborations that followed. Sarel always jumped at the opportunity to help and I am very grateful for his advice and support over the years. I have also enjoyed our many good times outside of work and I remember fondly our impromptu conversations in the hallway, over coffee, and over lunch.

I have learned quite a bit from my conversations with Jeff Erickson and from his insights and perspective on many topics. Jeff's advice and support during my job search was invaluable, and I am very thankful for it. I was fortunate to be Jeff's teaching assistant during my first years at UIUC. His approach to teaching and exposition have greatly influenced my own.

I was very fortunate to interact with and to learn from many co-authors, mentors, and colleagues. Each of them has shared with me key insights and I thank them for it. I would especially like to thank Jan Vondrák for inviting me to visit IBM and for his contributions to this thesis. Collaborating with Jan was an invaluable learning experience. I am very grateful to Julia Chuzhoy

for introducing me to routing and for our collaborations during my internships at TTI Chicago. Julia's approach to research and her tenacity and perseverance will always be an inspiration to me. Many thanks to Nitish Korula for being a wonderful mentor, collaborator, and friend; my first two years at UIUC would have been very different without his help. It was a joy to exchange ideas with Nitish and our collaborations were an important milestone in my graduate career. I could always count on Nitish for sound advice and support.

My internships and research visits have contributed significantly to my research and scholarship. I thank Martin Pál, Sudipto Guha, Diogo Andrade, and the Market Algorithms research group at Google NY for a great internship. I am grateful to Jan Vondrák and the theory group at IBM Almaden for two wonderful research visits and for countless informative discussions over lunch. My sincerest thanks to Julia Chuzhoy for two very helpful internships at TTI Chicago. I thank Viswanath Nagarajan, Rishi Saket, and the theory group at IBM TJ Watson for introducing me to a new area of research and for an enjoyable internship.

Prior to joining UIUC, I spent four wonderful years in Princeton that set the stage to what I hope to be a long and rewarding life as a theoretical computer scientist. I am very thankful to Princeton for a unique academic environment. I am especially grateful to Bob Tarjan for introducing me to research and for guiding my first research projects; my graduate path may not have existed without his help and support.

In the last five years, I have grown not only as a researcher but also as a person, and for the latter I am greatly indebted to my friends. I am very grateful for many rewarding and enduring friendships. I thank Dan Schreiber for being a wonderful and supportive friend. Dan's enthusiasm and optimism were contagious and uplifting. I have greatly enjoyed thinking about mathematics, Ingmar Berman's films, Glenn Gould's music, and many other topics on our long strolls through historic Urbana. I was quite fortunate that Ben Raichel joined the theory group at the perfect time; the long days at the office would not have been the same without Ben. I have enjoyed exchanging ideas with Ben and our many conversations have been fascinating and informative. Many thanks to the other members of the theory group; I am very grateful to Kyle Fox, Sungjin Im, Nirman Kumar, Hemanta Maji, Ben Moseley, and Amir Nayyeri for their friendship and support over the years. I thank Andreea Stuparu for being a great friend; my visits to Chicago would not have been

the same without Andreea.

Dankeschön to Andy Wiese for many wonderful conversations and for his support from afar. Many thanks to Andy for cheering me up during the long nights when I was writing this thesis and for always being there for me despite the distance and the time difference. Andy's optimism, compassion, and encouragements have helped me through some of the most trying moments. Our hiking trips on the west coast will always be with me.

I thank my family for their unconditional love and support over the years. I especially thank my mother for her faith in me, her guidance, and her dedication to my success. I hope some of her optimism, tenacity, and inspiring work ethic have rubbed off on me. My academic journey would not have been possible without my mother and I dedicate this thesis to her.

Lastly, I am very grateful for financial support from a Chirag Foundation graduate fellowship, Chandra Chekuri's NSF grants CCF-0728782 and CCF-1016684, and Julia Chuzhoy's NSF grant CCF-0844872.

Table of Contents

Chapter 1	Introduction	1
1.1	Preliminaries	3
1.1.1	Submodular functions	3
1.1.2	Approximation algorithms and hardness of approximation	5
1.2	Thesis contributions and organization	8
1.2.1	Submodular optimization	8
1.2.2	Routing	11
1.2.3	Network design	12
Chapter 2	The Submodular Cost Allocation Problem	15
2.1	Introduction	15
2.1.1	Problems related to MSCA	16
2.1.2	Overview of results and techniques	18
2.2	Approximation algorithms and integrality gap for MonMSCA	19
2.3	Approximation algorithms and integrality gap for SubLabel	24
2.3.1	Labeling in hypergraphs	24
2.3.2	Labeling with a symmetric separation cost	29
2.3.3	Approximation algorithm for SubLabel	31
2.4	Hardness of SubLabel	33
2.5	Hardness of MSCA	35
2.5.1	Representation of lattices by directed graphs	36
2.5.2	NP-completeness of Partition Matching and Lattice Matching	38
2.6	The Lovász extension and solving LE-Rel	40
2.7	Concluding remarks	42
Chapter 3	The Submodular Multiway Partition Problem	43
3.1	Introduction	43
3.1.1	Overview of rounding and the main technical result	45
3.1.2	Discussion and other related work	47
3.2	Approximation algorithm and integrality gap for SymSubMP	48
3.2.1	A 1.5 approximation algorithm	48
3.2.2	A $(1.5 - 1/k)$ approximation algorithm	50
3.3	Approximation algorithm and integrality gap for SubMP	51
3.3.1	A weaker analysis	52
3.3.2	An improved analysis	52
3.4	Proof of the main theorem	57
3.5	A variant of the main theorem	59

3.6	Concluding remarks	61
Chapter 4	The Maximum Node Disjoint Paths Problem	63
4.1	Introduction	63
4.1.1	High-level overview of algorithm and technical contribution	66
4.2	Preliminaries and setup	70
4.3	Expander embedding and routing algorithm	77
4.3.1	Family of good clusters	77
4.3.2	Connecting the good sets and expander embedding	78
4.3.3	Routing using the embedded expander	81
4.4	Proof of Theorem 4.3.2 on good clustering	83
4.5	Proof of Theorem 4.3.4 on connecting good clusters	85
4.5.1	Proof of Theorem 4.5.2	90
4.6	Concluding remarks	92
Chapter 5	The All-or-Nothing Flow Problem	94
5.1	Introduction	94
5.1.1	Motivation and connection to related problems	95
5.1.2	High-level overview of the algorithm and technical contributions	99
5.1.3	Discussion of related work	101
5.2	Approximation algorithm for SymANF	102
5.2.1	Preliminaries and setup	102
5.2.2	The approximation algorithm for SymANF	111
5.3	Well-linked decomposition	111
5.4	From fractional well-linked sets to well-linked sets	114
5.5	Concluding remarks	127
Chapter 6	Prize-collecting Steiner Tree and Forest in Planar Graphs	128
6.1	Introduction	128
6.1.1	Overview of techniques	130
6.2	A primal-dual algorithm for Prize-collecting Steiner Forest	135
6.3	The reduction to fixed treewidth: building spanners	140
6.3.1	Scaling penalties to capture important terminals	141
6.3.2	Completing the reduction	145
6.4	Proof of Theorem 6.3.3	146
6.5	Proof of Theorem 6.3.4	147
6.6	The Prize-collecting Traveling Salesperson problem	154
6.7	Prize-collecting Steiner Tree in graphs of fixed treewidth	155
6.7.1	A dynamic program for Prize-collecting Steiner Tree	156
6.8	Concluding remarks	163
Chapter 7	Node-weighted Network Design in Planar and Minor-free Graphs . .	165
7.1	Introduction	165
7.2	Algorithm for node-weighted EC-SNDP and proper functions	169
7.2.1	A primal-dual algorithm for the augmentation problem	171
7.3	Proof of Theorem 7.3.1	176
7.4	Concluding remarks	181
References	182

Chapter 1

Introduction

This thesis considers several discrete optimization problems. At a high level, these problems can be divided into two broad classes. The first class consists of problems involving submodular objective functions. We have a finite collection V of elements and a function f that assigns a value $f(S)$ to each subset S of the elements, and the goal is to select a subset $S \subseteq V$ that maximizes or minimizes $f(S)$ subject to certain constraints on S . In this thesis, we consider valuation functions f that are *submodular*. Submodularity is a central concept in combinatorial optimization. Starting with the milestone paper of Edmonds [69] from the '70s, the special structure of submodular functions has been shown to underpin the tractability¹ of many optimization problems. Submodular functions have received considerable attention in other fields such as Economics; submodular functions capture the intuitive notion of diminishing returns and this property makes them perfectly suited for modeling the valuations of players. More recently, there has been a renewed interest in submodularity that is driven by applications in areas such as Machine Learning and Computer Vision, and algorithmic Game Theory and auctions. In this thesis, we consider a class of submodular optimization problems that can be viewed as *allocation* or *labeling* problems. We introduce a model that captures several problems of interest and we give a general approach for designing algorithms for these problems based on a mathematical programming relaxation.

We also consider optimization problems on graphs² and networks. Graphs are a mathematical abstraction of real-world networks and systems, such as road and communication networks. Many real-world problems can be modeled as optimization problems involving graphs. Several of these applications involve transportation networks in which the goal is to route certain commodities through the network subject to various capacity constraints. A fundamental optimization problem

¹Following Edmonds, we identify tractability with polynomial-time solvability.

²In this thesis, we use standard graph theory terms and definitions. We refer the reader to [167] for an introduction to graph theory and the relevant terminology.

in this space is the single-commodity (or s - t) flow problem in which we have a graph with capacities on the edges and the goal is to route as much flow as possible from a source node s to a destination node t subject to the constraint that the amount of flow on each edge is at most its capacity. The s - t flow problem is intimately connected to other central graph optimization problems such as the minimum s - t cut and the maximum disjoint s - t paths problems; the minimum s - t cut problem asks for a set of edges with minimum total capacity whose removal disconnects s from t , and the maximum disjoint s - t paths problem asks for the maximum number of paths from s to t that do not share any edges or internal nodes. Flows, cuts, and disjoint paths are ubiquitous in applications and they serve as building blocks in the design of efficient algorithms for a variety of network optimization problems. In this thesis, we consider routing problems in the multicommodity setting in which there are several source-sink pairs and the goal is to route flow between these pairs according to various objectives and constraints.

Several other applications in real-world networks involve the following type of problem: we are given a graph with costs associated with the edges or the nodes, and the goal is to choose a minimum cost subgraph that meets certain connectivity requirements. A central problem in this area is the **Steiner Tree** problem in which we are also given a set of nodes called terminals and the goal is to choose a minimum cost set of edges that connects the terminals. In certain applications, it is critical to design networks that are robust to edge or node failures, and the problems that arise in this setting are collectively referred to as the **Survivable Network Design** problem. In this thesis, we consider several network design problems, including the **Survivable Network Design** problem and several generalizations of the **Steiner Tree** problem. We focus on the setting in which the input graph is planar or more generally, it belongs to a proper minor-closed family of graphs. In addition to being of theoretical interest, planar and minor-free graphs model the restricted topology of several real-world networks.

The problems that we study in this thesis are **NP**-hard. It is widely believed that $\mathbf{P} \neq \mathbf{NP}$ and, under this assumption, these problems do not admit *efficient* (polynomial time) exact algorithms. A very successful approach for coping with the intractability of these problems has been to settle for *near-optimal* solutions, i.e., a solution whose value is within a small *multiplicative* factor³ of the

³There are other notions of near-optimality that have been studied, such as *additive* approximations. Multiplicative notions are very robust and they are prevalent in the approximation algorithms literature.

value of the optimal solution. In this thesis, we follow this approach and we study these problems from the point of view of polynomial-time *approximation* algorithms. In addition to their many practical applications, approximation algorithms are of mathematical interest. A rich theory has been developed for designing approximation algorithms for **NP**-hard problems [101, 164, 169].

In this thesis, we describe approximation algorithms for several **NP**-hard optimization problems involving submodular functions and graphs. We make progress towards understanding the approximation threshold of these problems primarily by providing algorithms with improved approximation guarantees.

In the following sections, we give some background on submodular functions and approximation algorithms, and we summarize the main contributions of this thesis.

1.1 Preliminaries

1.1.1 Submodular functions

In this section, we review the main definitions associated with set functions that we use in Chapter 2 and Chapter 3.

Definition 1 (Non-negative function). *Let V be a finite ground set. A set function $f : 2^V \rightarrow \mathbb{R}$ is **non-negative** if $f(A) \geq 0$ for each subset $A \subseteq V$.*

Definition 2 (Symmetric function). *Let V be a finite ground set. A set function $f : 2^V \rightarrow \mathbb{R}$ is **symmetric** if $f(A) = f(V - A)$ for each subset $A \subseteq V$.*

Definition 3 (Monotone function). *Let V be a finite ground set. A set function $f : 2^V \rightarrow \mathbb{R}$ is **monotone** if $f(A) \leq f(B)$ for any two subsets A and B of V such that $A \subseteq B$.*

Definition 4 (Submodular function). *Let V be a finite ground set. A set function $f : 2^V \rightarrow \mathbb{R}$ is **submodular** if, for any two subsets A and B of V , we have*

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B).$$

An equivalent definition is the following. The function is submodular if, for any two subsets A and B such that $A \subseteq B$ and any element $v \in V - B$, we have

$$f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B).$$

The second definition of submodularity captures the intuitive notion of “diminishing returns.”

We note that a linear function satisfies the first submodularity inequality with equality, i.e., for any two subsets A and B , we have $f(A) + f(B) = f(A \cap B) + f(A \cup B)$. Linear functions are also referred to as *modular* functions.

Definition 5 (Posi-modular function). *Let V be a finite ground set. A set function $f : 2^V \rightarrow \mathbb{R}$ is **posi-modular** if, for any two subsets A and B of V , we have*

$$f(A) + f(B) \geq f(A - B) + f(B - A).$$

It is well-known that a symmetric submodular function is posi-modular, but not all posi-modular functions are symmetric submodular functions.

Definition 1.1.1. *Let V be a finite ground set. A set function $f : 2^V \rightarrow \mathbb{R}$ is **sub-additive** if, for any two subsets A and B of V , we have*

$$f(A) + f(B) \geq f(A \cup B).$$

A non-negative submodular function is sub-additive.

Examples of submodular functions. In Chapter 2 and Chapter 3 we will encounter the following submodular functions associated with graphs and hypergraphs.

A classical example of a submodular function is the *graph cut* function, which is defined as follows. Let $G = (V, E)$ be a graph with weights $w : E \rightarrow \mathbb{R}_+$ on the edges. For each set S of vertices, let $\delta(S)$ be the set of all edges of G with one endpoint in S and the other endpoint in $V - S$. The graph cut function is the function $f : 2^V \rightarrow \mathbb{R}_+$ such that $f(S) = \sum_{e \in \delta(S)} w(e)$ for each subset $S \subseteq V$. The graph cut function is submodular and symmetric.

In directed graphs, the cut function is the function $f : 2^V \rightarrow \mathbb{R}_+$ such that $f(S) = \sum_{e \in \delta^{\text{out}}(S)} w(e)$, where $\delta^{\text{out}}(S)$ is the set of all edges whose tail is in S and its head is in $V - S$. The directed cut function is submodular but it is not symmetric.

The *hypergraph cut* function is defined similarly. Let $\mathcal{G} = (V, \mathcal{E})$ be a hypergraph with weights $w : \mathcal{E} \rightarrow \mathbb{R}_+$ on the hyperedges. For each set S of vertices, let $\delta(S)$ be the set of all hyperedges e of \mathcal{G} with a vertex in S and a vertex in $V - S$; differently said, $S \cap e$ and $(V - S) \cap e$ are both non-empty.

The hypergraph cut function is the function $g : 2^V \rightarrow \mathbb{R}_+$ such that $g(S) = \sum_{e \in \delta(S)} w(e)$. The hypergraph cut function is submodular and symmetric.

Another example of a submodular function associated with a hypergraph is the following function, which we call the *hypergraph separation* function. Let $\mathcal{G} = (V, \mathcal{E})$ be a hypergraph with weights $w : \mathcal{E} \rightarrow \mathbb{R}_+$ on the hyperedges. Each hyperedge $e \in \mathcal{E}$ has a unique representative vertex $r(e) \in e$. The hypergraph separation function is the function $h : 2^V \rightarrow \mathbb{R}_+$ such that $h(S) = \sum_{e: e \in \delta(S), r(e) \in S} w(e)$. In words, $h(S)$ is the total weight of the hyperedges of $\delta(S)$ whose representatives are in S . The function h is submodular but it is not symmetric.

Our final example of a submodular function is the *coverage function* associated with a set system. Let N be a finite set and let $\{X_1, X_2, \dots, X_m\}$ be a collection of subsets of N . Let $[m]$ denote the set $\{1, 2, \dots, m\}$. The coverage function is the function $f : 2^{[m]} \rightarrow \mathbb{Z}_+$ such that $f(S) = |\bigcup_{i \in S} X_i|$ is the number of distinct elements of N that appear in the subsets indexed by S .

Representation of submodular functions. In Chapter 2 and Chapter 3, we study optimization problems whose input involves one or more submodular functions. This gives rise to the issue of how these functions are represented. One option is to represent f explicitly by listing its values on all possible subsets of V . This representation is exponential in the size of V and it is not suitable for our purposes. In order to overcome this issue, we represent each function implicitly using a *value oracle*. More precisely, we assume that we are given access to an oracle that takes as input any subset $A \subseteq V$ and it returns the value $f(A)$.

In some cases, there is a *compact* representation of the function whose size is polynomial in $|V|$ and there is an algorithm that, given the representation and a set $A \subseteq V$, it computes the value $f(A)$ in polynomial time. The graph cut function is such an example, where the graph itself provides us with the compact representation of the function. However, not all submodular functions have compact representations and thus an implicit representation is the only viable option in general.

1.1.2 Approximation algorithms and hardness of approximation

The problems we consider in this thesis are **NP** optimization problems (abbreviated **NPO**). We formally define the class **NPO** below.

Definition 6. An **NPO** problem is a tuple $(\mathcal{I}, F, \text{value}, \text{goal})$, where

- \mathcal{I} is the set of instances of the problem and it is recognizable in polynomial time.
- For each input $x \in \mathcal{I}$, $F(x)$ is the collection of feasible solutions. For each feasible solution $s \in F(x)$, the size of s is upper bounded by a polynomial in the size of x . Additionally, there is a polynomial time algorithm that takes as input x and s and it decides whether $s \in F(x)$.
- For each input $x \in \mathcal{I}$ and each feasible solution $s \in F(x)$, $\text{value}(x, s)$ is the value of the solution s , and it is a positive real number. There is a polynomial time algorithm that takes as input x and s and it outputs $\text{value}(x, s)$.
- The **goal** specifies whether the problem is a minimization or maximization problem, i.e., $\text{goal} \in \{\min, \max\}$.

For example, in the classical Minimum Spanning Tree problem, an instance of the problem consists of a graph $G = (V, E)$ with costs $c : E \rightarrow \mathbb{R}_+$ on the edges. The set of feasible solutions for the instance is the collection of spanning trees of G , i.e., subgraphs of G that are trees and they contain all the vertices of G . The value of a tree is the sum of the costs of the edges in the tree.

The decision versions of **NPO** problems are problems in **NP**. More precisely, for a minimization problem, the language $\{x \mid \exists s \in F(x) \text{ s.t. } \text{value}(x, s) \leq B\}$ is in **NP** for any real number B . Similarly, for a maximization problem, the language $\{s \mid \exists s \in F(x) \text{ s.t. } \text{value}(x, s) \geq B\}$ is in **NP** for any real number B . An **NPO** problem is **NP**-hard if its decision version is **NP**-complete.

Given an instance x of an **NPO** problem Π , let $\text{OPT}(x)$ be the value of the optimal solution for x . An algorithm \mathcal{A} achieves an α -approximation for the problem if, for any instance x , it constructs a solution whose value is at most $\alpha \cdot \text{OPT}(x)$ if Π is a minimization problem and it is at least $\text{OPT}(x)/\alpha$ if Π is a maximization problem. The asymmetry in the definition is to ensure that $\alpha \geq 1$ for all approximation algorithms. The *approximation ratio* of \mathcal{A} is the smallest value α such that \mathcal{A} achieves an α -approximation, and it could depend on the size of the input. A *polynomial time approximation scheme* (**PTAS**) is an algorithm such that, for any fixed $\epsilon > 0$, it constructs a $(1 + \epsilon)$ -approximate solution in time that is polynomial in the input size (the running time depends on ϵ). The complexity class **PTAS** consists of all **NPO** problems that admit a polynomial-time approximation scheme. A superclass of **PTAS** is the class **APX** (standing for approximable) that is comprised of all **NPO** problems that admit constant factor approximations. **APX** contains

PTAS but the inclusion is proper unless $\mathbf{P} \neq \mathbf{NP}$; classical problems such as Bin Packing and Metric Traveling Salesperson are in **APX** but not **PTAS** unless $\mathbf{P} \neq \mathbf{NP}$.

On the negative side, an **NPO** problem is β -hard to approximate if, under a suitable complexity theoretic assumption (typically $\mathbf{P} \neq \mathbf{NP}$), there does not exist a polynomial-time algorithm for the problem that achieves a β -approximation. We briefly mention two of the main techniques used to show that a problem is hard to approximate.

(1) Reductions from **NP**-complete problems: Consider an **NPO** problem Π and suppose for simplicity that Π is a minimization problem (the case in which Π is a maximization problem is analogous). Suppose that there is an **NP**-complete decision problem \mathcal{D} , a polynomial-time computable function f , and two non-negative constants α and $\beta > \alpha$ with the following properties:

- The function f maps each instance x of \mathcal{D} to an instance $f(x)$ of Π .
- If x is a Yes instance of \mathcal{D} , we have $\text{OPT}(f(x)) \leq \alpha$.
- If x is a No instance of \mathcal{D} , we have $\text{OPT}(f(x)) \geq \beta$.

Then it follows that, unless $\mathbf{P} = \mathbf{NP}$, Π is γ -hard to approximate for any $\gamma < \beta/\alpha$. This observation extends to the case in which α and β depend on the size of the input and it gives super-constant hardness of approximation results.

(2) Approximation-preserving reductions from **NPO** problems that are known to be hard to approximate: Consider two **NPO** problems $\Pi = (\mathcal{I}, F, \text{value}, \text{goal})$ and $\Pi' = (\mathcal{I}', F', \text{value}', \text{goal})$ with the same goal, and suppose for simplicity that they are minimization problems (the case in which they are maximization problems is analogous). Suppose that there are polynomial-time computable functions f and g and two non-negative constants α and β with the following properties:

- The function f maps each instance x of Π to an instance $f(x)$ of Π' with the property that $\text{OPT}(f(x)) \leq \alpha \text{OPT}(x)$.
- Consider an instance x of Π and let $x' = f(x)$. The function g maps each solution $s' \in F'(x')$ to a solution $g(s') \in F(x)$ with the property that $\text{value}(x, g(s')) - \text{OPT}(x) \leq$

$$\beta \left(\text{value}'(x', s') - \text{OPT}(x') \right).$$

Then it follows that, if Π is γ -hard to approximate, Π' is $\left(1 + \frac{\gamma-1}{\alpha\beta}\right)$ -hard to approximate (under the same complexity theoretic assumption). A pair of functions (f, g) with the properties above is called an *L-reduction* [141].

The hardness of approximation results described in this thesis belong to the categories above. We refer the reader to [8, 99, 101, 164, 169] for additional background on approximation algorithms and hardness of approximation.

1.2 Thesis contributions and organization

1.2.1 Submodular optimization

In Chapter 2 and Chapter 3, we consider allocation problems with submodular costs. These problems fall under the umbrella of the following abstract problem.

Problem 1 (Minimum Submodular-Cost Allocation (MSCA)). Let V be a finite ground set and let f_1, \dots, f_k be k non-negative submodular set functions on V . In the MSCA problem the goal is to partition the ground set V into k (possibly empty) sets A_1, \dots, A_k such that the sum $\sum_{i=1}^k f_i(A_i)$ is minimized.

We remark that the problem is interesting only if the f_i 's are different, since otherwise allocating all of V to f_1 is trivially an optimal solution. The special case of this problem in which all of the functions are monotone has been previously considered by Svitkina and Tardos [162]. In Chapter 2, we consider the problem with both monotone and non-monotone functions. We show that several well-studied problems such as non-metric facility location, multiway cut problems in graphs and hypergraphs, uniform metric labeling and its generalization to hub location among others can be cast as special cases of MSCA; we refer the reader to Chapter 2 for the definitions of these problems.

We show that, for any $k \geq 3$, the MSCA problem is inapproximable even if each function f_i is the sum of a linear function and the cut function of a directed graph. More precisely, we show that it is **NP**-hard to decide whether the optimal value of an instance of MSCA is zero or non-zero. On the positive side, we design approximation algorithms for several special cases of the problem.

All of our algorithms are based on a simple and natural convex programming relaxation for MSCA and a generic rounding approach. Using this framework, we achieve improved approximations for several problems of interest that we discuss below.

Problem 2 (Monotone MSCA (MonMSCA)). The MonMSCA problem is the special case of MSCA in which each function f_i is monotone.

Svitkina and Tardos [162] considered MonMSCA and they gave an $O(\log |V|)$ -approximation for the problem. They also showed that this approximation is best possible via an approximation preserving reduction from the Set Cover problem. We show that, for instances of MonMSCA, our convex relaxation has an $O(\log |V|)$ integrality gap.

Problem 3 (Submodular Labeling (SubLabel)). The SubLabel problem is the special case of MSCA in which each function f_i satisfies $f_i(S) = g_i(S) + h(S)$, where g_i is a monotone submodular function and h is an arbitrary submodular function. We refer to each function g_i as the assignment cost function for label i . We refer to the function h as the separation cost function; we emphasize that the function h is the same for all of the labels.

The SubLabel problem generalizes several labeling problems in graphs and hypergraphs that have applications in Computer Vision. One such problem is the Uniform Metric Labeling problem, which is the special case of SubLabel in which each assignment cost function g_i is linear and the separation cost function h is the cut function of a graph. Kleinberg and Tardos gave a 2-approximation for the Uniform Metric Labeling problem via an interesting LP relaxation and rounding. We give an $O(\log |V|)$ approximation for SubLabel when h is symmetric and we show that this is best possible unless $\mathbf{P} = \mathbf{NP}$ via an approximation preserving reduction from Set Cover. We give an $O(k \log |V|)$ approximation for SubLabel when h is an arbitrary submodular function. We also consider the case in which h is the hypergraph separation function that we defined in Subsection 1.1.1, and we give an $O(\log |V| + \Delta)$ approximation for the problem, where Δ is the maximum hyperedge size; the approximation guarantee improves to $O(\Delta)$ if each assignment cost function g_i is linear.

Problem 4 (Submodular Multiway Partition (SubMP)). Let $f : 2^V \rightarrow \mathbb{R}_+$ be a non-negative submodular set function over a finite ground set V and let $\mathcal{S} = \{s_1, s_2, \dots, s_k\} \subseteq V$ be a set of k terminals. The SubMP problem is to partition A_1, \dots, A_k of V to minimize $\sum_{i=1}^k f(A_i)$ such that

for $1 \leq i \leq k$, $s_i \in A_i$. An important special case is when f is symmetric and we refer to it as **SymSubMP**. The problem can be cast as a special case of **MSCA** as follows. The ground set of the **MSCA** instance is the set $V' = V - \mathcal{S}$ of non-terminals. Additionally, for each terminal s_i , we define a function $f_i : 2^{V'} \rightarrow \mathbb{R}_+$ as follows: $f_i(S) = f(S \cup \{s_i\})$ for each set $S \subseteq V'$.

The **SubMP** problem was introduced by Zhao, Nagamochi, and Ibaraki [173] and it captures classical multiway cut problems in graphs and hypergraphs. Zhao, Nagamochi, and Ibaraki gave a $(k - 1)$ -approximation for **SubMP** and a $2(1 - 1/k)$ -approximation for **SymSubMP**, which are obtained using a simple greedy splitting approach. A well-studied special case of **SymSubMP** is the **Graph Multiway Cut** problem for which the function f is the cut function of a graph. In a breakthrough result, Calinescu, Karloff, and Rabani [60] obtained a $(1.5 - 1/k)$ -approximation for the **Graph Multiway Cut** problem via an interesting geometric relaxation and rounding. The relaxation was key to improving the $2(1 - 1/k)$ -approximation that is achievable via greedy strategies. A natural question is whether **SymSubMP** also admits an approximation that is better than $2(1 - 1/k)$. We answer this question affirmatively in Chapter 3 where we give a matching $(1.5 - 1/k)$ -approximation for **SymSubMP**. Our result is based on the convex programming relaxation that we introduced for the **MSCA** problem and a rounding strategy. The convex program is the first mathematical programming relaxation for **SymSubMP** (and thus **SubMP**) and surprisingly, for instances of the **Graph Multiway Cut** problem, it is equivalent to the geometric relaxation of [60]. Our rounding strategy provides an alternate rounding for the **Graph Multiway Cut** problem. The *node-weighted* **Graph Multiway Cut** and the **Hypergraph Multiway Cut** problems have also received considerable attention. The two problems are equivalent from an approximation point of view (there are approximation preserving reductions between the two), and they are special cases of **SubMP** (but not **SymSubMP**). Garg, Vazirani, and Yannakakis [85] gave a $2(1 - 1/k)$ -approximation for the node-weighted **Graph Multiway Cut** via a distance-based LP relaxation for the problem. The rounding strategy of [85] is based on the fact that an LP solution is half integral, which is quite non-trivial to show. Our convex program provides a new mathematical programming relaxation for the *node-weighted* **Graph Multiway Cut** problem. We also give a rounding strategy that achieves a $2(1 - 1/k)$ -approximation for the **SubMP** problem. Our algorithm improves the previous $(k - 1)$ -approximation of [173].

1.2.2 Routing

In Chapter 4 and Chapter 5, we turn our attention to routing problems in graphs. In Chapter 4, we consider the optimization version of the classical Node Disjoint Paths problem in undirected graphs.

Problem 5 (Maximum Node Disjoint Paths (MNDP)). In the MNDP problem, we are given an undirected graph $G = (V, E)$ and a collection $\mathcal{M} = \{(s_1, t_1), \dots, (s_k, t_k)\} \subseteq V \times V$ of k source-sink pairs. The goal is to *route* a maximum cardinality subset of the pairs via *node-disjoint* paths. Formally, the goal is to select a maximum cardinality subset $\mathcal{M}' \subseteq \mathcal{M}$ and a collection of node-disjoint paths \mathcal{P} such that, for each pair $(s_i, t_i) \in \mathcal{M}'$, there is a path in \mathcal{P} with endpoints s_i and t_i .

The edge counterpart of MNDP is the Maximum Edge Disjoint Paths (MEDP) problem in which the goal is to route a subset of the pairs on *edge-disjoint* paths. Approximation algorithms for MEDP and MNDP have been studied extensively with MEDP receiving the most attention. The best approximation for both MEDP and MNDP is an $O(\sqrt{n})$ approximation [44, 127], where n is the number of nodes in the graph. Following a long line of work, a breakthrough result of Chuzhoy [53] showed that we can achieve much better approximations for MEDP in the setting in which we are allowed some *congestion* on the edges (the congestion is the maximum number of paths that use an edge). Chuzhoy obtained a poly-logarithmic approximation for MEDP with congestion 14; Chuzhoy and Li [57] improved the congestion to 2. In Chapter 4, we give a poly-logarithmic approximation for MNDP with constant node congestion. Prior to our work, the best approximation with congestion c was an $O(n^{1/c})$ approximation [127, 159].

In Chapter 5, we consider some fundamental maximum throughput routing problems in *directed* graphs. In this setting, we are given a capacitated directed graph $G = (V, E)$ with n nodes and m edges. We are also given source-destination pairs of nodes $(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)$. The goal is to select a largest subset of the pairs that are simultaneously *routable* subject to the capacities; a set of pairs is routable if there is a multicommodity flow for the pairs satisfying certain constraints that vary from problem to problem (e.g., integrality, unsplittability, edge or node capacities). Two well-studied optimization problems in this context are MEDP and the All-or-Nothing Flow (ANF) problem. In MEDP, a set of pairs is routable if there are edge-disjoint directed paths connecting s_i to t_i for each pair (s_i, t_i) . In ANF, a set of pairs is routable if there is a feasible multicommodity

flow that fractionally routes one unit of flow from s_i to t_i for each routed pair (s_i, t_i) . ANF, introduced in [40, 49], can be seen as a relaxation of MEDP in which the flow for the routed pairs is not required to be integral. MEDP and ANF are both **NP**-hard and their approximability has attracted substantial attention over the years. As we have already seen, MEDP and ANF admit poly-logarithmic approximation in *undirected* graphs if one allows constant congestion. In sharp contrast, both problems are hard to approximate to within a *polynomial* factor ($\Omega(n^\delta)$ for some fixed $\delta > 0$) in *directed* graphs even if constant congestion is allowed and the graph is acyclic. In Chapter 5, we study maximum throughput routing problems in directed graphs in the setting where the demand pairs are *symmetric*. Informally, in a symmetric demand pair instance, the input pairs are *unordered* and a pair $s_i t_i$ is routed only if both the ordered pairs (s_i, t_i) and (t_i, s_i) are routed. In particular, we focus our attention on the following problem.

Problem 6 (Symmetric All-or-Nothing Flow (SymANF)). In the SymANF problem, we are given a directed graph $G = (V, E)$ and a collection of (unordered) pairs of nodes $\mathcal{M} = \{s_1 t_1, s_2 t_2, \dots, s_k t_k\}$. A subset \mathcal{M}' of the pairs is *routable* if there is a feasible multicommodity flow in G such that, for each pair $s_i t_i \in \mathcal{M}'$, the amount of flow from s_i to t_i is at least one and the amount of flow from t_i to s_i is at least one. The goal is to find a maximum cardinality subset of the given pairs that can be routed.

We obtain a poly-logarithmic approximation with constant node congestion for the SymANF problem in directed graphs. Our result shows a strong separation between SymANF and its asymmetric counterpart, the ANF problem; as mentioned above, under a certain complexity theoretic assumption, the best approximation that we can achieve for the ANF problem in directed graphs is only a polynomial approximation.

1.2.3 Network design

The Steiner Tree and Steiner Forest problems are fundamental and well-studied problems in Network Design. In the Steiner Tree problem we have an undirected graph with costs on the edges and a set of nodes called terminals, and the goal is to select a minimum cost tree that connects the terminals. In the more general Steiner Forest problem we are given pairs of terminals and the goal is to select a minimum cost forest that connects each pair. In Chapter 6, we study the *prize-collecting* variants

of these problems.

Problem 7 (Prize-collecting Steiner Tree (PCST) and Prize-collecting Steiner Forest (PCSF)). In the PCST problem, we are given a graph $G = (V, E)$ with costs $c : E \rightarrow \mathbb{R}_+$ on the edges and penalties $\pi : V \rightarrow \mathbb{R}_+$ on the vertices. The goal is to find a tree $T \subseteq E$ that minimizes the sum of the edge-cost of T and the penalties of the vertices not included in T ; formally, the objective is to minimize $\sum_{e \in E(T)} c(e) + \sum_{v \notin V(T)} \pi(v)$. In the PCSF problem, we have a penalty $\pi(uv)$ for each pair uv of vertices, and the goal is to find a forest $F \subseteq E$ that minimizes the sum of the costs of the edges in F and the sum of the penalties of the pairs uv that are not connected by F .

The Steiner Tree problem is the special case of PCST in which the terminals have infinite penalty and the non-terminals have zero penalty. Similarly, the Steiner Forest problem is the special case of PCSF in which the terminal pairs have infinite penalty and all other pairs have zero penalty. The Steiner Tree and Steiner Forest problems and their prize-collecting counterparts have received considerable attention in the approximation algorithms literature [2, 7, 19, 28, 91, 96, 97]. In particular, they admit constant factor approximations and they are **APX**-hard to approximate [51]. There has also been significant interest in obtaining better approximations for these problems in restricted graphs; Arora [9] gave a polynomial time approximation scheme (**PTAS**) for Steiner Tree and related problems in low-dimensional Euclidean spaces, and Borradaile, Klein and Mathieu [24] obtained a **PTAS** for Steiner Forest in the Euclidean plane. When G is a planar graph, Borradaile, Klein, and Mathieu [25] obtained a **PTAS** for the Steiner Tree problem, and Bateni, Hajiaghayi, and Marx [17] obtained a **PTAS** for the Steiner Forest problem. In Chapter 6, we describe a **PTAS** for PCST in planar graphs. This result has two main components: an approximation preserving reduction from the PCST problem in planar graphs to the problem in graphs of fixed treewidth, and an exact algorithm for PCST in graphs of fixed treewidth. We also give an approximation preserving reduction from the PCSF problem in planar graphs to the problem in graphs of fixed treewidth. In contrast to PCST, the PCSF problem is **APX**-hard even in series-parallel graphs [16], which are planar and have treewidth two, and therefore we do not expect a **PTAS** in such graphs.

In Chapter 7, we consider certain generalizations of Steiner Tree and Steiner Forest in which the goal is to design a network that provides *higher connectivity* to the terminals. These problems are collectively referred to as the Survivable Network Design problem.

Problem 8 (Survivable Network Design). We are given a graph $G = (V, E)$ with weights on the edges or the nodes. We are also given integer requirements $r(uv)$ for each unordered pair uv of nodes. In the edge-connectivity Survivable Network Design (EC-SNDP) problem, the goal is to find a minimum cost subgraph H of G that contains $r(uv)$ *edge-disjoint* paths between u and v for each pair uv .

EC-SNDP arises naturally in the design of fault-tolerant networks where the goal is to find a minimum cost subgraph that satisfies the connectivity requirements. The cost of a network is dependent on the application. A common model is the edge-weighted model. A more general problem is obtained when each node of G has a weight and the cost of a subgraph H is the total weight of the nodes in H . Node weights are relevant in several applications. For example, in telecommunication networks, the node weights model the cost of setting up routing and switching infrastructure at a given node. The node-weighted versions of network design problems often turn out to be strictly harder to approximate than their corresponding edge-weighted versions. For instance, the edge-weighted Steiner Tree problem admits a 1.39-approximation [28]. In contrast, Klein and Ravi [124] showed, via a simple reduction from the Set Cover problem, that the node-weighted Steiner Tree problem is hard to approximate to within an $\Omega(\log n)$ factor unless $\mathbf{P} = \mathbf{NP}$, where n is the number of nodes in G . They also described a $(2 \log h)$ -approximation where h is the number of terminals. A more dramatic difference emerges if we consider EC-SNDP. Jain [106] gave a 2-approximation for the edge-weighted EC-SNDP problem. The best known approximation for the node-weighted EC-SNDP problem is $O(k \log n)$ [136], where $k = \max_{uv} r(uv)$ is the maximum connectivity requirement. Nutov [136] gives evidence, via a reduction from the k -Densest Subgraph problem, that a dependence on k in the approximation ratio is necessary for the node-weighted problem. Demaine, Hajiaghayi and Klein [65] considered the approximability of the node-weighted Steiner Tree problem in planar graphs. In an interesting result, they adapted a well-known primal-dual algorithm for the edge-weighted problem [2, 88] to the node-weighted setting and they showed that it gives a 6-approximation in planar and minor-closed families of graphs. In Chapter 7, we extend the result of [65] to higher connectivity. Our main result is an $O(k)$ approximation for the node-weighted EC-SNDP problem in planar and minor-closed families of graphs, where k is the maximum requirement of a pair.

Chapter 2

The Submodular Cost Allocation Problem

2.1 Introduction

In this chapter¹, we consider the following allocation problem with submodular costs.

Minimum Submodular-Cost Allocation (MSCA). Let V be a finite ground set and let f_1, \dots, f_k be k non-negative submodular set functions on V . In the MSCA problem the goal is to partition the ground set V into k (possibly empty) sets A_1, \dots, A_k such that the sum $\sum_{i=1}^k f_i(A_i)$ is minimized.

We observe that the problem is interesting only if the f_i 's are different for otherwise allocating all of V to f_1 is trivially an optimal solution. The special case of this problem in which all of the functions are monotone has been previously considered by Svitkina and Tardos [162]. In this chapter, we consider the problem with both monotone and non-monotone functions. We show that several well-studied problems such as non-metric facility location, multiway cut problems in graphs and hypergraphs, uniform metric labeling and its generalization to hub location among others can be cast as special cases of MSCA. In particular, we investigate the integrality gap of a simple and natural convex-programming relaxation for MSCA that is obtained via the use of the Lovász extension of a submodular function.

Lovász extension and a convex program for MSCA: Let V be a finite ground set of cardinality n . Each real-valued set function on V corresponds to a function $f : \{0, 1\}^n \rightarrow \mathbb{R}$ on the vertices of the n -dimensional hypercube. The Lovász extension of f to the continuous domain $[0, 1]^n$ denoted by \hat{f} is defined as²

$$\hat{f}(\mathbf{x}) = \mathbb{E}_{\theta \in [0, 1]} [f(\mathbf{x}^\theta)] = \int_0^1 f(\mathbf{x}^\theta) d\theta,$$

¹This chapter is based on a paper with Chandra Chekuri [34] and a manuscript with Jan Vondrák [70]. Copyrights to the conference version of [34] are held by Springer.

²The definition is not the standard one but is equivalent to it; see [166] or Section 2.6. This definition is convenient to us in describing and understanding rounding procedures.

where $\mathbf{x}^\theta \in \{0, 1\}^n$ for a given vector $\mathbf{x} \in [0, 1]^n$ is defined as: $x_i^\theta = 1$ if $x_i \geq \theta$ and 0 otherwise.

Lovász showed that \hat{f} is convex if and only if f is a submodular set function [132]. Moreover, it is easy to see that, given \mathbf{x} , the value $\hat{f}(\mathbf{x})$ can be computed in polynomial time by using a value oracle for f . Via this extension, we obtain a straightforward relaxation for MSCA with a convex objective function and linear constraints. Let v_1, \dots, v_n denote the elements of V . The relaxation has variables $x(v, i)$ for $v \in V$ and $1 \leq i \leq k$ with the interpretation that $x(v, i)$ is 1 if v is assigned to A_i and 0 otherwise. Let $\mathbf{x}_i = (x(v_1, i), \dots, x(v_n, i))$. The relaxation is given below.

LE-Rel

$$\begin{aligned} \min \quad & \sum_{i=1}^k \hat{f}_i(\mathbf{x}_i) \\ & \sum_{i=1}^k x(v, i) = 1 \quad \forall v \\ & x(v, i) \geq 0 \quad \forall v, i \end{aligned}$$

Throughout, we use OPT and OPT_{frac} to denote the value of an optimal integral and an optimal fractional solution to LE-Rel (respectively).

We remark that LE-Rel can be solved in time that is polynomial in n and $\log(\max_{S \subseteq V} f(S))$ via the ellipsoid method; we give some of the details in Section 2.6. Moreover, for some problems of interest the above convex program can be rewritten into an equivalent linear program. We now describe several problems that can be cast as special cases of MSCA, and also how some previously considered linear-programming relaxations can be seen as being equivalent to the convex program above.

2.1.1 Problems related to MSCA

Monotone MSCA and Facility Location: In facility location, we have a set of facilities \mathcal{F} and a set of clients or demands \mathcal{D} . There is a non-negative cost c_{ij} to connect facility i to client j (we do not necessarily assume that these costs form a metric). Opening facility $i \in \mathcal{F}$ costs f_i . The goal is to open a subset of the facilities and assign each client to an open facility so as to minimize the sum of the facility opening cost and the connection costs. Svitkina and Tardos [162] considered the setting where the cost of opening a facility i is a monotone submodular function g_i

of the clients assigned to it, and gave an $(1 + \ln |\mathcal{D}|)$ -approximation, and matching hardness via a reduction from Set Cover. We note that this problem is equivalent to MSCA when all the f_i are monotone submodular functions, which we refer to as MonMSCA. In [162] a greedy algorithm via submodular function minimization is used to derive the approximation. Here we prove that the integrality gap of LE-Rel is $(1 + \ln |\mathcal{D}|)$, and describe how certain rounding algorithms achieve this bound. These algorithms are useful when considering functions that are not necessarily monotone.

Submodular Multiway Partition: Let $f : 2^V \rightarrow \mathbb{R}_+$ be a submodular set function over V and let $\mathcal{S} = \{s_1, s_2, \dots, s_k\}$ be k terminals in V . The Submodular Multiway Partition (SubMP) problem asks for a partition of V into A_1, \dots, A_k such that $s_i \in A_i$ and $\sum_{i=1}^k f(A_i)$ is minimized. This problem has been previously considered by Zhao, Nagamochi and Ibaraki [173]. We can verify that the problem is a special case of MSCA as follows. Define the ground set to be $V' = V \setminus \mathcal{S}$ and, for $1 \leq i \leq k$, $f_i : 2^{V'} \rightarrow \mathbb{R}_+$ is the function defined as $f_i(S) = f(S \cup \{s_i\})$. We refer to the special case of the problem in which f is symmetric ($f(A) = f(V - A)$ for all A) as the symmetric SubMP (SymSubMP) problem. Note that, although the problem is based on a single function f , k different submodular functions (induced by the terminals) are needed to reduce it to MSCA. The Submodular Multiway Partition problem generalizes the classical multiway cut problem in graphs and hypergraphs. We discuss the problem and its special cases in Chapter 3.

Uniform Metric Labeling and Submodular Cost Labeling: The metric labeling problem was introduced by Kleinberg and Tardos [122] as a general classification problem. We are given an undirected edge-weighted graph $G = (V, E)$ and k labels and the goal is to assign a label to each vertex to minimize the labeling cost and the edge-cut cost. Assigning label i to v incurs a cost $c_i(v)$ and if an edge uv of weight $w(uv)$ has u labeled with i and v labeled with j then the edge-cut cost incurred is $w(uv) \cdot d(ij)$. The uniform metric labeling problem is obtained when $d(ij) = 1$ for all $i \neq j$. We consider the following generalization of the uniform metric labeling problem that we call the Submodular Cost Labeling (SubLabel) problem, which is a special case of MSCA. The k labels correspond to the k functions f_1, \dots, f_k . We define f_i as the sum of two functions, a monotone function g_i that models the label assignment cost, and a non-monotone function h that models the cut-cost. The goal is to partition V into A_1, \dots, A_k to minimize $\sum_{i=1}^k (g_i(A_i) + h(A_i))$. Note that the uniform metric labeling problem is the special case in which g_i are modular and h is the graph

cut function, which is symmetric. We are motivated to consider this generalization by problems that have been considered previously, such as the uniform metric labeling problem on hypergraphs, the hub location problem [86], and the extension of uniform metric labeling to handle label opening costs [63].

2.1.2 Overview of results and techniques

In this chapter, we examine the complexity of MSCA primarily through the “integrality gap” of the convex relaxation LE-Rel which can be optimized in polynomial time. All the problems we consider are **NP**-hard and our focus is on polynomial time approximation algorithms.

We highlight the naturalness of MSCA and the Lovász-extension based relaxation LE-Rel by showing connections to previously studied problems, linear programming relaxations, and rounding strategies. Viewing these problems in the more abstract setting of submodularity gives insights into prior algorithms. In the process, we obtain new and interesting results. Although one would like to obtain a single unifying algorithm that achieves a good approximation for MSCA, it turns out that the MSCA problem is inapproximable; we show in Section 2.5 that it is **NP**-hard to achieve any finite approximation factor for the problem.

Rounding the convex relaxation: Recall that the objective function in LE-Rel is $\sum_{i=1}^k \hat{f}_i(\mathbf{x}_i)$, where $\hat{f}_i(\mathbf{x}_i) = \mathbb{E}_{\theta \in [0,1]}[f(\mathbf{x}_i^\theta)]$. How do we round while preserving the objective function? If we focus on a specific i , the objective function suggests that we pick θ randomly from $[0, 1]$ and assign the elements in \mathbf{x}_i^θ to i ; we call this θ -rounding. However, there are two issues to contend with. First, if we independently round for each i then the same element may be assigned multiple times. Second, we need to ensure that all elements are assigned, which is not guaranteed by the θ -rounding.

Since the MSCA problem is inapproximable, there is no rounding strategy that works in general. Our approach is to understand the rounding process by considering various special cases of interest. In particular, we consider monotone functions, symmetric functions, the hypergraph separation cost function (which is asymmetric), and combinations of such functions. Monotonicity helps in that if elements are assigned to a label i , they can be removed without increasing the fractional cost. Although one can use different strategies to obtain an $O(\ln n)$ -approximation and integrality gap for MonMSCA, a useful strategy here is the rounding of Kleinberg and Tardos [122] that they

introduced for metric labeling. This has the additional property of ensuring that an element u is assigned to i with probability exactly $x(u, i)$. We use **KT Rounding** for **SubLabel**, since it is a reasonable strategy to approximately preserve the assignment costs, which are monotone. For the uniform metric labeling problem, [122] showed that **KT Rounding** approximately (to within a factor of 2) preserves the fractional connection cost in the case of graphs. We show bounds for hypergraph cut functions in an analogous fashion. Our insights enable us to develop a variant of the rounding that gives an $O(\log n)$ -approximation for **SubLabel** when the cut function is an arbitrary symmetric submodular function.

In this chapter, we focus on **MonMSCA** and **SubLabel**. In Chapter 3, we consider the **SubMP** problem and its special case, the **Symmetric Submodular Multiway Partition** problem, for which the function is also symmetric.

Other related work: There has been much recent interest in optimizing with submodular set functions. In particular, maximization problems have been examined via combinatorial techniques as well as the multilinear relaxation [59]. The submodular welfare problem [165] is similar in spirit to **MSCA** except that one is interested in maximizing the value of an allocation rather than minimizing the cost. Minimization problems with submodular costs have also received substantial attention [87, 90, 103, 160] with several negative results for basic problems as well as positive approximation results for problems such as the submodular cost vertex cover problem [87, 103]. Lovász-extension based convex programs have been effectively used for these problems. Various submodular cut and partition problems and their special cases, such as the hypergraph cut and partition, have been studied recently [79, 138, 171, 173]; however, these papers have typically focussed on greedy and divide-and-conquer based approaches while we use **LE-Rel**.

2.2 Approximation algorithms and integrality gap for **MonMSCA**

In this section we consider **MonMSCA** where f_1, \dots, f_k are monotone submodular functions. We will assume for simplicity that $f_i(\emptyset) = 0$ for all i . Throughout this chapter, we use n to denote the size of the ground set V . Svitkina and Tardos [162] considered this problem in the context of facility location and gave a $(1 + \ln n)$ -approximation and matching hardness via an approximation preserving reduction from set cover. Let $\alpha = \min_{S \subseteq V, 1 \leq i \leq k} f_i(S)/|S|$. The main observation

MonMSCA Greedy:

```

let  $x$  be a solution to LE-Rel
 $A_i \leftarrow \emptyset$  for all  $i$  ( $1 \leq i \leq k$ ) «the set of vertices that will be assigned to  $i$ »
 $U \leftarrow V$  «the set of unassigned vertices»
while  $U \neq \emptyset$ 
  let  $\tilde{x}$  be the restriction of  $x$  to  $U$ 
  for each  $\theta$ , let  $A(i, \theta) = \{v \mid v \in U, \tilde{x}(v, i) \geq \theta\}$ 
  let  $0 = \theta_{i,1} < \theta_{i,2} < \dots < \theta_{i,\ell_i} = 1$  be the distinct entries of  $\tilde{\mathbf{x}}_i$ 
  let  $(i', j')$  be the pair of indices in the set  $\{(i, j) \mid 1 \leq i \leq k, 1 \leq j < \ell_i\}$ 
    that minimizes the ratio  $f_i(A(i, \theta_{i,j}))/|A(i, \theta_{i,j})|$ 
   $A_{i'} \leftarrow A_{i'} \cup A(i', \theta_{i',j'})$ 
   $U \leftarrow U - A(i', \theta_{i',j'})$ 

```

Figure 2.1: Greedy rounding algorithm for MonMSCA.

in [162] is that $\alpha \leq \text{OPT}/n$, and moreover a pair (S, i) such that $f_i(S)/|S| = \alpha$ can be computed in polynomial time via submodular function minimization. One can then iterate using a greedy scheme, by using the monotonicity of the functions, to obtain a $(1 + \ln n)$ -approximation. Using a similar argument, we can prove the following theorem.

Theorem 2.2.1. *The integrality gap of LE-Rel for MonMSCA is at most $(1 + \ln n)$. In particular, $\alpha \leq \text{OPT}_{\text{frac}}/n$.*

Our greedy rounding strategy is given in Figure 2.1. In the following, we show that it achieves an H_n -approximation for MonMSCA.

Theorem 2.2.2. *MonMSCA Greedy achieves an H_n -approximation for MonMSCA.*

We start by introducing some notation. Consider an iteration of the while loop of MonMSCA Greedy. Let U be the set of elements that are unassigned at the beginning of the iteration, and let \tilde{x} denote the restriction of x to U ; more precisely, $\tilde{x}(v, i) = x(v, i)$ for each label i and each vertex $v \in U$. For any threshold $\theta \in [0, 1]$, let $A(i, \theta) = \{v \mid v \in U, \tilde{x}(v, i) \geq \theta\}$. Let $0 = \theta_{i,1} < \theta_{i,2} < \dots < \theta_{i,\ell_i} = 1$ be the distinct entries of $\tilde{\mathbf{x}}_i$. Let $\text{OPT}_{\text{frac}} = \sum_{i=1}^k f_i(\mathbf{x}_i)$. Theorem 2.2.2 is an immediate corollary of the following lemma.

Lemma 2.2.3. *We have*

$$\min_{1 \leq i \leq k} \min_{0 \leq j < \ell_i} \frac{f_i(A(i, \theta_{i,j}))}{|A(i, \theta_{i,j})|} \leq \frac{\text{OPT}_{\text{frac}}}{|U|}.$$

In order to prove Lemma 2.2.3, we will show that, if we choose a label $i \in \{1, 2, \dots, k\}$ and a threshold $\theta \in [0, 1]$ uniformly at random, the ratio $\mathbb{E}[f_i(A(i, \theta))] / \mathbb{E}[|A(i, \theta)|]$ is at most $\text{OPT}_{\text{frac}} / |U|$. We analyze $\mathbb{E}[f_i(A(i, \theta))]$ and $\mathbb{E}[|A(i, \theta)|]$ in the following propositions.

Proposition 2.2.4. *We have*

$$\mathbb{E}_{i, \theta}[f_i(A(i, \theta))] \leq \frac{1}{k} \text{OPT}_{\text{frac}}.$$

Proof. Consider a label i . For any θ , $A(i, \theta)$ is a subset of $\{v \mid v \in V, x(v, i) \geq \theta\}$. Since f_i is monotone, $f_i(A(i, \theta)) \leq f_i(\{v \mid v \in V, x(v, i) \geq \theta\})$. Therefore

$$\begin{aligned} \mathbb{E}_{\theta \in [0, 1]}[A(i, \theta)] &= \int_0^1 f_i(A(i, \theta)) d\theta \\ &\leq \int_0^1 f_i(\{v \mid v \in V, x(v, i) \geq \theta\}) d\theta \\ &= \hat{f}_i(\mathbf{x}_i) \end{aligned}$$

Finally,

$$\mathbb{E}_{i, \theta}[f_i(A(i, \theta))] = \frac{1}{k} \sum_{i=1}^k \mathbb{E}_{\theta \in [0, 1]}[f_i(A(i, \theta))] \leq \frac{1}{k} \text{OPT}_{\text{frac}}.$$

□

Proposition 2.2.5. *We have*

$$\mathbb{E}_{i, \theta}[|A(i, \theta)|] = \frac{1}{k} |U|.$$

Proof. Note that

$$\mathbb{E}_{i, \theta}[|A(i, \theta)|] = \frac{1}{k} \sum_{i=1}^k \mathbb{E}_{\theta \in [0, 1]}[|A(i, \theta)|].$$

We can prove by induction on the size of U that $\sum_{i=1}^k \mathbb{E}_{\theta \in [0, 1]}[|A(i, \theta)|]$ is equal to $|U|$.

If U is empty, the claim trivially holds. Therefore we may assume that U contains at least one element z . Let $U' = U - \{z\}$, and let \tilde{x}' be the restriction of \tilde{x} to U' ; more precisely, $\tilde{x}'(v, i) = \tilde{x}(v, i)$ for each $v \in U'$ and each label i . Let $A'(i, \theta) = \{v \mid v \in U', \tilde{x}'(v, i) \geq \theta\}$. Note that $A'(i, \theta)$ is equal

to $A(i, \theta) - \{z\}$ if θ is smaller than $x(z, i)$, and $A'(i, \theta)$ is equal to $A(i, \theta)$ otherwise. Therefore

$$\begin{aligned}
\sum_{i=1}^k \mathbb{E}_{\theta \in [0,1]} [|A(i, \theta)|] &= \sum_{i=1}^k \int_0^1 |A(i, \theta)| d\theta \\
&= \sum_{i=1}^k \int_0^{x(z,i)} |A'(i, \theta) \cup \{z\}| d\theta + \sum_{i=1}^k \int_{x(z,i)}^1 |A'(i, \theta)| d\theta \\
&= \sum_{i=1}^k \int_0^1 |A'(i, \theta)| d\theta + \sum_{i=1}^k x(z, i) \\
&= \sum_{i=1}^k \int_0^1 |A'(i, \theta)| d\theta + 1 \\
&= |U'| + 1 \quad (\text{By induction}) \\
&= |U|
\end{aligned}$$

Therefore $\mathbb{E}_{i,\theta} [|A(i, \theta)|] = |U|/k$, as desired. \square

Proof of Lemma 2.2.3: Let i be a label selected uniformly at random. Let θ be a threshold selected uniformly at random from the interval $[0, 1]$. It follows from Proposition 2.2.4 and Proposition 2.2.5 that the ratio $\mathbb{E}[f_i(A(i, \theta))]/\mathbb{E}[|A(i, \theta)|]$ is at most $\text{OPT}_{\text{frac}}/|U|$. By linearity of expectation,

$$\mathbb{E}_{i,\theta} \left[f_i(A(i, \theta)) - \frac{\text{OPT}_{\text{frac}}}{|U|} \cdot |A(i, \theta)| \right] \leq 0,$$

and therefore there exists a terminal i' and a threshold θ' for which the ratio $f_{i'}(A(i', \theta'))/|A(i', \theta')|$ is at most $\text{OPT}_{\text{frac}}/|U|$. Let j' be the smallest index j that satisfies $0 \leq j < \ell_{i'}$ and $\theta_{i',j} \geq \theta'$. Since $A(i', \theta_{i',j'}) = A(i', \theta')$, (i', j') is the desired pair. \square

In the remainder of this section, we consider a different algorithm that achieves an $O(\ln n)$ -approximation for MonMSCA. We will use this algorithm as a building block for submodular cost labeling algorithms (see Section 2.3). The algorithm **KT Rounding** is derived from the work of Kleinberg and Tardos on metric labeling [122] and it is given in Figure 2.2.

We prove the following theorem by building on some useful properties of the rounding algorithm. One of these properties, which was shown in [122], is that the probability that v gets assigned to i in the rounding is precisely $x(v, i)$. In particular, if the assignment costs are modular, the expected assignment cost of the integral solution constructed by the algorithm is equal to the assignment

KT Rounding

```

let  $x$  be a solution to LE-Rel
 $S \leftarrow \emptyset$      $\langle\langle$ set of all assigned vertices $\rangle\rangle$ 
 $\langle\langle$ set of vertices that are eventually assigned to  $i$  $\rangle\rangle$ 
 $A_i \leftarrow \emptyset$  for all  $i$  ( $1 \leq i \leq k$ )
while  $S \neq V$ 
    pick  $i \in \{1, 2, \dots, k\}$  uniformly at random
    pick  $\theta \in [0, 1]$  uniformly at random
     $A_i \leftarrow A_i \cup (\{v \mid x(v, i) \geq \theta\} - S)$ 
     $S \leftarrow S \cup A_i$ 
return  $(A_1, \dots, A_k)$ 

```

Figure 2.2: KT rounding for MonMSCA.

cost of the fractional solution.

Lemma 2.2.6 (Lemma 3.1 in [122]). *For each vertex v and each label i , the probability that v is assigned label i by KT Rounding is equal to $x(v, i)$.*

Lemma 2.2.7. *The number of iterations of the while loop of KT Rounding is $O(k \ln n)$ with probability at least $1 - 1/n$.*

Proof. Let $N = c \ln n$, where c is a sufficiently large constant. Let i_ℓ and θ_ℓ be the label and the threshold chosen in iteration ℓ of the algorithm. For each vertex v , we have

$$\begin{aligned}
 \Pr[x(v, i_\ell) < \theta_\ell] &= \sum_{i=1}^k \Pr[x(v, i_\ell) < \theta_\ell \mid i_\ell = i] \Pr[i_\ell = i] \\
 &= \frac{1}{k} \sum_{i=1}^k (1 - x(v, i)) = 1 - \frac{1}{k}
 \end{aligned}$$

Since the iterations are independent, the probability that v does not have a label after N iterations is at most

$$\left(1 - \frac{1}{k}\right)^N \leq \exp\left(-\frac{N}{k}\right) = \frac{1}{n^c}.$$

By the union bound, the probability that there exists an unlabeled vertex after N iterations is at most $1/n^{c-1}$. \square

Theorem 2.2.8. *Let A_1, \dots, A_k be the labeling constructed by KT Rounding. We have $\sum_{i=1}^k \mathbb{E}[f_i(A_i)] \leq O(\ln n) \sum_{i=1}^k \hat{f}_i(\mathbf{x}_i)$. Therefore KT Rounding achieves a randomized $O(\ln n)$ -*

approximation for MonMSCA.

Proof. By Lemma 2.2.7, the rounding terminates in $O(k \ln n)$ iterations of the while loop with probability at least $(1 - 1/n)$. Consider an iteration and let i and θ be the label and the threshold chosen. Let $A(i, \theta) = \{v \mid x(v, i) \geq \theta\}$. We have

$$\begin{aligned} \mathbb{E}_{i, \theta}[f_i(A(i, \theta))] &= \sum_{j=1}^k \mathbb{E}_{\theta}[f_i(A(i, \theta)) \mid i = j] \Pr[i = j] \\ &= \frac{1}{k} \sum_{j=1}^k \hat{f}_j(\mathbf{x}_j) = \frac{1}{k} \text{OPT}_{\text{frac}} \end{aligned}$$

Since the algorithm terminates in $O(k \ln n)$ iterations with probability at least $(1 - 1/n)$, by linearity of expectation and the sub-additivity of the functions (since the functions are submodular and $f(\emptyset) = 0$), the total expected cost is $O(\ln n) \text{OPT}_{\text{frac}}$. \square

2.3 Approximation algorithms and integrality gap for SubLabel

In this section we consider SubLabel, which generalizes MonMSCA, uniform metric labeling, hub location, and other problems. A natural algorithm for this problem is KT Rounding, which we have already introduced in Section 2.2. We also describe a different algorithm, SymSubLabel Rounding, which is appropriate for SubLabel when the separation cost function is a symmetric submodular function. Finally, we build on the SymSubLabel Rounding algorithm to get an approximation algorithm for the general SubLabel problem.

2.3.1 Labeling in hypergraphs

In this section, we consider the special case of SubLabel in which the separation cost function h is the following function. Let $\mathcal{G} = (V, \mathcal{E})$ be a hypergraph with edge weights $w(e)$. For each edge hyperedge e , we pick an arbitrary representative node $r(e) \in e$. We define the function $h : 2^V \rightarrow \mathbb{R}_+$ as follows: for $A \subseteq V$, let $f(A) = \sum_{e: r(e) \in A, e \not\subseteq A} w(e)$ be the weight of hyperedges whose representatives are in A and they cross A (a hyperedge e crosses a set A if e has a vertex in A and a vertex in $V - A$). We refer to this function as the hypergraph separation cost function.

Note that, if A_1, \dots, A_k is a partition of the vertices of the hypergraph, the total separation

cost $\sum_{i=1}^k f(A_i)$ is equal to the number of hyperedges that cross the partition, i.e., the hyperedges with a vertex in at least two parts.

In the following, we use $\Delta = \max_{e \in \mathcal{E}} |e|$ to denote the maximum hyperedge size. In the remainder of this section, we analyze the KT Rounding algorithm for the special case of **SubLabel** in which h is the hypergraph separation cost function, and we show that it achieves a Δ approximation when the assignment costs are modular and an $O(\ln n + \Delta)$ approximation when the assignment costs are arbitrary monotone submodular functions.

Theorem 2.3.1. *If h is the hypergraph separation cost function and each g_i is modular, KT Rounding achieves a randomized Δ -approximation for **SubLabel**, where Δ is the maximum hyperedge size.*

Theorem 2.3.2. *If h is the hypergraph separation cost function and each g_i is a monotone submodular function, KT Rounding achieves a randomized $O(\ln n + \Delta)$ approximation for **SubLabel**, where Δ is the maximum hyperedge size.*

We start by introducing some notation. Let \mathbf{x} be a solution to **LE-Rel** for the instance. The cost of the fractional solution is $\sum_{i=1}^k \hat{g}_i(\mathbf{x}_i) + \sum_{i=1}^k \hat{h}(\mathbf{x}_i)$. Let $\text{LP}_g = \sum_{i=1}^k \hat{g}_i(\mathbf{x}_i)$ be the assignment cost paid by the fractional solution, and let $\text{LP}_h = \sum_{i=1}^k \hat{h}(\mathbf{x}_i)$ be the separation cost paid by the fractional solution.

We can write the cost LP_h in a more convenient form as follows. For each hyperedge e and each label i , let $d(e, i) = x(r(e), i) - \min_{v \in e} x(v, i)$. Let $d(e) = \sum_{i=1}^k d(e, i) = 1 - \sum_{i=1}^k \min_{v \in e} x(v, i)$. The contribution of e to LP_h is $w(e)d(e)$, i.e., $\text{LP}_h = \sum_{e \in \mathcal{E}} w(e)d(e)$.

Lemma 2.3.3. *For each label i , we have $\hat{h}(\mathbf{x}_i) = \sum_{e \in \mathcal{E}} w(e)d(e, i)$. Thus $\text{LP}_h = \sum_{e \in \mathcal{E}} w(e)d(e)$.*

Proof. Consider a label i . For a threshold $\theta \in [0, 1]$, we let $A(i, \theta) = \{v \mid x(v, i) \geq \theta\}$ be the set of all vertices that are above the threshold for label i . By the definition of the Lovász extension, we have $\hat{h}(\mathbf{x}_i) = \mathbb{E}_{\theta \in [0, 1]} [h(A(i, \theta))]$. For any θ , $h(A(i, \theta))$ is the total weight of hyperedges e such that the representative of e is in $A(i, \theta)$ and e crosses $A(i, \theta)$. The representative $r(e)$ of e is in $A(i, \theta)$ if and only if $x(r(e), i) \geq \theta$. Additionally, $A(i, \theta)$ contains a vertex of e only if $\min_{v \in e} x(v, i) \leq \theta$. Therefore the contribution of e to $\hat{h}(\mathbf{x}_i)$ is $w(e)(x(r(e), i) - \min_{v \in e} x(v, i)) = w(e)d(e, i)$. Thus $\hat{h}(\mathbf{x}_i) = \sum_{e \in \mathcal{E}} w(e)d(e, i)$. \square

Let A_1, \dots, A_k be the labeling constructed by KT Rounding. Let $\text{ALG}_g = \sum_{i=1}^k g_i(A_i)$ be the assignment cost of the solution, and let $\text{ALG}_h = \sum_{i=1}^k h(A_i)$ be the separation cost of the solution. In the following, we relate $\mathbb{E}[\text{ALG}_g]$ to LP_g and $\mathbb{E}[\text{ALG}_h]$ to LP_h .

By Lemma 2.2.6, for each vertex v and each label i , the probability that KT Rounding assigns label i to v is equal to $x(v, i)$. Therefore the rounding preserves the assignment costs if the cost functions g_i are modular, i.e., $\mathbb{E}[\text{ALG}_g] = \text{LP}_g$.

If the assignment cost functions g_i are monotone submodular functions, Theorem 2.2.8 shows that $\mathbb{E}[\text{ALG}_g] \leq O(\ln n) \text{LP}_g$.

Thus it suffices to upper bound the expected separation cost of the algorithm.

Theorem 2.3.4. *We have $\mathbb{E}[\text{ALG}_h] \leq \Delta \cdot \text{LP}_h$.*

In order to upper bound the expected separation cost of the labeling constructed by KT Rounding, we consider each hyperedge separately and we upper bound the probability that the vertices of the hyperedge have at least two different labels. We say that a hyperedge e is *split* in some iteration of KT Rounding if there exists an iteration ℓ such that at least one vertex of e is assigned a label in iteration ℓ but not all vertices of e are assigned a label in iteration ℓ . The following lemma upper bounds the probability that a hyperedge e is split.

Lemma 2.3.5. *For each hyperedge e , the probability that e is split is at most $\Delta d(e)$.*

Theorem 2.3.4 follows from the lemma above.

Proof of Theorem 2.3.4: The expected separation cost of the labeling constructed by KT Rounding is the expected weight of the hyperedges that are split. More precisely, we have

$$\begin{aligned} \mathbb{E}[\text{ALG}_h] &= \sum_{e \in \mathcal{E}} w(e) \Pr[e \text{ is split}] \\ &\leq \Delta \sum_{e \in \mathcal{E}} w(e) d(e) && \text{(By Lemma 2.3.5)} \\ &= \Delta \cdot \text{LP}_h && \text{(By Lemma 2.3.3)} \end{aligned}$$

□

We devote the rest of this section to the proof of Lemma 2.3.5. We will need the following lemma.

Lemma 2.3.6. *Let e be any hyperedge, and let z be any vertex in e . Let $R(z) = \{i \mid x(z, i) = \max_{v \in e} x(v, i)\}$. Then $\sum_{i \in R(z)} (\max_{v \in e} x(v, i) - \min_{v \in e} x(v, i)) \leq d(e)$.*

Proof. Let u be the representative of e . If $z = u$, the lemma is immediate. Therefore we may assume that $z \neq u$. We partition $\{1, 2, \dots, k\}$ into two sets: the set $S(z)$ consisting of all coordinates i such that $x(z, i)$ is smaller than $x(u, i)$, and the set $B(z)$ consisting of all coordinates i such that $x(z, i)$ is at least $x(u, i)$. Since $\sum_{i=1}^k x(z, i)$ and $\sum_{i=1}^k x(u, i)$ are both equal to 1, the total difference between $x(u, i)$ and $x(z, i)$ over all coordinates $i \in S(z)$ is equal to the total difference between $x(z, i)$ and $x(u, i)$ over all coordinates $i \in B(z)$. Therefore

$$\begin{aligned} \sum_{i \in B(z)} (x(z, i) - x(u, i)) &= \sum_{i \in S(z)} (x(u, i) - x(z, i)) \\ &\leq \sum_{i \in S(z)} \left(x(u, i) - \min_{v \in e} x(v, i) \right) \\ &= d(e) - \sum_{i \notin S(z)} \left(x(u, i) - \min_{v \in e} x(v, i) \right) \\ &\leq d(e) - \sum_{i \in B(z)} \left(x(u, i) - \min_{v \in e} x(v, i) \right) \end{aligned}$$

Since $R(z)$ is a subset of $B(z)$, the lemma follows. \square

Proof of Lemma 2.3.5: Let θ_ℓ be the threshold chosen in iteration ℓ of the algorithm. We say that iteration ℓ *cuts* e if $\theta_\ell \in [\min_{v \in e} x(v, i_\ell), \max_{v \in e} x(v, i_\ell)]$. We say that iteration ℓ *touches* e if θ_ℓ is in the interval $[0, \max_{v \in e} x(v, i_\ell)]$. Let X_ℓ and Z_ℓ be the events that ℓ cuts and touches e (respectively). We have

$$\Pr[X_\ell] \leq \frac{1}{k} \sum_{i=1}^k \left(\max_{v \in e} x(v, i) - \min_{v \in e} x(v, i) \right) \leq \frac{\Delta d(e)}{k}$$

where the last inequality follows from Lemma 2.3.6. Additionally,

$$\Pr[Z_\ell] = \frac{1}{k} \sum_{i=1}^k \max_{v \in e} x(v, i) \geq \frac{1}{k}$$

The last inequality follows from the fact that, for any vertex $v \in e$, $\sum_{i=1}^k x(v, i) = 1$. It follows

that the probability that e is split in iteration j is at most $\Delta d(e)$.

Let L be a random variable that is equal to the first iteration that touches e . Note that e is split by the algorithm — that is, its vertices receive at least two different labels — only if L cuts e ; otherwise, all the vertices of e receive the same label in iteration L . Therefore we have

$$\begin{aligned}
\Pr[e \text{ is split} \mid L = \ell] &\leq \Pr[X_\ell \mid L = \ell] \\
&= \Pr[X_\ell \mid Z_\ell \wedge (\wedge_{j < \ell} \bar{Z}_j)] \\
&= \Pr[X_\ell \mid Z_\ell] \quad (\text{Different iterations are independent}) \\
&= \frac{\Pr[X_\ell \wedge Z_\ell]}{\Pr[Z_\ell]} = \frac{\Pr[X_\ell]}{\Pr[Z_\ell]} \leq \Delta d(e)
\end{aligned}$$

where the last inequality follows from the bounds on $\Pr[X_\ell]$ and $\Pr[Z_\ell]$ shown above. Thus

$$\begin{aligned}
\Pr[e \text{ is split}] &= \sum_{\ell \geq 1} \Pr[e \text{ is split} \mid L = \ell] \Pr[L = \ell] \\
&\leq \Delta d(e) \sum_{\ell \geq 1} \Pr[L = \ell] = \Delta d(e)
\end{aligned}$$

□

Let $\text{ALG} = \text{ALG}_g + \text{ALG}_h$ be the total cost of the labeling, and let $\text{LP} = \text{LP}_g + \text{LP}_h$ be the total cost of the fractional solution. If each of the functions g_i is modular, we have

$$\begin{aligned}
\mathbb{E}[\text{ALG}] &= \mathbb{E}[\text{ALG}_g] + \mathbb{E}[\text{ALG}_h] \\
&\leq \text{LP}_g + \Delta \cdot \text{LP}_h \\
&\leq \Delta \cdot \text{LP}
\end{aligned}$$

If each of the functions g_i is a monotone submodular function, we have

$$\begin{aligned}
\mathbb{E}[\text{ALG}] &= \mathbb{E}[\text{ALG}_g] + \mathbb{E}[\text{ALG}_h] \\
&\leq O(\ln n) \cdot \text{LP}_g + \Delta \cdot \text{LP}_h \\
&\leq O(\ln n + \Delta) \cdot \text{LP}
\end{aligned}$$

SymSubLabel Rounding

```

let  $x$  be a solution to LE-Rel
 $S \leftarrow \emptyset$      $\langle\langle$ set of all assigned vertices $\rangle\rangle$ 
 $A_i \leftarrow \emptyset$  for all  $i$  ( $1 \leq i \leq k$ )
while  $S \neq V$ 
    pick  $i \in \{1, 2, \dots, k\}$  uniformly at random
    pick  $\theta \in [0, 1]$  uniformly at random
     $A_i \leftarrow A_i \cup \{v \mid x(v, i) \geq \theta\}$ 
     $S \leftarrow S \cup \{v \mid x(v, i) \geq \theta\}$ 
 $\langle\langle$ uncross  $A_1, \dots, A_k$  $\rangle\rangle$ 
 $A'_i \leftarrow A_i$  for all  $i$  ( $1 \leq i \leq k$ )
while there exist  $i \neq j$  such that  $A'_i \cap A'_j \neq \emptyset$ 
    if  $(f(A'_i) + f(A'_j - A'_i)) \leq f(A'_i) + f(A'_j)$ 
         $A'_j \leftarrow A'_j - A'_i$ 
    else
         $A'_i \leftarrow A'_i - A'_j$ 
return  $(A'_1, \dots, A'_k)$ 

```

Figure 2.3: Rounding algorithm for the special case of SubLabel in which the separation cost function h is symmetric.

This completes the proofs of Theorem 2.3.1 and Theorem 2.3.2.

2.3.2 Labeling with a symmetric separation cost

In this section, we consider the special case of SubLabel in which the separation cost function h is symmetric.

Theorem 2.3.7. *If h is a symmetric submodular function and each g_i is a monotone submodular function, SymSubLabel Rounding achieves a randomized $O(\ln n)$ -approximation for SubLabel.*

We will need the following lemma that shows that the uncrossing step does not increase the separation cost of the labeling. The lemma is standard and it has been used in previous work [161]. We remark that the algorithm guaranteed by the lemma is the uncrossing algorithm described in SymSubLabel Rounding.

Lemma 2.3.8. *Let f be a symmetric submodular set function over V and let A_1, \dots, A_k be subsets of V . Then there exist sets A'_1, \dots, A'_k such that (i) $A'_i \subseteq A_i$ for $1 \leq i \leq k$, (ii) A'_1, \dots, A'_k are mutually disjoint (iii) $\cup_i A'_i = \cup_i A_i$ and (iv) $\sum_i f(A'_i) \leq \sum_i f(A_i)$. Moreover, given the A_i 's a*

collection of sets A'_i satisfying the above properties can be found in polynomial time via a value oracle for f .

Proof. Since f is symmetric, it satisfies posi-modularity; that is, $f(X) + f(Y) \geq f(X - Y) + f(Y - X)$. From this we see that either $f(X) + f(Y - X)$ or $f(Y) + f(X - Y)$ is no larger than $f(X) + f(Y)$. This allows us to uncross A_1, \dots, A_k as follows. If the A_i 's are mutually disjoint then we can set $A'_i = A_i$ for each i and they satisfy the desired properties. Otherwise, there exist distinct i and j such that $A_i \cap A_j \neq \emptyset$. We can replace A_i and A_j with A_i and $A_j - A_i$ if $f(A_i) + f(A_j - A_i) \leq f(A_i) + f(A_j)$; otherwise, we replace them by $A_i - A_j$ and A_j . We repeat this process to get the desired sets. \square

Proof of Theorem 2.3.7: Let A_1, \dots, A_k be the collection of sets constructed by the first while loop of SymSubLabel Rounding. Let A'_1, \dots, A'_k be the final labeling. For each i , A'_i is a subset of A_i . Since the assignment cost functions g_i are monotone, we have

$$\sum_{i=1}^k g_i(A'_i) \leq \sum_{i=1}^k g_i(A_i).$$

Since h is symmetric, it follows from Lemma 2.3.8 that

$$\sum_{i=1}^k h(A'_i) \leq \sum_{i=1}^k h(A_i).$$

Therefore it suffices to upper bound the expected cost of the sets A_1, \dots, A_k . Let $\text{ALG}_g = \sum_{i=1}^k g_i(A_i)$ and $\text{ALG}_h = \sum_{i=1}^k h(A_i)$. Let $\text{LP}_g = \sum_{i=1}^k \hat{g}_i(\mathbf{x}_i)$ be the total assignment cost paid by the fractional solution, and let $\text{LP}_h = \sum_{i=1}^k \hat{h}_i(\mathbf{x}_i)$ be the total separation cost paid by the fractional solution. We can relate $\mathbb{E}[\text{ALG}_g]$ to LP_g and $\mathbb{E}[\text{ALG}_h]$ to LP_h as follows.

Using the argument in the proof of Theorem 2.2.8, we can show that $\mathbb{E}[\text{ALG}_g] \leq O(\ln n) \cdot \text{LP}_g$. Therefore it suffices to analyze the separation cost. Let θ_ℓ and i_ℓ be the threshold and the label

chosen in iteration ℓ of the first while loop of the algorithm. Let $X_\ell = \{v \mid x(v, i_\ell) \geq \theta_\ell\}$. We have

$$\begin{aligned} \mathbb{E}_{\theta_\ell, i_\ell} [h(X_\ell)] &= \sum_{i=1}^k \mathbb{E}_{\theta_\ell} [h(X_\ell) \mid i_\ell = i] \Pr[i_\ell = i] \\ &= \frac{1}{k} \sum_{i=1}^k \hat{h}(\mathbf{x}_i) = \frac{\text{LP}_h}{k} \end{aligned}$$

The set A_i is the union of the sets X_ℓ for which $i_\ell = i$. A non-negative submodular function is subadditive and therefore

$$h(A_i) \leq \sum_{\ell: i_\ell = i} h(X_\ell).$$

Thus we have

$$\sum_{i=1}^k h(A_i) \leq \sum_{\ell \geq 1} h(X_\ell).$$

By Lemma 2.2.7, the number of iterations of the first while loop of the algorithm is $O(k \ln n)$ with probability at least $(1 - 1/n)$. Therefore $\mathbb{E}[\text{ALG}_h] = O(\ln n) \text{LP}_h$, and the theorem follows. \square

2.3.3 Approximation algorithm for **SubLabel**

In this section, we consider the general **SubLabel** problem in which each assignment cost function g_i is an arbitrary monotone submodular function and the separation cost function h is an arbitrary submodular function. Let h' be the following function: $h'(S) = h(S) + h(V - S)$ for each set S . The following proposition is well-known and we include a proof for completeness.

Proposition 2.3.9. *Let $f : 2^V \rightarrow \mathbb{R}$ be a submodular function. Let f' be the following function: $f'(S) = f(S) + f(V - S)$ for each set $S \subseteq V$. Then f' is submodular and symmetric.*

Proof. It is clear that f' is symmetric. Thus it suffices to verify that it is submodular. Let A and B be two sets. We need to verify that $f'(A) + f'(B) \geq f'(A \cap B) + f'(A \cup B)$. Since f is submodular, we have

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$$

and

$$\begin{aligned} f(V - A) + f(V - B) &\geq f((V - A) \cap (V - B)) + f((V - A) \cup (V - B)) \\ &= f(V - (A \cup B)) + f(V - (A \cap B)) \end{aligned}$$

By summing the two inequalities, we get that

$$f'(A) + f'(B) \geq f'(A \cap B) + f'(A \cup B),$$

as desired. □

Since h' is symmetric, we can use the algorithm **SymSubLabel Rounding** from Subsection 2.3.2 to construct a labeling for the instance of **SubLabel** in which the assignment costs are given by g_i and the separation cost is given by h' . For any labeling, we can relate its h' cost to its h cost as follows.

Proposition 2.3.10. *Let A_1, \dots, A_k be a labeling. We have*

$$\sum_{i=1}^k h(A_i) \leq \sum_{i=1}^k h'(A_i) \leq k \sum_{i=1}^k h(A_i).$$

Proof. The first inequality follows from the fact that h is non-negative. Therefore it suffices to show the second inequality. A non-negative submodular function is sub-additive and thus we have

$$h(V - A_i) = h(\cup_{j \neq i} A_j) \leq \sum_{j \neq i} h(A_j).$$

Therefore

$$\sum_{i=1}^k h(V - A_i) \leq (k-1) \sum_{i=1}^k h(A_i).$$

Thus

$$\sum_{i=1}^k h'(A_i) \leq k \sum_{i=1}^k h(A_i).$$

□

Let OPT and OPT' be the costs of the optimal solution for the original instance in which the

separation cost function is h and the symmetric instance in which the separation cost function is h' , respectively. Let A_1, \dots, A_k be the solution constructed by SymSubLabel Rounding for the symmetric instance. We have

$$\mathbb{E} \left[\sum_{i=1}^k g_i(A_i) + \sum_{i=1}^k h(A_i) \right] \leq \mathbb{E} \left[\sum_{i=1}^k g_i(A_i) + \sum_{i=1}^k h'(A_i) \right] \leq O(\ln n) \text{OPT}'.$$

The first inequality follows from the proposition above and the second inequality follows from Theorem 2.3.7. It follows from the proposition above that $\text{OPT}' \leq k\text{OPT}$, and therefore A_1, \dots, A_k is an $O(k \ln n)$ approximate solution for the original instance. Therefore we have the following theorem.

Theorem 2.3.11. *There is a randomized $O(k \ln n)$ approximation for the SubLabel problem.*

2.4 Hardness of SubLabel

In this section, we show that SubLabel is Set Cover hard even if the assignment cost functions g_i are modular and the separation cost function h is symmetric.

Theorem 2.4.1. *There is an approximation preserving reduction from the Set Cover problem to the special case of SubLabel in which each assignment cost function g_i is modular and the separation cost function h is symmetric. Moreover, each function g_i satisfies $g_i(S) = \sum_{v \in S} c(v, i)$, where $c(v, i)$ is either zero or infinity. The function h satisfies $h(A) = 0$ if $A \in \{\emptyset, V\}$ and $h(A) = 1$ otherwise.*

Remark 2.4.2. The function $h : 2^V \rightarrow \mathbb{R}$ that satisfies $h(A) = 0$ if $A \in \{\emptyset, V\}$ and $h(A) = 1$ otherwise is the cut function of a hypergraph on the vertex set V that has a single hyperedge containing all the vertices.

Our reduction is based on the reduction of Svitkina and Tardos [162] for MonMSCA. Consider an instance of Set Cover consisting of a set $V = \{v_1, \dots, v_n\}$ of n elements and a collection $\mathcal{S} = \{S_1, \dots, S_k\}$ of k sets. We construct an instance of SubLabel as follows. The ground set is the set V of elements in the Set Cover instance. We have a label i for each set S_i in \mathcal{S} . For each element v and each label i , we have an assignment cost $c(v, i)$ that is equal to zero if $v \in S_i$ and ∞ otherwise.

The assignment cost function g_i for the i -th label is defined as follows: $g_i(A) = \sum_{v \in A} c(v, i)$ for each set $A \subseteq V$. The separation function h is defined as follows: $h(A) = 0$ if $A \in \{\emptyset, V\}$ and $h(A) = 1$ otherwise.

Proposition 2.4.3. *Let $h : 2^V \rightarrow \mathbb{R}_+$ be the function such that $h(A) = 0$ if $A \in \{\emptyset, V\}$ and $h(A) = 1$ otherwise. The function h is submodular and symmetric.*

Proof. It is clear that h is symmetric. We can verify that h is submodular as follows. Let A and B be any two sets. We can verify that $h(A) + h(B) \geq h(A \cap B) + h(A \cup B)$ as follows. We have the following cases:

- $h(A \cap B) = h(A \cup B) = 0$. Since $h(A) \geq 0$ and $h(B) \geq 0$, the inequality clearly holds.
- $h(A \cap B) = 0$ and $h(A \cup B) = 1$. Then one of A or B is non-empty and both of them are not equal to V . Therefore $h(A) + h(B) \geq 1$ and the inequality holds.
- $h(A \cap B) = 1$ and $h(A \cup B) = 0$. Since $A \cap B$ is non-empty, both A and B are non-empty. Since $A \cap B$ is not equal to V , one of A or B is not equal to V . Therefore $h(A) + h(B) \geq 1$ and the inequality holds.
- $h(A \cap B) = h(A \cup B) = 1$. Since $A \cap B$ is non-empty, both A and B are non-empty. Since $A \cup B$ is not equal to V , neither A nor B is equal to V . Therefore $h(A) = h(B) = 1$ and the inequality holds.

Therefore h is submodular. □

Note that we may assume that there does not exist a set in \mathcal{S} that covers all the elements, since otherwise the solution consisting of such a set is an optimal solution.

Lemma 2.4.4. *Suppose that there does not exist a set in \mathcal{S} that covers all the elements. Then the Set Cover instance has a solution consisting of t sets if and only if the SubLabel instance has a solution of cost t .*

Proof. Consider a solution $\mathcal{S}' \subseteq \mathcal{S}$ for the Set Cover instance. We construct a labeling A_1, \dots, A_k inductively as follows. We let $A_1 = S_1$ if $S_1 \in \mathcal{S}'$ and $A_1 = \emptyset$ otherwise. Consider an index $i \geq 2$. We let $A_i = S_i - (A_1 \cup \dots \cup A_{i-1})$ if $S_i \in \mathcal{S}'$ and $A_i = \emptyset$ otherwise.

Note that the resulting sets A_1, \dots, A_k are disjoint and they cover all the elements. Since $A_i \subseteq S_i$, we have $c(v, i) = 0$ for each $v \in A_i$ and thus $g_i(A_i) = 0$. Additionally, $h(A_i) = 1$ only if $S_i \in \mathcal{S}'$. Therefore the total separation cost of the labeling is at most $|\mathcal{S}'|$.

Conversely, consider a solution A_1, \dots, A_k for the **SubLabel** instance. Note that we may assume that the solution has finite cost and thus $g_i(A_i) = 0$ for all labels i . It follows that $A_i \subseteq S_i$ for each i . We construct a set cover \mathcal{S}' as follows. For each i such that A_i is non-empty, we add S_i to \mathcal{S}' . Since the sets A_i cover all the elements and $A_i \subseteq S_i$ for each i , the sets of \mathcal{S}' cover all the elements as well. Since $V \notin \mathcal{S}$, $A_i \neq V$ for all i . Thus the cost of the labeling is equal to the number of non-empty sets in the labeling, which in turn it is equal to $|\mathcal{S}'|$. \square

2.5 Hardness of MSCA

If $k = 2$, the MSCA problem can be reduced to submodular function minimization as follows. Let $f : 2^V \rightarrow \mathbb{R}_+$ be the function such that $f(S) = f_1(S) + f_2(V - S)$. As shown in the following proposition, f is submodular. A submodular function can be minimized in polynomial time [61, 74, 104, 142, 154], and therefore MSCA is in **P** when $k = 2$.

Proposition 2.5.1. *Let $f_1 : 2^V \rightarrow \mathbb{R}$ and $f_2 : 2^V \rightarrow \mathbb{R}$ be two submodular functions. Let $f : 2^V \rightarrow \mathbb{R}$ be the function such that $f(S) = f_1(S) + f_2(V - S)$ for all $S \subseteq V$. The function f is submodular.*

Proof. Let g_2 be the function such that $g_2(S) = f_2(V - S)$ for all S . We can verify that g_2 is submodular as follows. Consider any two sets A and B . Since f_2 is submodular, we have

$$\begin{aligned} g_2(A) + g_2(B) &= f_2(V - A) + f_2(V - B) \\ &\geq f_2((V - A) \cap (V - B)) + f_2((V - A) \cup (V - B)) \\ &= f_2(V - (A \cup B)) + f_2(V - (A \cap B)) \\ &= g_2(A \cup B) + g_2(A \cap B) \end{aligned}$$

Since f_1 and g_2 are submodular, their sum is also submodular. \square

In this section, we prove that, for any $k \geq 3$, the MSCA problem does not admit any finite approximation factor. The reason is that for $k \geq 3$ it is **NP**-hard to decide whether the optimal

solution has value zero or nonzero.

Theorem 2.5.2. *It is **NP**-hard to decide whether the optimal value for an instance of MSCA is zero, even for $k = 3$ and functions of the form $f_i(S) = c_i(S) + \delta_{G_i}(S)$ where c_i is a linear function and δ_{G_i} is the cut function of a directed graph G_i .*

We start with the following known fact. For any non-negative submodular function, the collection of sets of zero value forms a *lattice*. (We recall that a boolean lattice is a family of sets $\mathcal{L} \subseteq 2^V$ closed under unions and intersections, i.e. $\forall A, B \in \mathcal{L}; A \cap B \in \mathcal{L}, A \cup B \in \mathcal{L}$.) This observation suggests that the question of checking for solutions of zero value is related to the following problem that we call **Lattice Matching**.

Lattice Matching. *Given k boolean lattices $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_k \subset 2^V$ (by a suitable compact representation), find k disjoint sets $S_1 \in \mathcal{L}_1, \dots, S_k \in \mathcal{L}_k$ such that $\bigcup_{i=1}^k S_i = V$.*

We show that this problem is **NP**-complete for $k \geq 3$, by a reduction from 1-in-3 SAT [153]. A technical issue is the question of representing a lattice compactly on the input. For this purpose we use a representation of lattices by directed graphs that goes back to Birkhoff [20]. In fact this construction also provides explicit submodular functions whose zeros are exactly the points of the respective lattice, and hence we prove Theorem 2.5.2.

A special case of a lattice is the collection of all unions of sets from some partition (collection of disjoint sets) \mathcal{P}_i . Thus the **Lattice Matching** problem has the following natural special case.

Partition Matching. *Given k collections of sets $\mathcal{P}_1, \dots, \mathcal{P}_k \subset 2^V$, where each \mathcal{P}_i is a partition of a subset of V (the sets in \mathcal{P}_i are disjoint), find disjoint sets $S_1, \dots, S_\ell \in \bigcup_{i=1}^k \mathcal{P}_i$ such that $\bigcup_{i=1}^\ell S_i = V$.*

Just like the problems above, the **Partition Matching** problem is in **P** for $k \leq 2$. We prove that this problem is **NP**-complete for $k = 5$. We leave it as an open question whether it is **NP**-complete for $k = 3$ and $k = 4$.

2.5.1 Representation of lattices by directed graphs

We recall the following well-known fact.

Lemma 2.5.3. *For every non-negative submodular function $f : 2^V \rightarrow \mathbb{R}_+$, the set $\mathcal{L} = \{S \subseteq V : f(S) = 0\}$ is a lattice, i.e., it is closed under unions and intersections.*

In the following, we describe how to encode a boolean lattice using a directed graph, and how this connects MSCA and Lattice Matching. We follow the construction of [78].

Definition 2.5.4. *Given a lattice $\mathcal{L} \subseteq 2^V$ such that $\emptyset \in \mathcal{L}$, let $V_{\mathcal{L}} = \bigcup \{S : S \in \mathcal{L}\}$. For each $v \in V_{\mathcal{L}}$, let $D(v) = \bigcap \{S : S \in \mathcal{L}, v \in S\}$. We define a directed graph $G_{\mathcal{L}} = (V_{\mathcal{L}}, A)$ where $A = \{(v, w) : v \in V_{\mathcal{L}}, w \in D(v), w \neq v\}$.*

The following lemma is implicit in [20, 78]. We include a simple proof for completeness.

Lemma 2.5.5. *For every lattice $\mathcal{L} \subseteq 2^V$ such that $\emptyset \in \mathcal{L}$, the directed graph $G_{\mathcal{L}}$ encodes the lattice in the sense that $S \in \mathcal{L}$ if and only if $S \subseteq V_{\mathcal{L}}$ and $G_{\mathcal{L}}$ has no arcs from S to $V_{\mathcal{L}} \setminus S$.*

Proof. If $S \in \mathcal{L}$, then $S \subseteq V_{\mathcal{L}}$ by the properties of the lattice. Also, for each $v \in S$, $D(v) = \bigcap \{S' \in \mathcal{L} : v \in S'\} \subseteq S$ and hence all arcs originating at v stay within S .

Conversely, if $S \subseteq V_{\mathcal{L}}$ and there are no arcs leaving S , we know that for each $v \in S$, $D(v) \subseteq S$. By the properties of lattices, $D(v) = \bigcap \{S' \in \mathcal{L}, v \in S'\} \in \mathcal{L}$. If $S \neq \emptyset$, we get that $S = \bigcup_{v \in S} D(v) \in \mathcal{L}$. If $S = \emptyset$, then $S \in \mathcal{L}$ by assumption. \square

Thus the directed graph $G_{\mathcal{L}}$ encodes the lattice \mathcal{L} in a compact way: its description has size $O(n^2)$, where $n = |V|$. Furthermore, we observe that this description provides a submodular function whose zeros are exactly the sets in \mathcal{L} .

Lemma 2.5.6. *For a lattice $\mathcal{L} \subseteq 2^V$ defined by the directed graph $G_{\mathcal{L}}$, the following function is submodular and its zeros are exactly the sets in \mathcal{L} :*

$$f_{\mathcal{L}}(S) = |S \setminus V_{\mathcal{L}}| + \delta_{G_{\mathcal{L}}}(S)$$

where $\delta_{G_{\mathcal{L}}}(S) = |\{(v, w) \in A(G_{\mathcal{L}}) : v \in S, w \notin S\}|$ is the directed cut function of $G_{\mathcal{L}}$.

Proof. By Lemma 2.5.5, $S \in \mathcal{L}$ if and only if $S \subseteq V_{\mathcal{L}}$ and there are no arcs from S to outside of S in $G_{\mathcal{L}}$, which occurs if and only if $f_{\mathcal{L}}(S) = 0$. \square

It follows that the **Lattice Matching** problem — where the lattices are given by the associated directed graphs — is equivalent to checking whether the **MSCA** instance in which the functions are $\{f_{\mathcal{L}_i} \mid i \in [k]\}$ has zero cost. To prove Theorem 2.5.2, it remains to prove the **NP**-completeness of **Lattice Matching** under this encoding.

2.5.2 NP-completeness of **Partition Matching** and **Lattice Matching**

In this section, we prove that the **Lattice Matching** problem is **NP**-complete for $k = 3$. First, as a warm-up, let us prove that its special case, the **Partition Matching** problem, is **NP**-complete for $k = 7$. We reduce from the following **NP**-complete problem [81].

3-bounded 3-set Packing. *Given a system of triples $\mathcal{T} \subseteq 2^V$ such that each element of V is contained in at most 3 triples, it is **NP**-complete to decide whether there exists a collection of disjoint triples covering V .*

Theorem 2.5.7. *Partition Matching is **NP**-complete for $k = 7$.*

Proof. Given an instance of 3-bounded 3-set Packing, we observe that each triple intersects at most 6 other triples (2 for each of its elements). Thus we can greedily color the triples with 7 colors in such a way that intersecting triples get different colors. We define \mathcal{P}_i to be the collection of all triples of color i . We obtain an instance of **Partition Matching** with $k = 7$, for which it is **NP**-complete to decide whether there exists a collection of disjoint triples covering V . \square

For lower values of k , we use more careful reductions from the **Monotone 1-in-3 SAT** problem [153].

Monotone 1-in-3 SAT. *Given a formula $\bigwedge_{i=1}^m (x_{i_1} \vee x_{i_2} \vee x_{i_3})$ (without negations), it is **NP**-complete to find a boolean assignment such that in each clause $(x_{i_1} \vee x_{i_2} \vee x_{i_3})$, exactly one variable is True and two variables are False.*

Theorem 2.5.8. *Partition Matching is **NP**-complete for $k = 5$.*

Proof. Given an instance of **Monotone 1-in-3 SAT**, we produce an instance of **Partition Matching** as follows. We define a ground set V consisting of

- An element v_j for each variable x_j .

- Two elements $x_j^i, \neg x_j^i$ for each occurrence of a variable x_j in clause i .

On this ground set, we define the following 5 collections of sets:

- \mathcal{P}_1 contains for each variable x_j a set $\{v_j\} \cup \{x_j^i : \forall \text{clause } i \text{ containing variable } x_j\}$.
- \mathcal{P}_2 contains for each variable x_j a set $\{v_j\} \cup \{\neg x_j^i : \forall \text{clause } i \text{ containing variable } x_j\}$.
- \mathcal{P}_3 contains for each clause $x_{i_1} \vee x_{i_2} \vee x_{i_3}$ a set $\{x_{i_1}^i, \neg x_{i_2}^i, \neg x_{i_3}^i\}$.
- \mathcal{P}_4 contains for each clause $x_{i_1} \vee x_{i_2} \vee x_{i_3}$ a set $\{\neg x_{i_1}^i, x_{i_2}^i, \neg x_{i_3}^i\}$.
- \mathcal{P}_5 contains for each clause $x_{i_1} \vee x_{i_2} \vee x_{i_3}$ a set $\{\neg x_{i_1}^i, \neg x_{i_2}^i, x_{i_3}^i\}$.

We call the sets in $\mathcal{P}_1 \cup \mathcal{P}_2$ *variable-assignment* sets, and the sets in $\mathcal{P}_3 \cup \mathcal{P}_4 \cup \mathcal{P}_5$ *clause-assignment* sets. Observe that in each collection \mathcal{P}_i , the sets are pairwise disjoint. I.e., we have an instance of **Partition Matching** with $k = 5$.

Now, if there is a boolean assignment such that exactly one variable in each clause is satisfied, we produce a solution of **Partition Matching** as follows. For each variable $x_j = \text{True}$, we choose the variable-assignment set containing v_j that is in \mathcal{P}_2 (i.e. containing all the elements $\neg x_j^i$). For each variable $x_j = \text{False}$, we choose the variable-assignment set containing v_j that is in \mathcal{P}_1 (i.e. containing all the elements x_j^i). Finally, for each clause $x_{i_1} \vee x_{i_2} \vee x_{i_3}$, we choose the set corresponding to its assignment, from either \mathcal{P}_3 , \mathcal{P}_4 or \mathcal{P}_5 . It is easy to verify that these sets are disjoint and cover the entire ground set V .

Conversely, let us assume that there is a collection of disjoint sets $\mathcal{F} \subset \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3 \cup \mathcal{P}_4 \cup \mathcal{P}_5$ that covers the ground set V . Since \mathcal{F} must cover the element v_j for each variable, it must contain a variable-assignment set in either \mathcal{P}_1 or \mathcal{P}_2 (no other sets contain v_j). This choice determines a boolean assignment: we set $x_j = \text{True}$ if v_j is covered by a set from \mathcal{P}_2 , and $x_j = \text{False}$ if v_j is covered by a set from \mathcal{P}_1 . Now, consider the 6 elements $x_{i_1}^i, \neg x_{i_1}^i, x_{i_2}^i, \neg x_{i_2}^i, x_{i_3}^i, \neg x_{i_3}^i$ for clause i . Exactly 3 of these elements are covered by sets from \mathcal{P}_1 and \mathcal{P}_2 , hence the remaining 3 elements must form a clause-assignment set in $\mathcal{P}_3 \cup \mathcal{P}_4 \cup \mathcal{P}_5$. These clause-assignment sets correspond to satisfying assignments and hence the formula is satisfied by the boolean assignment that we defined. \square

Theorem 2.5.9. *Lattice Matching is NP-complete for $k = 3$.*

Proof. We use the same reduction as in the proof of Theorem 2.5.8, but we combine the 5 partitions into 3 lattices. Specifically, using the notation from the proof above, we define

- $\mathcal{L}_1 = \text{cl}(\mathcal{P}_1 \cup \mathcal{P}_3)$
- $\mathcal{L}_2 = \text{cl}(\mathcal{P}_2)$
- $\mathcal{L}_3 = \text{cl}(\mathcal{P}_4 \cup \mathcal{P}_5)$

where $\text{cl}(\mathcal{P})$ means all the sets that can be generated from \mathcal{P} by taking unions and intersections. By construction, $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ are lattices that contain all the sets that were contained in $\mathcal{P}_1, \dots, \mathcal{P}_5$ (as well as some additional sets). In the case of a satisfiable formula, we can still choose disjoint sets covering V as above. Let S_i be the union of those of these sets that are contained in \mathcal{L}_i (i.e., S_i is also in \mathcal{L}_i), and $S_1 \cup S_2 \cup S_3 = V$ is a feasible solution of the **Lattice Matching** problem.

The potential issue with this construction is that we might have created a feasible solution of the **Lattice Matching** problem in case the formula is not satisfiable. Let us argue that this is not the case. In \mathcal{L}_1 , we get new sets obtained by intersecting sets in \mathcal{P}_1 and \mathcal{P}_3 : specifically, this is the singleton $\{x_{i_1}^i\}$ for each clause $x_{i_1} \vee x_{i_2} \vee x_{i_3}$ (and sets obtained by taking unions of these singletons with other sets in $\mathcal{P}_1 \cup \mathcal{P}_3$). In \mathcal{L}_2 , we obtain only the unions of sets in \mathcal{P}_2 (which are disjoint). In \mathcal{L}_3 , we obtain by intersection the singletons $\{\neg x_{i_1}^i\}$ for each clause $x_{i_1} \vee x_{i_2} \vee x_{i_3}$, and again sets obtained by unions with other sets in $\mathcal{P}_4 \cup \mathcal{P}_5$.

However, observe that for each variable, the element v_j is still contained only in the variable-assignment sets arising from \mathcal{P}_1 and \mathcal{P}_2 (and larger sets in $\mathcal{L}_1, \mathcal{L}_2$). As before, v_j must be covered by a variable-assignment set and this defines a boolean assignment. This also covers either all the elements x_j^i for variable x_j , or all the elements $\neg x_j^i$. Moreover, for each clause, the elements $x_{i_2}^2$ and $\neg x_{i_2}^2$ can be covered only by triples corresponding to satisfying assignments of the clause (singletons are present only for the first variable in each clause). Thus we still need to use a triple corresponding to a satisfying assignment for each clause, and this is possible only if a globally satisfying assignment exists. \square

2.6 The Lovász extension and solving **LE-Rel**

Let $f : \{0, 1\}^n \rightarrow \mathbb{R}$ be a function. The Lovász extension \hat{f} of f is the function $\hat{f} : [0, 1]^n \rightarrow \mathbb{R}$ defined as follows. Let \mathbf{x} be a vector in $[0, 1]^n$. We relabel the vertices as $1, 2, \dots, n$ so that $x_1 \leq x_2 \leq \dots \leq x_n$; for ease of notation, let $x_0 = 1$ and $x_{n+1} = 0$. Let $S_i = \{1, 2, \dots, i\}$. The value

	LE-Rel-Primal		LE-Rel-Dual
min	$\sum_{i=1}^k \sum_{S \subseteq V} \lambda(S, i) f_i(S)$		max $\sum_{i=1}^k \beta_i + \sum_{v \in V} \gamma_v$
	$\sum_{S: v \in S} \lambda(S, i) = x(v, i) \quad \forall v, i$		$\sum_{v \in S} \alpha(v, i) + \beta_i \leq f_i(S) \quad \forall S, i$
	$\sum_{S \subseteq V} \lambda(S, i) = 1 \quad \forall i$		$\gamma_v \leq \alpha(v, i) \quad \forall v, i$
	$\sum_{i=1}^k x(v, i) = 1 \quad \forall v$		
	$\lambda(S, i) \geq 0 \quad \forall S, i$		
	$x(v, i) \geq 0 \quad \forall v, i$		

Figure 2.4: An equivalent formulation of LE-Rel and its dual.

of \hat{f} at \mathbf{x} is equal to

$$\hat{f}(\mathbf{x}) = \sum_{i=0}^n (x_{i+1} - x_i) f(S_i)$$

It is straightforward to verify that $\sum_{i=0}^n (x_{i+1} - x_i) f(S_i) = \mathbb{E}_{\theta \in [0,1]} [f(\mathbf{x}^\theta)]$.

Another useful extension for a function f is its convex closure, which is defined as follows. For each set $S \subseteq V$, we let $\mathbf{1}_S$ denote the characteristic vector of S ; that is, the i -th coordinate of $\mathbf{1}_S$ is equal to 1 if i is in S and 0 otherwise. The convex closure f^- of f is the function $f^- : [0, 1]^n \rightarrow \mathbb{R}$ where $f^-(\mathbf{x}) = \min\{\sum_{S \subseteq V} \lambda_S f(S) : \sum_{S \subseteq V} \lambda_S \mathbf{1}_S = \mathbf{x}, \sum_{S \subseteq V} \lambda_S = 1, \lambda_S \geq 0\}$. The Lovász extension \hat{f} of f is equal to the convex closure f^- of f iff f is submodular; see for instance [68]. Using this result, we can show that LE-Rel can be solved in time that is polynomial in n and $\log(\max_{i, S \subseteq V} f_i(S))$ via the ellipsoid method.

Since \hat{f} is equal to f^- , the relaxation (LE-Rel-Primal) shown in Figure 2.4 is equivalent to LE-Rel.

Separation oracle for LE-Rel-Dual. Fix an assignment of values to the variables α, β, γ in **LE-Rel-Dual**. It is easy to check in polynomial time whether $\gamma_v \leq \alpha(v, i)$ for all v, i since there are only nk such constraints. Let $g_i(S) = \sum_{v \in S} \alpha(v, i) + \beta_i$. Note that g_i is a *modular* function and therefore $f_i - g_i$ is a submodular function. Using a polynomial time algorithm for submodular function minimization, for a given i , we can check whether $f_i(S) - g_i(S) \geq 0$ for all sets $S \subseteq V$.

Therefore we can solve **LE-Rel-Dual** in time that is polynomial in n and $\log(\max_{i, S \subseteq V} f_i(S))$

using the ellipsoid method. Using standard techniques, we can also construct an optimal solution for the primal; we omit the details here.

2.7 Concluding remarks

In this chapter, we introduce the **MSCA** problem as a model for submodular optimization problems that can be viewed as labeling or allocation problems. We show that several well-studied problems are captured by this model. We give a framework for designing approximation algorithms for these problems that is based on a mathematical programming relaxation for **MSCA** and a generic rounding strategy. We study several special cases of **MSCA** through the integrality gap of this relaxation. In the process, we obtain improved approximation algorithms for several special cases of **MSCA**.

An interesting direction for future work is related to the approximability of the **SubLabel** problem. In this chapter, we give an $O(\log n)$ approximation for the special case in which the separation cost function is symmetric and we show that this is best possible even if the assignment costs are linear, via a reduction from **Set Cover**. For the general **SubLabel** problem, we give an $O(k \log n)$ approximation; it would be very interesting to either improve this ratio or show that it is tight. The hypergraph labeling problem that we consider in Subsection 2.3.1 might be a good starting point. In this problem, the separation cost function is the hypergraph separation function, which is asymmetric. In Subsection 2.3.1, we give a Δ approximation for this special case, where Δ is the maximum size of a hyperedge. In subsequent work that is not included in this thesis, we improved the approximation guarantee to $O(\log n)$; this algorithm builds on an algorithm for the **Hypergraph Multiway Cut** problem that we give in [34]. An interesting question is whether $O(\log n)$ is best possible for this problem when the assignment costs are linear or whether there exists an $O(1)$ approximation. It might be possible to give a reduction from **Set Cover** or an example that shows that the integrality gap of **LE-Rel** is $\Omega(\ln n)$, but this does not seem to be straightforward.

Chapter 3

The Submodular Multiway Partition Problem

3.1 Introduction

In this chapter¹ we consider the **Submodular Multiway Partition (SubMP)** problem introduced in Chapter 2. Let $f : 2^V \rightarrow \mathbb{R}_+$ be a non-negative submodular set function over a finite ground set V and let $S = \{s_1, s_2, \dots, s_k\} \subseteq V$ be a set of k terminals. The **SubMP** problem asks for a partition A_1, \dots, A_k of V such that $s_i \in A_i$ for each i such that $1 \leq i \leq k$ and the sum $\sum_{i=1}^k f(A_i)$ is minimized. An important special case is when f is symmetric and we refer to it as **SymSubMP**.

Motivation and Related Problems: We are motivated to consider **SubMP** for two reasons. First, **SubMP** generalizes several problems that have been well-studied. One of the most well-known special cases is the **Graph Multiway Cut (GraphMC)** problem in which we are given an undirected edge-weighted graph $G = (V, E)$ and a set $S \subseteq V$ of terminals, and the objective is to remove a minimum weight set of edges to separate the terminals [62]. Although the objective is stated in terms of edge-removals, the problem can also be viewed as a special case of **SymSubMP** in which the function f is the cut function of G .

One obtains two interesting and related problems if one generalizes **GraphMC** to *hypergraphs*. Let $\mathcal{G} = (V, \mathcal{E})$ be an edge-weighted hypergraph. In the **Hypergraph Multiway Cut (HyperMC)** problem, the goal is to remove a minimum-weight set of hyperedges to disconnect the given set of terminals. The **Hypergraph Multiway Partition (HyperMP)** problem is the special case of **SymSubMP** where f is the cut function of \mathcal{G} : $f(A) = \sum_{e \in \delta(A)} w(e)$ where $w(e)$ is the weight of e and $\delta(A)$ is the set of all hyperedges that intersect A but are not contained in A . The distinction between **HyperMC** and **HyperMP** is that in **HyperMC** a hyperedge incurs a cost only once if the vertices in it are split across

¹This chapter is based on a paper with Chandra Chekuri [33] and a paper with Jan Vondrák and Yi Wu [71]. Copyrights to the conference versions of [33] and [71] are held by the Institute of Electrical and Electronics Engineers (IEEE) and the Society for Industrial and Applied Mathematics (SIAM), respectively.

terminals while in **HyperMP** the cost paid for a hyperedge is the number of non-trivial pieces it is partitioned into. Both problems have applications, in particular for circuit partitioning problems in VLSI design [3]. We wish to draw special attention to **HyperMC** since it is equivalent from an approximation point of view to the *node-weighted Graph Multiway Cut* (**Node-wt-MC**) problem where the nodes have weights and the goal is to remove a minimum-weight subset of nodes to disconnect a given set of terminals [85]. An important motivation to consider **SubMP** is that **HyperMC** can be cast as a special case of it [138]; the reduction is simple, yet interesting, and we stress that the resulting function f is *not* symmetric. From the above discussion it follows that **Node-wt-MC**, via **HyperMC**, can be viewed indirectly as a partition problem with an appropriate submodular function. We believe this is a useful observation. In fact, **SubMP** (and related generalizations) were introduced by Zhao, Nagamochi and Ibaraki [173] partly motivated by the applications to hypergraph cut and partition problems.

A second important motivation to consider **SubMP** and **SymSubMP** is the following question. To what extent do current algorithms and techniques for **GraphMC** and **Node-wt-MC** depend on the fact that the underlying structure is a graph or a hypergraph? Is it the case that the submodularity of the cut function is the key underlying phenomenon? For **GraphMC**, Dahlhaus *et al.* [62] gave a simple $2(1 - 1/k)$ -approximation via the isolating cut heuristic. Queyranne [143] showed that this same bound can be achieved for **SymSubMP** (see also [173]). For **GraphMC**, Calinescu, Karloff, and Rabani [60], in a breakthrough, obtained a $(1.5 - 1/k)$ approximation via an interesting geometric relaxation. Shortly after, Karger *et al.* [111] showed an improved integrality gap of 1.3438 for this relaxation. Recently, Buchbinder *et al.* [27] improved the gap even further to 1.32388. Once again it is natural to ask if this geometric relaxation is specific to graphs and whether corresponding results exist for **SymSubMP**. Further, the current best approximation for **SubMP** is $(k - 1)$ [173] and is obtained via a simple greedy splitting algorithm. **SubMP** generalizes **Node-wt-MC** and the latter has a constant factor approximation [85] but it is a non-trivial LP relaxation based algorithm. Therefore it is reasonable to expect that one needs a mathematical programming relaxation to obtain a constant factor approximation for **SubMP**. In Chapter 2, we gave a convex programming relaxation **LE-Rel** for the **MSCA** problem. As we have already seen, **SubMP** is a special case of **MSCA**. The **LE-Rel** relaxation is the first mathematical programming relaxation for **SubMP** and, as

we show in this chapter, it leads to improved approximation guarantees for SubMP and SymSubMP. For instances of SubMP, LE-Rel is equivalent to the following relaxation.

SubMP-Rel			
min	$\sum_{i=1}^k \hat{f}(\mathbf{x}_i)$		
	$\sum_{i=1}^k x(v, i)$	=	1 $\quad \forall v$
	$x(s_i, i)$	=	1 $\quad \forall i$
	$x(v, i)$	\geq	0 $\quad \forall v, i$

As we have seen in Chapter 2, the relaxation SubMP-Rel can be solved *exactly* in polynomial time. We give algorithms to round an optimum fractional solution to SubMP-Rel and obtain the following two results which also establish corresponding upper bounds on the integrality gap of SubMP-Rel.

Theorem 3.1.1. *There is a $(1.5 - 1/k)$ -approximation for SymSubMP.*

Theorem 3.1.2. *There is a $2(1 - 1/k)$ -approximation for SubMP.*

We give the proofs of Theorem 3.1.1 and Theorem 3.1.2 in Section 3.2 and Section 3.3 respectively.

Remark 3.1.3. Related to SubMP and SymSubMP are k -way partition problems k-way Sub-MP and k-way Sym-Sub-MP where no terminals are specified but the goal is to partition V into k non-empty sets A_1, \dots, A_k to minimize $\sum_{i=1}^k f(A_i)$. When k is part of the input these problems are **NP**-hard. For a fixed $k \geq 4$, it is open whether k-way Sym-Sub-MP is polynomial-time solvable or it is **NP**-hard. When k is a constant, one can reduce k-way Sub-MP to SubMP by guessing k terminals and this leads to a $2(1 - 1/k)$ -approximation via Theorem 3.1.2, improving the previously known ratio of $(k + 1 - 2\sqrt{k-1})$ [138].

3.1.1 Overview of rounding and the main technical result

Let \mathbf{x} be a fractional allocation and $\sum_i \hat{f}(\mathbf{x}_i)$ the corresponding objective function value. How do we round \mathbf{x} to an integral allocation while approximately preserving the convex objective function? The simple insight that we used in Chapter 2 is to simply follow the definition of the Lovász function and do θ -rounding: pick a (random) threshold $\theta \in [0, 1]$ and set $x(v_j, i) = 1$ if and only if

$x(v_j, i) \geq \theta$. Let \mathbf{x}^θ be the resulting integer vector. If we pick θ uniformly at random in $[0, 1]$ then the expected cost of $\sum_i \mathbb{E}[f(\mathbf{x}_i^\theta)] = \sum_i \hat{f}(\mathbf{x}_i)$. However, the difficulty is that \mathbf{x}^θ may not correspond to a feasible allocation. Let $A(i, \theta)$ be the support of \mathbf{x}_i^θ , that is, the set of vertices assigned to s_i for a given θ . The reason that \mathbf{x}^θ may not be a feasible allocation is two-fold. First, a vertex v may be assigned to multiple terminals, that is, the sets $A(i, \theta)$ for $i = 1, \dots, k$ may not be disjoint. Second, the vertices $U(\theta) = V - \cup_{i=1}^k A(i, \theta)$ are unallocated. We let $A(\theta) = \cup_{i=1}^k A(i, \theta)$ be the allocated set.

Our main insight is that the expected cost of the unallocated set, that is $f(U(\theta))$, can be upper bounded effectively. We can then assign the set $U(\theta)$ to an arbitrary terminal and use sub-additivity of f (since it is submodular and non-negative). Before we formalize this, we discuss how to overcome the overlap in the sets $A(i, \theta)$. If f is symmetric then it is also posi-modular and one can do a simple uncrossing of the sets to make them disjoint without increasing the cost. If f is not symmetric we cannot resort to this trick. In this case we ensure that the sets $A(i, \theta)$ are disjoint by picking θ uniformly in $(1/2, 1]$ rather than $[0, 1]$ (we call this half-rounding); the sets $A(i, \theta)$ are disjoint because for any v there can be at most one i such that $x(v, i) > 1/2$. Now the unallocated set and the expected cost of the initial allocation are some what more complex. We analyze both these scenarios using the following theorem which is our main result. The theorem below has a parameter $\delta \in [1/2, 1]$ and this corresponds to rounding where we pick θ uniformly from the interval $(1 - \delta, 1]$.

Theorem 3.1.4. *Let $f \geq 0$ be a submodular function such that $f(\emptyset) = 0$. Let \mathbf{x} be a feasible solution to SubMP-Rel. For $\theta \in [0, 1]$ let $A(i, \theta) = \{v \mid x(v, i) \geq \theta\}$, $A(\theta) = \cup_{i=1}^k A(i, \theta)$ and $U(\theta) = V - A(\theta)$. For any $\delta \in [1/2, 1]$ the following holds:*

$$\sum_{i=1}^k \int_0^\delta f(A(i, \theta)) d\theta \geq \int_0^\delta f(A(\theta)) d\theta + \int_0^1 f(U(\theta)) d\theta.$$

By setting $\delta = 1$, we obtain the following corollary.

Corollary 3.1.5.

$$\sum_{i=1}^k \hat{f}(\mathbf{x}_i) = \sum_{i=1}^k \int_0^1 f(A(i, \theta)) d\theta \geq \int_0^1 f(A(\theta)) d\theta + \int_0^1 f(U(\theta)) d\theta.$$

Theorem 3.1.4 gives a unified analysis of our algorithms for **SymSubMP** and **SubMP**. More precisely, we get Theorem 3.1.1 and Theorem 3.1.2 as rather simple corollaries. Corollary 3.1.5 is sufficient to show that the **SymSubMP** algorithm achieves a 1.5-approximation and that the **SubMP** algorithm achieves a 4-approximation. In order to show that the **SubMP** algorithm achieves a 2-approximation, we need the stronger statement of Theorem 3.1.4. We further improve the approximation guarantees for **SymSubMP** and **SubMP** to $(1.5 - 1/k)$ and $2(1 - 1/k)$ (respectively) by modifying our algorithms slightly; although the modifications are very simple, the analyses are much more involved.

3.1.2 Discussion and other related work

As we already noted, a classical special case of **SymSubMP** (and thus **SubMP**) is the **GraphMC** problem. Calinescu, Karloof, and Rabani [60] obtained a $(1.5 - 1/k)$ approximation for **GraphMC** via a very interesting LP relaxation for the problem. Geometry plays a key role in the formulation, rounding and analysis of the relaxation of [60]. The subsequent work of Karger *et al.* exploits the geometric aspects further to obtain an improvement in the ratio to 1.3438. If one views **GraphMC** as a special case of **SymSubMP** then the function f under consideration is the cut function. The cut function f can be decomposed into several simple submodular functions, corresponding to the edges, each of which depends only on two vertices. This allows one to focus on the probability that an edge is cut in the rounding process. Our analysis differs substantially in that we no longer have a local handle on f and hence the need for Theorem 3.1.4. It is interesting that the integrality gap of **SubMP-Rel** is at most $(1.5 - \frac{1}{k})$ for any symmetric function f , matching the bound achieved by [60] for **GraphMC**. Our rounding differs from that in [60]; both do θ -rounding but our algorithm uncrosses the sets $A(i, \theta)$ to make them disjoint while the rounding of [60] does it by picking a random permutation. One can understand the random permutation as an oblivious uncrossing operation that is particularly suited for submodular functions that depend on only two variables (in this case the edges); it is unclear whether this is suitable for arbitrary symmetric functions.

As we remarked, **SymSubMP** and **SubMP** were considered in several papers [138, 143, 173] with **HyperMC** and **k-way Hypergraph Cut** as interesting applications for **SubMP**. These papers primarily relied on greedy methods. It was noted in [138] that **HyperMC** and **Node-wt-MC** are essentially equivalent problems. Garg, Vazirani, and Yannakakis [85] gave a $2(1 - 1/k)$ -approximation for

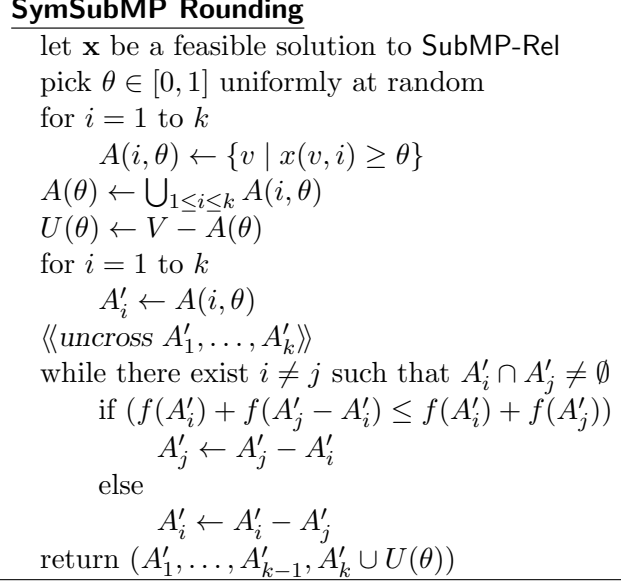


Figure 3.1: Algorithm for SymSubMP.

Node-wt-MC [85] via a natural distance based LP relaxation; we note that this result is non-trivial and relies on proving the existence of a half-integral optimum fractional solution. SubMP-Rel gives a new and strictly stronger relaxation for Node-wt-MC (via the connection to HyperMC). The previous best approximation for SubMP was $(k - 1)$ [173]. As we already remarked, obtaining a constant factor approximation for SubMP without a mathematical programming relaxation like SubMP-Rel is difficult given the lack of combinatorial algorithms for special cases like Node-wt-MC.

3.2 Approximation algorithm and integrality gap for SymSubMP

In this section, we describe our rounding algorithm for SymSubMP. As a warm-up, we give an algorithm that achieves a 1.5 approximation; this algorithm is straightforward to analyze using Theorem 3.1.4 and it is given in Figure 3.1. We then show how to modify the algorithm in order to achieve an improved $(1.5 - 1/k)$ approximation.

3.2.1 A 1.5 approximation algorithm

In this section, we describe the algorithm for SymSubMP. The algorithm rounds a fractional solution \mathbf{x} to SubMP-Rel using threshold rounding, and it is given in Figure 3.1.

The algorithm runs in polynomial time. Additionally, one can easily derandomize the algorithm as follows. The only randomness is in the choice of θ . As θ ranges in the interval $[0, 1]$, the collection of sets $\{A(i, \theta) \mid 1 \leq i \leq k\}$ changes only when θ crosses some $x(v_j, i)$ value. Thus there are at most nk such distinct values. We can try each of them as a choice for θ and pick the least cost partition obtained among all the choices.

In the remainder of this section, we show that SymSubMP Rounding achieves a 1.5 approximation for SymSubMP. Let $\text{OPT}_{\text{frac}} = \sum_{i=1}^k \hat{f}(\mathbf{x}_i)$ (the notation is with the implicit understanding that \mathbf{x} is an optimal fractional solution). Since f is symmetric, it follows from Theorem 3.1.4 that the expected cost of the unallocated set is at most $\text{OPT}_{\text{frac}}/2$.

Lemma 3.2.1. *If f is a symmetric submodular function,*

$$\mathbb{E}_{\theta \in [0,1]}[f(U(\theta))] \leq \frac{1}{2} \text{OPT}_{\text{frac}}.$$

Proof. By setting $\delta = 1$ in Theorem 3.1.4, we get

$$\text{OPT}_{\text{frac}} \geq \int_0^1 f(V - U(\theta)) d\theta + \int_0^1 f(U(\theta)) d\theta.$$

Since f is symmetric, $f(V - U(\theta)) = f(U(\theta))$ for all θ and hence,

$$\text{OPT}_{\text{frac}} \geq 2 \int_0^1 f(U(\theta)) d\theta = 2 \mathbb{E}_{\theta \in [0,1]}[f(U(\theta))].$$

□

The random partition constructed by the algorithm is $(A'_1, \dots, A'_{k-1}, A'_k \cup U(\theta))$. A non-negative submodular function is sub-additive and therefore $f(A'_k \cup U(\theta)) \leq f(A'_k) + f(U(\theta))$. The expected cost of the partition is

$$\begin{aligned} \sum_{i=1}^{k-1} \mathbb{E}[f(A'_i)] + \mathbb{E}[f(A'_k \cup U(\theta))] &\leq \sum_{i=1}^k \mathbb{E}[f(A'_i)] + \mathbb{E}[f(U(\theta))] \\ &\leq \sum_{i=1}^k \mathbb{E}[f(A(i, \theta))] + \mathbb{E}[f(U(\theta))] \quad (\text{Using Lemma 2.3.8}) \\ &\leq \text{OPT}_{\text{frac}} + \frac{1}{2} \text{OPT}_{\text{frac}} \quad (\text{Using Lemma 3.2.1}) \\ &= 1.5 \text{OPT}_{\text{frac}}. \end{aligned}$$

Improved SymSubMP Rounding

```

let  $\mathbf{x}$  be a feasible solution to SubMP-Rel
relabel the terminals so that  $k = \arg \max_i \hat{f}(\mathbf{x}_i)$ 
pick  $\theta \in [0, 1]$  uniformly at random
for  $i = 1$  to  $k - 1$ 
     $A(i, \theta) \leftarrow \{v \mid x(v, i) \geq \theta\}$ 
 $A(\theta) \leftarrow \bigcup_{1 \leq i \leq k-1} A(i, \theta)$ 
 $U(\theta) \leftarrow V - A(\theta)$ 
for  $i = 1$  to  $k - 1$ 
     $A'_i \leftarrow A(i, \theta)$ 
 $\langle\langle \text{uncross } A'_1, \dots, A'_{k-1} \rangle\rangle$ 
while there exist  $i \neq j$  such that  $A'_i \cap A'_j \neq \emptyset$ 
    if  $(f(A'_i) + f(A'_j - A'_i)) \leq f(A'_i) + f(A'_j)$ 
         $A'_j \leftarrow A'_j - A'_i$ 
    else
         $A'_i \leftarrow A'_i - A'_j$ 
return  $(A'_1, \dots, A'_{k-1}, U(\theta))$ 

```

Figure 3.2: Improved algorithm for SymSubMP.

3.2.2 A $(1.5 - 1/k)$ approximation algorithm

In this section, we modify the previous algorithm to achieve an improved $(1.5 - 1/k)$ approximation. The rounding algorithm is given in Figure 3.2.

We relabel the terminals so that $k = \arg \max_i \hat{f}(\mathbf{x}_i)$. The algorithm performs θ -rounding with respect to the first $k - 1$ terminals in order to get the sets $A(i, \theta)$ for each $i \neq k$, and we let $U(\theta) = V - \bigcup_{1 \leq i \leq k-1} A(i, \theta)$. We uncross the sets $\{A(i, \theta) \mid 1 \leq i < k\}$ to get $k - 1$ disjoint sets A'_i , and we return $(A'_1, \dots, A'_{k-1}, U(\theta))$.

In Section 3.5, we prove a variant of Theorem 3.1.4 that shows that the expected cost of $U(\theta)$ is at most $\text{OPT}_{\text{frac}}/2$, even when $U(\theta)$ is the set of all vertices that are unallocated when we perform θ -rounding with respect to only the first k' terminals, for any $k' \leq k$.

Lemma 3.2.2. *If f is a symmetric submodular function,*

$$\mathbb{E}_{\theta \in [0,1]}[f(U(\theta))] \leq \frac{1}{2} \text{OPT}_{\text{frac}}.$$

Proof. By Theorem 3.5.1, we have

$$\text{OPT}_{\text{frac}} \geq \int_0^1 f(V - U(\theta)) d\theta + \int_0^1 f(U(\theta)) d\theta.$$

SubMP-Half-Rounding

let \mathbf{x} be a feasible solution to SubMP-Rel
 pick $\theta \in (1/2, 1]$ uniformly at random
 for $i = 1$ to k
 $A(i, \theta) \leftarrow \{v \mid x(v, i) \geq \theta\}$
 $A(\theta) \leftarrow \bigcup_{1 \leq i \leq k} A(i, \theta)$
 $U(\theta) \leftarrow V - A(\theta)$
 allocate each $A(i, \theta)$ to terminal i
 allocate $U(\theta)$ to a terminal i' chosen uniformly at random

Figure 3.3: Algorithm for SubMP.

Since f is symmetric, $f(V - U(\theta)) = f(U(\theta))$ for all θ and hence,

$$\text{OPT}_{\text{frac}} \geq 2 \int_0^1 f(U(\theta)) d\theta = 2 \mathbb{E}_{\theta \in [0,1]} [f(U(\theta))].$$

□

The random partition constructed by the algorithm is $(A'_1, \dots, A'_{k-1}, U(\theta))$. The expected cost of the partition is

$$\begin{aligned}
 \sum_{i=1}^{k-1} \mathbb{E}[f(A'_i)] + \mathbb{E}[f(U(\theta))] &\leq \sum_{i=1}^{k-1} \mathbb{E}[f(A(i, \theta)) + f(U(\theta))] \quad (\text{Using Lemma 2.3.8}) \\
 &\leq \text{OPT}_{\text{frac}} - \hat{f}(\mathbf{x}_k) + \frac{1}{2} \text{OPT}_{\text{frac}} \quad (\text{Using Lemma 3.2.1}) \\
 &\leq \left(1 - \frac{1}{k}\right) \text{OPT}_{\text{frac}} + \frac{1}{2} \text{OPT}_{\text{frac}} \quad \left(\hat{f}(\mathbf{x}_k) \geq \text{OPT}_{\text{frac}}/k\right) \\
 &= \left(1.5 - \frac{1}{k}\right) \text{OPT}_{\text{frac}}.
 \end{aligned}$$

3.3 Approximation algorithm and integrality gap for SubMP

In this section we consider SubMP when f is an arbitrary non-negative submodular function. As before, the algorithm is based on threshold rounding. Since f is not symmetric, we cannot uncross the sets and make them disjoint. Instead, we choose the threshold θ uniformly at random from the set $(1/2, 1]$ instead of $[0, 1]$. Since θ is greater than $1/2$, the resulting sets will be disjoint. We give the rounding algorithm in Figure 3.3. The algorithm can be derandomized in the same fashion as the one for symmetric functions.

In this section, we prove that the algorithm achieves a $2(1 - 1/k)$ approximation for **SubMP**. We first give a much easier analysis that gives an upper bound of 2 on the approximation ratio. This weaker analysis holds even if we allocate the unallocated set $U(\theta)$ to an arbitrary terminal instead of a random terminal. However, allocating $U(\theta)$ to a random terminal allows us to improve the approximation ratio, but the analysis is much more involved.

3.3.1 A weaker analysis

In the following, we show that **SubMP-Half-Rounding** achieves a 2-approximation for **SubMP-Half-Rounding**. As before, let $\text{OPT}_{\text{frac}} = \sum_{i=1}^k \hat{f}(\mathbf{x}_i)$. Since f is sub-additive, for any terminal i' , we have

$$\begin{aligned} & \mathbb{E}_{\theta \in (1/2, 1]} \left[\sum_{i \neq i'} f(A(i, \theta)) + f(A(i', \theta) \cup U(\theta)) \right] \\ & \leq \mathbb{E}_{\theta \in (1/2, 1]} \left[\sum_{i=1}^k f(A(i, \theta)) + f(U(\theta)) \right] \quad (f \text{ is sub-additive}) \\ & = 2 \left(\sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta)) d\theta + \int_{1/2}^1 f(U(\theta)) d\theta \right) \\ & = 2 \left(\text{OPT}_{\text{frac}} - \sum_{i=1}^k \int_0^{1/2} f(A(i, \theta)) d\theta + \int_{1/2}^1 f(U(\theta)) d\theta \right). \end{aligned}$$

To show that the expected cost is at most $2\text{OPT}_{\text{frac}}$ it suffices to show that $\sum_{i=1}^k \int_0^{1/2} f(A(i, \theta)) d\theta \geq \int_{1/2}^1 f(U(\theta)) d\theta$. Setting $\delta = 1/2$ in Theorem 3.1.4 we obtain

$$\begin{aligned} \sum_{i=1}^k \int_0^{1/2} f(A(i, \theta)) d\theta & \geq \int_0^{1/2} f(V - U(\theta)) d\theta + \int_0^1 f(U(\theta)) d\theta \\ & \geq \int_{1/2}^1 f(U(\theta)) d\theta \quad (f \text{ is non-negative}). \end{aligned}$$

Thus **SubMP-Half-Rounding** achieves a randomized 2-approximation for **SubMP**.

3.3.2 An improved analysis

Theorem 3.3.1. *The rounding algorithm described in Figure 3.3 constructs a feasible solution of expected value at most $(2 - \frac{2}{k}) \sum_{i=1}^k \hat{f}(\mathbf{x}_i)$.*

Theorem 3.1.2 follows from Theorem 3.3.1. In the following, we assume that $f(\emptyset) = 0$. This is without loss of generality, as the value of the empty set can be decreased without violating submodularity and this does not affect the problem (since terminals are always assigned to themselves).

We start by defining several sets, parameterized by θ , that will be important in the analysis.

- $A(i, \theta) = \{j : x_{ij} > \theta\}$.
- $A(\theta) = \bigcup_{i=1}^k A(i, \theta) = \{j : \max_i x_{ij} > \theta\}$.
- $U(\theta) = V \setminus A(\theta) = \{j : \max_i x_{ij} \leq \theta\}$.
- $B(\theta) = U(1 - \theta) = \{j : 1 - \max_i x_{ij} \geq \theta\}$.

We can express the LP cost and the cost of the rounded solution in terms of these sets as follows. The following lemma follows immediately from the definition of the Lovász extension.

Lemma 3.3.2. *The cost of the LP solution is*

$$\text{OPT}_{\text{frac}} = \sum_{i=1}^k \int_0^1 f(A(i, \theta)) d\theta.$$

The next lemma gives an expression for the expected value achieved by the algorithm.

Lemma 3.3.3. *The expected cost of the rounded solution is*

$$\text{ALG} = \left(2 - \frac{2}{k}\right) \sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta)) d\theta + \frac{2}{k} \sum_{i=1}^k \int_0^{1/2} f(A(i, \theta) \cup B(\theta)) d\theta.$$

Proof. The set allocated to terminal i is $A(i, \theta)$ with probability $1 - 1/k$, and $A(i, \theta) \cup U(\theta)$ with probability $1/k$. We are choosing θ uniformly between $\frac{1}{2}$ and 1. This gives the expression

$$\text{ALG} = \left(2 - \frac{2}{k}\right) \sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta)) d\theta + \frac{2}{k} \sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta) \cup U(\theta)) d\theta.$$

We claim that for $\theta \in [\frac{1}{2}, 1]$, $A(i, \theta) \cup U(\theta)$ can be written equivalently as $A(i, 1 - \theta) \cup B(1 - \theta)$.

We consider three cases for each element j :

- If $x_{ij} > \frac{1}{2}$, then $j \in A(i, \theta) \cup U(\theta)$ for every $\theta \in [\frac{1}{2}, 1]$, because $x_{i'j} < \frac{1}{2}$ for every other $i' \neq i$ and hence j cannot be allocated to any other terminal. Similarly, $j \in A(i, 1 - \theta) \cup B(1 - \theta)$ for every $\theta \in [\frac{1}{2}, 1]$, because $1 - \theta \leq \frac{1}{2}$ and so $j \in A(i, 1 - \theta)$.
- If $x_{ij} \leq \frac{1}{2}$ and $x_{ij} = \max_{i'} x_{i'j}$, then again $j \in A(i, \theta) \cup U(\theta)$ for every $\theta \in [\frac{1}{2}, 1]$, because j is always in the unallocated set $U(\theta)$. Also, $j \in A(i, 1 - \theta) \cup B(1 - \theta)$, because $B(1 - \theta) = U(\theta)$.
- If $x_{ij} \leq \frac{1}{2}$ and $x_{ij} < \max_{i'} x_{i'j}$, then $j \in A(i, \theta) \cup U(\theta)$ if and only if $j \in U(\theta) = B(1 - \theta)$. Also, we have $x_{ij} = 1 - \sum_{i' \neq i} x_{i'j} \leq 1 - \max_{i'} x_{i'j}$, and therefore $j \in A(i, 1 - \theta) \cup B(1 - \theta)$ if and only if $j \in B(1 - \theta)$.

To summarize, for every $\theta \in [\frac{1}{2}, 1]$, $j \in A(i, \theta) \cup U(\theta)$ if and only if $j \in A(i, 1 - \theta) \cup B(1 - \theta)$.

Therefore, the total expected cost can be written as

$$\begin{aligned}
\text{ALG} &= \left(2 - \frac{2}{k}\right) \sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta)) d\theta + \frac{2}{k} \sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta) \cup U(\theta)) d\theta \\
&= \left(2 - \frac{2}{k}\right) \sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta)) d\theta + \frac{2}{k} \sum_{i=1}^k \int_{1/2}^1 f(A(i, 1 - \theta) \cup B(1 - \theta)) d\theta \\
&= \left(2 - \frac{2}{k}\right) \sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta)) d\theta + \frac{2}{k} \sum_{i=1}^k \int_0^{1/2} f(A(i, \theta) \cup B(\theta)) d\theta.
\end{aligned}$$

□

In the rest of the analysis, we prove several inequalities that relate the OPT_{frac} cost to the ALG cost. Note that the integrals $\int_{1/2}^1 f(A(i, \theta)) d\theta$ appear in both OPT_{frac} and ALG. The interesting part is how to relate $\int_0^{1/2} f(A(i, \theta)) d\theta$ to $\int_0^{1/2} f(A(i, \theta) \cup B(\theta)) d\theta$.

The following inequality is a corollary of our main theorem, Theorem 3.1.4.

Lemma 3.3.4.

$$\sum_{i=1}^k \int_0^{1/2} f(A(i, \theta)) d\theta \geq \int_0^{1/2} f(B(\theta)) d\theta.$$

Proof. Considering Theorem 3.1.4, we simply note that $U(\theta) = B(1 - \theta)$. We discard the contribution of $f(A(\theta))$ and keep only one half of the integral involving $B(1 - \theta)$. □

We combine this bound with the following lemma.

Lemma 3.3.5.

$$\sum_{i=1}^k \int_0^{1/2} f(A(i, \theta)) d\theta \geq \sum_{i=1}^k \int_0^{1/2} f(A(i, \theta) \cup B(\theta)) d\theta - (k-2) \int_0^{1/2} f(B(\theta)) d\theta.$$

Proof. In order to simplify the notation, in this proof we let $A_i = A(i, \theta)$, $B = B(\theta)$, and $U = U(\theta)$.

By submodularity, we have $f(A_i) + f(B) \geq f(A_i \cup B) + f(A_i \cap B)$. Therefore,

$$\begin{aligned} \sum_{i=1}^k f(A_i) &\geq \sum_{i=1}^k (f(A_i \cup B) + f(A_i \cap B) - f(B)) \\ &= \sum_{i=1}^k f(A_i \cup B) + \sum_{i=1}^k f(A_i \cap B) - k \cdot f(B) \end{aligned}$$

This would already prove the lemma with k instead of $k-2$; however, we use $\sum_{i=1}^k f(A_i \cap B)$ to save the additional terms. We apply a sequence of inequalities using submodularity, starting with $f(A_1 \cap B) + f(A_2 \cap B) \geq f(A_1 \cap A_2 \cap B) + f((A_1 \cup A_2) \cap B)$, then $f((A_1 \cup A_2) \cap B) + f(A_3 \cap B) \geq f((A_1 \cup A_2) \cap A_3 \cap B) + f((A_1 \cup A_2 \cup A_3) \cap B)$, etc. until we obtain

$$\sum_{i=1}^k f(A_i \cap B) \geq \sum_{i=1}^{k-1} f((A_1 \cup \dots \cup A_i) \cap A_{i+1} \cap B) + f((A_1 \cup \dots \cup A_k) \cap B).$$

The last term is equal to $f(A \cap B)$. Moreover, we observe that for every element v , at most one variable $x(v, i)$ can be larger than $1 - \max_{i'} x(v, i')$ (because otherwise the two variables would sum up to more than 1). Therefore for every i , $(A_1 \cup \dots \cup A_i) \cap A_{i+1} \subseteq B$. So we get

$$\sum_{i=1}^k f(A_i \cap B) \geq \sum_{i=1}^{k-1} f((A_1 \cup \dots \cup A_i) \cap A_{i+1}) + f(A \cap B).$$

Integrating from 0 to $1/2$, we get

$$\sum_{i=1}^k \int_0^{1/2} f(A_i \cap B) d\theta \geq \sum_{i=1}^{k-1} \int_0^{1/2} f((A_1 \cup \dots \cup A_i) \cap A_{i+1}) d\theta + \int_0^{1/2} f(A \cap B) d\theta.$$

By Lemma 3.4.1 (recalling that $A_i = A(i, \theta)$), we obtain

$$\sum_{i=1}^k \int_0^{1/2} f(A_i \cap B) d\theta \geq \int_0^1 f(U) d\theta + \int_0^{1/2} f(A \cap B) d\theta.$$

Using $B(\theta) = U(1 - \theta)$, submodularity, and the fact that U is the complement of A , we obtain

$$\begin{aligned} \sum_{i=1}^k \int_0^{1/2} f(A_i \cap B) d\theta &\geq \int_0^{1/2} f(B) d\theta + \int_0^{1/2} f(U) d\theta + \int_0^{1/2} f(A \cap B) d\theta \\ &\geq \int_0^{1/2} f(B) d\theta + \int_0^{1/2} f(U \cup (A \cap B)) d\theta \\ &= \int_0^{1/2} f(B) d\theta + \int_0^{1/2} f(U \cup B) d\theta \end{aligned}$$

Finally, for $\theta \in [0, \frac{1}{2}]$, we claim that $U \cup B = B$. This is because if $\max_i x_{ij} > \frac{1}{2}$, then $j \notin U$, and hence the membership on both sides depends only on $j \in B$. If $\max_i x_{ij} \leq \frac{1}{2}$, then $j \in B$ and hence also $j \in U \cup B$. We conclude that

$$\sum_{i=1}^k \int_0^{1/2} f(A_i \cap B) d\theta \geq 2 \int_0^{1/2} f(B) d\theta$$

and

$$\begin{aligned} \sum_{i=1}^k \int_0^{1/2} f(A_i) d\theta &\geq \sum_{i=1}^k \int_0^{1/2} (f(A_i \cup B) + f(A_i \cap B) - f(B)) d\theta \\ &\geq \sum_{i=1}^k \int_0^{1/2} f(A_i \cup B) d\theta - (k-2) \int_0^{1/2} f(B) d\theta \end{aligned}$$

which finishes the proof. \square

A combination of Lemma 3.3.4 and Lemma 3.3.5 relates $\sum_{i=1}^k \int_0^{1/2} f(A(i, \theta)) d\theta$ to $\sum_{i=1}^k \int_0^{1/2} f(A(i, \theta) \cup B(\theta)) d\theta$, and finishes the analysis.

Proof of Theorem 3.3.1: Add up $\frac{k-2}{k-1} \times$ Lemma 3.3.4 + $\frac{1}{k-1} \times$ Lemma 3.3.5:

$$\sum_{i=1}^k \int_0^{1/2} f(A(i, \theta)) d\theta \geq \frac{1}{k-1} \sum_{i=1}^k \int_0^{1/2} f(A(i, \theta) \cup B(\theta)) d\theta.$$

Adding $\sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta)) d\theta$ to both sides gives us that

$$\sum_{i=1}^k \int_0^1 f(A(i, \theta)) d\theta \geq \sum_{i=1}^k \int_{1/2}^1 f(A(i, \theta)) d\theta + \frac{1}{k-1} \cdot \sum_{i=1}^k \int_0^{1/2} f(A(i, \theta) \cup B(\theta)) d\theta.$$

The left-hand side is equal to OPT_{frac} , while the right-hand side is equal to $\text{ALG}/(2 - 2/k)$ (see Lemma 3.3.3). \square

3.4 Proof of the main theorem

In this section, we prove our main theorem.

Theorem 3.1.4. Let $f \geq 0$ be a submodular function such that $f(\emptyset) = 0$. Let \mathbf{x} be a feasible solution to SubMP-Rel. For $\theta \in [0, 1]$ let $A(i, \theta) = \{v \mid x(v, i) \geq \theta\}$, $A(\theta) = \cup_{i=1}^k A(i, \theta)$ and $U(\theta) = V - A(\theta)$. For any $\delta \in [1/2, 1]$ the following holds:

$$\sum_{i=1}^k \int_0^\delta f(A(i, \theta)) d\theta \geq \int_0^\delta f(A(\theta)) d\theta + \int_0^1 f(U(\theta)) d\theta.$$

The theorem is a straightforward corollary of the following inequality.

Lemma 3.4.1. For any $\delta \in [1/2, 1]$,

$$\sum_{i=1}^{k-1} \int_0^\delta f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) d\theta \geq \int_0^1 f(U(\theta)) d\theta.$$

Proof. First consider $\delta = 1$. We can view the value $\int_0^1 f(A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta) d\theta$ as the Lovász extension evaluated on a suitable vector², $\mathbf{y}_i = (\mathbf{x}_1 \vee \dots \vee \mathbf{x}_i) \wedge \mathbf{x}_{i+1}$. In other words, $y(v, i) = \min \{\max \{x(v, 1), \dots, x(v, i)\}, x(v, i+1)\}$ for each $v \in V$. Note that v is in $(A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)$ if and only if $y(v, i) \geq \theta$. Therefore

$$\int_0^1 f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) d\theta = \hat{f}(\mathbf{y}_i).$$

We can also view the value $\int_0^1 f(U(\theta)) d\theta$ as the Lovász extension evaluated on the vector $\mathbf{u} = \sum_{i=1}^{k-1} \mathbf{y}_i$.

²If \mathbf{a} and \mathbf{b} are two d -dimensional vectors, $\mathbf{a} \vee \mathbf{b}$ is the d -dimensional vector whose i -th entry is $\max \{a_i, b_i\}$, and $\mathbf{a} \wedge \mathbf{b}$ is the d -dimensional vector whose i -th entry is $\min \{a_i, b_i\}$.

Note that $\mathbf{u} = \mathbf{1} - (\mathbf{x}_1 \vee \cdots \vee \mathbf{x}_k)$; this holds because $\sum_{i=1}^{k-1} \mathbf{y}_i + (\mathbf{x}_1 \vee \cdots \vee \mathbf{x}_k) = \sum_{i=1}^{k-1} ((\mathbf{x}_1 \vee \cdots \vee \mathbf{x}_i) \wedge \mathbf{x}_{i+1}) + (\mathbf{x}_1 \vee \cdots \vee \mathbf{x}_k) = \sum_{i=1}^k \mathbf{x}_i$, which can be proved by repeated use of the rule $(\mathbf{u} \wedge \mathbf{v}) + (\mathbf{u} \vee \mathbf{v}) = \mathbf{u} + \mathbf{v}$, and finally $\sum_{i=1}^k \mathbf{x}_i = \mathbf{1}$. Note that v is in $U(\theta)$ if and only if $u(v) \geq 1 - \theta$. Therefore

$$\int_0^1 f(U(\theta)) d\theta = \hat{f}(\mathbf{u}).$$

Additionally, we have

$$\begin{aligned} \frac{1}{k-1} \sum_{i=1}^{k-1} \hat{f}(\mathbf{y}_i) &\geq \hat{f}\left(\frac{1}{k-1} \sum_{i=1}^{k-1} \mathbf{y}_i\right) \quad (\hat{f} \text{ is convex}) \\ &= \hat{f}\left(\frac{1}{k-1} \mathbf{u}\right) = \frac{1}{k-1} \hat{f}(\mathbf{u}) \end{aligned}$$

where we also used the fact that $\hat{f}(\alpha \mathbf{x}) = \alpha \hat{f}(\mathbf{x})$ for any $\alpha \in [0, 1]$ ($\hat{f}(\mathbf{x})$ is linear along any line through the origin). Equivalently,

$$\sum_{i=1}^{k-1} \int_0^1 f((A(1, \theta) \cup \cdots \cup A(i, \theta)) \cap A(i+1, \theta)) d\theta \geq \int_0^1 f(U(\theta)) d\theta.$$

Now note that, if $\theta > \delta$, the sets $(A(1, \theta) \cup \cdots \cup A(i, \theta)) \cap A(i+1, \theta)$ are empty, since $\sum_{i=1}^k \mathbf{x}_i = \mathbf{1}$ and hence two vectors $\mathbf{x}_i, \mathbf{x}_j$ cannot have the same coordinate larger than $\theta > \delta \geq 1/2$. We also assumed that $f(\emptyset) = 0$, so we proved in fact

$$\sum_{i=1}^{k-1} \int_0^\delta f((A(1, \theta) \cup \cdots \cup A(i, \theta)) \cap A(i+1, \theta)) d\theta \geq \int_0^1 f(U(\theta)) d\theta,$$

as desired. □

Given this inequality, Theorem 3.1.4 follows easily.

Proof of Theorem 3.1.4: By applying submodularity inductively to the sets $A(1, \theta) \cup \cdots \cup A(i, \theta)$

and $A(i+1, \theta)$, we get

$$\begin{aligned} \sum_{i=1}^k f(A(i, \theta)) &\geq \sum_{i=1}^{k-1} f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) + f(A(1, \theta) \cup \dots \cup A(k, \theta)) \\ &= \sum_{i=1}^{k-1} f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) + f(A(\theta)). \end{aligned}$$

Integrating from 0 to δ and using Lemma 3.4.1, we obtain

$$\begin{aligned} \sum_{i=1}^k \int_0^\delta f(A(i, \theta)) d\theta &\geq \sum_{i=1}^{k-1} \int_0^\delta f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) d\theta + \int_0^\delta f(A(\theta)) d\theta \\ &\geq \int_0^1 f(U(\theta)) d\theta + \int_0^\delta A(\theta) d\theta. \end{aligned}$$

□

We conclude this section with the following theorem, which is an extension of Theorem 3.1.4.

3.5 A variant of the main theorem

In this section, we prove the following theorem, which can be viewed as an extension of the special case of Theorem 3.1.4 in which δ is equal to one.

Theorem 3.5.1. *Let $f \geq 0$ be a submodular function such that $f(\emptyset) = 0$. Let ℓ be an index such that $1 \leq \ell \leq k$. Let \mathbf{x} be a feasible solution to SubMP-Rel. For $\theta \in [0, 1]$ let $A(i, \theta) = \{v \mid x(v, i) \geq \theta\}$, $A(\theta) = \cup_{i=1}^\ell A(i, \theta)$ and $U(\theta) = V - A(\theta)$. We have*

$$\sum_{i=1}^k \int_0^1 f(A(i, \theta)) d\theta \geq \int_0^1 f(A(\theta)) d\theta + \int_0^1 f(U(\theta)) d\theta.$$

Note that, in the preceding theorem, the allocated set $A(\theta)$ is the union of the first ℓ sets $A(i, \theta)$, whereas in Theorem 3.1.4 the set $A(\theta)$ is the union of all k sets $A(i, \theta)$. The proof is similar to the proof of Theorem 3.1.4. The theorem is a straightforward corollary of the following inequality.

Lemma 3.5.2. *We have*

$$\sum_{i=\ell+1}^k \int_0^1 f(A(i, \theta)) d\theta + \sum_{i=1}^{\ell-1} \int_0^1 f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) d\theta \geq \int_0^1 f(U(\theta)) d\theta.$$

Proof. We can view the value $\int_0^1 f(A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta) d\theta$ as the Lovász extension evaluated on a suitable vector, $\mathbf{y}_i = (\mathbf{x}_1 \vee \dots \vee \mathbf{x}_i) \wedge \mathbf{x}_{i+1}$. In other words, $y(v, i) = \min \{ \max \{ x(v, 1), \dots, x(v, i) \}, x(v, i+1) \}$ for each $v \in V$. Note that v is in $(A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)$ if and only if $y(v, i) \geq \theta$. Therefore

$$\int_0^1 f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) d\theta = \hat{f}(\mathbf{y}_i).$$

We can also view the value $\int_0^1 f(U(\theta))$ as the Lovász extension evaluated on the vector $\mathbf{u} = \sum_{i=\ell+1}^k \mathbf{x}_i + \sum_{i=1}^{\ell-1} \mathbf{y}_i$. Note that $\mathbf{u} = \mathbf{1} - (\mathbf{x}_1 \vee \dots \vee \mathbf{x}_\ell)$; this holds because $\sum_{i=1}^{\ell-1} \mathbf{y}_i + (\mathbf{x}_1 \vee \dots \vee \mathbf{x}_\ell) = \sum_{i=1}^{\ell-1} ((\mathbf{x}_1 \vee \dots \vee \mathbf{x}_i) \wedge \mathbf{x}_{i+1}) + (\mathbf{x}_1 \vee \dots \vee \mathbf{x}_\ell) = \sum_{i=1}^{\ell} \mathbf{x}_i$, which can be proved by repeated use of the rule $(\mathbf{u} \wedge \mathbf{v}) + (\mathbf{u} \vee \mathbf{v}) = \mathbf{u} + \mathbf{v}$, and finally $\sum_{i=1}^{\ell} \mathbf{x}_i = \mathbf{1} - \sum_{i=\ell+1}^k \mathbf{x}_i$. Note that v is in $U(\theta)$ if and only if $u(v) \geq 1 - \theta$. Therefore

$$\int_0^1 f(U(\theta)) d\theta = \hat{f}(\mathbf{u}).$$

Additionally, we have

$$\begin{aligned} \frac{1}{k-1} \left(\sum_{i=\ell+1}^k \hat{f}(\mathbf{x}_i) + \sum_{i=1}^{\ell-1} \hat{f}(\mathbf{y}_i) \right) &\geq \hat{f} \left(\frac{1}{k-1} \left(\sum_{i=\ell+1}^k \mathbf{x}_i + \sum_{i=1}^{\ell-1} \mathbf{y}_i \right) \right) \quad (\hat{f} \text{ is convex}) \\ &= \hat{f} \left(\frac{1}{k-1} \mathbf{u} \right) = \frac{1}{k-1} \hat{f}(\mathbf{u}) \end{aligned}$$

where we also used the fact that $\hat{f}(\alpha \mathbf{x}) = \alpha \hat{f}(\mathbf{x})$ for any $\alpha \in [0, 1]$ ($\hat{f}(\mathbf{x})$ is linear along any line through the origin). Equivalently,

$$\sum_{i=\ell+1}^k \int_0^1 f(A(i, \theta)) d\theta + \sum_{i=1}^{\ell-1} \int_0^1 f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) d\theta \geq \int_0^1 f(U(\theta)) d\theta.$$

□

Given this inequality, Theorem 3.5.1 follows easily.

Proof of Theorem 3.1.4: By applying submodularity inductively to the sets $A(1, \theta) \cup \dots \cup A(i, \theta)$ and $A(i+1, \theta)$, we get

$$\begin{aligned} \sum_{i=1}^{\ell} f(A(i, \theta)) &\geq \sum_{i=1}^{\ell-1} f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) + f(A(1, \theta) \cup \dots \cup A(\ell, \theta)) \\ &= \sum_{i=1}^{\ell-1} f((A(1, \theta) \cup \dots \cup A(i, \theta)) \cap A(i+1, \theta)) + f(A(\theta)). \end{aligned}$$

Integrating from 0 to 1 and using Lemma 3.5.2, we obtain

$$\sum_{i=1}^k \int_0^1 f(A(i, \theta)) d\theta \geq \int_0^1 f(U(\theta)) d\theta + \int_0^1 A(\theta) d\theta.$$

□

3.6 Concluding remarks

In this chapter, we consider the **SymSubMP** and **SubMP** problems. These problems are special cases of the **MSCA** problem from Chapter 2 and they capture classical multiway cut problems in graphs and hypergraphs. Our main results are a $(1.5 - 1/k)$ approximation for **SymSubMP** and a $2(1 - 1/k)$ -approximation for **SubMP**.

In joint work with Jan Vondrák and Yi Wu [71], we showed that the approximation guarantee for **SubMP** is optimal in two senses: (1) a $2(1 - 1/k - \epsilon)$ -approximation for **SubMP** would require exponentially many value queries (in the oracle model), or imply $\mathbf{NP} = \mathbf{RP}$ (for certain explicit submodular functions), and (2) for **HyperMC** and **Node-wt-MC**, a $2(1 - 1/k - \epsilon)$ -approximation is \mathbf{NP} -hard assuming the Unique Games Conjecture. For **SymSubMP**, we showed that, for any fixed k sufficiently large, a 1.26-approximation would require exponentially many value queries (in the oracle model), or imply $\mathbf{NP} = \mathbf{RP}$ (for certain explicit submodular functions).

The main question that is left open by our work is whether the integrality gap of **SubMP-Rel** for **SymSubMP** is strictly smaller than the bound of $(1.5 - 1/k)$ we showed in this chapter. Karger *et al.* [111] rely extensively on the geometry of the simplex to obtain a bound of 1.3438 for **GraphMC** via the relaxation from [60]. However, we mention that the rounding algorithms used in [111] have

natural analogues for rounding **SubMP-Rel** but analyzing them appears challenging for an arbitrary symmetric submodular function. The recent work of Buchbinder *et al.* [27] improved the gap of the relaxation of [60] to $4/3 - \epsilon$. It would be very interesting to generalize their approach to the **SymSubMP** problem.

Zhao, Nagamochi and Ibaraki [173] considered a common generalization of **SubMP** and **k-way Sub-MP** where we are given a set S of terminals with $|S| \geq k$ and the goal is to partition V into k sets A_1, \dots, A_k such that each A_i contains at least one terminal and $\sum_{i=1}^k f(A_i)$ is minimized. Note that when $|S| = k$ we get **SubMP** and when $S = V$ we get **k-way Sub-MP**. The advantage of the greedy splitting algorithms developed in [173] is that they extend to these more general problems. However, unlike the case of **SubMP**, it is unclear how to formulate a mathematical programming relaxation for this more general problem; see [38] for a relaxation in the case of graphs. An interesting open problem is whether the k -way cut problem in graphs admits an approximation better than $2(1 - 1/k)$.

Related to the above questions is the complexity of **k-way Sub-MP** when k is a fixed constant. For **SymSubMP** a polynomial-time algorithm was claimed in [143] although no formal proof has been published; this generalizes the polynomial-time algorithm for the **Graph k -Cut** problem first developed by Goldschmidt and Hochbaum [93]. There has been particular interest in the special case of **k-way Sub-MP**, namely, the **Hypergraph k -Cut** problem [79, 138, 171]. It is an open problem whether the **Hypergraph k -Cut** problem has a polynomial time algorithm for $k = 4$.

Chapter 4

The Maximum Node Disjoint Paths Problem

4.1 Introduction

In this chapter¹, we consider the Maximum Node Disjoint Paths (MNDP) problem in undirected graphs. An instance of MNDP consists of an undirected graph $G = (V, E)$ and a collection $\mathcal{M} = \{(s_1, t_1), \dots, (s_k, t_k)\} \subseteq V \times V$ of k source-sink pairs. The goal is to *route* a maximum cardinality subset of the source-sink pairs via *node-disjoint* paths. Formally, the goal is to select a maximum cardinality subset $\mathcal{M}' \subseteq \mathcal{M}$ and a collection of node-disjoint² paths \mathcal{P} such that, for each pair $(s_i, t_i) \in \mathcal{M}'$, there is a path in \mathcal{P} with endpoints s_i and t_i . MNDP is an optimization version of the classical decision problem NDP in which the goal is to decide whether all the pairs in \mathcal{M} can be routed in G via node disjoint paths. NDP and MNDP are related to, and generalize, the corresponding problems EDP and MEDP where the paths for the pairs are required to be *edge-disjoint*. These disjoint paths problems have been well-studied because of their connections to algorithms, combinatorial optimization, and graph theory. In addition, these problems and their variants can be used to model a variety of routing problems in networks, and have several applications in practice.

Karp [113] showed that NDP is **NP**-complete when k is part of the input (in his original list of **NP**-complete problems). EDP is also **NP**-complete [72]. Over the years it has been shown that very restricted instances of disjoint paths problems are **NP**-complete, see [135] for a survey. In contrast, when k is a fixed constant, Robertson and Seymour [150], building on several tools from their seminal graph minor project, gave a polynomial-time algorithm for NDP. In this chapter we

¹This chapter is based on joint work with Chandra Chekuri and it has appeared in [36]. Copyrights to the conference version of [36] are held by the Society for Industrial and Applied Mathematics (SIAM).

²It is important that the paths do not share endpoints as well as internal nodes. This is in contrast to some settings in which one may want the paths to be disjoint only on the internal nodes.

are concerned with the case when k is part of the input and consider the approximability of MNDP. MNDP is easily seen to be **NP**-hard via a reduction from NDP. There is also hardness coming from the problem of choosing which pairs to connect. This can be seen from the fact that MEDP is **NP**-hard (and in fact **APX**-hard to approximate) in capacitated trees [84]; routing is trivial in trees since there is a unique path connecting any given pair, nevertheless, the subset selection is hard.

Approximation algorithms for MEDP and MNDP have been studied extensively with MEDP receiving the most attention. The starting point for many approximation algorithms for disjoint paths problems is a natural multicommodity flow based linear programming relaxation (see Section 4.2 for the formal description); random-walk based algorithms for routing on expanders are an exception. However, a simple example shows that the integrality gap of this flow relaxation is $\Omega(\sqrt{n})$ (here n is the number of nodes in G) for both MEDP and MNDP even on planar graph instances [84]. This arises due to a crossing obstruction in the plane that allows only one pair to be routed while the fractional solution can route $1/2$ unit of flow for $\Omega(\sqrt{n})$ pairs. This example naturally led to the question of whether the integrality gap of the flow relaxation becomes substantially smaller if one removes the topological obstruction by allowing up to two paths to use an edge (or assuming that edges/nodes have capacity 2). More generally, what is the trade-off between the congestion c (the number of paths that are allowed to use an edge/node) and the integrality gap of the flow relaxation? Raghavan and Thompson [145] showed via randomized rounding that a constant factor approximation is achievable if $c = \Omega(\log n / \log \log n)$ (even in directed graphs). It is only in the last few years that substantial and exciting progress was achieved for $c = o(\log n / \log \log n)$. In a recent breakthrough, Chuzhoy [53] obtained a poly-logarithmic approximation for MEDP with congestion 14; Chuzhoy and Li [57] improved the congestion to 2. Our main result is a poly-logarithmic approximation for MNDP with constant congestion and is encapsulated in the following theorem.

Theorem 4.1.1. *There is a randomized polynomial time algorithm that, given an instance of MNDP in an undirected graph G with n nodes and k pairs, routes $\Omega(\text{OPT}/\text{poly log } k)$ pairs with $O(1)$ congestion, where OPT is the value of an optimum solution to the multicommodity flow relaxation.*

The congestion we can guarantee is 51. We believe that it can be further reduced, however, we have

not attempted to optimize it in the interests of keeping the algorithmic details and proofs simple and clear. An $O(\sqrt{n})$ -approximation is known for MNDP [127]; with congestion c the best previous approximation in general graphs is $O(n^{1/c})$ which follows from randomized rounding. Moreover, via known hardness results for MEDP [5], for any constant c there is no $O(1)$ -approximation for MNDP with congestion c unless $\mathbf{NP} \subseteq \mathbf{ZPTIME}(n^{\text{poly log } n})$. We discuss a specific motivation for our study of MNDP.

MEDP, MNDP and connections to treewidth and graph theory: Chuzhoy’s work on MEDP introduces beautiful new ideas while utilizing several concepts and tools from previous work [4, 42, 116, 144, 146, 158]. In particular, the work of [42] on well-linked decompositions for routing problems reduced the poly-logarithmic approximability of MEDP and MNDP with constant congestion to the following graph-theoretic question. *If a graph G has a well-linked set³ of size k , does it have a constant congestion crossbar of size $\Omega(k/\text{poly log } k)$?* A crossbar is a switch-like routing structure that can route any permutation at its interface — see [42] for the precise definitions. Chuzhoy’s work answered this question affirmatively by embedding, with constant congestion, an expander of size $\Omega(k/\text{poly log } k)$ into any graph G that has a well-linked set of size k ; the edges of the expander are embedded as paths in G that cause constant edge congestion. Our result shows that the graph-theoretic question has an affirmative answer for node-disjoint routing as well. In the node-capacitated case, a graph G has a well-linked set of size k iff the treewidth of G is $\Omega(k)$. Thus, our result implies that if G has treewidth k then one can embed an expander of size $\Omega(k/\text{poly log } k)$ such that the edges of G can be mapped to paths that cause constant node congestion. This has the following interesting connection to a central result in the graph minor project of Robertson and Seymour.

Theorem 4.1.2 (Robertson and Seymour [149]). *Let $r, h > 0$ be integers and let G be a graph of treewidth greater than $r^{4h^2(r+2)}$. Then G contains either the $r \times r$ grid or the complete graph K_h as a minor.*

The quantitative bounds have been subsequently improved by Robertson, Seymour and Thomas.

³A set X is well-linked in G iff, for any two equal-sized subsets Y and Z of X , there is a collection of disjoint Y - Z paths in G .

Theorem 4.1.3 ([151]). *Let $r, h > 0$ be integers and let G be a graph of treewidth at least 20^{5gh^3} . Then G contains either the $r \times r$ grid or the complete graph K_h as a minor.*

We note that a clique minor K_h is a crossbar of size h ; an $h \times h$ grid minor is also a crossbar of size h but it requires congestion 2. Thus, Theorem 4.1.3 says that if a graph G has treewidth at least k then it is guaranteed to have a congestion 2 crossbar of size $\Omega(\log^{1/4} k)$. Our result shows that, by allowing constant congestion, one can obtain a crossbar of size $\Omega(k/\text{poly log } k)$, which is a significant improvement. As previously mentioned, Chuzhoy and Li [57] obtained a congestion 2 poly-logarithmic approximation for MEDP, a remarkably tight result! It is conceivable that one can extend their result (although we suspect it will be technically quite challenging) to obtain a similar result for MNDP. These results and some of the techniques developed along the way may have other applications in (algorithmic) graph theory.

Our algorithm for MNDP follows Chuzhoy’s high-level framework for MEDP. Node problems in routing and network design have similarities to their corresponding edge problems, but often exhibit non-trivial differences in the technical details or approximability. The algorithm in [53] uses several tools; some of them are straightforward to generalize to the node setting and some are not. In the following subsection, we give a high-level overview of Chuzhoy’s algorithm for MEDP and our adaptation of it to MNDP, while indicating which parts generalize easily and which require new technical ideas. This also serves as a roadmap for the reader who may not be familiar with [53]. This description will gloss over several details; Section 4.3 gives a formal description of the algorithm.

4.1.1 High-level overview of algorithm and technical contribution

Let (G, \mathcal{M}) be an instance of the MNDP problem. We may assume without loss of generality that the nodes participating in the pairs of \mathcal{M} are distinct and they have degree one in G . Let \mathcal{T} denote the set of all nodes participating in the pairs of \mathcal{M} ; we refer to the nodes in \mathcal{T} as *terminals*.

Reduction to well-linked instances: The first (and a key) step is to reduce a general instance of the problem to one in which the terminals are *well-linked*. This is done via the well-linked decomposition framework of [42] which applies to edge as well as node problems. An important technical ingredient is a grouping technique that, given a set X of nodes that is approximately

well-linked, it identifies a subset X' of X that is well-linked. This boosting technique is simple in the edge-capacitated setting [40] but is more involved in the node-capacitated setting [42, 43].

Embedding an expander in a well-linked instance: Suppose \mathcal{T} is well-linked. The second ingredient is to show that an expander of sufficiently large size can be embedded into G with constant node congestion. Once this is done, a large number of pairs can be routed via the expander (there are technical details on how to embed an expander that can be reached by \mathcal{T}). To embed the expander, Chuzhoy [53] builds on a crucial idea from Rao and Zhou’s [146] work on approximating MEDP when G has a large (poly-logarithmic) global minimum-cut value. They embedded an expander as follows. Khandekar, Rao, and Vazirani [116] describe an algorithm that, given a graph G with a well-linked set X of size k , embeds an expander of size k in G but with congestion $O(\log^2 k)$. This may not seem so useful but the important fact about the KRV algorithm is that the expander is embedded in $O(\log^2 k)$ rounds where in each round the well-linkedness of X is used to find a collection of $|X|/2$ disjoint paths. Rao and Zhou used the large minimum cut assumption to split G into $\Omega(\log^2 k)$ *edge-disjoint* subgraphs of G via Karger’s sampling scheme [110], and simulated each round of the KRV algorithm in a separate subgraph. Subsequently, Andrews [4], using some ideas from [146] and properties of Raecke’s hierarchical decomposition [144], obtained a poly-logarithmic approximation with $O(\text{poly}(\log \log n))$ congestion; Andrews’ result was the first approximation algorithm that achieves a sub-polynomial approximation factor using $o(\log n)$ congestion. We note that both Rao-Zhou’s sampling approach and Andrews’ approach based on the Raecke tree decomposition do not admit an easy extension to node-disjoint routing.

Chuzhoy’s key high-level contribution is based on a new approach to simulating the KRV algorithm and it consists of two ingredients.

Finding good subsets: Chuzhoy shows that if G has a well-linked set X of size k , for any integer parameter h , one can find h *node-disjoint* subsets S_1, S_2, \dots, S_h such that each S_i has a boundary B_i and a subset $D_i \subseteq B_i$ such that $|D_i| = \Omega(k/(\text{poly}(h, \log k)))$ and D_i is approximately well-linked in $G[S_i]$; Chuzhoy refers to such a set as a good subset. In other words, G can be split into h disjoint subgraphs each of which has a well-linked set of size $\Omega(k/\text{poly}(h, \log k))$. The idea is to simulate each iteration of the KRV algorithm inside a separate subgraph $G[S_i]$ by choosing $h = \Omega(\log^2 k)$. The algorithm of Chuzhoy for finding such a partitioning relies on edge-well-linked

sets and their properties, and it is quite technical and non-trivial. We believe that the algorithm can be generalized to apply directly to node-well-linked sets, however, we do not have such an algorithm yet. Instead, we apply a simple idea to use the algorithm of Chuzhoy in a black box fashion. Using a preprocessing step, we can assume that the graph G has maximum degree $O(\log n)$. An edge-well-linked set is an approximately node-well-linked where the approximation depends linearly on the degree. Since good subsets are already weakly well-linked (the well-linkedness parameters are later boosted using the grouping technique that we mentioned earlier), this loss does not matter too much and we can absorb it into the approximation ratio rather than in the congestion which we cannot afford to do.

Connecting good subsets via disjoint trees: The second ingredient in Chuzhoy’s approach for embedding an expander is the following. The good subsets allow each iteration of KRV to be simulated inside a separate part of G so that the edges used in each iteration are disjoint. However, to embed an expander, one needs to have for each node v of the expander a representative v_i in the boundary of $G[S_i]$; further, all the representatives of v have to simulate a single node. For this purpose Chuzhoy ensures that the boundaries of the good sets are in fact connected to the terminals \mathcal{T} and hence are well-linked themselves. Moreover, she uses this property and several technical tools based on connectivity, to find $k' = \Omega(k/\text{poly log } k)$ trees $T_1, T_2, \dots, T_{k'}$ such that (i) the trees are nearly edge-disjoint, in the sense that no edge of G is in more than a constant number of trees, (ii) each tree T_j has a leaf in the boundary of each good subset S_i , and (iii) the leaves of the different trees are disjoint. Thus, each T_j simulates a node v of the expander and the leaves of T_j are the representatives of v in each good subset.

Our main technical contribution is in implementing the preceding step for MNDP; we need to find trees that are nearly node-disjoint. The algorithm of Chuzhoy for finding the disjoint trees is involved; she uses the splitting-off operation that preserves edge-connectivity [76, 105, 133] to create an auxiliary graph and then shows the existence of a bounded degree spanning tree T in the auxiliary graph via a result of Singh and Lau [158]. This tree T is then used as a “template” to generate the required nearly edge-disjoint trees. To obtain nearly node-disjoint trees, we instead rely on an element-connectivity reduction step [47, 50, 100]; however, this reduction step is not as clean as the edge-connectivity step and does not eliminate “Steiner” nodes that can have high

degree. Nevertheless, we are able to apply the rough high-level idea of finding a bounded-degree spanning tree but we use a different (a weaker but somewhat more intuitive) argument that is based on the notion of toughness of a graph [80, 170]. We then do a postprocessing step to reduce the high-degree Steiner nodes and make them to essentially behave as edges. We refer the reader to Section 4.3 and Section 4.5 for more details.

Other related work: We refer the reader to [135, 155] for tractable cases of EDP. We mostly restrict our attention to undirected graphs. The literature on approximation algorithms for disjoint paths problems has focused primarily on MEDP. The best known approximation for MEDP is $O(\sqrt{n})$ [44], and there is a matching approximation for MNDP [127]; here n is the number of nodes in the input graph. Various special classes of graphs have been studied; constant factor and poly-logarithmic factor approximations for MEDP are known for trees [49, 84], graphs with bounded treewidth [46], grids and grid-like graphs [117, 120, 121], Eulerian planar graphs [114, 118], graphs with good expansion [26, 77, 119, 128], and graphs with large global minimum cut [146]. MEDP and MNDP with congestion have also been well-studied, especially given the integrality gap of the flow relaxation; it is known that randomized rounding techniques give an $O(d^{1/c})$ approximation, where d is the maximum flow path length in the fractional solution and c is the congestion parameter [12, 29, 127, 159]; this holds even for directed graphs and it leads to an $O(n^{1/c})$ approximation. Improved bounds are obtained by taking advantage of fractional solutions with short paths, for example, in expanders. The well-linked decomposition ideas in [40–42] led to an $O(\log k)$ -approximation for MEDP and MNDP in planar graphs with congestion 2; the approximation for MEDP was subsequently improved to an $O(1)$ -approximation in [45, 156]. Finally, [115] obtained an $O(n^{3/7} \text{poly log } n)$ -approximation ratio for MEDP with congestion 2 prior to the result of [57] that obtained a poly-logarithmic approximation.

In terms of hardness of approximation, despite the polynomial-factor upper bounds, the first non-trivial lower bounds were established fairly recently. It is known that MEDP (and hence also MNDP) does not admit an $O(\log^{1/2-\varepsilon} n)$ -approximation unless $\mathbf{NP} \subseteq \mathbf{ZPTIME}(n^{\text{poly log } n})$ [5]; under the same hardness assumption there is no $O\left((\log n)^{\frac{1-\varepsilon}{c+1}}\right)$ -approximation with congestion c [5]. As we mentioned, the latter result rules out for MEDP and MNDP a constant factor approximation for any constant congestion. MEDP is significantly harder in directed graphs; there is no $n^{1/2-\varepsilon}$ -

approximation unless $\mathbf{P} = \mathbf{NP}$ [95], and with congestion c there is no $n^{\Omega(1/c)}$ -approximation unless $\mathbf{NP} \subseteq \mathbf{ZPTIME}(n^{\text{poly log } n})$ [54]; these hardness bounds match the approximation ratios guaranteed by randomized rounding.

Chapter outline: We build on several tools from previous work; Section 4.2 discusses the relevant definitions and theorems. Section 4.3 describes our algorithm and its proof assuming two key technical theorems on embedding an expander in a graph with a well-linked set. These theorems are proved in Section 4.4 and Section 4.5, respectively.

4.2 Preliminaries and setup

In the following, we work with an instance (G, \mathcal{M}) of the MNDP problem, where $G = (V, E)$ is an undirected graph and $\mathcal{M} = \{(s_1, t_1), \dots, (s_k, t_k)\}$ is a collection k node pairs. We let \mathcal{T} denote the set of all nodes participating in the pairs of \mathcal{M} . We refer to the nodes in \mathcal{T} as *terminals*. Each terminal has degree one in G and \mathcal{M} is a perfect matching on \mathcal{T} .

For a set S of nodes, we let $\text{out}_G(S)$ denote the set of all edges $e \in E(G)$ such that e has exactly one endpoint in S . Let $\text{bd}_G(S)$ denote the set of all nodes $v \in S$ such that v is incident to some edge of $\text{out}_G(S)$; we refer to $\text{bd}_G(S)$ as the *inner boundary* of S . Let $N_G(S)$ be the set of all nodes $v \notin S$ such that v is incident to some edge of $\text{out}_G(S)$; we refer to $N_G(S)$ as the *outer boundary* of S . We may omit the subscript if the graph G is clear from the context.

Given two disjoint subsets $A, B \subset V$ in a graph G such that $|A| \leq |B|$ we say that \mathcal{P} is a collection of A - B paths if the following properties hold: (i) the endpoints of the paths in \mathcal{P} are in $A \cup B$ and (ii) each node of A is the endpoint of exactly one path and each node of B is the endpoint of at most one path.

LP relaxation: We consider a standard multicommodity flow relaxation for the MNDP problem. Let \mathcal{P}_i denote the collection of all paths that connect s_i to t_i in G , and let $\mathcal{P} = \cup_i \mathcal{P}_i$; from our assumption on \mathcal{M} forming a perfect matching on \mathcal{T} , \mathcal{P}_i and \mathcal{P}_j are disjoint for $i \neq j$. For each path $p \in \mathcal{P}$, the relaxation has a variable $f(p)$ which represents the amount of flow that is sent on p . We let x_i denote the total flow routed for the pair (s_i, t_i) . We use f to denote the vector that has an entry for each path $p \in \mathcal{P}$ that is equal to $f(p)$, and we let $|f| = \sum_i x_i$.

$$\begin{array}{ll}
\text{(NDP-LP)} & \\
\max & \sum_{i=1}^k x_i \\
\text{s.t.} & \sum_{p \in \mathcal{P}_i} f(p) = x_i \quad 1 \leq i \leq k \\
& \sum_{p: v \in p} f(p) \leq 1 \quad v \in V(G) \\
& 0 \leq x_i \leq 1 \quad 1 \leq i \leq k \\
& f(p) \geq 0 \quad p \in \mathcal{P}
\end{array}$$

The above path-based relaxation has an exponential (in n) number of variables but a polynomial number of non-trivial constraints. It can be solved in polynomial time since the separation oracle for the dual is a shortest path problem. Alternatively, there is an equivalent LP formulation that is polynomial-sized.

Sparse node separators: We need an approximation algorithm for finding a sparse node separator. This will be used (in a black box fashion) to obtain a well-linked decomposition. With this goal in mind, let $\text{cap} : V \rightarrow \mathbb{R}_+$ be a node-capacity function and let $\pi : V \rightarrow \mathbb{R}_+$ be a weight function. A node separator is a set $S \subset V$ that partitions $V - S$ into A and B such that there are no edges between A and B . The sparsity⁴ of S is defined to be $\frac{\text{cap}(S)}{\pi(A \cup S)\pi(B \cup S)}$. An $O(\sqrt{\log k})$ -approximation for sparse node separators is given in [73] where k is the support of π (the number of nodes with non-zero π value); see also [1].

Well-linked sets: As we discussed already, well-linked sets and the reduction to well-linked instances of disjoint paths is an important ingredient. Let $X \subseteq V$ be a set of nodes and $\pi : X \rightarrow [0, 1]$ be a weight function on X . We are primarily concerned here with node-well-linked sets but we start with the easier to define notion of edge-well-linkedness. We say that X is π -edge-well-linked in G if $|\delta_G(S)| \geq \pi(X \cap S)$ for all $S \subset V$ such that $\pi(X \cap S) \leq \pi(X \cap (V \setminus S))$. If $\pi(u) = \alpha$ for all $u \in X$ we say that X is α -edge-well-linked, and in particular X is edge-well-linked if $\alpha = 1$. Using Menger's theorem, it is straightforward to show that if X is edge-well-linked then for any two disjoint subsets Y, Z of X such that $|Y| = |Z| = k$ there are k edge-disjoint Y - Z paths in

⁴One has to be a somewhat careful in defining sparsity of node separators. This is in contrast to the definitions for sparsity of edge separators. We refer the reader [73]; we follow their definition.

G . In the context of node-well-linked sets, a separator-based definition was given in [42]; here we give a slightly refined and precise definition that is helpful. We say that X is node-well-linked if for any two disjoint subsets Y, Z of X such that $|Y| = |Z| = k$ there are k node-disjoint Y - Z paths in G . This is the definition that is standard in the literature on treewidth. We would like to extend it to weight functions $\pi : X \rightarrow [0, 1]$. Assume that each node in X has degree 1 in G and no two nodes $u, v \in X$ share a neighbor in G . In this case we say that X is π -node-well-linked if $|N_G(S)| \geq \pi(X \cap S)$ for all sets S such that $\pi(X \cap S) \leq \pi(X \cap (V \setminus S))$. In general, we say that X is π -node-well-linked in G if X' is π' -node-well-linked in G' , where G' , X' , and π' are defined as follows. For each node $u \in X$, we attach a new leaf node u' to u ; we let G' denote the resulting graph and we let X' denote the set of all new leaf nodes. For each node $u' \in X'$, we set $\pi'(u') = \pi(u)$. As in the edge case, we say that X is α -node-well-linked if $\pi(u) = \alpha$ for all $u \in X$. The following lemma follows easily from Menger's theorem.

Lemma 4.2.1. *Let G be a graph and let X be a set that is α -node-well-linked in G , where $\alpha \in (0, 1]$. For any two subsets Y and Z of X such that $|Y| = |Z|$, there is a collection of Y - Z paths \mathcal{P} such that each node of G appears in at most $\lceil 1/\alpha \rceil$ paths of \mathcal{P} .*

The following proposition relates the two notions of edge and node well-linkedness.

Proposition 4.2.2. *Let G be a graph with maximum degree Δ . Let X be a set of vertices of G and let π be a weight function on X . If X is π -edge-well-linked in G then X is (π/Δ) -node-well-linked in G .*

Well-linked decomposition: The following theorem allows us to reduce a general instance of MNDP to one in which the terminals are (approximately) node-well-linked.

Theorem 4.2.3 ([42]). *Let OPT be the value of a solution to NDP-LP for a given instance (G, \mathcal{M}) of MNDP in a graph G . Let $\beta(G) \geq 1$ be an upper bound on the approximation ratio of a polynomial time algorithm for the sparsest node separator problem in G . Then there is a polynomial time algorithm that partitions G into node-disjoint induced subgraphs G_1, G_2, \dots, G_ℓ and it assigns a weight function $\pi_i : V(G_i) \rightarrow \mathbb{R}_+$ to each graph G_i such that the function π_i has the following properties. Let \mathcal{M}_i be the set of all pairs of \mathcal{M} that are contained in G_i , and let \mathcal{T}_i be the set of terminals of \mathcal{M}_i .*

(1) $\pi_i(u) = \pi_i(v)$ for $uv \in \mathcal{M}_i$.

(2) \mathcal{T}_i is π_i -node-well-linked in G_i .

(3) $\sum_{i=1}^{\ell} \pi_i(\mathcal{T}_i) = \Omega(\text{OPT}/(\beta(G) \log \text{OPT})) = \Omega(\text{OPT}/\log^{1.5} k)$.

Grouping technique: We also need a technique to boost well-linkedness in the following sense: Given a set X that is α -well-linked, find a subset X' that is well-linked. This is relatively easy for edge-well-linkedness [40] but is more involved in the node case. The following theorem strengthens weaker versions that were initially given in [42].

Theorem 4.2.4 ([43]). *Let B be a π -node-well-linked set in G and let M be a perfect matching on B such that $\pi(u) = \pi(v)$ for all $uv \in M$. Then there is a matching $M' \subseteq M$ with endpoints $B' \subseteq B$ such that B' is $1/4$ -node-well-linked in G and $|M'| = 2|B'| = \Omega(\pi(B))$. Moreover, we can find B' and M' in polynomial time.*

Combining Theorem 4.2.3 and Theorem 4.2.4, we can reduce an arbitrary instance of MNDP to one in which the terminals are $1/4$ -node-well-linked at the loss of a $O(\log^{1.5} k)$ -factor in the approximation ratio. Here we use the $O(\sqrt{\log k})$ -approximation for sparse node separators from [73].

Routing in edge expanders: A graph $G = (V, E)$ is an α -edge-expander iff $\min_{S \subseteq V: |S| \leq |V|/2} \frac{|\delta(S)|}{|S|} \geq \alpha$. We make use of the following theorem on routing along node-disjoint paths in edge-expanders that have a bound on the degree (and hence are also node-expanders with a weaker parameter that depends on the degree).

Theorem 4.2.5 ([146]). *Let $G = (V, E)$ be a d -regular α -edge-expander on n nodes. Suppose that n is even and the nodes of G are partitioned into $n/2$ disjoint demand pairs \mathcal{M} . There is a polynomial time algorithm that routes a subset $\mathcal{M}' \subseteq \mathcal{M}$ of size $\Omega(\alpha n / (d^2 \log n))$ on node-disjoint paths of G .*

The cut-matching game: Following [53, 146], we use the cut-matching game of Khandekar, Rao, and Vazirani [116] in order to embed an expander into G . In the cut-matching game, there is a set V of nodes, where $|V|$ is even, and two players, the cut player and the matching player. The

goal of the cut player is to construct an edge-expander in as few iterations as possible, whereas the goal of the matching player is to prevent the construction of the edge-expander for as long as possible. The two players start with a graph \mathcal{X} with node set V and an empty edge set. The game then proceeds in iterations, each of which adds a set of edges to \mathcal{X} . In iteration j , the cut player chooses a partition (Y_j, Z_j) of V such that $|Y_j| = |Z_j|$ and the matching player chooses a perfect matching M_j that matches the nodes of Y_j to the nodes of Z_j . The edges of M_j are then added to \mathcal{X} . Khandekar, Rao, and Vazirani [116] showed that there is a strategy for the cut player that guarantees that after $O(\log^2 |V|)$ iterations the graph \mathcal{X} is a $1/2$ -edge-expander. Orecchia *et al.* [139] strengthened this result by showing that after $O(\log^2 |V|)$ iterations the graph \mathcal{X} is a $\Omega(\log |V|)$ -edge-expander.

Theorem 4.2.6 ([139]). *There is a probabilistic algorithm for the cut player such that, no matter how the matching player plays, after $\gamma_{\text{CMG}}(|V|) = O(\log^2 |V|)$ iterations, the graph \mathcal{X} is an $\Omega(\log |V|)$ -edge-expander with constant probability.*

We use $\gamma_{\text{CMG}}(n)$ to denote the number of iterations of the cut-matching game required in the proof of the preceding theorem for $|V| = n$. Note that the resulting expander is regular with degree equal to $\gamma_{\text{CMG}}(n)$.

Element connectivity and a reduction lemma: Let $G = (V, E)$ be an undirected graph and let V be partitioned into black nodes B and white nodes W . The element connectivity of two black nodes u and v , denoted by $\kappa'_G(u, v)$, is the maximum number of paths in G from u to v that are disjoint in edges and white nodes (the edges and white nodes are the elements). By Menger's theorem, the element connectivity of u and v is equal to the minimum number of elements whose removal disconnects u and v . It is convenient to assume that the black nodes form an independent set by subdividing any edge connecting two black nodes and placing a new white node. In this case $\kappa'_G(u, v)$ for $u, v \in B$ is equal to the maximum number of u - v paths that are disjoint in white nodes. Hind and Oellermann [100] described a graph reduction step that preserves the global element connectivity of the black nodes. Chekuri and Korula [47] generalized this result in order to preserve the pairwise element-connectivity of the black nodes.

Lemma 4.2.7 ([47]). *Let G be an undirected graph and B be a set of black nodes. Let pq be any edge where $p, q \in V(G) - B$ and let $G_1 = G - pq$ and $G_2 = G/pq$, where $G - pq$ is the graph obtained*

from G by removing the edge pq and G/pq is the graph obtained from G by contracting the edge pq . Then one of the following holds: (i) for all $u, v \in B$, $\kappa'_{G_1}(u, v) = \kappa'_G(u, v)$, or (ii) for all $u, v \in B$, $\kappa'_{G_2}(u, v) = \kappa'_G(u, v)$.

The preceding lemma can be used to transform the original graph G into a new graph G' in which the element connectivity of the black nodes is preserved and the white nodes form an independent set; hence G' is bipartite if the black nodes also form an independent set. A white node v in G' corresponds to a connected subgraph of G consisting of only white nodes; this is the subgraph that was contracted to form v .

Degree reduction: For technical reasons we need to work with a graph that has low degree. We accomplish this as follows. Using a standard randomized rounding argument, we can convert a feasible solution to the flow relaxation to another feasible solution such that the flow on each path is either zero or $\Omega(1/\log n)$, while losing only a constant factor in the value of the flow.

Lemma 4.2.8 (Lemma 1.1 in [42]). *Let f be a feasible solution to NDP-LP. Then there is a solution f' such that $|f'| = \Omega(|f|)$ and $f'(p) = 0$ or $f'(p) = \Omega(1/\log n)$ for all $p \in \mathcal{P}$.*

Let G' be the subgraph of G induced by the support of the modified flow f' ; that is, G' consists of all edges of G used in some path p with $f'(p) > 0$. Since each node capacity is 1, the maximum number of edges incident to any node is $O(\log n)$. We will henceforth assume that the graph we are working with has degree at most $c \log n$ for some fixed constant c .

In the following, we show that the degree can also be upper bounded by $O(\log^2 k)$ at a loss of a $O(\log^{5.5} k)$ factor in the approximation ratio using techniques from [42, 116, 146]. If we use this degree reduction procedure instead, the approximation ratio of the algorithm becomes polylogarithmic in k instead of n .

Using the cut-matching game, we can show the following lemma.

Lemma 4.2.9. *Let G be a graph. Let X be an α -node-well-linked set in G of size k . There is an efficient randomized algorithm that constructs a subgraph G' of G such that the maximum degree in G' is $O(\gamma_{\text{CMG}}(k)/\alpha)$. Additionally, X is $\Omega(\alpha^2/\gamma_{\text{CMG}}^2(k))$ -node-well-linked in G' with constant probability.*

Proof. We assume that k is even; if this is not the case, we simply remove a vertex from X . We use the cut-matching game of [116] to construct an expander $\mathcal{X} = (X, F)$ and embed it into G with congestion $O(\gamma_{\text{CMG}}(k)/\alpha)$. The embedding of \mathcal{X} into G will map each vertex of X to itself, and it will map each edge uv of \mathcal{X} to a u - v path in G . The congestion of the embedding is the maximum number of times a node of G appears on the paths associated with the edges of \mathcal{X} .

We use the following strategies for the cut and matching players. The strategy of the cut player is the strategy guaranteed by Theorem 4.2.6. The strategy of the matching player is the following strategy. In each iteration i of the cut-matching game, the cut player chooses a partition (A_i, B_i) of X into two equal-sized sets. When presented with the partition, the matching player chooses a perfect matching between A_i and B_i as follows. Since X is α -node-well-linked in G , it follows from Lemma 4.2.1 that there is a collection of $|A_i|$ paths \mathcal{P}_i in G such that each vertex in A_i is the start vertex of exactly one of the paths, each vertex in B_i is the end vertex of exactly one of the paths, and \mathcal{P}_i has node congestion $O(1/\alpha)$. Moreover, we can compute such a collection \mathcal{P} of paths using a polynomial-time maximum flow computation in G . The endpoints of \mathcal{P}_i define a perfect matching M_i between A_i and B_i , and we let this matching be the response of the matching player. The game stops after $\gamma = \gamma_{\text{CMG}}(k)$ iterations.

Let $\mathcal{P} = \bigcup_{i=1}^{\gamma} \mathcal{P}_i$ and let G' be the subgraph of G whose edge set is the union of the edges that appear on the paths in \mathcal{P} . Let $\mathcal{X} = (X, F)$, where $F = \bigcup_{i=1}^{\gamma} M_i$. Recall that each edge $uv \in F$ corresponds to a u - v path in G ; we let this path be the embedding of the edge into G . Note that \mathcal{X} is embedded into G' with congestion $O(\gamma_{\text{CMG}}(k)/\alpha)$.

By Theorem 4.2.6, \mathcal{X} is an $\Omega(\log k)$ -edge-expander with constant probability; in the following, we assume that this is the case. Since \mathcal{X} is an $\Omega(\log k)$ -edge-expander, X is $\Omega(1)$ -edge-well-linked in \mathcal{X} . Since \mathcal{X} is embedded in G' with congestion $O(\gamma_{\text{CMG}}(k)/\alpha)$, X is $\Omega(\alpha/\gamma_{\text{CMG}}(k))$ -edge-well-linked in G' . Since the maximum degree in G' is $O(\gamma_{\text{CMG}}(k)/\alpha)$, it follows from Proposition 4.2.2 that X is $\Omega(\alpha^2/\gamma_{\text{CMG}}^2(k))$ -node-well-linked in G' . \square

Now we are ready to describe the degree reduction step. Using the well-linked decomposition technique (Theorem 4.2.3) and the grouping technique (Theorem 4.2.4), we may assume that the terminals are $1/4$ -node-well-linked at the loss of a $O(\log^{1.5} k)$ factor in the approximation ratio. Using the algorithm guaranteed by Lemma 4.2.9, we construct a graph G' such that the maximum

degree of G' is $O(\gamma_{\text{CMG}}(k)) = O(\log^2 k)$ and the terminals are $\Omega(1/\gamma_{\text{CMG}}^2(k)) = \Omega(1/\log^4 k)$ -node-well-linked in G' . We use the grouping technique to select a subset of the terminals that are $\Omega(1)$ -node-well-linked in G' ; this final clustering step ensures that the terminals are $\Omega(1)$ -node-well-linked in G' at the loss of another $O(\log^4 k)$ factor in the approximation ratio.

4.3 Expander embedding and routing algorithm

In this section we describe the details of the routing algorithm; the main ingredient is the expander embedding algorithm that relies on the framework and approach of [53]. We state and use two theorems that are proved in subsequent sections. The algorithm can be described as follows.

- (1) Solve the flow relaxation NDP-LP to obtain a fractional solution (x, f) . Use degree reduction (Lemma 4.2.8), well-linked decomposition (Theorem 4.2.3), and grouping (Theorem 4.2.4) to reduce the original instance to a collection of separate instances such that the graph in each instance has maximum degree $\Delta = O(\log n)$ and the terminals are $1/4$ -node-well-linked.
- (2) Let (G, \mathcal{M}) have k pairs such that the terminals are $1/4$ -node-well-linked and $\Delta(G) = O(\log n)$. Embed, with $O(1)$ congestion, an expander in G of size $k' = \Omega(k/\text{poly}(\Delta, \log k))$ and degree $\text{poly} \log(k)$.
- (3) Use the expander to route $\Omega(k'/\text{poly} \log k)$ pairs from \mathcal{M} with $O(1)$ congestion.

The first step incurs an $O(\log^{1.5} k)$ -factor loss in the approximation. The heart of the matter is the second step. Following the outline in Subsection 4.1.1, it consists of two steps: (i) finding a node-disjoint collection of “good” clusters to simulate the KRV cut-matching game, and (ii) finding representatives in each cluster and connecting them via (nearly-disjoint) trees to simulate each node of the expander. We start with the clustering step and we prove Theorem 5.1.1 at the end of the section.

4.3.1 Family of good clusters

We extend the definition of a good set from [53] to the node-capacitated setting as follows; we refer to such sets as clusters.

Definition 4.3.1. A subset $S \subseteq V(G) - \mathcal{T}$ of nodes is a (h, α) -good-cluster iff there is a subset $B \subseteq \text{bd}_G(S)$ of nodes with the following properties:

- $|B| \geq h$.
- B is α -node-well-linked in $G[S]$.
- There is a collection of B - \mathcal{T} paths \mathcal{P} in G that are node-disjoint.

The subset $B \subseteq \text{bd}_G(S)$ of boundary nodes is part of the definition of a good cluster. In the following, when we say that we are given a good cluster S , we mean that we are given the set S and the subset $B \subseteq \text{bd}_G(S)$. The parameters in the definition give flexibility in finding good clusters; the grouping technique allows us to boost the well-linkedness later.

Theorem 4.3.2 below shows that one can find a family of node-disjoint good clusters in graph G (with appropriate parameters) if G has a well-linked set. We prove it in Section 4.4 via a corresponding theorem in [53] for the edge-capacitated case.

Theorem 4.3.2. Let G be a graph that contains a set \mathcal{T} of nodes with the following properties: \mathcal{T} is $1/4$ -node-well-linked in G , $|\mathcal{T}| = 2k$, and each node in \mathcal{T} has degree one in G . Let Δ be the maximum degree in G . There is an efficient randomized algorithm that with high probability constructs a family $\mathcal{F} = \{S_1, \dots, S_\gamma\}$ of $\gamma = \gamma_{\text{CMG}}(k) = \Theta(\log^2 k)$ node-disjoint sets such that each S_i is a $(k^*, 1/4)$ good cluster, where $k^* = \Omega\left(\frac{k}{\Delta^3 \log^{14} k \log \log k}\right)$. Moreover, the algorithm also outputs for each S_j a set $B_j \subset \text{bd}_G(S_j)$ such that (i) $|B_j| = k^*$, (ii) B_j is $1/4$ -well-linked in $G[S_j]$, and (iii) there is a collection of node-disjoint $B_j - \mathcal{T}$ paths in G .

Remark 4.3.3. There is a technical requirement in the preceding theorem that Δ is sufficiently small compared to k . A poly-logarithmic approximation is easy if the condition is not satisfied. This is explained in Section 4.4.

4.3.2 Connecting the good sets and expander embedding

We now describe the algorithm that uses the good clustering from the previous subsection to embed an expander \mathcal{X} in G . We work with two parameters, k^* and k' , where k^* is the parameter guaranteed by Theorem 4.3.2; it is helpful to simply think of k^* as $k/\text{poly} \log(k + n)$. We set

$k' = k^*/(6\gamma^4)$; we round k' down to the nearest even integer. The embedding follows the approach from [53, 146].

The expander $\mathcal{X} = (V(\mathcal{X}), E(\mathcal{X}))$ has k' nodes $v_1, v_2, \dots, v_{k'}$. The embedding maps each node $v_i \in V(\mathcal{X})$ to a connected subgraph C_i of G ; the k' connected subgraphs $C_1, \dots, C_{k'}$ are nearly disjoint in that each node of G appears in only a constant number of them. The embedding of each edge $v_i v_{i'} \in E(\mathcal{X})$ is a path of G connecting C_i to $C_{i'}$; the paths corresponding to the edges of \mathcal{X} also have constant node congestion in G . We also need the expander to be reachable from the terminals. For this purpose an additional property that is guaranteed is that each C_i contains a unique terminal; by relabeling terminals C_i contains $t_i \in \mathcal{T}$. We can thus identify v_i with the terminal t_i and interpret the expander as being embedded on a subset of the terminals.

Recall that the plan for embedding the expander is to simulate iteration j of the KRV cut-matching game [116] in cluster S_j (thus the number of clusters is $\gamma = \gamma_{\text{CMG}}(k)$) using the well-linked set $B_j \subset \text{bd}_G(S_j)$. Each node v_i of \mathcal{X} has a representative $b_{i,j} \in B_j$ for each j such that $1 \leq j \leq \gamma$, and the subgraph C_i connects the nodes $b_{i,1}, \dots, b_{i,\gamma}$ and t_i . In fact, the algorithm first finds the nearly-disjoint connected subgraphs $C_1, \dots, C_{k'}$ such that each C_i has a representative in each B_j ; the well-linkedness of B_j implies that the identity of the representative for C_i in B_j is not important. The theorem, whose proof is given in Section 4.5, formally states the properties of the polynomial time algorithm that finds the desired sets $C_1, \dots, C_{k'}$.

Theorem 4.3.4. *Let $\mathcal{F} = \{S_1, S_2, \dots, S_\gamma\}$ be the good clusters guaranteed by Theorem 4.3.2. There is a polynomial time algorithm that finds a subset $\mathcal{T}' \subset \mathcal{T}$ of k' terminals, connected subgraphs $C_1, \dots, C_{k'}$, and a collection of node sets $D_1, \dots, D_{k'}$ with the following properties.*

- *Each node of G belongs to at most 43 of the subgraphs $C_1, \dots, C_{k'}$.*
- *For each i such that $1 \leq i \leq k'$, $D_i \subset C_i$ and $D_i = \{b_{i,1}, b_{i,2}, \dots, b_{i,\gamma}\}$, where $b_{i,j} \in B_j$ for all j such that $1 \leq j \leq \gamma$.*
- *The sets $D_1, \dots, D_{k'}$ are mutually disjoint.*
- *We can label the terminals in \mathcal{T}' as $t_1, \dots, t_{k'}$ such that $t_i \in V(C_i)$ for each i such that $1 \leq i \leq k'$.*

Theorem 4.3.4 gives us an embedding of the nodes of the edge-expander in which each node v_i is embedded into G using a connected subgraph containing the terminal t_i . We use the cut-matching game of Theorem 4.2.6 to define the edges of \mathcal{X} and an embedding of these edges into G .

Recall that we have γ good clusters $S_1, S_2, \dots, S_\gamma$, where $\gamma = \gamma_{\text{CMG}}(k) \geq \gamma_{\text{CMG}}(k')$. We use the cut-matching game as follows. The cut player will follow the strategy guaranteed by Theorem 4.2.6. In each iteration j of the cut-matching game, the algorithm implements the matching-player as follows. The matching player receives a partition of $V(\mathcal{X})$ into two sets Y_j and Z_j of equal size and needs to find a perfect matching between them. Let $Y'_j = \{b_{i,j} : v_i \in Y\} \subset D_j$ be the representatives in D_j of the expander nodes Y_j , and similarly let $Z'_j = \{b_{i,j} : v_i \in Z\} \subset D_j$ be the representatives in D_j of the expander nodes Z_j . From Theorem 4.3.4, the sets Y'_j and Z'_j are disjoint and D_j is $1/4$ -node-well-linked in $G[S_j]$. Hence there is a collection \mathcal{P} of Y'_j - Z'_j paths in $G[S_j]$ with node congestion 4 (moreover, given Y'_j, Z'_j such a collection of paths can be found in polynomial time via a maximum-flow algorithm). This collection of paths induces a perfect matching M'_j between Y'_j and Z'_j and hence also a perfect matching M_j between Y_j and Z_j ; each edge $u'v' \in M'_j$ corresponds to an edge uv in M_j where u' is the representative of u in D_j and v' is the representative of v in D_j . The matching player outputs M_j in iteration j . We associate each edge $uv \in M_j$ with the path between u' and v' in $G[S_j]$.

It follows from Theorem 4.2.6 that, after γ iterations, the graph \mathcal{X} is an edge-expander with constant probability. Since the sets S_1, \dots, S_γ are node disjoint, the collection of paths in G corresponding to all the edges added to \mathcal{X} in the cut-matching game has node congestion 4. Based on the preceding argument we obtain the following theorem on embedding \mathcal{X} .

Theorem 4.3.5. *There exists a set $\mathcal{T}' \subseteq \mathcal{T}$ of k' terminals and a graph \mathcal{X} with node set \mathcal{T}' with the following properties.*

- *The graph \mathcal{X} is a γ -regular $\Omega(\log k')$ -edge-expander.*
- *For each $t_i \in \mathcal{T}'$ there is a connected subgraph C_i of G such that C_i contains t_i and the node congestion of the subgraphs $\{C_i \mid 1 \leq i \leq k'\}$ is at most 43.*
- *For each edge $e = v_i v_{i'} \in E(\mathcal{X})$, there is a path q_e in G connecting C_i to $C_{i'}$ such that the node congestion of the paths $\{q_e \mid e \in E(\mathcal{X})\}$ is at most 4.*

Moreover, there is a randomized polynomial time algorithm that constructs a set \mathcal{T}' and a graph \mathcal{X} with these properties with constant probability.

4.3.3 Routing using the embedded expander

Let \mathcal{T}' and \mathcal{X} be the set of terminals and the edge-expander guaranteed by Theorem 4.3.5. We can use the edge-expander \mathcal{X} to route a large subset of the pairs of \mathcal{M} .

Theorem 4.3.6. *Let $\mathcal{M}_0 \subseteq \mathcal{M}$ be any subset of $k'/2$ pairs. There is a randomized polynomial time algorithm that, with high probability, routes in G a subset of $\Omega(|\mathcal{M}_0|/\gamma^2)$ of the pairs in \mathcal{M}_0 with node congestion at most 51.*

Proof. Let \mathcal{T}_0 denote the set of all terminals participating in the pairs in \mathcal{M}_0 . Note that $|\mathcal{T}_0| = 2|\mathcal{M}_0| = k'$. Since the set \mathcal{T} of terminals is $1/4$ -node-well-linked in G , it follows from Lemma 4.2.1, there is a collection \mathcal{P} of $\mathcal{T}_0 - \mathcal{T}'$ paths with congestion 4. Using the paths in \mathcal{P} we can translate the matching \mathcal{M}_0 to a matching \mathcal{M}' on \mathcal{T}' as follows: for any pair $uv \in \mathcal{M}_0$, we add the pair $u'v'$ to \mathcal{M}' , where u' and v' are the nodes of \mathcal{T}' that are the endpoints of the paths of \mathcal{P} that start at u and v , respectively.

Since \mathcal{X} is a γ -regular $\Omega(\log k')$ -edge-expander, we can route a subset $\mathcal{M}'' \subseteq \mathcal{M}'$ of $\Omega(|\mathcal{M}_0|/\gamma^2)$ demand pairs on node-disjoint paths of \mathcal{X} (see Theorem 4.2.5). Let \mathcal{P} be the collection of these paths. We map these paths to a collection of paths \mathcal{P}' in G resulting in a routing of \mathcal{M}'' in G . Let p be a $t_i-t_{i'}$ path in \mathcal{X} for a pair $(t_i, t_{i'}) \in \mathcal{M}''$. Let $e = t_h t_\ell$ be an edge on p . From Theorem 4.3.5 there is a path $q(e)$ in G connecting C_h and C_ℓ ; since C_h and C_ℓ are connected subgraphs of G containing t_h and t_ℓ respectively, there is a t_h-t_ℓ path $q'(e)$ in G whose nodes are contained in $C_h \cup C_\ell \cup q(e)$. We map the $t_i-t_{i'}$ path p in \mathcal{X} to a $t_i-t_{i'}$ walk p' in G obtained by replacing each edge $e \in p$ by the path $q'(e)$; this walk contains a simple $t_i-t_{i'}$ path in G . This procedure, done separately for each path $p \in \mathcal{P}$, gives the desired path collection \mathcal{P}' for routing the pairs \mathcal{M}'' in G . It is easy to see that \mathcal{P}' can be generated efficiently given the embedding from Theorem 4.3.5. We now consider the node congestion in G caused by the path collection \mathcal{P}' . Since the paths in \mathcal{P} are node disjoint in \mathcal{X} (and hence also trivially edge disjoint) the node congestion of \mathcal{P}' is upper bounded by the sum of the node congestion of the collection $\{C_i \mid 1 \leq i \leq k'\}$ and $\{q(e) \mid e \in E(\mathcal{X})\}$, which, by Theorem 4.3.5, is at most $43 + 4 = 47$.

The paths in $\mathcal{P}(\mathcal{T}_0, \mathcal{T}')$ concatenated with the paths corresponding to the routing of \mathcal{M}'' in G gives a routing with congestion $47 + 4 = 51$ of a subset $\mathcal{M}_1 \subseteq \mathcal{M}_0$ of size $\Omega(|\mathcal{M}_0|/\gamma^2)$. Theorem 4.3.5 guarantees that \mathcal{X} has the desired expansion properties with constant probability. One can independently repeat the expander embedding algorithm and the subsequent routing via the expander, to obtain a high probability bound on the success of routing a poly-logarithmic fraction of the pairs. \square

We can now put the ingredients together to prove our main result.

Proof of Theorem 5.1.1: Let (G, \mathcal{M}) be an instance of MNDP on a graph with n nodes and k pairs. We follow the outline of the algorithm stated at the beginning of this section. The algorithm solves the NDP-LP relaxation to obtain an optimal fractional solution (x, f) . Let OPT be the value of this solution. First, we assume that the number of pairs and OPT are at least $\log^c n$ for a sufficiently large constant c , since otherwise we can obtain a poly-logarithmic approximation by routing an arbitrarily chosen pair from \mathcal{M} (if no pair can be connected in G then $\text{OPT} = 0$ and the problem is trivial). We then use the fractional solution and apply the degree reduction and well-linked decomposition to reduce the given instance to a collection of separate instances on subgraphs G_1, \dots, G_ℓ of G such that the resulting instances are $1/4$ -node-well-linked for the terminals and the graph in each instance has degree $O(\log n)$; an α -approximation for these restricted instances implies an $O(\alpha \log^{1.5} k)$ -approximation for the original instance (from Theorem 4.2.3 and Theorem 4.2.4) where the approximation is with respect to the fractional solution value OPT.

Given a well-linked instance with k terminals and a graph with n nodes and maximum degree $O(\log n)$, Theorems 4.3.2, 4.3.4, and 4.3.5 together give an efficient algorithm that embeds an expander \mathcal{X} of size k' in G with constant congestion where $k' = \Omega(k/\text{poly log}(k+n))$; the expansion of \mathcal{X} is $\Omega(\log k')$. Theorem 4.3.6 shows that the expander can be used to route $\Omega(k'/\log^4 k)$ pairs from the given instance with congestion 51. Thus, the algorithm routes $\Omega(k/\text{poly log}(k+n))$ pairs in G with congestion 51. The dependence on n in the approximation ratio is due to the fact that the maximum degree is $\Delta = O(\log n)$. As shown in Section 4.2, we can also ensure that $\Delta = O(\log^2 k)$, which leads to an efficient algorithm that routes $\Omega(\text{OPT}/\text{poly log } k)$ pairs. \square

4.4 Proof of Theorem 4.3.2 on good clustering

Chuzhoy [53] gave a clustering algorithm for the edge-capacitated case. We believe there should be a “natural” extension of it to the node-capacitated case; however, the proof in [53] is rather technical and non-trivial. Here, we use her result in a black-box fashion and take advantage of the fact that edge-well-linkedness and node-well-linkedness can be related if we have an upper bound on the degree; we lose factors that are polynomial in the degree in this translation; since the degree bound we have is $O(\log n)$, it affects the final approximation ratio by only a poly-logarithmic factor.

Chuzhoy [53] uses the following definition of well-linkedness, which we call *edge-well-linkedness*. For a set S of nodes, we let $\text{out}_G(S)$ denote the set of all edges of G with an endpoint in S and the other endpoint outside of S .

Definition 4.4.1 ([53]). *Let G be a graph and let S be a set of nodes. Let $F \subseteq \text{out}_G(S)$ be a set of edges. We say that S is α -edge-well-linked for F iff, for any partition (X, Y) such that $X \cup Y = S$, we have*

$$|E_G(X, Y)| \geq \alpha \cdot \min\{|\text{out}_G(X) \cap F|, |\text{out}_G(Y) \cap F|\}$$

where $E_G(X, Y)$ is the set of all edges of G with one endpoint in X and the other in Y .

We observe that Chuzhoy’s definition is equivalent to the following. Subdivide each edge $e \in F$ using a node v_e ; let X be the set of these new nodes. The set S is α -edge-well-linked for F iff X is α -edge-well-linked (according to the definition given in Section 4.2) in the graph $G[S \cup X]$.

One of the main technical ingredients of the algorithm of [53] is a clustering procedure that selects a family of $\gamma = \gamma_{\text{CMG}} = \Theta(\log^2 k)$ disjoint subsets of nodes called *good subsets*. Following [53], we use the parameters $k_1 = \Omega\left(\frac{k}{\log^6 k \log \log k}\right)$ and $\alpha_{\text{WL}}(k) = \Omega\left(\frac{1}{\log^{3.5} k}\right)$.

Definition 4.4.2 (Definition 4 in [53]). *A subset $S \subseteq V(G) - \mathcal{T}$ of nodes is a good subset iff there is a subset $\Gamma \subseteq \text{out}_G(S)$ of edges with the following properties:*

- $|\Gamma| = k_1$.
- S is $\alpha_{\text{WL}}(k)$ -edge-well-linked for Γ .
- There is a flow F in graph G , where every edge $e \in \Gamma$ sends one flow unit to a distinct terminal $t_e \in \mathcal{T}$ (so for $e \neq e'$, $t_e \neq t_{e'}$), and the congestion caused by F is at most $O(\log^{4.5} k)$.

A family $\mathcal{F} = \{S_1, \dots, S_\gamma\}$ of $\gamma = \gamma_{\text{CMG}}(k) = \Theta(\log^2 k)$ subsets of nodes is good iff each subset S_j is a good subset of nodes of G , and S_1, \dots, S_γ are pairwise disjoint.

Theorem 4.4.3 (Corollary 2 in [53]). *Let G be a graph that contains a set \mathcal{T} such that $|\mathcal{T}| = 2k$ and \mathcal{T} is $1/4$ -edge-well-linked in G . If the maximum degree Δ of G is at most k_1 , there is a polynomial time randomized algorithm that computes a good family of subsets in G with high probability.*

Remark 4.4.4. In [53] the statement of Theorem 4.4.3 assumes that \mathcal{T} is flow-well-linked. However, the proof only uses cut-well-linkedness. Further, this distinction is not crucial since flow and cut well-linkedness are approximately the same (within a logarithmic factor). We refer the reader to [42] for a definition of flow and cut well-linkedness and the approximate equivalence between the two notions. Additionally, \mathcal{T} is assumed to be $1/2$ -edge-well-linked. Replacing the $1/2$ by $1/4$ only weakens the parameters for the good sets by a constant factor. Alternatively, we can make a copy of each edge of the graph in order to boost the well-linkedness of \mathcal{T} from $1/4$ to $1/2$ by losing a constant factor in the congestion.

In our setting we have $\Delta = O(\log n)$. We will assume that $\Delta \leq k_1$, since otherwise $k = O(\text{poly } \log n)$ and it is trivial to get a k approximation for MNDP by simply routing one pair.

The following propositions relate the notions of edge and node well-linkedness, and they are straightforward to verify.

Proposition 4.4.5. *Let G be a graph with maximum degree Δ . Let S be a set of nodes and let $F \subseteq \text{out}_G(S)$ be a set of edges. Let B be the set of all endpoints of edges in F that are in S . If S is α -edge-well-linked for F then B is $\Omega(\alpha/\Delta)$ -node-well-linked in $G[S]$, where $G[S]$ is the subgraph of G induced by S .*

Proposition 4.4.6. *If X is α -node-well-linked in G then X is α -edge-well-linked in G .*

We can use Theorem 4.4.3 to complete the proof of Theorem 4.3.2 as follows.

Proof of Theorem 4.3.2: Since \mathcal{T} is $1/4$ -node-well-linked in G , it follows from Proposition 4.4.6 that \mathcal{T} is $1/4$ -edge-well-linked in G . Therefore G and \mathcal{T} satisfy the conditions of Theorem 4.4.3. Let $\mathcal{F} = \{S_1, \dots, S_\gamma\}$ be the good family guaranteed by Theorem 4.4.3. These will be our good clusters, however, the parameters will be weaker and we also need to identify a set $B_j \subset \text{bd}_G(S_j)$ for each S_j .

For each set $S_j \in \mathcal{F}$, we have a subset $\Gamma_j \subseteq \text{out}_G(S_j)$ of edges. For each index j , let B_j be the set of all endpoints of the edges in Γ_j that are in S_j . We select a subset $B_j'' \subseteq B_j$ such that S_j and B_j'' form a $(k^*, 1/4)$ -good-cluster as follows.

Consider an index j . We start by selecting a subset $B_j' \subseteq B_j$ such that there is a collection of B_j' - \mathcal{T} paths in G that are node-disjoint. Recall that there is a flow F in G where each edge $e \in \Gamma_j$ sends one flow unit to a distinct terminal in \mathcal{T} and the edge congestion caused by F is at most $O(\log^{4.5} k)$. We reinterpret the flow F as originating at the nodes in B_j and ending in \mathcal{T} . Note that, since the maximum degree in G is Δ , the node congestion caused by F is at most $O(\Delta \log^{4.5} k)$. Additionally, every node in B_j sends at least one unit of flow and at most Δ units of flow. Thus, if we scale down the flow F by $O(\Delta^2 \log^{4.5} k)$, we get a flow F' from B_j to \mathcal{T} that respects node capacities (1 on every node) of the graph including the endpoints B_j and \mathcal{T} . We can use the flow F' to select the subset B_j' as follows. We add a source node s and an edge from s to each node in B_j . We add a sink t and an edge from each node in \mathcal{T} to t . We assign a capacity of one to each node. Note that the flow F' gives us a feasible s - t flow of value $\Omega(k_1/(\Delta^2 \log^{4.5} k))$. Since the node capacities are integral, there is an integral s - t flow of value $\Omega(k_1/(\Delta^2 \log^{4.5} k))$. We take a path decomposition of such an integral flow in order to get a collection of s - t paths that are internally node-disjoint; by removing the endpoints s, t from these paths, we get a collection of node-disjoint paths in G connecting a subset $B_j' \subseteq B_j$ to \mathcal{T} , where $|B_j'| = \Omega(k_1/(\Delta^2 \log^{4.5} k))$.

Since S_j is $\alpha_{\text{WL}}(k)$ -edge-well-linked for Γ_j , it follows from Proposition 4.4.5 that B_j is $\Omega(\alpha_{\text{WL}}(k)/\Delta)$ -node-well-linked in $G[S_j]$. We apply Theorem 4.2.4 to B_j' in order to get a subset $B_j'' \subseteq B_j'$ such that B_j'' is $1/4$ -node-well-linked in $G[S_j]$ and $|B_j''| = \Omega(|B_j'| \alpha_{\text{WL}}(k)/\Delta) = \Omega(k_1 \alpha_{\text{WL}}(k)/(\Delta^3 \log^{4.5} k))$.

Therefore the set S_j together with the boundary set $B_j'' \subseteq \text{bd}_G(S_j)$ gives us a $(k^*, 1/4)$ -good-cluster, where $k^* = \Omega(k_1 \alpha_{\text{WL}}(k)/(\Delta^3 \log^{4.5} k))$. \square

4.5 Proof of Theorem 4.3.4 on connecting good clusters

We recall the properties of good clusters guaranteed by Theorem 4.3.2. There are γ good clusters S_1, \dots, S_γ ; each S_j has a set $B_j \subset \text{bd}_G(S_j)$ such that B_j is $1/4$ -node-well-linked in $G[S_j]$ and there is a collection of $B_j - \mathcal{T}$ node disjoint paths in G . Recall that $|B_j| \geq k^*$ for $1 \leq j \leq \gamma$. In this

section we assume that $|B_j| = k^*$, which we can ensure by selecting arbitrarily a subset of B_j of cardinality k^* ; this will be important later.

We prove Theorem 4.3.4 in this section; it guarantees $k' = k^*/(64\gamma^2)$ connected subgraphs $C_1, \dots, C_{k'}$ in G . Each C_i has a representative $b_{i,j}$ in B_j for each j . At a high level we find these subgraphs via the same approach as that in [53]; Chuzhoy uses edge-connectivity based techniques to find many trees, each of which has a boundary node from each good set. We use element-connectivity techniques to find many trees, each of which has a boundary node from each good cluster. Moreover, each node of G is in at most a constant number of these trees. Once we have the trees, we connect a subset of the terminals to the trees to get the desired connected subgraphs.

To construct the trees, we first create a new graph G' from G as follows. We add γ new (super-)nodes $s_1, s_2, \dots, s_\gamma$; for $1 \leq j \leq \gamma$, s_j is connected to each node in B_j . We think of the nodes of G' as being partitioned into black and white nodes; the black nodes are the new super-nodes and the white nodes are the nodes of G . We note that the degree of each black node is exactly equal to k^* . We refer the reader to element-connectivity definitions from Section 4.2.

Proposition 4.5.1. *For any two black nodes s_i and s_j , we have $\kappa'_{G'}(s_i, s_j) \geq \lceil k^*/6 \rceil$; that is, s_i and s_j are $\lceil k^*/6 \rceil$ element-connected in G' .*

Proof. Consider B_i and B_j ; we show a collection of B_i - B_j paths \mathcal{P} in G such that any node in G is in at most 6 paths in \mathcal{P} . This implies that there are $\lceil k^*/6 \rceil$ paths from s_i to s_j in G' that are disjoint in the white nodes, which proves the proposition. Recall that in G there is a collection \mathcal{P}_1 of node-disjoint B_i - \mathcal{T} paths and similarly there is a collection \mathcal{P}_2 of node disjoint B_j - \mathcal{T} paths. Let $Y \subset \mathcal{T}$ be the endpoints of the paths in \mathcal{P}_1 and $Z \subset \mathcal{T}$ be the endpoints of the paths in \mathcal{P}_2 . Note that $|Y| = |Z| = k^*$. Since \mathcal{T} is $1/4$ -node-well-linked in G , it follows from Lemma 4.2.1 that there is a collection of Y - Z paths \mathcal{P}_3 with node congestion 4. We obtain a collection of paths \mathcal{P} by concatenating the paths in \mathcal{P}_1 , \mathcal{P}_3 , and \mathcal{P}_2 in the natural way. That is, if p is a u - t path in \mathcal{P}_1 from a node $u \in B_i$ to a terminal $t \in Y$, q is a t - t' path in \mathcal{P}_3 from t to $t' \in Z$, and p' is a t' - v path in \mathcal{P}_2 from t' to $v \in B_j$ then we obtain a u - v path in \mathcal{P} from the union of the paths p, q, p' . The node congestion of \mathcal{P} is at most 6 since \mathcal{P}_1 and \mathcal{P}_2 have congestion 1, and \mathcal{P}_3 has congestion at most 4. \square

We now apply the element-connectivity reduction step from Lemma 4.2.7 to G' . This results in a bipartite graph G'' in which the element-connectivity between each pair of black nodes is at least $\lceil k^*/6 \rceil$. Moreover, each white node v in G'' is obtained by contracting a connected subgraph of G . We now create an auxiliary graph H as follows. The node set of H is $V(H) = \{s_1, s_2, \dots, s_\gamma\}$. There is an edge $s_i s_j$ in H iff there are at least $k^*/(6\gamma^3)$ white nodes v in G'' such that v is adjacent to both s_i and s_j . Theorem 4.5.2 captures the important facts about the auxiliary graph, in particular the existence of a constant degree spanning tree that will be used as a “template” to find many trees.

Theorem 4.5.2. *There is a spanning tree T^* in H with the following properties:*

- *The maximum degree of T^* is at most 10.*
- *For each edge $e = s_i s_j$ of T^* , there is a collection \mathcal{P}_e of at least $k^*/(6\gamma^4)$ paths of G' such that, for each path $p \in \mathcal{P}_e$, the endpoints of p are s_i and s_j , and the internal nodes of p are white.*
- *The paths in $\mathcal{P} = \cup_{e \in E(T^*)} \mathcal{P}_e$ are disjoint in white nodes.*

Moreover, we can find the tree T^* and the collections of paths $\{\mathcal{P}_e \mid e \in E(T^*)\}$ in polynomial time.

Remark 4.5.3. Chuzhoy [53] uses the edge-connectivity preserving splitting-off operation [76, 105, 133] to remove all white nodes to directly obtain an auxiliary graph on the super-nodes and then uses a theorem of Singh and Lau [158] to find a spanning tree of degree at most 3 in the auxiliary graph (via a feasible solution to an LP relaxation). The presence of white nodes in the graph G' which have different degrees does not allow us to use a similar argument. Hence, we rely on a different argument based on the notion of toughness; this gives a bound of 10 on the degree of the spanning tree (which affects the final congestion bound) and we also lose additional poly-logarithmic factors in the approximation.

We give the proof of Theorem 4.5.2 in Subsection 4.5.1. Using Theorem 4.5.2, we can complete the proof of Theorem 4.3.4 as follows.

Proof of Theorem 4.3.4: Let T^* be the tree guaranteed by Theorem 4.5.2. We first consider the special case in which T^* is a path; as we will see shortly, we can extend the argument for this

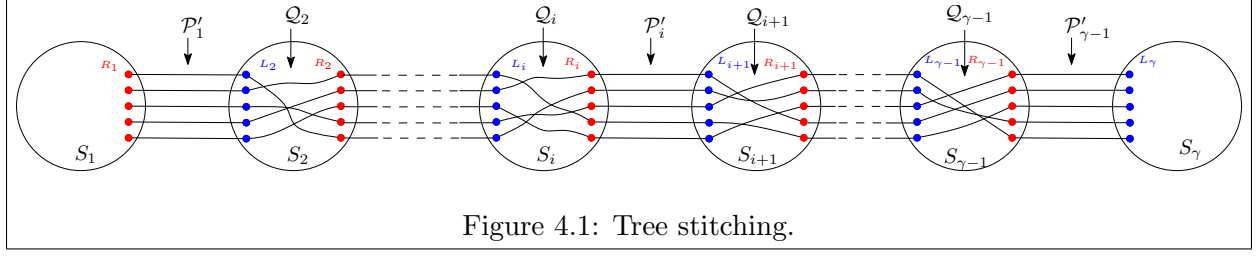


Figure 4.1: Tree stitching.

special case to the general case by making a copy of each edge of T^* and considering an Eulerian walk of the resulting graph.

Suppose that T^* is a path. We think of the nodes and edges of T^* as being ordered from left to right. By relabeling the nodes, we may assume that the nodes of T^* are $s_1, s_2, \dots, s_\gamma$ from left to right. Let \mathcal{P}_j be the collection of $k' = k^*/(6\gamma^4)$ s_j - s_{j+1} paths guaranteed by Theorem 4.5.2. By removing the endpoints of the paths in \mathcal{P}_j , we get a collection \mathcal{P}'_j of R_j - L_{j+1} paths in G where $R_j \subseteq B_j$ and $L_{j+1} \subseteq B_{j+1}$. (Recall that $B_j \subseteq \text{bd}_G(S_j)$ is the set of boundary nodes of the good cluster S_j and s_j is connected only to B_j in G' .) Note that the paths in $\mathcal{P}' = \uplus_j \mathcal{P}'_j$ are node disjoint. From this it follows that the node sets $R_1, L_2, R_2, L_3, R_3, \dots, L_{\gamma-1}, R_{\gamma-1}, L_\gamma$ are disjoint and all have the same cardinality k' . Since $|L_j| = |R_j| = k'$ and B_j is $1/4$ -node-well-linked in $G[S_j]$, there is a collection \mathcal{Q}_j of L_j - R_j paths that are contained in $G[S_j]$ and they have node congestion at most 4. These paths can be concatenated together to generate k' walks in G (see Figure 4.1). Once we have the walks, we attach a subset of the terminals in order to get the connected subgraphs $C_1, \dots, C_{k'}$.

In the following, we give a more formal overview of the stitching described in Figure 4.1. The path collection \mathcal{P}'_j defines a perfect matching M_j between R_j and L_{j+1} , and the path collection \mathcal{Q}_j defines a perfect matching M'_j between L_j and R_j . Consider the layered graph with node set $R_1 \uplus \left(\uplus_{j=2}^{\gamma-1} L_j \uplus R_j \right) \uplus L_\gamma$ and edge set $M_1 \cup \left(\cup_{j=2}^{\gamma-1} M'_j \cup M_j \right)$. Each layer has the same cardinality k' and the edge set consists of perfect matchings between adjacent layers. It is easy to see that the edge set can be decomposed into k' paths where each path consists of a sequence of nodes, one from each layer, starting with a node in the first layer R_1 and ending at the last layer L_γ ; these paths also partition the nodes. Let $p_1, \dots, p_{k'}$ be these paths. Let $p_i = v_{i,1}, u_{i,2}, v_{i,2}, \dots, u_{i,\gamma-1}, v_{i,\gamma-1}, u_{i,\gamma}$ where $u_{i,j}$ is the node from L_j on p_i and $v_{i,j}$ is the node from R_j on p_i . For each i we obtain a connected subgraph (in fact a walk) C_i in G by replacing the edges of p_i with corresponding paths

from G : an edge $v_{i,j}u_{i,j+1}$ in p_i corresponds to a unique path in \mathcal{P}'_j and an edge $u_{i,j}v_{i,j}$ corresponds to a unique path in \mathcal{Q}'_j . Recall that the paths in \mathcal{P}' are node-disjoint in G and the paths in \mathcal{Q}'_j are contained in $G[S_j]$ and have node congestion 4. It therefore follows that no node of G is in more than 5 of the subgraphs $C_1, \dots, C_{k'}$. We also need to choose $D_i \subset C_i$ such that $|D_i \cap B_j| = 1$ for each $1 \leq j \leq \gamma$ and such that $D_1, \dots, D_{k'}$ are disjoint; we let $D_i = \{v_{i,j} | 1 \leq j < \gamma\} \cup \{u_{i,\gamma}\}$; in other words we set $b_{i,j} = v_{i,j}$ for $1 \leq j < \gamma$ and $b_{i,\gamma} = u_{i,\gamma}$. It is easy to verify that $D_1, \dots, D_{k'}$ satisfy the desired properties.

Finally, we need to ensure that each C_i contains a distinct terminal. Let $B'_1 = \{b_{i,1} | 1 \leq i \leq k'\} \subseteq B_1$. Since S_1 is a good cluster, there is a collection of node-disjoint paths in G connecting B'_1 to a subset $\mathcal{T}' \subseteq \mathcal{T}$; for each i , we add to C_i the path connecting $b_{i,1}$ to \mathcal{T} . This final step increases the congestion by 1 so we have that no node is in more than 6 of the subgraphs $C_1, C_2, \dots, C_{k'}$.

Now we extend the argument to the case in which T^* is not a path. We make a copy of each edge of T^* and consider an Eulerian walk of the resulting Eulerian graph. We view this walk as a path (after removing the final edge in the walk) in which each edge of T^* appears at most twice and each node s_i of T^* appears at most 10 times (the degree of s_i to be precise). We apply the previous argument to this path. For this purpose each edge and node of T^* that occurs more than once is assumed to be distinct; the path collection associated with each edge and node of T^* in the previous argument will use separate copies of the nodes of G . We will subsequently analyze the congestion caused by these copies. The path argument gives k' connected components $C_1, \dots, C_{k'}$ and k' sets $D_1, \dots, D_{k'}$. We claim that the connected components have node congestion at most $1 \cdot 2 + 4 \cdot 10 + 1 = 43$. Since each edge of T^* appears twice in the Eulerian walk, the stitching described in Figure 4.1 uses the paths of $\{\mathcal{P}'_e | e \in E(T^*)\}$ at most twice; thus we have a congestion of at most 1.2 from these paths. Since each node s_i appears at most 10 times in the Eulerian walk, the stitching uses the subgraph $G[S_i]$ at most 10 times. Each of those uses requires a collection of paths \mathcal{Q}'_i between two disjoint sets $L_i, R_i \subset B_i$; since B_i is $1/4$ -node-well-linked in $G[S_i]$ such a path collection with node congestion 4 can be found. Hence the overall congestion of all these paths in $G[S_i]$ is $4 \cdot 10$; since the good clusters $S_1, S_2, \dots, S_\gamma$ are disjoint, the total congestion of the union of these paths is also upper bounded by $4 \cdot 10$. Finally, we use a collection of node-disjoint

paths to connect a subset of B_1 to a subset of \mathcal{T} . □

Remark 4.5.4. In the proof of Theorem 4.3.4, we used an Eulerian walk of T^* in order to make the argument more transparent. In order to get an Eulerian walk, we duplicated the edges of T^* . Instead, we can root T^* at an arbitrary leaf and stitch the paths in a bottom-up fashion in order to get the desired connected components; this is the scheme used in [53] and requires a more involved description and proof. The bottom-up argument avoids duplicating the edges of T^* and therefore it improves the congestion of the resulting connected subgraphs by 1.

4.5.1 Proof of Theorem 4.5.2

First, we show that H has a spanning tree T^* that has constant degree. Chvátal [58] introduced the notion of graph toughness. Win [170] showed existence of a low-degree spanning tree in a graph where the degree bound was related to its toughness.

The toughness of a graph G , denoted by $\tau(G)$, is defined as follows. Consider a subset $S \subset V(G)$ of the nodes of G . Let $c(S)$ denote the number of connected components of the graph $G - S$ obtained from G by removing the nodes in S . The *toughness* of G is the ratio $\tau(G) = \min_{S \subset V} |S|/c(S)$, where the minimum is taken over all sets $S \subset V$ such that $c(S) > 1$. We use the following result of Fürer and Raghavachari [80] that is slightly stronger than the result of Win [170].

Theorem 4.5.5 ([80]). *Let G be a graph and let $\tau(G)$ be its toughness. Let Δ^* be the smallest number such that G has a spanning tree with maximum degree Δ^* . Then $\Delta^* - 3 < 1/\tau(G) \leq \Delta^*$.*

Fürer and Raghavachari also described a polynomial time algorithm that constructs a spanning tree of degree at most $\Delta^* + 1$, where Δ^* is the optimal degree. Therefore, in order to prove that we can find in polynomial time a spanning tree of H with degree at most 10, it suffices to show that $1/\tau(H)$ is at most 7.

Lemma 4.5.6. *We have $1/\tau(H) \leq 6\gamma/(\gamma - 1) \leq 7$.*

Proof. It follows from the definition of $\tau(H)$ that we need to verify that $c(S) \leq (6\gamma/(\gamma - 1))|S|$ for each set $S \subset V(H)$ such that $c(S) > 1$. Consider such a set S and let C_1, C_2, \dots, C_ℓ denote the connected components of $H - S$ (see Figure 4.2).

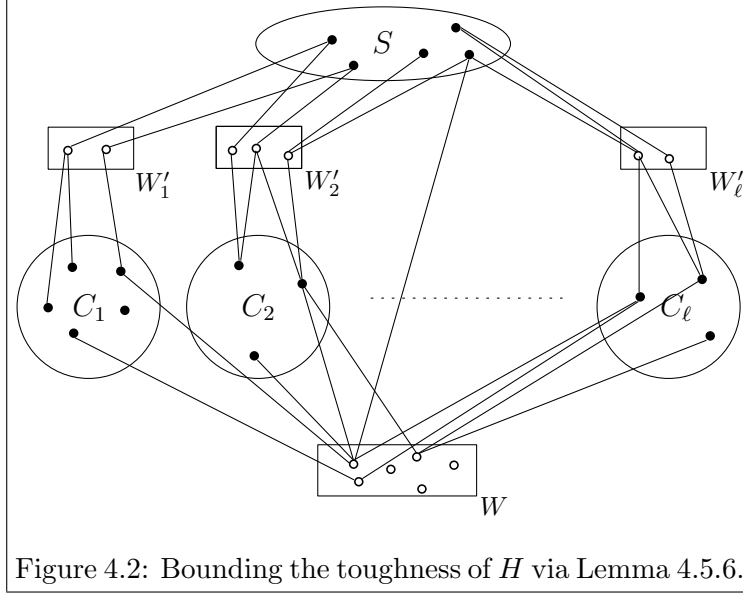


Figure 4.2: Bounding the toughness of H via Lemma 4.5.6.

Let M be the set of all pairs $s_i s_j$ such that s_i and s_j are in different components of $H - S$. Trivially, $|M| \leq \gamma^2$, since there are γ black nodes. Let W be the set of all white nodes v in G'' such that v is adjacent to s_i and s_j for some pair $s_i s_j$ in M .

We claim that $|W| \leq k^*/(6\gamma)$. If not, there is some pair $s_i s_j \in M$ such that there are more than $(1/\gamma^2) \cdot k^*/(6\gamma) = k^*/(6\gamma^3)$ white nodes in G'' that are adjacent to both s_i and s_j . But then $s_i s_j$ is an edge in H , which contradicts the assumption that s_i and s_j are in different connected components of $H - S$.

Now we claim that, for each connected component C_i of $H - S$, there is a set W'_i of white nodes in G'' with the following properties: (1) $|W'_i| \geq (1 - 1/\gamma)k^*/6$, (2) $N_{G''}(W'_i) \subseteq C_i \cup S$, and (3) each node in W'_i is adjacent in G'' to S . Since the black nodes are $\lceil k^*/6 \rceil$ element-connected in G'' , there is a set W_i of white nodes such that $|W_i| \geq \lceil k^*/6 \rceil$ and each node in W_i is adjacent in G'' to C_i and $V(H) - C_i$. Let W'_i be the subset of W_i consisting of all nodes that are only adjacent in G'' to $C_i \cup S$. By the claim in the preceding paragraph, there are at most $k^*/(6\gamma)$ nodes in $W_i - W'_i$ and therefore $|W'_i| \geq (1 - 1/\gamma)k^*/6$.

Let $W' = W'_1 \cup \dots \cup W'_\ell$. Note that, if $i \neq j$, W'_i and W'_j are disjoint and therefore $|W'| \geq (1 - 1/\gamma)k^*\ell/6$. Since each node in W' contributes at least one edge to the total degree in G'' of the nodes in S , it follows that $\sum_{u \in S} d_{G''}(u) \geq |W'|$. Since each black node has degree k^* in G'' , we have $|W'| \leq k^*|S|$. Therefore $c(S) = \ell \leq 6|S|/(1 - 1/\gamma) \leq 7|S|$. (Recall that $\gamma = \gamma_{\text{CMG}} = \Omega(\log^2 k)$; we

can ensure that $6/(1 - 1/\gamma) \leq 7$ by assuming that k is a sufficiently large constant.) \square

It follows from Theorem 4.5.5 and Lemma 4.5.6 that H has a spanning tree with maximum degree $\Delta^* < 3 + (1/\tau(H)) \leq 10$; since Δ^* is an integer, we have $\Delta^* \leq 9$. As shown by Fürer and Raghavachari [80], we can find in polynomial time a spanning tree T^* with maximum degree at most $\Delta^* + 1 \leq 10$. This proves the first part of Theorem 4.5.2.

For each edge $e = s_i s_j$ of T^* , we construct a collection \mathcal{P}_e of paths as follows. Since T^* is a subgraph of H , for each edge $s_i s_j$ of T^* , the graph G'' has a set W'_e of at least $k^*/(6\gamma^3)$ white nodes that are adjacent in G'' to both s_i and s_j . The sets $\{W'_e \mid e \in E(T^*)\}$ may not be disjoint, but we can select a subset $W''_e \subseteq W'_e$ for each edge e such that $|W''_e| \geq k^*/(6\gamma^4)$ and the sets $\{W''_e \mid e \in E(T^*)\}$ are disjoint. We construct the sets $\{W''_e \mid e \in E(T^*)\}$ greedily as follows. We order the edges of T^* arbitrarily. We consider the edges in this order. Let e be the current edge and suppose that, for each edge e' that comes before e in the order, we have already selected a set $W''_{e'} \subseteq W'_{e'}$ of size $k^*/(6\gamma^4)$. Since W'_e has at least $k^*/(6\gamma^3)$ nodes and there are less than $\gamma - 1$ edges that appear before e , W'_e contains a subset W''_e of $k^*/(6\gamma^4)$ nodes such that $W''_e \cap W''_{e'} = \emptyset$ for each edge e' that appears before e in the ordering.

Recall that the white nodes of G'' resulted from the contraction of disjoint subgraphs of G' that are connected and they consist of only white nodes of G' . Since $v \in W''_e$ is adjacent to s_i and s_j — where s_i and s_j are the endpoints of e — we can find a path p_v in G' from s_i to s_j through the subgraph corresponding to v . Thus we obtain $|W''_e|$ paths in G' from s_i to s_j . This is the desired collection \mathcal{P}_e for edge $e = s_i s_j$. The sets $\{W''_e \mid e \in E(T^*)\}$ are mutually disjoint by construction, and hence the paths in $\{\mathcal{P}_e \mid e \in E(T^*)\}$ have the desired properties.

This completes the proof of Theorem 4.5.2. \square

4.6 Concluding remarks

In this chapter, we study the Maximum Node Disjoint Paths problem in undirected graphs. Our main result is a poly-logarithmic approximation with constant node congestion for the problem. In the process, we prove a conjecture of Chekuri, Khanna, and Shepherd [41] on the connection between the treewidth of the graph and the existence of a good routing structure called a crossbar.

We remark that the techniques introduced in the subsequent work of Chekuri and Chuzhoy [32] lead to a poly-logarithmic approximation with congestion 2 for the Maximum Node Disjoint Paths problem.

Chapter 5

The All-or-Nothing Flow Problem

5.1 Introduction

In this chapter¹, we consider some fundamental maximum throughput routing problems in *directed* graphs. In this setting, we are given a capacitated directed graph $G = (V, E)$ with n nodes and m edges. We are also given source-destination pairs of nodes $(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)$. The goal is to select a largest subset of the pairs that are simultaneously *routable* subject to the capacities; a set of pairs is routable if there is a multicommodity flow for the pairs satisfying certain constraints that vary from problem to problem (e.g., integrality, unsplittability, edge or node capacities). Two well-studied optimization problems in this context are the Maximum Edge Disjoint Paths (MEDP) and the All-or-Nothing Flow (ANF) problem. In MEDP, a set of pairs is routable if the pairs can be connected using edge-disjoint paths. In ANF, a set of pairs is routable if there is a feasible multicommodity flow that fractionally routes one unit of flow from s_i to t_i for each routed pair (s_i, t_i) . ANF, introduced in [40, 49], can be seen as a relaxed version of MEDP where the flow for the routed pairs is not required to be integral.

MEDP and ANF are both **NP**-hard and their approximability has attracted substantial attention over the years. Over the last decade, several breakthrough results on both upper bounds and lower bounds have led to a much better understanding of these problems. At a high level, one can summarize this progress as follows. MEDP and ANF admit poly-logarithmic approximation in *undirected* graphs if one allows constant congestion²; in fact, a congestion of 2 is sufficient, and moreover the problems are hard to approximate to within a factor of $\Omega(\log^\delta n)$ for some fixed $\delta > 0$ even with constant congestion. In sharp contrast, both problems are hard to approximate

¹This chapter is based on joint work with Chandra Chekuri [35].

²A routing has congestion c if it violates the capacities by a factor of at most c . ANF for edge-capacitated graphs admits a poly-logarithmic approximation without the need for extra congestion.

to within a *polynomial* factor ($\Omega(n^\delta)$ for some fixed $\delta > 0$) in *directed* graphs even if constant congestion is allowed, and the graph is acyclic. The upper bounds and lower bounds on the approximability are closely related to corresponding integrality gap bounds on a multicommodity flow relaxation for these problems.

In this chapter, with several interrelated motivations in mind that we discuss in detail subsequently, we initiate the study of maximum throughput routing problems in directed graphs in the setting where the demand pairs are *symmetric*. Informally, in a symmetric demand pair instance, the input pairs are *unordered* and a pair $s_i t_i$ is routed only if both the ordered pairs (s_i, t_i) and (t_i, s_i) are routed. In particular, we focus our attention on the SymANF problem. The input consists of a directed graph $G = (V, E)$ and a collection of (unordered) pairs of nodes $\mathcal{M} = \{s_1 t_1, s_2 t_2, \dots, s_k t_k\}$. A subset \mathcal{M}' of the pairs is *routable* if there is a feasible multicommodity flow in G such that, for each pair $s_i t_i \in \mathcal{M}'$, the amount of flow from s_i to t_i is at least one *and* the amount of flow from t_i to s_i is at least one. The goal is to find a maximum cardinality subset of the given pairs that can be routed. Our main result is the following theorem that gives a poly-logarithmic approximation with constant congestion for SymANF.

Theorem 5.1.1. *There is a polynomial time algorithm that, given any instance of the SymANF problem in directed graphs, it routes $\Omega(\text{OPT}/\log^2 k)$ pairs with constant node congestion, where OPT is the value of an optimal solution for the instance.*

The congestion that we guarantee is 64. We believe that the congestion can be improved, but we have not attempted to optimize the constant. Our algorithm uses a natural LP relaxation for the problem as a starting point and we also show a poly-logarithmic upper bound on the integrality gap of the relaxation.

We observe that, via existing results on the hardness of ANF in undirected graphs with congestion [6], one can conclude that SymANF with congestion c is hard to approximate to within a factor of $\log^{\Omega(1/c)} n$ for any fixed c unless $\text{NP} \subseteq \text{ZPTIME}(n^{\text{poly log } n})$.

5.1.1 Motivation and connection to related problems

The study of routing problems is motivated by several real-world applications but also by the fundamental role that flows and cuts play in algorithms, combinatorial optimization, and graph

theory. For a single pair (s, t) it is well-known that the value of a maximum s - t flow in a directed graph is equal to the value of a minimum s - t cut; moreover, when the capacities are integral, the maximum fractional s - t flow is equal to the maximum integral s - t flow. These nice structural properties do not hold in the multicommodity setting even for a small constant number of pairs in both undirected and directed graphs.

The study of *approximate* flow-cut gap results, starting with the seminal work of Leighton and Rao [130], has been extremely fruitful and we now have an optimal upper bound of $\Theta(\log k)$ on multicommodity flow-cut gaps in undirected graphs in a variety of settings [39, 73, 84, 131]. In contrast, it was only fairly recently that polynomial-factor lower bounds were established for flow-cut gaps in directed graphs [55]. On the other hand, poly-logarithmic upper bounds on the flow-cut gap are also known in directed graphs with symmetric demand pairs [123], although the lower and upper bounds are not known to be tight and several open problems remain.

Maximum throughput routing problems, such as MEDP and the related problem of congestion minimization, aim to construct integer flows. These problems are typically tackled via relaxations based on multicommodity flows. Rounding these relaxations is equivalent to upper bounding the gap between fractional and integral flows. A technical issue that arises is the following. For MEDP, even in undirected graphs, the integrality gap of the flow relaxation is known to be $\Omega(\sqrt{n})$ because of a simple topological obstruction in the plane [84]. For this reason, the main focus has been on understanding the following question: what is the gap between the maximum fractional flow and the maximum integral flow with constant congestion? Understanding this question has been a very challenging open problem until the recent breakthrough work of Chuzhoy [53]. She showed a poly-logarithmic upper bound with constant congestion (subsequently, the congestion has been brought down to 2 in [57]). In Chapter 4, we obtained a poly-logarithmic upper bound with constant congestion for the maximum *node-disjoint* paths problem in undirected graphs; note that in undirected graphs the distinction between node-disjoint routing and edge-disjoint routing is important, with the former being more general. Again, in contrast to the undirected graph case, as we already mentioned, in directed graphs there is a polynomial-factor lower bound on the integrality gap of the flow relaxation for any constant congestion.

A natural meta-question is the following. Why are routing problems “easy” in undirected graphs

and “hard” in directed graphs? In the same vein, why are flow-cut gaps and fractional flow-integral flow gaps “small” (poly-logarithmic) in undirected graphs and “large” (polynomial) in directed graphs? What is the most general setting in which one can obtain good results? It is difficult to give a clean answer to these questions but one can observe that routing problems in directed graphs with symmetric demand pairs straddle the boundary between routing in undirected and directed graphs. Moreover, we already know that the flow-cut gap is poly-logarithmic in directed graphs if the demand pairs are symmetric. Our primary motivation is to understand whether the gap between fractional flows and integral flows is also small in directed graphs with symmetric demand pairs. A direct benefit is a generalization of existing results that show poly-logarithmic gaps for edge *and* node disjoint paths in undirected graphs (with constant congestion). This research agenda will involve the development and understanding of several technical ingredients that have auxiliary benefits. We briefly elaborate on this point below.

Structure of graphs with large (directed) treewidth: Recent progress on routing problems has been accomplished via the following scheme. The well-linked decomposition framework of [42] showed that one can use flow-cut gap results to reduce the problem (to within poly-logarithmic factors) to a graph theoretic question: if the graph G has a “well-linked” set of size k , does it have a routing structure (called a *crossbar*) of size³ $\tilde{\Omega}(k)$? For node-capacitated routing problems in undirected graphs, the question can be phrased in terms of the well-understood notion of *treewidth*: If G has treewidth k , does G have a crossbar of size $\tilde{\Omega}(k)$? The question was answered affirmatively (in [36], following Chuzhoy’s framework for MEDP) by embedding, in a technical sense, in G an expander of size $\tilde{\Omega}(k)$ with constant congestion; the expander is the desired crossbar. This high-level structural result required several technical ingredients. A key tool developed in this process was a graph splitting procedure that has already found several powerful applications in fixed parameter tractability and graph theory [31]. These results are also closely related to the well-known grid minor theorem of Robertson and Seymour that shows that G contains a grid minor of size k as long as the treewidth of G is at least $f(k)$ for some function f [149]. Very recently the ideas from routing led to a proof that $f(k)$ can be chosen to be a polynomial [32], improving upon a previous exponential bound [151].

³The $\tilde{\Omega}$ notation hides poly-logarithmic factors.

Johnson, Robertson, Seymour, and Thomas [108] introduced the notion of *directed* treewidth which is also related to the notion of well-linked sets in directed graphs (see [148]). Although there are several similarities between treewidth and directed treewidth, the latter concept is more difficult to work with and it is believed that understanding it better would yield significant dividends in graph theory and algorithms. Our second motivation for studying routing problems in directed graphs with symmetric demand pairs stems from their connections to directed treewidth. In this chapter, we extend the well-linked decomposition framework of [42] to this setting and this leads to the following question: If a directed graph G has directed treewidth k , does it have a “routing structure” of size $\tilde{\Omega}(k)$? Answering this question affirmatively would lead to algorithms for disjoint path routing for symmetric pair instances. This question seems to pose several non-trivial technical challenges despite the substantial recent progress made in the undirected graph case. We note that [108] formulated a conjecture that can be viewed as a directed counterpart of the grid minor theorem of Robertson and Seymour. The conjecture posits that there is a function $f(k)$ such that, if G is a directed graph with directed treewidth at least k , then G has a cylinder of size k as a minor⁴; a cylinder is a directed analogue of a grid and it is a crossbar. The cylinder minor conjecture is still open for general graphs. Johnson *et al.* [109] showed that the cylinder conjecture is true in planar graphs for some function $f(k)$. Proving it for general graphs seems very challenging. Moreover, to obtain good approximations for routing, one needs a very strong quantitative bound on f ; to obtain an approximation ratio of $\alpha(k)$, we need $f(k)$ to be $\Omega(k/\alpha(k))$. On the other hand, there is flexibility in routing applications in that the crossbar structure that we need to exhibit need not be a cylinder minor, and moreover we can allow constant congestion. In fact such flexibility is needed to obtain a crossbar whose size is a near-linear function of k .

Flow-cut gap in planar graphs: Another interesting direction for future work, and a motivation for us, is to show the following: if G is a planar directed graph with directed treewidth k , then it has a crossbar of size $\Omega(k)$. A linear relationship between treewidth and grid-minor size is known in undirected planar graphs [151] (and also in a more general class of graphs [66]) but the known quantitative relationship between directed treewidth and the cylinder minor size in planar graphs is so weak that it is not explicitly specified [109]. A linear relationship would have applications

⁴There are several ways to define a minor in directed graphs. The minor notion used in the cylinder conjecture is the notion of a butterfly minor. We refer the reader to [108] for the definition of a butterfly minor and a cylinder.

to routing on disjoint paths, but would also give an improved upper bound on the flow-cut gap for symmetric product multicommodity flows in planar directed graphs. Currently, the best upper bound on the flow-cut gap is $O(\log n)$ for product multicommodity flows in both general and planar graphs [130]. The existence of a crossbar of size $\Omega(k)$ will imply that the flow-cut gap is $O(1)$ in planar directed graphs⁵, which in turn it will give constant factor approximation guarantees for problems such as the **Uniform Sparsest Cut** problem in planar directed graphs; such results are known for planar undirected graphs [126]. Currently, the best approximation for the **Uniform Sparsest Cut** problem is $O(\sqrt{\log n})$ [1] in both planar and general directed graphs. We remark that the crossbar we need for a flow-cut gap result can be much weaker than what one needs for disjoint path problems; it only needs to support fractional routing instead of integral routing. Exploring weaker notions of crossbars, we believe, may help make progress on the difficult questions while also yielding results that are independently interesting.

In this chapter, we study the **SymANF** problem as a first step towards understanding maximum throughput routing problems in directed graphs. We now give a high-level description of our algorithm and we describe in more detail some specific technical contributions that enable us to prove Theorem 5.1.1.

5.1.2 High-level overview of the algorithm and technical contributions

Let (G, \mathcal{M}) be an instance of **SymANF**. Let \mathcal{T} be the set of all nodes that participate in the pairs of \mathcal{M} ; we refer to the nodes in \mathcal{T} as the *terminals*. Our algorithm for **SymANF** in directed graphs follows the framework of Chekuri, Khanna, and Shepherd [40, 42] for the **ANF** problem in undirected graphs. In a nutshell, the framework decomposes an arbitrary instance of **ANF** into several instances that are *flow-well-linked*. The set of terminals $\mathcal{T} = \{s_1, t_1, \dots, s_k, t_k\}$ is *flow-well-linked* if *any* matching on the terminals is routable. If the terminals are flow-well-linked, we can route all the input pairs. Thus the heart of the matter is to show that we can decompose an arbitrary instance into well-linked instances without losing too much flow.

The decomposition has two main components. The first step is a weaker decomposition in which we take a fractional solution⁶ to the LP and use it to decompose the instance into instances

⁵The implications of crossbar results for product multicommodity flow-cut gaps is pointed out in [42].

⁶We work with a natural multicommodity flow LP relaxation for the problem which we describe in Subsection 5.2.1.

that are only *fractionally* flow-well-linked. More precisely, there is a weight function $\pi : \mathcal{T} \rightarrow [0, 1]$ and the terminals are flow-well-linked with respect to these weights; if all terminals have weight 1 then they are flow-well-linked. The second step is a clustering step in which we take a fractionally flow-well-linked instance and we identify a large subset of the pairs such that their endpoints are flow-well-linked. In this chapter, we show how to implement these two steps for the **SymANF** problem in directed graphs. In the first step, we extend the approach of Chekuri, Khanna, and Shepherd [42] to our setting; we refer the reader to Section 5.3 for the details of the decomposition. A crucial ingredient in this decomposition is the fact that the flow-cut gap for symmetric instances is poly-logarithmic.

The second step poses several technical difficulties in directed graphs and it is our main technical contribution. We briefly highlight some of the difficulties involved in the clustering step, and we refer the reader to Section 5.4 for the details. Chekuri, Khanna, and Shepherd [42] gave a simple clustering technique for *edge-capacitated* undirected graphs. Roughly speaking, the approach is to take a spanning tree and to partition it into edge-disjoint subtrees where each subtree gathers roughly a unit weight from π . These subtrees are then used to find the desired flow-well-linked subset of pairs/terminals; one terminal is picked from each subtree. The clustering step is more involved in *node-capacitated* undirected graphs. The spanning tree approach, combined with some preprocessing to reduce the degree, gives a clustering for node-capacitated graphs with slightly weaker parameters [42]. In [43], the authors gave a stronger clustering for the node-capacitated setting; this approach is more involved than the spanning tree clustering and it exploits a connection between well-linked sets and treewidth. In directed graphs, there is no simple clustering process akin to using a spanning tree (or even an arborescence). Instead, our approach exploits the connection between well-linked sets and *directed treewidth*. However, the notion of directed treewidth is different from that of undirected treewidth and this discrepancy poses several non-trivial technical challenges. We also mention that, in addition to finding a large flow-well-linked set Y from a fractionally flow-well-linked set X , we also need to ensure that Y contains a large enough matching from the original set of pairs. For this purpose, we crucially rely on a flow augmentation tool developed in [44]. These difficulties are also the reason why we are only able to obtain a constant congestion for **SymANF** while **ANF** admits a poly-logarithmic ratio with congestion 1 in

edge-capacitated graphs [40] and with congestion $(1 + \varepsilon)$ in node-capacitated graphs [42].

5.1.3 Discussion of related work

The ANF problem, the MEDP problem and its node-capacitated counterpart, the Maximum Node Disjoint Paths (MNDP) problem have been studied extensively in both undirected and directed graphs. We first discuss the decision versions of these problems where we are given G and the pairs, and the goal is to decide if all of them can be routed. It is easy to see that the decision version of ANF is polynomial-time solvable via linear programming — one needs to check whether there is a multicommodity flow that routes one unit of flow for each input pair. On the other hand, the decision versions of MEDP and MNDP, denoted by EDP and NDP respectively, are **NP**-complete if k is part of the input [72, 113]. If k is fixed, Robertson and Seymour, building on their seminal work on graph minors, gave a polynomial-time algorithm for NDP (and hence also for EDP) in undirected graphs. Interestingly, EDP is already **NP**-complete for $k = 2$ in directed graphs [75]. It is useful to note that the undirected graph algorithm of Robertson and Seymour relies heavily on treewidth and the structure of graphs with large treewidth.

ANF, MEDP and MNDP are optimization problems. Although the decision version of ANF is poly-time solvable, ANF is NP-Hard, and APX-hard to approximate, even in capacitated trees [84]; routing is trivial in trees, selecting the pairs to route is not. The best approximation guarantees that are known for the ANF problem in undirected graphs are an $O(\log^2 k)$ approximation in edge-capacitated graphs [42] and an $O(\log^4 k \log n)$ approximation with congestion $(1 + \epsilon)$ in node-capacitated graphs [42]; these ratios improve by a logarithmic factor for planar graphs. For node-capacitated graphs, an unpublished manuscript [43] gives an $O(\log^2 k)$ -approximation if constant congestion is allowed. The MEDP problem with congestion $c = o(\log n / \log \log n)$ is $\Omega(\log^{O(1/c)})$ -hard to approximate in undirected graphs [6] and $\Omega(n^{O(1/c)})$ -hard to approximate in directed graphs [54], unless **NP** \subseteq **ZPTIME** $(n^{\text{poly} \log n})$. It is useful to note that these hardness results also hold for the ANF problem, which suggests that the current techniques do not distinguish between the difficulty of ANF and MEDP. For MEDP in undirected graphs there is an $O(\sqrt{n})$ -approximation with congestion 1 [44] and as we already mentioned, recent work obtains a poly-logarithmic approximation with congestion 2 [57]. In directed graphs, MEDP has an $n^{O(1/c)}$ -

approximation with congestion c [159], and this approximation carries over to ANF as well. These approximation results use the natural multicommodity flow relaxations as a starting point and they also establish the same upper bound on the integrality gap of the relaxations.

Chapter outline: The rest of this chapter is organized as follows. Section 5.2 introduces the main definitions and technical tools that we use, and it describes the approximation algorithm for SymANF. Section 5.3 and Section 5.4 describe the well-linked decomposition and clustering technique for directed graphs with symmetric demand pairs.

5.2 Approximation algorithm for SymANF

5.2.1 Preliminaries and setup

In the following, we work with an instance (G, \mathcal{M}) of the SymANF problem, where $G = (V, E)$ is a directed graph and $\mathcal{M} = \{s_1 t_1, \dots, s_k t_k\}$ is a collection of node pairs. We refer to the nodes participating in the pairs of \mathcal{M} as terminals, and we use \mathcal{T} to denote the set of all terminals. We may assume, without loss of generality, that the pairs \mathcal{M} form a perfect matching on \mathcal{T} ; if a node participates in several pairs, we make a copy of the node for each pair, we attach the copy to the original node as a leaf using an edge in each direction, and we replace the node by its copy in the pair. Similarly, we may assume without loss of generality that each terminal is a leaf in G , i.e., each terminal is connected to a single neighbor using an edge in each direction.

LP relaxation: We consider a standard multicommodity flow relaxation for the SymANF problem. For each ordered pair (u, v) of nodes of G , let $\mathcal{P}(u, v)$ be the set of all paths in G from u to v . Since \mathcal{M} forms a matching on \mathcal{T} , for all $i \neq j$, the sets $\mathcal{P}(s_i, t_i)$, $\mathcal{P}(t_i, s_i)$, $\mathcal{P}(s_j, t_j)$, and $\mathcal{P}(t_j, s_j)$ are pairwise disjoint. Let $\mathcal{P} = \bigcup_{i=1}^k (\mathcal{P}(s_i, t_i) \cup \mathcal{P}(t_i, s_i))$. For each pair (s_i, t_i) , we have a variable x_i that is equal to the amount of flow routed for the pair. For each path $p \in \mathcal{P}$, we have a variable $f(p)$ that is equal to the amount of flow on p .

$$\begin{array}{ll}
\text{(symANF-LP)} \\
\max & \sum_{i=1}^k x_i \\
\text{s.t.} & \sum_{p \in \mathcal{P}(s_i, t_i)} f(p) \geq x_i \quad 1 \leq i \leq k \\
& \sum_{p \in \mathcal{P}(t_i, s_i)} f(p) \geq x_i \quad 1 \leq i \leq k \\
& \sum_{p: v \in p} f(p) \leq 1 \quad v \in V(G) \\
& x_i \leq 1 \quad 1 \leq i \leq k \\
& f(p) \geq 0 \quad p \in \mathcal{P}
\end{array}$$

The dual of the symANF-LP relaxation has polynomially many variables and exponentially many constraints. The separation oracle for the dual is the shortest paths problem. Thus we can solve the relaxation in polynomial time. Alternatively, we can write an equivalent LP relaxation that is polynomial sized.

Multicommodity flows and sparse node separators: Let $G = (V, E, \text{cap})$ be a directed node-capacitated graph with node capacities given by cap . In this chapter, we work with path-based flows f that assign a non-negative real value $f(p)$ to each path in G . A flow f is *feasible* if it satisfies the capacity constraints; more precisely, for each node v , $\sum_{p: v \in p} f(p) \leq \text{cap}(v)$. For any ordered pair (u, v) of nodes, the total flow from u to v is $\sum_{p \in \mathcal{P}(u, v)} f(p)$, where $\mathcal{P}(u, v)$ is the set of all paths of G from u to v .

A *multicommodity flow* instance in G is a demand vector \mathbf{d} that assigns a non-negative real value $d(u, v)$ to each ordered pair (u, v) of nodes of G ; we refer to $d(u, v)$ as the demand of the pair (u, v) . A multicommodity flow instance is *symmetric* if $d(u, v) = d(v, u)$ for all ordered pairs (u, v) . A multicommodity flow instance \mathbf{d} is a *product* multicommodity flow instance if $d(u, v) = w(u)w(v)$, where $w : V \rightarrow \mathbb{R}_+$ is a weight function on the nodes of G . Note that a product multicommodity flow instance is symmetric. In the following, we only consider symmetric multicommodity flow instances. A multicommodity flow instance \mathbf{d} is *routable* if there is a feasible multicommodity flow in which, for each ordered pair (u, v) , the total flow on the paths from u to v is at least $d(u, v)$. We work with the following quantities associated with a *symmetric* multicommodity flow instance,

the maximum concurrent flow and the sparsest node separator. The *maximum concurrent flow* is the maximum value $\lambda \geq 0$ such that $\lambda \mathbf{d}$ is routable. A *node separator* is a set $C \subseteq V$ of nodes. The removal of a node separator gives us one or more strongly connected components; we say that a pair uv is separated by C if u and v are in different strongly connected components of $G - C$. The demand separated by C , denoted by $\text{dem}_{\mathbf{d}}(C)$, is the total demand of all of the unordered pairs separated by C ; more precisely, $\text{dem}_{\mathbf{d}}(C) = \sum_{uv \text{ separated by } C} d(u, v)$. The *sparsity* of a node separator C is $\text{cap}(C)/\text{dem}_{\mathbf{d}}(C)$. A *sparsest node separator* is a separator with minimum sparsity. It is straightforward to verify that, for a symmetric multicommodity flow, the minimum sparsity of a node separator is an upper bound on the maximum concurrent flow. The *flow-cut gap* in G is the maximum value — over all symmetric multicommodity flow instances \mathbf{d} in G — of the ratio between the minimum sparsity of a node separator and the maximum concurrent flow. The flow-cut gap in any graph is $O(\log^2 k)$, where k is the number of commodities (each pair (u, v) with non-zero demand is a commodity) [123]. For product multicommodity flows, the flow-cut gap is $O(\log k)$ [130]. Moreover, it was shown in [130] that there is a polynomial time algorithm that, given a product multicommodity flow instance \mathbf{d} in G , it constructs a node separator C whose sparsity is at most $O(\log k)\lambda$, where λ is the maximum concurrent flow for \mathbf{d} ; we use such an algorithm in a black box fashion in the well-linked decomposition step that we describe in more detail below.

A *node separation* in G is a partition (A, B, C) of the nodes of G such that there is no edge of G from A to B (note that there can be an edge of G from B to A). The following proposition relates a node separator to a node separation.

Proposition 5.2.1. *Let $G = (V, E)$ be a directed graph and let $\pi : V \rightarrow \mathbb{R}_+$ be a weight function. Let \mathbf{d} be the following product multicommodity flow: $d(u, v) = \pi(u)\pi(v)/\pi(V)$ for each pair (u, v) of nodes. Let C be a node separator in G . There is a node separation (A, B, C) such that $\text{dem}_{\mathbf{d}}(C) \leq 2 \min \{\pi(A), \pi(B)\}$. Moreover, given C , we can compute such a node separation in polynomial time.*

Proof. Let C be a node separator. Let K_1, K_2, \dots, K_ℓ be a topological ordering of the strongly connected components of $G - C$ in which each edge of $G - C$ connecting different strongly connected components is oriented from right to left.

Suppose that $\pi(K_i) \leq \pi(V - C)/2$ for each i . Let p be the smallest index such that $\pi(K_1 \cup \dots \cup K_p) \geq \pi(V - C)/4$. Let $A = V(K_1) \cup \dots \cup V(K_p)$ and $B = V(K_{p+1}) \cup \dots \cup V(K_\ell)$. Since $\pi(K_1 \cup \dots \cup K_{p-1}) < \pi(V - C)/4$ and $\pi(K_p) \leq \pi(V - C)/2$, we have $\pi(A) \leq 3\pi(V - C)/4$ and therefore $\pi(B) \geq \pi(V - C)/4$. Thus (A, B, C) is a node separation satisfying $\min\{\pi(A), \pi(B)\} \geq \pi(V - C)/4$. Note that the total demand of the pairs $(u, v) \in (V - C) \times (V - C)$ is $\pi(V - C) \cdot \pi(V - C)/\pi(V) \leq \pi(V - C)$. Therefore $\text{dem}_{\mathbf{d}}(C) \leq \pi(V - C)/2 \leq 2 \min\{\pi(A), \pi(B)\}$.

Therefore we may assume that $\max_i \pi(K_i) > \pi(V - C)/2$. Let K_q be the strongly connected component with maximum π -weight; more precisely, $q = \arg \max_i \pi(K_i)$. We define a partition (A, B) of $V - C$ as follows. If $\pi(K_1 \cup \dots \cup K_{q-1}) \geq \pi(K_{q+1} \cup \dots \cup K_\ell)$, we let $A = V(K_1) \cup \dots \cup V(K_{q-1})$ and $B = V(K_q) \cup \dots \cup V(K_\ell)$. Otherwise, we let $A = V(K_1) \cup \dots \cup V(K_q)$ and $B = V(K_{q+1}) \cup \dots \cup V(K_\ell)$. The partition (A, B, C) is a node separation satisfying $\min\{\pi(A), \pi(B)\} \geq \pi(V - (C \cup K_q))/2$. Note that the total demand of the pairs $(u, v) \in (V - (C \cup K_q)) \times (V - (C \cup K_q))$ is $\pi(V - (C \cup K_q)) \cdot \pi(V - (C \cup K_q))/\pi(V)$. Additionally, the total demand of the pairs $(u, v) \in K_q \times (V - (C \cup K_q))$ is $\pi(K_q)\pi(V - (C \cup K_q))/\pi(V)$. Therefore we have

$$\begin{aligned} \text{dem}_{\mathbf{d}}(C) &\leq \frac{\pi(K_q)\pi(V - (C \cup K_q))}{\pi(V)} + \frac{\pi(V - (C \cup K_q))\pi(V - (C \cup K_q))}{2\pi(V)} \\ &= \frac{\pi(V - (C \cup K_q))(\pi(K_q) + \pi(V - C))}{2\pi(V)} \\ &\leq \pi(V - (C \cup K_q)) \end{aligned}$$

Therefore $\text{dem}_{\mathbf{d}}(C) \leq 2 \min\{\pi(A), \pi(B)\}$. □

Well-linked sets: There are two notions of well-linkedness that have been used for routing problems in undirected graphs [42]; one is based on a flow requirement and the other is based on a cut requirement. In the following, we define directed versions of these two notions and we show some basic properties of these notions.

Flow-well-linked sets: Let G be a directed graph with unit capacities on the nodes. A set $X \subseteq V$ is **flow-well-linked** in G iff, for any matching M on X , M can be routed fractionally in G . More precisely, let \mathbf{d} be the following demand vector: $d(u, v) = d(v, u) = 1$ for each pair $uv \in M$, and $d(u, v) = 0$ for all other pairs; the vector \mathbf{d} is routable in G .

An equivalent definition is the following. Let \mathbf{d} be the following demand vector: $d(u, v) = 1/|X|$ for each pair (u, v) of nodes in X . The set X is flow-well-linked iff \mathbf{d} is routable in G .

We define a fractional version of flow-well-linkedness as follows. Let $\pi : X \rightarrow [0, 1]$ be a weight function on X . Let \mathbf{d} be the following demand vector: $d(u, v) = \pi(u)\pi(v)/\pi(X)$ for each ordered pair (u, v) of nodes in X . The set X is π -**flow-well-linked** in G iff \mathbf{d} is routable in G .

Cut-well-linked sets: A set $X \subseteq V$ is **cut-well-linked** in G iff, for any two disjoint subsets Y and Z of X of equal size, there are $|Y|$ node-disjoint paths from Y to Z in G .

Recall that a node is a leaf in G if it is connected to a single neighbor using an edge in each direction. If the nodes of X are leaves in G , an equivalent definition is the following. The set X is cut-well-linked iff, for any node separation (A, B, C) , we have $|C| \geq \min\{|X \cap A|, |X \cap B|\}$.

We define a fractional version of cut-well-linkedness as follows. Let X be a set of nodes of G and let $\pi : X \rightarrow [0, 1]$ be a weight function on X . Suppose that all the nodes in X are leaves of G . The set X is π -**cut-well-linked** in G if, for any node separation (A, B, C) , we have $|C| \geq \min\{\pi(A), \pi(B)\}$. Note that, since the nodes in X are leaves, it suffices to check this condition for separations (A, B, C) for which $\pi(C) = 0$. Now consider a set X that contains nodes that are not leaves. For each node $x \in X$, we add a new node x' and connect x' to x using two edges, one in each direction. Let X' be the set of new nodes, let G' be the resulting graph, and let $\pi' : X' \rightarrow [0, 1]$ be the weight function $\pi'(x') = \pi(x)$ for each node $x \in X$. The set X is π -cut-well-linked in G iff X' is π' -cut-well-linked in G' .

The following proposition relates the two notions of well-linkedness.

Proposition 5.2.2. *Let $G = (V, E)$ be a directed graph. Let X be a set of nodes and let $\pi : X \rightarrow [0, 1]$ be a weight function on X . Let $\alpha = \alpha(G) \geq 1$ be an upper bound on the worst case flow-cut gap for product multicommodity flows in G . If X is π -flow-well-linked in G then X is $(\pi/2)$ -cut-well-linked in G . If X is π -cut-well-linked in G then X is $(\pi/(2\alpha))$ -flow-well-linked in G .*

Proof. Let \mathbf{d} be the following product multicommodity flow: $d(u, v) = \pi(u)\pi(v)/\pi(X)$ for each pair (u, v) of nodes in X , and $d(\cdot)$ is zero for all other pairs.

Suppose that X is π -flow-well-linked. Recall that, in order to show that X is $(\pi/2)$ -cut-well-linked, it suffices to verify that, for each node separation (A, B, C) such that $\pi(C) = 0$, we have

$|C| \geq \min\{\pi(A), \pi(B)\}/2$. Consider a node separation (A, B, C) such that $\pi(C) = 0$. Since X is π -flow-well-linked, \mathbf{d} is routable and therefore $|C| \geq \text{dem}_{\mathbf{d}}(C) = \pi(A)\pi(B)/\pi(X)$. Since $\pi(X) = \pi(A) + \pi(B)$, we have $\pi(A)\pi(B)/\pi(X) \geq \min\{\pi(A), \pi(B)\}/2$, as desired.

Conversely, suppose that X is π -cut-well-linked. By definition, X is $(\pi/(2\alpha))$ -flow-well-linked if $\mathbf{d}/(2\alpha)$ is routable. Thus, in order to show that X is $(\pi/(2\alpha))$ -flow-well-linked, it suffices to verify that each node separator has sparsity at least $1/2$.

Let C be a sparsest node separator. By Proposition 5.2.1, there is a node separation (A, B, C) such that $\text{dem}_{\mathbf{d}}(C) \leq 2 \min\{\pi(A), \pi(B)\}$. Since X is π -cut-well-linked, we have $|C| \geq \min\{\pi(A), \pi(B)\}$. Therefore the sparsity of C is at least $1/2$. \square

We will need the following simple observation in our clustering algorithm.

Proposition 5.2.3. *Let G be a directed graph. Let X be a set of nodes of G and let $\pi : X \rightarrow [0, 1]$ be a weight function on X . Suppose that X is π -cut-well-linked in G . Then for any set Z such that $|Z| < \pi(X)/4$, there is a unique strongly connected component $\beta(Z)$ of $G - Z$ such that $\pi(\beta(Z)) > \pi(X)/2$.*

Proof. Suppose for contradiction that there is a set Z such that $|Z| < \pi(X)/4$ and, for each strongly connected component H of $G - Z$, we have $\pi(H) \leq \pi(X)/2$. Let H_1, H_2, \dots, H_ℓ be a topological ordering of the strongly connected components of $G - Z$ in which each edge of $G - Z$ that connects different strongly connected components is oriented from right to left. Let p be the smallest index such that $\pi(H_1 \cup \dots \cup H_p) \geq \pi(X)/4$. Note that, since $\pi(H_1 \cup \dots \cup H_{p-1}) < \pi(X)/4$ and $\pi(H_p) \leq \pi(X)/2$, we have $\pi(H_1 \cup \dots \cup H_p) < 3\pi(X)/4$. Thus we have $\pi(H_{p+1} \cup \dots \cup H_\ell) > \pi(X)/4$. Let A be the set of all vertices in $H_1 \cup \dots \cup H_p$ and let B be the set of all vertices in $H_{p+1} \cup \dots \cup H_\ell$. Note that (A, B, Z) is a node separation in G . Since X is π -cut-well-linked, it follows that $|Z| \geq \min\{\pi(A), \pi(B)\} \geq \pi(X)/4$, which is a contradiction. \square

Lemma 5.2.4. *Let $G = (V, E)$ be a node-capacitated directed network. Let A and B be two sets of nodes in G . Let $\pi : A \rightarrow \mathbb{R}_+$ and $\pi' : B \rightarrow \mathbb{R}_+$ be two weight functions. Suppose that A and B satisfy the following conditions:*

- A is π -flow-well-linked.

- There is a feasible single-commodity flow f_1 in G from B to A such that each node $b \in B$ sends $\pi'(b)$ units of flow to A and each node $a \in A$ receives at most $\pi(a)$ units of flow.
- There is a feasible single-commodity flow f_2 in G from A to B such that each node $a \in A$ sends at most $\pi(a)$ units of flow and each node $b \in B$ receives $\pi'(b)$ units of flow.

Then B is $(\pi'/4)$ -flow-well-linked in G .

Proof. Let \mathbf{d}_1 be the following multicommodity flow instance: $d_1(b, a) = \pi(a)\pi'(b)/\pi(A)$ for each pair $(b, a) \in B \times A$, and $\mathbf{d}_1(\cdot)$ is zero for all other pairs. We claim that we can route \mathbf{d}_1 using congestion at most two. In order to prove the claim, we combine the flow f_1 and the flow f that routes the following product multicommodity flow instance \mathbf{d} : $d(a, a') = \pi(a)\pi(a')/\pi(A)$ for all pairs of nodes $(a, a') \in A \times A$. Let $F_1(b, a)$ be the amount of flow sent by f_1 from b to a . We split the flow of f_1 from b to a among the nodes of A as follows: for each node $a' \in A$, the amount of f_1 -flow from b to a that we allocate to a' is $F_1(b, a)\pi(a')/\pi(A)$. We split the flow of f from a to a' among the nodes of B as follows: for each node $b \in B$, the amount of f -flow from a to a' that we allocate to b is $F_1(b, a)\pi(a')/\pi(A)$; since $\sum_b F_1(b, a) = \pi(a)\pi'(B)/\pi(A) \leq \pi(a)$, there is enough f -flow from a to a' to allocate to B . Finally, we concatenate the allocated flow paths as follows. Consider a node $b \in B$ and two nodes $a, a' \in A$. We allocated $F_1(b, a)\pi(a')/\pi(A)$ units of f_1 -flow to a' ; we can represent the allocated flow as a collection $\{(P_i, \epsilon_i)\}$, where P_i is a path from b to a and ϵ_i is the amount of f_1 -flow on P_i that we allocated. We allocated $F_1(b, a)\pi(a')/\pi(A)$ units of f -flow to a' ; we can represent the allocated flow as a collection $\{(Q_j, \delta_j)\}$, where Q_j is a path from a to a' and δ_j is the amount of f -flow on Q_j that we allocated. By making multiple copies of each path, we may assume that $\epsilon_i = \delta_j = \epsilon$ for all i and j ; that is, all flow paths have the same amount ϵ of flow. For each i , we send ϵ units of flow on the path obtained by concatenating P_i and Q_i ; more precisely, we replace the flow paths $\{(P_i, \epsilon)\}$ and $\{(Q_i, \epsilon)\}$ by the flow paths $\{(P_i Q_i, \epsilon)\}$. By concatenating all of the allocated flow paths, we get a flow with congestion at most two. For each pair $(b, a') \in B \times A$, the amount of flow from b to a' is $\sum_{a \in A} F_1(b, a)\pi(a')/\pi(A) = \pi'(b)\pi(a')/\pi(A) = d_1(b, a')$.

Let \mathbf{d}_2 be the following multicommodity flow instance: $d_2(a, b) = \pi(a)\pi'(b)/\pi(A)$ for each pair $(a, b) \in A \times B$, and $d_2(\cdot)$ is zero for all other pairs. By combining the flows f_2 and f , we can show that \mathbf{d}_2 is routable with congestion at most two; the argument is very similar to the previous argument and we omit it.

Let g_1 and g_2 be the congestion two flows that route \mathbf{d}_1 and \mathbf{d}_2 , respectively. In the following, we show how to combine g_1 and g_2 to get a congestion four flow that routes the following product multicommodity flow instance \mathbf{d}' : $d'(b, b') = \pi'(b)\pi'(b')/\pi'(B)$ for each pair of nodes $(b, b') \in B \times B$. Consider a node $a \in A$ and two nodes $b_1, b_2 \in B$. The amount of g_1 -flow from b_1 to a is $\pi(a)\pi'(b_1)/\pi(A)$; we allocate $\pi(a)\pi'(b_1)\pi'(b_2)/(\pi(A)\pi'(B))$ of this flow to b_2 . The amount of g_2 -flow from a to b_2 is $\pi(a)\pi'(b_2)/\pi(A)$; we allocate $\pi(a)\pi'(b_1)\pi'(b_2)/(\pi(A)\pi'(B))$ of this flow to b_1 . By concatenating the allocated flow paths, we can send $\pi(a)\pi'(b_1)\pi'(b_2)/(\pi(A)\pi'(B))$ units of flow from b_1 to b_2 through a ; summing over all nodes $a \in A$, the total flow from b_1 to b_2 is $\pi'(b_1)\pi'(b_2)/\pi'(B)$. Therefore the \mathbf{d}' is routable with congestion at most four. Thus B is $(\pi'/4)$ -flow-well-linked. \square

Well-linked decomposition: The following theorem is an extension to directed graphs of the well-linked decomposition technique introduced by [42] for routing problems in undirected graphs. The proof follows the outline of the approach in [42] and it can be found in Section 5.3.

Theorem 5.2.5. *Let OPT be the value of a solution to the symANF-LP relaxation for a given instance (G, \mathcal{M}) of SymANF . Let $\alpha = \alpha(G) \geq 1$ be an upper bound on the worst case flow-cut gap for product multicommodity flows in G . There is a partition of G into node-disjoint induced subgraphs G_1, G_2, \dots, G_ℓ and weight functions $\pi_i : V(G_i) \rightarrow \mathbb{R}_+$ with the following properties. Let \mathcal{M}_i be the induced pairs of \mathcal{M} in G_i and let X_i be the endpoints of the pairs in \mathcal{M}_i . We have*

- (a) $\pi_i(u) = \pi_i(v)$ for each pair $uv \in \mathcal{M}_i$.
- (b) X_i is π_i -flow-well-linked in G_i .
- (c) $\sum_{i=1}^\ell \pi_i(X_i) = \Omega(\text{OPT}/(\alpha \log \text{OPT})) = \Omega(\text{OPT}/\log^2 k)$.

Moreover, such a partition is computable in polynomial time if there is a polynomial time algorithm for computing a node separator with sparsity at most $\alpha(G)$ times the maximum concurrent flow.

From fractional well-linked sets to well-linked sets: We prove the following theorem in Section 5.4.

Theorem 5.2.6. *Let X be a π -flow-well-linked set in G and let \mathcal{M} be a perfect matching on X such that $\pi(u) = \pi(v)$ for each pair $uv \in \mathcal{M}$. There is a matching \mathcal{M}' on a set $X' \subseteq X$ such that*

X' is $1/32$ -flow-well-linked in G and $|\mathcal{M}'| = 2|X'| = \Omega(\pi(X))$. Moreover, given X and \mathcal{M} , we can construct X' and \mathcal{M}' in polynomial time.

We can prove an analogous theorem for cut-well-linked sets using an argument that is very similar to the argument in Section 5.4.

Theorem 5.2.7. *Let X be a π -cut-well-linked set in G and let \mathcal{M} be a perfect matching on X such that $\pi(u) = \pi(v)$ for each pair $uv \in \mathcal{M}$. There is a matching \mathcal{M}' on a set $X' \subseteq X$ such that X' is $1/32$ -cut-well-linked in G and $|\mathcal{M}'| = 2|X'| = \Omega(\pi(X))$. Moreover, given X and \mathcal{M} , we can construct X' and \mathcal{M}' in polynomial time.*

Routing a flow-well-linked instance: Finally, we observe that, if an instance of **SymANF** is c -flow-well-linked for some $c \leq 1$, then we can route all of the pairs with congestion at most $2/c$.

Proposition 5.2.8. *Let (G, \mathcal{M}) be an instance of **SymANF** and let X be the set of all vertices that participate in the pairs of \mathcal{M} . If X is c -flow-well-linked for some $c \leq 1$, then we can route all of the pairs of \mathcal{M} with congestion at most $2/c$.*

Proof. Note that it suffices to show that we can route c units of flow for each pair using congestion at most 2; once we have this flow, we can simply scale it by $1/c$ to get a flow that routes one unit of flow for each pair.

Let X_1 be a set consisting of exactly one node from each pair of \mathcal{M} , and let $X_2 = X - X_1$ be the set of all partners of the nodes in X_1 . Let \mathbf{d} be the following demand vector: $d(u, v) = c/|X|$ for each pair (u, v) of nodes in X , and $d(\cdot)$ is zero for all other pairs. Since X is c -flow-well-linked, there is a feasible flow f that routes \mathbf{d} . Note that f gives us a feasible flow in which each node in X_1 sends c units of flow to its partner: consider a node $u \in X_1$ and let v be its partner; we combine the flow paths of f connecting u to X and the flow paths of f connecting X to v in order to get flow paths from u to v carrying at least c units of flow. Similarly, f also gives us a feasible flow in which each node in X_2 sends c units of flow to its partner. The sum of the two flows gives us a congestion two flow that routes c units of flow for each pair of \mathcal{M} . \square

5.2.2 The approximation algorithm for SymANF

In this section, we describe our algorithm for SymANF. Let (G, \mathcal{M}) be an instance of SymANF. The algorithm is the following.

- (1) Solve the relaxation symANF-LP to get an optimal fractional solution (x, f) for the instance (G, \mathcal{M}) .
- (2) Use the well-linked decomposition (Theorem 5.2.5) to get a collection $(G_1, \mathcal{M}_1, \pi_1), \dots, (G_\ell, \mathcal{M}_\ell, \pi_\ell)$ of disjoint instances and weight functions.
- (3) For each instance $(G_i, \mathcal{M}_i, \pi_i)$ in the decomposition, use the clustering technique (Theorem 5.2.6) to get an instance (G_i, \mathcal{M}'_i) .
- (4) For each instance (G_i, \mathcal{M}'_i) , route all of the pairs of \mathcal{M}'_i in G_i (Proposition 5.2.8). Output the union of these routings.

The number of pairs routed by the algorithm is $\sum_{i=1}^\ell |\mathcal{M}'_i| = \sum_{i=1}^\ell \Omega(\pi(V(\mathcal{M}_i))) = \Omega(\text{OPT}/\log^2 k)$. Since each instance (G_i, \mathcal{M}'_i) is $1/32$ -flow-well-linked, the routing in G_i has congestion at most 64. Since the instances are node disjoint, the congestion of the final routing is at most 64. This completes the proof of Theorem 5.1.1.

5.3 Well-linked decomposition

In this section, we prove Theorem 5.2.5. We follow the notation and the approach introduced in [42] for edge and node-capacitated multicommodity flow problems in undirected graphs.

Let (x, f) be a solution to the symANF-LP with value $\text{OPT} = \sum_{i=1}^k x_i$. The flow f is a symmetric multicommodity flow; as before, we view f as a path-based flow. Let H be a node-induced subgraph of G . For each ordered pair (u, v) of nodes in H , let $\gamma(u, v; H)$ be the total amount of f -flow on paths p from u to v that are completely contained in H . For each unordered pair uv of nodes in H , let $\gamma'(u, v; H) = \gamma'(v, u; H) = \min\{\gamma(u, v; H), \gamma(v, u; H)\}$. For each node u in H , let $w(u; H) = \sum_{v \in V(H)} \gamma'(u, v; H)$. Let $\mathbf{w}(H) = \sum_{u \in V(H)} w(u; H)$.

We will need the following observation.

Proposition 5.3.1. *Let $G = (V, E)$ be a directed graph. Let $\pi : V \rightarrow [0, 1]$ be a weight function. Let $\alpha = \alpha(G) \geq 1$ be an upper bound on the worst case flow-cut gap for product multicommodity flows in G . Suppose that V is not π -flow-well-linked in G . There is a node separation (A, B, C) such that $|C| \leq 2\alpha \min \{\pi(A), \pi(B)\}$. Moreover, we can construct such a separation in polynomial time if there is a polynomial time algorithm for computing a node separator with sparsity at most α times the maximum concurrent flow.*

Proof. Note that it follows from Proposition 5.2.2 that, for any weight function $\pi : V \rightarrow [0, 1]$, either V is π -flow-well-linked or there is a node separation (A, B, C) such that $|C| \leq 2\alpha \min \{\pi(A), \pi(B)\}$. Additionally, we can construct such a separation in polynomial time as follows.

Let \mathbf{d} be the following demand vector: $d(u, v) = \pi(u)\pi(v)/\pi(V)$ for each ordered pair (u, v) of nodes. Since \mathbf{d} is not routable, we can compute in polynomial time a node separator C such that $|C| \leq \alpha \text{dem}_{\mathbf{d}}(C)$. By Proposition 5.2.1, once we have C , we can compute in polynomial time a node separation (A, B, C) such that $\text{dem}_{\mathbf{d}}(C) \leq 2 \min \{\pi(A), \pi(B)\}$. The resulting separation (A, B, C) satisfies $|C| \leq 2\alpha \min \{\pi(A), \pi(B)\}$. \square

Note that, in the step (2b) of the algorithm, we used Proposition 5.3.1: let $\pi(u) = w(u; H)/(8\alpha \log \text{OPT})$ for each node u in H ; since $\lambda < 1/(8\alpha \log \text{OPT})$, $V(H)$ is not π -flow-well-linked and therefore there is a node separation (A, B, C) such that

$$|C| \leq 2\alpha \min \{\pi(A), \pi(B)\} = \min \left\{ \sum_{a \in A} w(a; H), \sum_{b \in B} w(b; H) \right\} / (4 \log \text{OPT}).$$

We apply the decomposition algorithm described above to each strongly connected component of G in order to get a decomposition of G into node-induced disjoint subgraphs G_1, G_2, \dots, G_ℓ with associated weight functions $\pi_1, \pi_2, \dots, \pi_\ell$. In the following, we show that this decomposition has the properties required by Theorem 5.2.5. It is straightforward to verify that the decomposition has the first two properties and thus we focus on the third property.

From the terminating conditions, it follows that $\pi_i(G_i) \geq \mathbf{w}(G_i)/(8\alpha \log \text{OPT})$ for each i . Therefore it suffices to show that $\sum_{i=1}^k \mathbf{w}(G_i) \geq \mathbf{w}(G)/2 = \text{OPT}/2$. Equivalently, the total flow lost is at most $\mathbf{w}(G)/2$, where the flow lost is $\mathbf{w}(G) - \sum_{i=1}^k \mathbf{w}(G_i)$.

We upper bound the total flow lost as follows. We say that a node of G was cut in the

Decomposition Algorithm

Input: Strongly connected subgraph H .

Output: Node-disjoint subgraphs H_1, H_2, \dots, H_ℓ with associated weight functions $\pi_1, \pi_2, \dots, \pi_\ell$, where each H_i is a node-induced subgraph of H .

- (1) Suppose that $0 < \mathbf{w}(H) \leq \alpha \log \text{OPT}$. Let $\pi(u) = w(u; H)/(8\alpha \log \text{OPT})$ for each node $u \in V(H)$. Stop and output H and π .
- (2) Suppose that $\mathbf{w}(H) > \alpha \log \text{OPT}$. Let \mathbf{d} be the following demand vector: $d(u, v) = w(u; H)w(v; H)/\mathbf{w}(H)$ for each ordered pair (u, v) of nodes in H . Let λ be the maximum concurrent flow for \mathbf{d} .
 - (a) If $\lambda \geq 1/(8\alpha \log \text{OPT})$, stop the recursive procedure. Let $\pi(u) = w(u; H)/(8\alpha \log \text{OPT})$ for each node $u \in V(H)$. Output H and π .
 - (b) Otherwise find a node separation (A, B, C) such that $|C| \leq \min \{ \sum_{a \in A} w(a; H), \sum_{b \in B} w(b; H) \} / (4 \log \text{OPT})$. Recursively decompose each strongly connected component of $H - C$. Output the decompositions of the strongly connected components.

Figure 5.1: Well-linked decomposition algorithm.

decomposition if the node belongs to a node separator C found in the step (2b). We first note that the total flow lost is at most twice the number of nodes that were cut by the decomposition. We can show this as follows. Let Z be the set of all nodes that are cut by the decomposition. Recall that, for each pair uv of nodes, the amount of f -flow from u to v is equal to the amount of f -flow from v to u ; we think of the flow from u to v and the flow from v to u as partner flows. Now consider the f -flow that does not contribute to $\sum_{i=1}^k \mathbf{w}(G_i)$: this flow can be partitioned into flows, each of which is on paths that intersect Z or it is the partner of a flow whose paths intersect Z . Since each node has unit capacity, the total f -flow on paths that intersect Z is at most $|Z|$ and thus the total flow lost is at most $2|Z|$. Thus it suffices to show that $|Z|$ is at most $\mathbf{w}(G)/4$.

Lemma 5.3.2. *The number of nodes cut by the well-linked decomposition is at most $\mathbf{w}(G)/4$.*

Proof. We charge the cut nodes as follows. Consider an iteration of the decomposition algorithm that cuts a node. Let H denote the graph considered in the current iteration and let (A, B, C) be

the node separation found in Step (2b). Recall that we have

$$|C| \leq \frac{1}{4 \log \text{OPT}} \min \left\{ \sum_{a \in A} w(a; H), \sum_{b \in B} w(b; H) \right\}.$$

We charge the nodes in C as follows. Let $D = A$ if $\sum_{a \in A} w(a; H) \leq \sum_{b \in B} w(b; H)$ and $D = B$ otherwise. We refer to D as the smaller side of the separation (A, B, C) . For each node $u \in D$, we charge $w(u; H)/(4 \log \text{OPT})$ to u . Note that the total charge to the nodes of D is $\sum_{u \in D} w(u; H)/(4 \log \text{OPT}) \geq |C|$.

The total amount charged by the charging scheme is at least the number of nodes that are cut by the decomposition and thus it suffices to upper bound the total amount charged. We can show that the total amount charged is at most $\mathbf{w}(G)/4$ as follows. For each node u , we claim that u is charged at most $w(u; G)/4$. A node u is charged only if it is on the smaller side of the separation found in Step (2b) and therefore it is charged at most $\log(\mathbf{w}(G)) = \log \text{OPT}$ times. Additionally, each charge to u is at most $w(u; G)/(4 \log \text{OPT})$. \square

5.4 From fractional well-linked sets to well-linked sets

In this section, we prove Theorem 5.2.6. We prove the theorem in two steps. In the first step, we show that there exists a set Y of cardinality $\Omega(\pi(X))$ such that Y is $\Omega(1)$ -flow-well-linked. Additionally, the set Y can send flow to X and receive flow from X . In the second step, we use Y to select a matching $\mathcal{M}' \subseteq \mathcal{M}$ of size $\Omega(|Y|)$.

First step: Finding a well-linked set. In the first step, we find a set Y with the following properties:

Theorem 5.4.1. *Let G be a directed graph. Let X be a set of nodes of G and let $\pi : X \rightarrow (0, 1]$ be a weight function on X . Suppose that X is π -flow-well-linked in G . There is a polynomial time algorithm that constructs a set $Y \subseteq V(G)$ with the following properties. Let H be the network obtained from G by assigning a capacity of one to each node. We have*

$$(P_1) \quad |Y| = \lfloor \pi(X)/8 \rfloor.$$

$$(P_2) \quad Y \text{ is } 1/4\text{-flow-well-linked in } G.$$

Additionally, for any subset $X' \subseteq X$ such that $\pi(X') \leq \pi(X)/15$, we have

(Q₁) *There is a single commodity flow in H from X' to Y such that each node $x \in X'$ sends $\pi(x)/64$ units of flow and each node in Y receives at most one unit of flow.*

(Q₂) *There is a single commodity flow in H from Y to X' such that each node $x \in X'$ receives $\pi(x)/64$ units of flow and each node in Y sends at most one unit of flow.*

The main ingredient in the proof of Theorem 5.4.1 is the following lemma. The lemma shows that, if we have a set X that is π -flow-well-linked, then there exists a set Y of size $\Omega(\pi(X))$ such that Y is $\Omega(1)$ -flow-well-linked. The main idea behind the lemma is the following. By Proposition 5.2.3, if X is π -cut-well-linked and Z is a node separator of size less than $\pi(X)/4$, there is a unique strongly connected component $\beta(Z)$ of $G - Z$ whose π -weight is more than half the weight of X . The main insight is that, if we consider the set Y of size $\lfloor \pi(X)/4 \rfloor$ for which $|Y \cup \beta(Y)|$ is minimum, this gives us the desired set. This gives us a non-constructive proof of the existence of such a set Y , and we show that a simple iterative procedure will construct such a set in polynomial time.

Lemma 5.4.2. *Let G be a directed graph. Let X be a set of nodes of G and let $\pi : X \rightarrow (0, 1]$ be a weight function on X . Suppose that X is π -cut-well-linked in G . There is a polynomial time algorithm that constructs a set $Y \subseteq V(G)$ with the following properties. Let H be the network obtained from G by assigning a capacity of one to each node. We have*

(R₁) $|Y| = \lfloor \pi(X)/4 \rfloor$.

(R₂) *There is a single commodity flow in H from X to Y such that each node $x \in X$ sends at most $\pi(x)$ units of flow and each node in Y receives one unit of flow.*

(R₃) *There is a single commodity flow in H from Y to X such that each node in Y sends one unit of flow and each node $x \in X$ receives at most $\pi(x)$ units of flow.*

Proof. We start by introducing some notation. If X is a π -cut-well-linked set in G , it follows from Proposition 5.2.3 that, for each set $Z \subseteq V(G)$ such that $|Z| < \pi(X)/4$, there is a unique strongly connected component $\beta(Z)$ of $G - Z$ such that $\pi(\beta(Z)) > \pi(X)/2$.

Let H_1 be the following network. We start with $H_1 = H$. For each node $x \in X$, we add a node x' to H_1 and an edge from x' to x ; the node x' receives a capacity of $\pi(x)$. We add a source node

s and a directed edge from s to each node x' . We add a sink node t and an edge from each node in Y to t .

Let H_2 be the following network. We start with $H_2 = H$. We add a source node s and a directed edge from s to each node in Y . For each node $x \in X$, we add a node x' to H_2 and an edge from x to x' ; the node x' receives a capacity of $\pi(x)$. We add a sink node t and a directed edge from each node x' to t .

We will maintain a set Y satisfying the first condition. If Y does not satisfy the second or the third condition, we show that we can find a set Y' satisfying the first condition such that $|Y' \cup \beta(Y')| < |Y \cup \beta(Y)|$. Initially, Y is an arbitrary subset of size $\lfloor \pi(X)/4 \rfloor$.

Suppose that Y does not satisfy the second condition. Consider the network H_1 and let X' be the set of all copies of the nodes in X . A triple (A, B, C) is an s - t separation in H_1 if the sets A, B, C partition $V(H_1)$, $s \in A$, $t \in B$, and there is no edge of H_1 from A to B . The capacity of a separation (A, B, C) is the capacity of the nodes in C . Let (A, B, C) be an s - t separation in H_1 with minimum capacity. Since Y does not satisfy the second condition, the capacity of C is smaller than $|Y|$. Let $A' = A - (X' \cup \{s\})$, $B' = B - (X' \cup \{t\})$, and $C' = C - X'$. Since t is in B and there is no edge of H_1 from A to B , we have $Y \subseteq B' \cup C'$.

In the following, we show that $\beta(C') \subseteq A' \cap \beta(Y)$. Since there is no edge of H_1 from A to B , for each node $x \in X \cap B$, we have $x' \in C$: if x' is in A , the edge from x' to x is connecting A to B ; if x' is in B , the edge from s to x' is connecting A to B . Therefore $\text{cap}(C) \geq \pi(B) = \pi(B')$ and thus $\pi(B') \leq \pi(X)/4$. Since $\beta(C')$ is a strongly connected component of $G - C'$ and there is no edge of G from A' to B' , we have that $\beta(C')$ is completely contained in one of A' and B' . Since $\pi(\beta(C')) > \pi(X)/2$ and $\pi(B') \leq \pi(X)/2$, we have $\beta(C') \subseteq A'$. Since $\beta(C')$ is contained in A' , $\beta(C')$ is a strongly connected subgraph of $G - (B' \cup C')$. Since $Y \subseteq B' \cup C'$, there is a unique strongly connected component K of $G - Y$ that contains $\beta(C')$. Since $\beta(C')$ and $\beta(Y)$ overlap at a vertex of X , we have $K = \beta(Y)$. Therefore $\beta(C') \subseteq \beta(Y)$ and thus $\beta(C') \subseteq A' \cap \beta(Y)$, as claimed.

Since $|C'| < |Y|$, we have $|C' \cup \beta(C')| = |C'| + |\beta(C')| < |Y| + |\beta(Y)| = |Y \cup \beta(Y)|$. We let Y' be the set consisting of C' together with an arbitrary subset of $\beta(C')$ of size $\lfloor \pi(X)/4 \rfloor - |C'|$. Then Y' is the desired set.

Therefore we may assume that Y does not satisfy the third condition. The argument is very

similar to the previous case, and we include it for completeness. Consider the network H_2 and let X' be the set of all copies of the nodes in X . A triple (A, B, C) is an s - t separation in H_2 if the sets A, B, C partition $V(H_2)$, $s \in A$, $t \in B$, and there is no edge of H_2 from A to B . The capacity of a separation (A, B, C) is the capacity of the nodes in C . Let (A, B, C) be an s - t separation in H_2 with minimum capacity. Since Y does not satisfy the third condition, the capacity of C is smaller than $|Y|$. Let $A' = A - (X' \cup \{s\})$, $B' = B - (X' \cup \{t\})$, and $C' = C - X'$. Since s is in A there is no edge of H_2 from A to B , we have $Y \subseteq A' \cup C'$.

In the following, we show that $\beta(C') \subseteq B' \cap \beta(Y)$. Since there is no edge of H_2 from A to B , for each node $x \in X \cap A$, we have $x' \in C$: if x' is in A , the edge from x' to t is connecting A to B ; if x' is in B , the edge from x to x' is connecting A to B . Therefore $\text{cap}(C) \geq \pi(A) = \pi(A')$ and thus $\pi(A') \leq \pi(X)/4$. Since $\beta(C')$ is a strongly connected component of $G - C'$ and there is no edge of G from A' to B' , we have that $\beta(C')$ is completely contained in one of A' and B' . Since $\pi(\beta(C')) > \pi(X)/2$ and $\pi(A') \leq \pi(X)/2$, we have $\beta(C') \subseteq B'$. Since $\beta(C')$ is contained in B' , $\beta(C')$ is a strongly connected subgraph of $G - (A' \cup C')$. Since $Y \subseteq A' \cup C'$, there is a unique strongly connected component K of $G - Y$ that contains $\beta(C')$. Since $\beta(C')$ and $\beta(Y)$ overlap at a vertex in X , we have $K = \beta(Y)$. Therefore $\beta(C') \subseteq \beta(Y)$ and thus $\beta(C') \subseteq B' \cap \beta(Y)$. Since $|C'| < |Y|$, we have $|C' \cup \beta(C')| = |C'| + |\beta(C')| < |Y| + |\beta(Y)| = |Y \cup \beta(Y)|$. We let Y' be the set consisting of C' together with an arbitrary subset of $\beta(C')$ of size $\lfloor \pi(X)/4 \rfloor - |C'|$. Then Y' is the desired set. \square

Now we are ready to prove Theorem 5.4.1.

Proof of Theorem 5.4.1: Since X is π -flow-well-linked in G , it follows from Proposition 5.2.2 that X is $(\pi/2)$ -cut-well-linked in G . By Lemma 5.4.2, there is a set Y with the following properties. Let H be the network obtained from G by assigning a capacity of one to each node.

- $|Y| = \lfloor \pi(X)/8 \rfloor$.
- There is a single commodity flow f_1 in H from X to Y such that each node $x \in X$ sends at most $\pi(x)/2$ units of flow and each node in Y receives one unit of flow.
- There is a single commodity flow f_2 in H from Y to X such that each node in Y sends one unit of flow and each node $x \in X$ receives at most $\pi(x)/2$ units of flow.

By Lemma 5.2.4, Y is $1/4$ -flow-well-linked; here we applied the lemma with $A = X$, $B = Y$, $\pi(x) = \pi(x)$ for each $x \in X$, and $\pi'(y) = 1$ for each $y \in Y$.

Let $X_1 \subseteq X$ be the set of all nodes $x \in X$ such that x sends at least $\pi(x)/32$ units of flow in f_1 . We can show that $\pi(X_1) \geq \pi(X)/15$ as follows. For a set $A \subseteq X$, let $F_1(A)$ be the total amount of f_1 -flow sent by the nodes in A . We have $F_1(X) = |Y| \geq \pi(X)/16$. Additionally, since each node $x \in X - X_1$ sends at most $\pi(x)/32$ units of flow in f_1 , we have $F_1(X - X_1) \leq (\pi(X) - \pi(X_1))/32$. It follows that $F_1(X_1) \geq (\pi(X) + \pi(X_1))/32$ and therefore $\pi(X_1)/2 \geq F_1(X_1) \geq (\pi(X) + \pi(X_1))/32$. Thus $\pi(X_1) \geq \pi(X)/15$.

Let $X_2 \subseteq X$ be the set of all nodes $x \in X$ such that x receives at least $\pi(x)/32$ units of flow in f_2 . As before, we have $\pi(X_2) \geq \pi(X)/15$.

Now consider a subset $X' \subseteq X$ such that $\pi(X') \leq \pi(X)/15$. Note that $\pi(X') \leq \pi(X_1)$ and $\pi(X') \leq \pi(X_2)$. Consider the following multicommodity flow instance \mathbf{d} : $d(x', x) = \pi(x')\pi(x)/(32\pi(X_1))$ for each pair $(x', x) \in X' \times X_1$, $d(x, x') = \pi(x)\pi(x')/(32\pi(X_2))$ for each pair $(x, x') \in X_2 \times X'$, and $d(\cdot)$ is zero for all other pairs. Since $d(a, b) \leq \pi(a)\pi(b)/\pi(X)$ for all pairs of nodes (a, b) , there is a feasible flow g that routes \mathbf{d} . The flow g satisfies the following properties:

- Each node $x \in X_1$ receives $\pi(x)\pi(X')/(32\pi(X_1)) \leq \pi(x)/32$ units of flow.
- Each node $x' \in X'$ sends $\pi(x')/32$ units of flow.
- Each node $x \in X_2$ sends $\pi(x)\pi(X')/(32\pi(X_2)) \leq \pi(x)/32$ units of flow.
- Each node $x' \in X'$ receives $\pi(x')/32$ units of flow.

By combining the flows f_1 and g , we get a congestion two flow from X' to Y in which each node in Y receives at most one unit of flow and each node $x' \in X'$ sends $\pi(x')/32$ units of flow. Similarly, by combining the flows f_2 and g , we get a congestion two flow from Y to X' in which each node in Y sends at most one unit of flow and each node $x' \in X'$ receives $\pi(x')/32$ units of flow. We scale down these flows by a factor of two to get feasible flows. \square

Second step: Finding a matching. In the second step, we use the set Y guaranteed by Theorem 5.4.1 in order to select a matching $\mathcal{M}' \subseteq \mathcal{M}$.

We will need the following theorem, which is a slight variant of Theorem 2.1 in [44]. The difference between Theorem 5.4.3 and Theorem 2.1 in [44] is that the former theorem requires that $b'_i \leq b_i$ for each terminal $i \neq j$, whereas the latter theorem requires that $b'_i \leq \lceil b_i \rceil$. One can prove the theorem above using essentially the same argument as in [44]; we include a proof for completeness.

Before stating the theorem, we introduce some notation. Let G be a directed graph with integer arc capacities given by c . Let s_1, s_2, \dots, s_k be distinct source nodes and let t be a sink node. A non-negative vector $\mathbf{b} = (b_1, b_2, \dots, b_k)$ is a *feasible* flow vector if there is a feasible flow in G in which each source s_i sends b_i units of flow to t and t receives $\sum_{i=1}^k b_i$ units of flow. Let \mathcal{B} be the set of all feasible flow vectors. For a vector $\mathbf{b} \in \mathcal{B}$, let $F(\mathbf{b}) = \sum_{i=1}^k b_i$ denote the total flow and let $I(\mathbf{b})$ be the set of all indices i such that b_i is an integer.

Theorem 5.4.3. *Given $\mathbf{b} \in \mathcal{B}$ and $j \notin I(\mathbf{b})$ with $b_j > 0$, we can compute $\mathbf{b}' \in \mathcal{B}$ in polynomial time with $b'_j = \lceil b_j \rceil$ and $F(\mathbf{b}') \geq F(\mathbf{b})$ such that*

- $b'_i \leq b_i$ for each $i \in [k] - \{j\}$, and
- $b'_i = b_i$ for each $i \in I(\mathbf{b})$.

Proof. Recall that $G = (V, A)$ is a network with arc capacities given by c . In the following, we assume for simplicity that, for each pair uv of nodes, at most one of the arcs $u \rightarrow v$, $v \rightarrow u$ is in A ; if this is not the case, we simply subdivide one of the arcs; more precisely, we introduce a new node w and we replace an arc $u \rightarrow v$ by two arcs $u \rightarrow w$ and $w \rightarrow v$, each with capacity $c(u \rightarrow v)$.

Since \mathbf{b} is feasible, there is a flow f that routes \mathbf{b} . We construct a residual graph G_f from G and f in a standard way. More precisely, we define a residual capacity function $c_f : V \times V \rightarrow \mathbb{R}_+$ as follows. For each pair (u, v) , we have

$$c_f(u \rightarrow v) = \begin{cases} c(u \rightarrow v) - f(u \rightarrow v) & \text{if } u \rightarrow v \in A \\ f(v \rightarrow u) & \text{if } v \rightarrow u \in A \\ 0 & \text{otherwise} \end{cases}$$

The residual graph G_f has the same vertex set as G and the following arc set. For each pair (u, v) such that $c_f(u \rightarrow v) > 0$, we add the arc $u \rightarrow v$ to G_f and we assign capacity $c_f(u \rightarrow v)$ to it.

Suppose that G_f has a directed path p that starts at s_j and it ends at either t or a terminal s_h with $h \notin I(\mathbf{b}) \cup \{j\}$. Consider the flow f' in G obtained from f by augmenting along p . More precisely, f' is the following flow in G . Let $\delta = \min_{u \rightarrow v \in p} c_f(u \rightarrow v)$ be the minimum residual capacity on p . Using a standard argument, we show that for each pair (u, v) , we have:

$$f'(u \rightarrow v) = \begin{cases} f(u \rightarrow v) + \delta & \text{if } u \rightarrow v \in p \\ f(u \rightarrow v) - \delta & \text{if } v \rightarrow u \in p \\ f(u \rightarrow v) & \text{otherwise} \end{cases}$$

Note that f' is a feasible flow in G . For each vertex u , let $\Delta_f(u) = \sum_{u \rightarrow v \in A} f(u \rightarrow v) - \sum_{v \rightarrow u \in A} f(v \rightarrow u)$ be the net f -flow from u to t . We define $\Delta_{f'}(u)$ analogously. We claim that, for each vertex u , we have

$$\Delta_{f'}(u) = \begin{cases} \Delta_f(u) + \delta & \text{if } u \text{ is the start node of } p \\ \Delta_f(u) - \delta & \text{if } u \text{ is the end node of } p \\ \Delta_f(u) & \text{otherwise} \end{cases}$$

Suppose that u is the start node of p . Then p contains only one arc $u \rightarrow z$ that is incident to u . For each arc $u \rightarrow v \in A$, we have

$$f'(u \rightarrow v) = \begin{cases} f(u \rightarrow v) + \delta & \text{if } v = z \\ f(u \rightarrow v) & \text{otherwise} \end{cases}$$

For each arc $v \rightarrow u \in A$, we have

$$f'(v \rightarrow u) = \begin{cases} f(v \rightarrow u) - \delta & \text{if } v = z \\ f(v \rightarrow u) & \text{otherwise} \end{cases}$$

Thus we have

$$\begin{aligned}
\Delta_{f'}(u) &= \sum_{u \rightarrow v \in A} f'(u \rightarrow v) - \sum_{v \rightarrow u \in A} f'(v \rightarrow u) \\
&= \Delta_f(u) + \delta \cdot (|\{u \rightarrow z, z \rightarrow u\} \cap A|) \\
&= \Delta_f(u) + \delta
\end{aligned}$$

In the last line we have used the fact that, since $c_f(u \rightarrow z) > 0$, one of the arcs $u \rightarrow z$ or $z \rightarrow u$ is present in G . Additionally, it follows from our assumption on G that only one of $u \rightarrow z$ and $z \rightarrow u$ is present in G .

Suppose that u is the end vertex of p . Then p contains only one arc $w \rightarrow u$ that is incident to u . For each arc $u \rightarrow v \in A$, we have

$$f'(u \rightarrow v) = \begin{cases} f(u \rightarrow v) - \delta & \text{if } v = w \\ f(u \rightarrow v) & \text{otherwise} \end{cases}$$

For each arc $v \rightarrow u \in A$, we have

$$f'(v \rightarrow u) = \begin{cases} f(v \rightarrow u) + \delta & \text{if } v = w \\ f(v \rightarrow u) & \text{otherwise} \end{cases}$$

Thus we have

$$\begin{aligned}
\Delta_{f'}(u) &= \sum_{u \rightarrow v \in A} f'(u \rightarrow v) - \sum_{v \rightarrow u \in A} f'(v \rightarrow u) \\
&= \Delta_f(u) - \delta \cdot (|\{u \rightarrow w, w \rightarrow u\} \cap A|) \\
&= \Delta_f(u) - \delta
\end{aligned}$$

As before, since $c_f(w \rightarrow u) > 0$, one of the arcs $u \rightarrow w$ and $w \rightarrow u$ is present in G . Additionally, by our assumption on G , only one of the arcs $u \rightarrow w$ and $w \rightarrow u$ is present in G .

Suppose that u is an intermediate node on p . Then p contains two arcs $u \rightarrow z$ and $w \rightarrow u$ that

are incident to u . For each arc $u \rightarrow v \in A$, we have

$$f'(u \rightarrow v) = \begin{cases} f(u \rightarrow v) + \delta & \text{if } v = z \\ f(u \rightarrow v) - \delta & \text{if } v = w \\ f(u \rightarrow v) & \text{otherwise} \end{cases}$$

For each arc $v \rightarrow u \in A$, we have

$$f'(v \rightarrow u) = \begin{cases} f(v \rightarrow u) + \delta & \text{if } v = w \\ f(v \rightarrow u) - \delta & \text{if } v = z \\ f(v \rightarrow u) & \text{otherwise} \end{cases}$$

Thus we have

$$\begin{aligned} \Delta_{f'}(u) &= \sum_{u \rightarrow v \in A} f'(u \rightarrow v) - \sum_{v \rightarrow u \in A} f'(v \rightarrow u) \\ &= \Delta_f(u) + \delta \cdot (|\{u \rightarrow z, z \rightarrow u\} \cap A|) - \delta \cdot (|\{u \rightarrow w, w \rightarrow u\} \cap A|) \\ &= \Delta_f(u) \end{aligned}$$

As before, since $c_f(u \rightarrow z) > 0$, one of the arcs $u \rightarrow z$ and $z \rightarrow u$ is present in G . Additionally, by our assumption on G , only one of the arcs $u \rightarrow z$ and $z \rightarrow u$ is present in G . Similarly, exactly one of the arcs $u \rightarrow w$ and $w \rightarrow u$ is present in G . Finally, if u does not appear on p , we have $\Delta_{f'}(u) = \Delta_f(u)$.

For each terminal s_ℓ , we have $b_\ell = \Delta_f(s_\ell)$. Let $b'_\ell = \Delta_{f'}(s_\ell)$. Since p starts at s_j , we have $b'_j = b_j + \delta > b_j$ and $b'_\ell \leq b_\ell$ for all $\ell \neq j$. Since the end vertex on p is not in $I(\mathbf{b}) \cup t$, $b'_\ell = b_\ell$ for each $\ell \in I(\mathbf{b})$. Finally, $F(\mathbf{b}') \geq F(\mathbf{b})$.

Thus, while there exists an augmenting path from s_j to either t or a terminal s_h with $h \notin I(\mathbf{b}) \cup \{j\}$, we can augment the flow along p . The augmentation will increase the flow of s_j while maintaining the properties in the theorem statement. Therefore it suffices to show that, as long as b_j is fractional, there is always such an augmenting path p .

Suppose for contradiction that b_j is fractional but there does not exist an augmenting path from

s_j to either t or a terminal s_h with $h \notin I(\mathbf{b}) \cup \{j\}$. Let S be the set of all nodes reachable from s_j in the residual graph G_f . It follows that $t \notin S$ and, for each terminal s_h with $h \notin I(\mathbf{b}) \cup \{j\}$, $s_h \notin S$. Since no arcs leave S in G_f , for each arc $a \in \delta_G^-(S)$ of G that enters S , we have $f(a) = 0$. Additionally, for each arc $a \in \delta_G^+(S)$ of G that leaves S , we have $f(a) = c(a)$. It follows that $\sum_{a \in \delta_G^+(S)} c(a) = \sum_{i \in S} b_i$. Note that the only terminals in S are s_j and some terminals from $I(\mathbf{b})$. Therefore $\sum_{i \in S} b_i$ cannot be integral, since s_j is fractional and all other terms in the sum are integral. But this contradicts the fact that the sum $\sum_{a \in \delta_G^+(S)} c(a)$ is integral.

Hence there always exists an augmenting path with the desired properties and the theorem follows. \square

Note that the flow augmentation theorem (Theorem 5.4.3) also applies to single-source networks and flows, since we can simply reverse the directions of all of the arcs. It also applies to node-capacitated routing using a standard reduction from node-capacitated directed networks to edge-capacitated directed networks.

Now we are ready to complete the proof of Theorem 5.2.6. Note that we may assume that $\pi(X) \geq c$, for some large enough constant c , since otherwise we can let \mathcal{M}' be a single pair (u, v) of \mathcal{M} that is routable in G . In the following, we assume that $\pi(X) \geq 2048$. Additionally, we may assume that $\pi(x) > 0$ for each node $x \in X$, since we can discard from X all the nodes x such that $\pi(x) = 0$.

Using Theorem 5.4.3, we can identify a large matching whose terminals can send one unit of flow to Y and receive one unit of flow from Y .

Lemma 5.4.4. *There is a matching $\mathcal{M}' \subseteq \mathcal{M}$ with the following properties. Let X'_1 be a set of nodes containing exactly one node from each pair in \mathcal{M}' , and let $X'_2 = V(\mathcal{M}') - X'_1$ be the partners of the nodes in X'_1 . Let H be the network obtained from G by assigning a capacity of one to each node. We have*

$$(C_1) \quad |\mathcal{M}'| = \Omega(|Y|).$$

(C₂) *There is a feasible single-commodity flow in H in which each node in X'_1 sends one unit of flow to Y .*

(C₃) *There is a feasible single-commodity flow in H in which each node in X'_1 receives one unit of flow from Y .*

(C₄) *There is a feasible single-commodity flow in H in which each node in X'_2 sends one unit of flow to Y .*

(C₅) *There is a feasible single-commodity flow in H in which each node in X'_2 receives one unit of flow from Y .*

Proof. Let \mathcal{M}'' be any subset of \mathcal{M} such that $\pi(X)/16 \leq \pi(X'') \leq \pi(X)/15$, where X'' is the set of nodes participating in the pairs of \mathcal{M}'' . Note that such a set \mathcal{M}'' exists, since $\pi(X) \geq 240$ and $\pi(x) \leq 1$ for each node $x \in X$. Let X''_1 be a set of nodes containing exactly one node from each pair in \mathcal{M}'' , and let $X''_2 = V(\mathcal{M}'') - X''_1$ be the partners of the nodes in X''_1 .

In the following, we use the flow augmentation theorem (Theorem 5.4.3) to select a matching $\mathcal{M}' \subseteq \mathcal{M}''$ with the desired properties.

We make four copies of G ; let G_1, G_2, G_3 , and G_4 denote the four copies of G . For each $i \in \{1, 3\}$, we construct a node-capacitated single-sink network H_i from G_i as follows. We start with $H_i = G_i$ and we assign a capacity of one to each node. We add to H_i a sink node t_i and a directed edge from each node in Y to t_i . For each $i \in \{2, 4\}$, we construct a node-capacitated single-source network H_i from G_i as follows. We start with $H_i = G_i$ and we assign a capacity of one to each node. We add to H_i a source node s_i and a directed edge from s_i to each node in Y .

For each $i \in \{1, 2, 3, 4\}$, we maintain a feasible flow vector \mathbf{b}_i in H_i . If $i \in \{1, 2\}$, \mathbf{b}_i has an entry $\mathbf{b}_i(x)$ for each node $x \in X''_1$. If $i \in \{3, 4\}$, \mathbf{b}_i has an entry $\mathbf{b}_i(x)$ for each node $x \in X''_2$.

We initialize the flow vectors \mathbf{b}_i as follows. Let f_1 be the flow from X'' to Y guaranteed by property (Q₁) (see the statement of Theorem 5.4.1). Let f_2 be the flow from Y to X'' guaranteed by property (Q₂). Note that, for each $i \in \{1, 3\}$, f_1 translates to a flow in H_i from X'' to t_i ; we let $\mathbf{b}_i(x)$ denote the amount of flow from x to t_i . Similarly, for each $i \in \{2, 4\}$, f_2 translates to a flow in H_i from s_i to X'' ; we let $\mathbf{b}_i(x)$ denote the amount of flow from s_i to x .

Our goal is to use the flow augmentation theorem (Theorem 5.4.3) in order to select a matching $\mathcal{M}' \subseteq \mathcal{M}''$. The main idea behind the approach is the following. If we have a pair $(u, v) \in \mathcal{M}''$ whose flow is fractional (that is, $\mathbf{b}_1(u) = \mathbf{b}_2(u) = \mathbf{b}_3(v) = \mathbf{b}_4(v) \in (0, 1)$), we use Theorem 5.4.3

in each copy G_i to increase the flow of u and v to 1. We repeatedly apply this procedure until the flow of each pair is either 0 or 1. The pairs with unit flow will give us the desired matching. We now describe this approach more formally.

If \mathbf{b} is a flow vector on $Z \subseteq X''$, we let $F(\mathbf{b}) = \sum_{x \in Z} \mathbf{b}(x)$ be the total flow and $I(\mathbf{b})$ denote the set of all nodes $x \in Z$ such that $\mathbf{b}(x) = 1$. We will maintain flow vectors \mathbf{b}_i that satisfy the following invariants:

(I_1) For each $u \in X_1''$, we have $\mathbf{b}_1(u) = \mathbf{b}_2(u) = \mathbf{b}_3(v) = \mathbf{b}_4(v)$, where v is the partner of u .

(I_2) For each $i \in \{1, 2, 3, 4\}$, we have $F(\mathbf{b}_i) \geq (|Y|/256) - 4|I(\mathbf{b}_1)|$.

We can verify that the initial flow vectors satisfy the invariants as follows. For each pair $uv \in \mathcal{M}$, we have $\mathbf{b}_1(u) = \mathbf{b}_2(u) = \pi(u)/64$ and $\mathbf{b}_3(v) = \mathbf{b}_4(v) = \pi(v)/64$. Since $\pi(a) = \pi(b)$ for each pair $ab \in \mathcal{M}$, the flow vectors satisfy the first invariant. Note that $I(\mathbf{b}_i)$ is empty, since $\mathbf{b}_i(x) = \pi(x)/64 < 1$ for each $x \in X''$. Moreover, we have $F(\mathbf{b}_i) = \pi(X_1'')/64 = \pi(X'')/128$. Since $\pi(X'') \geq \pi(X)/16$ and $|Y| = \lfloor \pi(X)/8 \rfloor$, we have $F(\mathbf{b}_i) = \pi(X'')/128 \geq \pi(X)/2048 \geq |Y|/256$. Thus the flow vectors satisfy the second invariant.

Now consider flow vectors \mathbf{b}_i that satisfy the invariants (I_1) and (I_2) above. Suppose that, for each $u \in X_1''$, we have $\mathbf{b}_1(u) \in \{0, 1\}$. Let $X_1' = I(\mathbf{b}_1)$ and $X_2' = I(\mathbf{b}_3)$; by (I_1), X_2' is the set of all partners of the nodes of X_1' . Let \mathcal{M}' be the set of all pairs $uv \in \mathcal{M}''$ such that $u \in X_1'$ and $v \in X_2'$. We can verify that \mathcal{M}' is the desired matching as follows. We have $|X_1'| = F(\mathbf{b}_1) \geq (|Y|/256) - 4|X_1'|$ and thus $|X_1'| \geq |Y|/1280$. Thus X_1' and X_2' satisfy the conditions (C_1)-(C_5) in the theorem statement and we are done.

Therefore we may assume that there is a node $u \in X_1''$ such that $\mathbf{b}_1(u) \in (0, 1)$. Let v be the partner of u . Recall that we have $\mathbf{b}_1(u) = \mathbf{b}_2(u) = \mathbf{b}_3(v) = \mathbf{b}_4(v)$. For each $i \in \{1, 2\}$, we apply Theorem 5.4.3 with $G = H_i$, $\mathbf{b} = \mathbf{b}_i$, and $b_j = u$ in order to get a feasible flow vector \mathbf{b}_i' such that $I(\mathbf{b}_i') \supseteq I(\mathbf{b}_i) \cup \{u\}$. For each $i \in \{3, 4\}$, we apply Theorem 5.4.3 with $G = H_i$, $\mathbf{b} = \mathbf{b}_i$, and $b_j = v$ in order to get a feasible flow vector \mathbf{b}_i' such that $I(\mathbf{b}_i') \supseteq I(\mathbf{b}_i) \cup \{v\}$. We construct flow vectors \mathbf{b}_i'' as follows. For each pair $zw \in \mathcal{M}''$, we let $\mathbf{b}_1''(z) = \mathbf{b}_2''(z) = \mathbf{b}_3''(w) = \mathbf{b}_4''(w) = \min \{\mathbf{b}_1'(z), \mathbf{b}_2'(z), \mathbf{b}_3'(w), \mathbf{b}_4'(w)\}$.

In the following, we show that the flows \mathbf{b}_i'' satisfy the second invariant. This follows from the properties guaranteed by Theorem 5.4.3. When we augment the flow of u to 1 in G_1 (or G_2), we

decrease the total flow of all other pairs by at most 1. Similarly, when we augment the flow of v to 1 in G_3 (or G_4), we decrease the total flow of all other pairs by at most 1. Thus the total flow decrease in G_1, G_2, G_3 , and G_4 is at most 4, and we charge this flow to the pair uv .

More formally, we claim that $F(\mathbf{b}_i'') \geq F(\mathbf{b}_i) - 4$ for each $i \in \{1, 2, 3, 4\}$. Consider an index $i \in \{1, 2\}$. Note that it suffices to show that $\sum_{z \in X_1'' - \{u\}} (\mathbf{b}_i(z) - \mathbf{b}_i''(z)) \leq 4$. We have

$$\begin{aligned}
& \sum_{z \in X_1'' - \{u\}} (\mathbf{b}_i(z) - \mathbf{b}_i''(z)) \\
&= \sum_{zw \in \mathcal{M}'' - \{uv\}} (\mathbf{b}_i(z) - \min \{\mathbf{b}_1'(z), \mathbf{b}_2'(z), \mathbf{b}_3'(w), \mathbf{b}_4'(w)\}) \\
&= \sum_{zw \in \mathcal{M}'' - \{uv\}} \max \{\mathbf{b}_1(z) - \mathbf{b}_1'(z), \mathbf{b}_2(z) - \mathbf{b}_2'(z), \mathbf{b}_3(w) - \mathbf{b}_3'(w), \mathbf{b}_4(w) - \mathbf{b}_4'(w)\} \\
&\quad \left(\text{Since } \mathbf{b}_1(z) = \mathbf{b}_2(z) = \mathbf{b}_3(w) = \mathbf{b}_4(w) \right) \\
&\leq \sum_{zw \in \mathcal{M}'' - \{uv\}} (\mathbf{b}_1(z) - \mathbf{b}_1'(z) + \mathbf{b}_2(z) - \mathbf{b}_2'(z) + \mathbf{b}_3(w) - \mathbf{b}_3'(w) + \mathbf{b}_4(w) - \mathbf{b}_4'(w)) \\
&\quad \left(\text{The terms in the max are non-negative by the first bullet in Theorem 5.4.3} \right) \\
&= \sum_{i=1}^4 (F(\mathbf{b}_i) - F(\mathbf{b}_i')) + \sum_{i=1}^4 (\mathbf{b}_i(u) - \mathbf{b}_i'(u)) + \sum_{i=1}^4 (\mathbf{b}_i(v) - \mathbf{b}_i'(v)) \\
&\leq 4
\end{aligned}$$

The last inequality follows from the fact that $F(\mathbf{b}_i) \leq F(\mathbf{b}_i')$ for all $i \in \{1, 2, 3, 4\}$, $\mathbf{b}_i(u) - \mathbf{b}_i'(u)$ is at most 0 if $i \in \{1, 2\}$ and at most 1 if $i \in \{3, 4\}$, $\mathbf{b}_i(v) - \mathbf{b}_i'(v)$ is at most 1 if $i \in \{1, 2\}$ and at most 0 if $i \in \{3, 4\}$.

A very similar argument shows that, for each $i \in \{3, 4\}$, we have $F(\mathbf{b}_i'') \geq F(\mathbf{b}_i) - 4$. Thus we have $F(\mathbf{b}_i'') \geq F(\mathbf{b}_i) - 4 \geq (|Y|/256) - 4(|I(\mathbf{b}_1)| + 1)$ for each $i \in \{1, 2, 3, 4\}$. Since $|I(\mathbf{b}_1'')| \geq |I(\mathbf{b}_1)| + 1$, we have $F(\mathbf{b}_i'') \geq (|Y|/256) - 4|I(\mathbf{b}_1'')|$ and thus the second invariant is also satisfied.

We repeatedly apply the flow augmentation procedure until the flow of each pair is either zero or one. The pairs with unit flow will give us the desired matching \mathcal{M}' . \square

Let \mathcal{M}' be the set of pairs guaranteed by Lemma 5.4.4 and let X' be the set of terminals participating in the pairs of \mathcal{M}' . We can show that X' is 1/32-flow-well-linked as follows. Note that the properties $(C_2) - (C_5)$ gives us the following flows: a congestion two flow from X' to Y in which

each node in X' sends one unit of flow and each node in Y receives at most two units of flow, and a congestion two flow from Y to X' in which each node in Y sends at most two units of flow and each node in X' receives one unit of flow. We scale these flows by a factor of 8 to ensure that each node in Y sends and receives at most $1/4$ units of flow. Since Y is $1/4$ -flow-well-linked, it follows from Lemma 5.2.4 that X' is $1/32$ -flow-well-linked; here we applied the lemma with $A = Y$, $\pi(y) = 1/4$ for each $y \in Y$, $B = X'$, $\pi'(x') = 1/8$ for each $x' \in X'$. This completes the proof of Theorem 5.2.6.

5.5 Concluding remarks

In this chapter, we initiate the study of throughput multicommodity flow problems in directed graphs with symmetric demand pairs. Classical routing problems such as **All-or-Nothing Flow** and **Maximum Edge Disjoint Paths** are known to be very hard to approximate in directed graphs; the best approximation we can achieve for these problems is a polynomial approximation even if we allow constant congestion. A closer look at these lower bounds leads to the observation that they rely on the asymmetry of the demand pairs. A natural question is whether we can achieve much better approximations for these problems when the demand pairs are symmetric. In this chapter, we answer this question affirmatively for the **All-or-Nothing** flow problem in directed graphs with symmetric demand pairs. In the process, we give a directed counterpart of the well-linked decomposition of Chekuri, Khanna, and Shepherd [42]. This decomposition is a key technical tool in the algorithms for several routing problems in undirected graphs, including the recent algorithms for routing on disjoint paths [4, 36, 53, 57]. This decomposition is also a first step in a broader research agenda in which one of the main goals is to understand the approximability of **MEDP**. As we remarked in the introduction, this research agenda is connected to several fundamental questions on the structure of graphs with large directed treewidth and, if successful, it will have several useful consequences and applications.

Chapter 6

Prize-collecting Steiner Tree and Forest in Planar Graphs

6.1 Introduction

In this chapter¹ we consider the prize-collecting Steiner Tree and Steiner Forest problems in planar graphs. The Steiner Tree and Steiner Forest problems are fundamental and well-studied problems in network design. In the Steiner Tree problem, we have an undirected graph $G = (V, E)$ with weights on the edges given by $w: E \rightarrow \mathbb{R}_+$, and a set of terminals $S \subseteq V$. The goal is to find a minimum weight tree in G that connects/contains the terminals. In the more general Steiner Forest problem we are given pairs of terminals s_1t_1, \dots, s_kt_k and the goal is to find a minimum weight forest which connects s_i and t_i for each i such that $1 \leq i \leq k$. Both problems have been studied for decades. The Steiner Tree problem was one of Karp's [113] original twenty-one **NP**-hard problems. The problems are also **APX**-hard to approximate [51], and they have received considerable attention in the approximation algorithms literature. A series of papers [18, 102, 112, 152, 163, 172] obtained successively better approximations for the Steiner Tree problem. This line of work culminated with a breakthrough result of Byrka *et al.* [28] that established a 1.388 approximation. For the Steiner Forest problem the best known approximation is $2 - 1/k$ [2], which is obtained via a natural cut-based LP relaxation for the problem.

There has also been a significant interest in obtaining better approximations for these natural problems in restricted classes of graphs. Arora [9] gave a **PTAS** for Steiner Tree and related problems in low-dimensional Euclidean spaces, and Borradaile, Klein, and Mathieu [24] obtained a **PTAS** for Steiner Forest in the Euclidean plane. When G is a planar graph, Borradaile, Klein, and Mathieu [25] obtained a **PTAS** for the Steiner Tree problem, and Bateni, Hajiaghayi, and

¹This chapter is based on joint work with Chandra Chekuri and Nitish Korula and it has appeared in part in [15]. Copyrights to the conference version of [15] are held by the Society for Industrial and Applied Mathematics (SIAM).

Marx [17] obtained a **PTAS** for the Steiner Forest problem.

One obtains more general problems if not all terminals (or terminal pairs in the case of Steiner Forest) need to be connected, but rather an appropriately chosen subset. For instance, in the k -MST problem [10, 11, 52, 82, 83, 147], one wishes to find a minimum weight tree that contains at least k terminals. In another variant, the Max-Prize Tree problem [21, 48], the goal is to connect as many terminals as possible subject to a constraint on the total weight of the tree. In this chapter, we consider the *prize-collecting* versions of the Steiner Tree and Steiner Forest problems, which have received considerable attention [7, 19, 91, 96, 97]. In these problems, it may be very expensive to connect certain terminals, and so one may choose to pay a specified penalty instead of connecting them.

More formally, in the prize-collecting version of Steiner Tree (PCST), we are given a root vertex r and non-negative penalties $\pi : V \rightarrow \mathbb{R}_+$ on the vertices of G . The goal is to find a tree T rooted at r that minimizes the sum of the edge weights of T and the penalties of the vertices not included in T ; formally, the objective is to minimize $\sum_{e \in E(T)} w(e) + \sum_{v \notin V(T)} \pi(v)$. The Steiner Tree problem is the special case with an infinite penalty on the terminals and zero penalty on non-terminals. In the prize-collecting Steiner Forest (PCSF) problem we have a penalty $\pi(uv)$ for each pair of vertices $u, v \in V$. The goal is to find a forest $F \subseteq E$ to minimize $\sum_{e \in F} w(e) + \sum_{uv \text{ not connected by } F} \pi(uv)$.

We are motivated to consider the prize-collecting problems for several reasons. First, they generalize the Steiner Tree and Steiner Forest problems. Second, they have practical applications in access network design and related problems in constructing telecommunications networks [107]. Third, the PCST problem can be interpreted as a Lagrangian relaxation of the k -MST problem [52], and as such, has played a crucial role in the design of algorithms for k -MST [10, 82], k -Stroll [30], and Orienteering [14, 21, 48]. Thus algorithms for the prize-collecting problems, and PCST in particular, are of both theoretical and practical interest.

The first approximation algorithm for the PCST problem was given by Bienstock *et al.* [19], who gave a 3-approximation. Goemans and Williamson [91] improved the approximation factor to 2 shortly after and, seventeen years later, Archer *et al.* [7] achieved an approximation guarantee of ≈ 1.992 . For the PCSF problem, the best known ratio is 2.54 due to Hajiaghayi and Jain [97].

In this chapter, we study these problems in *planar graphs*. For many practical applications

of network design problems such as PCST, PCSF, k -MST, and vehicle routing problems such as Orienteering, the input graphs are typically planar. Our main result is a **PTAS** for the PCST problem in planar graphs. We obtain this result via an approximation preserving reduction from the PCST problem in planar graphs to the problem in graphs of fixed treewidth. Similarly, for the PCSF problem, we reduce the problem in planar graphs to the problem in graphs of fixed treewidth. Our main contributions are summarized in the following theorems.

Theorem 6.1.1. *For each fixed $\varepsilon > 0$, there is a $\rho(1 + \varepsilon)$ -approximation for the prize-collecting Steiner Forest problem in planar graphs if, for each fixed integer k , there is a ρ -approximation for the prize-collecting Steiner Forest problem in graphs of treewidth at most k . In particular, there is a $\rho(1 + \varepsilon)$ -approximation for the prize-collecting Steiner Tree problem in planar graphs if there is a ρ -approximation for the prize-collecting Steiner Tree problem in graphs of fixed treewidth.*

Our next contribution is an exact algorithm for the PCST problem in graphs of fixed treewidth.

Theorem 6.1.2. *For any fixed integer k , there is a polynomial-time algorithm that exactly solves the prize-collecting Steiner Tree problem in graphs of treewidth at most k .*

The reduction guaranteed by Theorem 6.1.1 and the theorem above give us the following result.

Corollary 6.1.3. *There is a **PTAS** for the prize-collecting Steiner Tree problem in planar graphs.*

The reduction given by Theorem 6.1.1 would lead to a **PTAS** for the PCSF problem in planar graphs as well provided that we had an exact algorithm or a **PTAS** for the problem in fixed treewidth graphs. Unfortunately, the PCSF problem is **NP**-hard even in graphs of fixed treewidth [17] and thus we do not expect an exact algorithm in such graphs. Additionally, Bateni *et al.* [16] showed that the PCSF problem is **APX**-hard even in series-parallel graphs, which are planar and have treewidth two. Therefore we do not expect a **PTAS** for PCSF even in series-parallel graphs.

6.1.1 Overview of techniques

Throughout this chapter, we use OPT to denote the cost of an optimal solution to the given problem instance. Also, for any solution F for PCSF (or PCST), we use $\text{Length}(F)$ to denote $\sum_{e \in F} w(e)$, $\text{Penalty}(F)$ to denote $\sum_{uv \text{ not connected by } F} \pi(uv)$, and $\text{Cost}(F)$ to denote $\text{Length}(F) + \text{Penalty}(F)$.

For the PCST problem, we refer to all vertices with non-zero penalty as *terminals*; similarly, for PCSF, we refer to all pairs of vertices with non-zero penalty as *terminal pairs*. Finally, we assume that $\varepsilon \leq 1$ and (w.l.o.g.) that all terminals in the input graph G have degree 1; if this is not the case for some terminal v , simply connect it to a new vertex v' using an edge of cost 0, and use v' as a terminal in the place of v .

Planar graph approximation schemes for Steiner Tree [25] and Steiner Forest [17] build on many ideas starting with the framework of Baker [13] for **PTASes** for planar graphs and utilizing several subsequent technical tools. Before describing the new technical contributions needed to handle prize-collecting problems, we first give a very high-level overview of the approach common to our algorithms, and the previous approximation schemes of [17, 25].²

- (1) Given a planar graph G , construct a *spanner*, a subgraph $H \subseteq G$ such that there exists a solution in H of cost $(1 + \varepsilon)\text{OPT}$ and the total length of edges in H is $f(\varepsilon)\text{OPT}$, for some function f depending only on ε .
- (2) Partition the edges of H into $f(\varepsilon)/\varepsilon$ sets $E_1, E_2, \dots, E_{f(\varepsilon)/\varepsilon}$, such that contracting any set results in a graph of treewidth $O(f(\varepsilon)/\varepsilon)$ [67].
- (3) Pick the set E_i of minimum total edge length; this cost must be no more than εOPT . Contract the edges of this set E_i , yielding a fixed treewidth graph that contains a solution of cost $(1 + \varepsilon)\text{OPT}$. Solve the problem on this graph of bounded treewidth. This may not correspond to a solution in the original graph G ; add (uncontract) edges of E_i as necessary to obtain a feasible solution in G . The total cost of E_i is at most εOPT , and hence the cost of the solution is at most $(1 + 2\varepsilon)\text{OPT}$.

We note that the most difficult step is the first, finding a spanner H . Once this is accomplished, the rest is somewhat standard; applying a theorem of [67] allows one to decompose the edge set of H into pieces, and a simple averaging argument implies that one of these pieces has low cost; contracting this piece yields a graph of fixed treewidth. Depending on the problem being considered, it may be possible to solve the fixed treewidth instance exactly; if not, an approximation algorithm is used. (In [25], steps 2 and 3 are modified slightly to obtain a more efficient algorithm; we omit

²All the algorithms modify this approach in different ways; see the subsequent discussion.

details here.) For PCST, it is fairly straightforward to solve the problem exactly in graphs of fixed treewidth. As mentioned above, PCSF is **NP**-hard even in graphs of fixed treewidth; obtaining a small constant-factor approximation would be of interest as, from Theorem 6.1.1, this would immediately yield an algorithm for planar graphs achieving a similar approximation ratio. Thus, we focus below on Step 1, the construction of the spanner H ; we describe parts of the spanner constructions of [17, 25], and the modifications necessary for the prize-collecting variants.

Prize-collecting Steiner Tree: We first focus on the easier case of **Steiner Tree** and the scheme in [25]. The basic idea to find a spanner for the **Steiner Tree** instance is as follows. (We assume we are given an embedding of the input planar graph G .) The algorithm starts by computing a 2-approximate (any constant factor would do) Steiner tree T in G . One can use an Euler tour of T and splice open along the tour to obtain another plane graph G' in which the Euler tour of T is its outer face; note that the total cost of edges on this face is at most 4OPT . Now all the terminals are on the *outer face* of G' ; this fact is crucial. The algorithm in [25] then builds on the work of Klein [125] to obtain the desired spanner H . It begins with the outer face of G' (containing all the terminals), and adds edges of G of total cost proportional to the length of this outer face, while guaranteeing the existence of a near-optimal solution using only these added edges. We provide details of the construction in Section 6.5.

The main difficulty in extending the above **PTAS** to the PCST problem is the following. We again wish to find a spanner subgraph H of G by starting with an $O(1)$ -approximate tree T and making it the outer face. Unlike the **Steiner Tree** case, we run into a difficulty. The approximate tree T is not guaranteed to contain all the vertices that are connected to the root in an optimal solution. In fact, if we knew the vertices that need to be connected to the root in a near-optimal solution then we could simply reduce the problem to the **Steiner Tree** problem. We overcome this difficulty by proving the following.

Theorem 6.1.4. *There is a polynomial time algorithm that, given a prize-collecting **Steiner Tree** instance in a graph G , outputs a tree T of cost $O(1/\varepsilon)\text{OPT}$ such that there is a $(1+\varepsilon)$ -approximate solution T' that connects to the root a subset of the vertices in T .*

The algorithm to achieve the above is in fact the Goemans-Williamson primal-dual algorithm for the PCST problem but with modified potentials $\pi'(v) = \frac{2}{\varepsilon}\pi(v)$ for each $v \in V$. By exploiting

properties of the primal-dual algorithm we can prove the above theorem. We then proceed as in the **Steiner Tree** case, making this tree T the outer face, and adding edges to form a spanner. We note that we cannot use an (approximation) algorithm for the PCST problem as a black box in proving the above — it is important to rely on the properties of the primal-dual algorithm as we change the potentials.

Prize-collecting Steiner Forest: The **PTAS** for Steiner Forest [17] requires two new ideas. First, as with the **Steiner Tree PTAS**, one starts with an $O(1)$ -approximate Steiner forest. However, this forest has potentially many components and one cannot apply the spanner construction idea of [25] in making a single tree the outer face. At the same time one cannot directly argue that one can treat each tree in the forest separately; optimal (or near-optimal) solutions may connect vertices in different components of the forest. In [17] there is an additional key step in which the trees are grown via a primal-dual type argument and some of them are merged — after this step, the remaining trees are “far apart” and hence can be treated independently via the spanner approach of [25]. (More precisely, one obtains a collection of subgraphs H_1, H_2, \dots such that $\sum_i \text{Length}(H_i) \leq f(\varepsilon)\text{OPT}$. Further, if OPT_i denotes the cost of an optimal solution for the terminal pairs in H_i , then $\sum_i \text{OPT}_i \leq (1 + \varepsilon)\text{OPT}$.)

A second difficulty in obtaining a **PTAS** is in step 3, solving the problem on fixed treewidth graphs. As mentioned previously, a **PTAS** is developed in [17] for Steiner Forest in fixed treewidth graphs.

As for the PCST problem, the difficulty in the PCSF problem is that we do not know which terminal pairs to connect. We deal with this using an approach similar to that for the PCST problem. However, the 3-approximate primal-dual algorithm for the Steiner Forest problem, due to Hajiaghayi and Jain [97], is quite complex; each step requires $O(n)$ max-flow computations. It is unclear whether one can prove a theorem similar to Theorem 6.1.4 via the algorithm in [97]. We develop a simpler primal-dual algorithm and analysis for the Steiner Forest problem that gives a 4-approximation. We use the structure and analysis of our algorithm to prove the following theorem.

Theorem 6.1.5. *There is a polynomial time algorithm that, given a prize-collecting Steiner Forest instance in a graph G , outputs a forest F of cost $O(1/\varepsilon)\text{OPT}$ such that there is a $(1+\varepsilon)$ -approximate*

solution F' that connects a subset of the pairs connected by F .

Other related work: There is a substantial amount of literature on problems related to various aspects of the work discussed in this chapter. We refer the reader to [64] for an overview of the progress on obtaining approximation schemes for optimization problems on planar graphs; recent papers [17,25,125] have pushed the techniques further for network design problems. Prize-collecting versions of network design have received considerable attention following the work of Goemans and Williamson [91] on a primal-dual algorithm which had several applications to other problems such as k -MST. (We note that for a large class of problems such as TSP, Steiner Tree, and Steiner Forest, where the natural LP has a constant integrality gap, one can obtain a weaker constant-factor approximation for their prize-collecting variants.) Following Hajiaghayi and Jain's work on the PCSF problem [97], there have been several other papers on prize-collecting network design problems. Sharma, Swamy and Williamson [157] generalized the approach in [97] to obtain primal-dual constant factor approximation algorithms for prize-collecting constrained forest problems (in the framework of Goemans and Williamson [91]) with submodular penalty functions. Gutner [96] gave a very simple and efficient local-ratio based 3-approximation for the PCSF problem which also applies to the generalized version with more than two terminals in a group; although Gutner's algorithm is quite simple, it does not seem possible to prove Theorems 6.1.4 and 6.1.5 directly via his algorithm. Constant factor approximation ratios have also been obtained for more general problems with higher connectivity requirements [98,134].

Chapter outline: The rest of this chapter is organized as follows. In Section 6.2, we give a 4-approximate primal-dual algorithm for the PCSF problem. In Section 6.3, we prove Theorem 6.1.1, giving the complete reduction from the prize-collecting problems in planar graphs to their fixed treewidth versions. This reduction has three parts. First, in Section 6.3.1 we prove Theorems 6.1.4 and 6.1.5, showing that we can get $O(1/\varepsilon)$ -approximate solutions connecting almost all the terminals connected by an optimal solution. Second, we use these theorems to complete the spanner constructions; the details are provided in Section 6.5. Third, we give the remaining details of the reduction in Section 6.3.2. In Section 6.6, we show how to extend our approach for PCST in order to get a **PTAS** for the prize-collecting Traveling Salesperson problem. Finally, in Section 6.7, we give an exact algorithm for the PCST problem in graphs of fixed treewidth.

Primal-PCSF	Dual-PCSF
$\min \sum_e c_e x_e + \sum_i \pi_i z_i$	$\max \sum_i \sum_{S \in \mathcal{S}_i} y_{i,S}$
$\text{s.t.} \quad \sum_{e \in \delta(S)} x_e \geq (1 - z_i) \quad (\forall i, S \in \mathcal{S}_i)$	$\text{s.t.} \quad \sum_{S: e \in \delta(S)} \sum_{i: S \in \mathcal{S}_i} y_{i,S} \leq c_e \quad (\forall e)$
$x_e, z_i \geq 0 \quad (\forall e, i)$	$\sum_{S \in \mathcal{S}_i} y_{i,S} \leq \pi_i \quad (\forall i)$
	$y_{i,S} \geq 0 \quad (\forall i, S \in \mathcal{S}_i)$

6.2 A primal-dual algorithm for Prize-collecting Steiner Forest

Our algorithm is similar to the Goemans-Williamson [91] primal-dual algorithm for prize-collecting Steiner tree. For the prize-collecting Steiner tree, this gives a 2-approximation (with some additional guarantees), but this does not appear possible in the Steiner forest variant. In this section, we give a simple 4-approximation for the prize-collecting Steiner forest problem, and later exploit properties of this algorithm to prove Theorem 6.1.5.

We give primal and dual linear programming formulations for the prize-collecting Steiner forest problem below. For each pair (s_i, t_i) the variable z_i is 1 if we pay the penalty for not connecting the pair, and 0 otherwise; the variable x_e denotes whether the edge e is selected for the forest. We abuse notation and say that a set S separates (the terminal pair) i if it separates s_i from t_i . Let \mathcal{S}_i denote the collection of sets S that separate i .

A reader unfamiliar with the primal-dual algorithms of [2, 91] (for the Steiner forest and prize-collecting Steiner tree problems respectively) may wish to skip this paragraph and the next, proceeding directly to the description of our algorithm. Here, we briefly describe one way in which our algorithm differs from those of [2, 91]. A natural approach is to initially assign each terminal s_i or t_i a potential of approximately π_i , and make each one an active component. In both the Steiner forest and prize-collecting Steiner tree problems, a natural LP formulation has a single dual variable y_S for each active component S . One increases the y_S variable for each active component S uniformly, until either a component “runs out” of potential, or an edge becomes “tight”. In the former case, the component is deactivated; in the latter, one merges the two components adjacent

to the edge, and combines their potentials.

In the prize-collecting Steiner forest problem, however, we have a *collection* of dual variables $y_{i,S}$ for a single active component S . We still wish each active component S to grow at a uniform rate, but it is now not clear which pair (s_i, t_i) separated by S should pay for the growth of S . Below, we show that there is a natural way to share the cost of this growth among the pairs separated by S ; once this is done, the analysis proceeds as in [2, 91].

Before describing the algorithm, we present some useful notation. The primal-dual algorithm begins with a growth phase, followed by a deletion phase. At all times, our algorithm maintains a forest F of edges. The connected components of F are labeled either *active* or *inactive*; we use \mathcal{C} to denote the set of active components at any given time. Every component is active at the time it is formed; we use $\hat{\mathcal{C}}$ to denote the set of components that were ever formed during the algorithm's execution. (Note that as we only add edges to F during the growth phase, $\hat{\mathcal{C}}$ is a laminar family of components.) When a component $S \in \hat{\mathcal{C}}$ is formed, we assign it a potential $\text{Potential}(S)$ that corresponds (roughly) to the penalty of terminal pairs separated by S ; thus, this measures how much we are willing to pay in order to connect terminals in S to their partners outside. If we grow S by more than its potential without meeting the desired partners, then it is more effective to pay the penalty than connect the pairs separated by S ; at this point, we will mark S inactive. (Of course, some terminals in S may meet their partners while others may not; this is the key difference between Steiner tree and Steiner forest, and we describe how to make this intuition more precise below.)

We form new components by adding edges to F , merging existing components; when two components merge, they combine their potentials. We say that $S \in \hat{\mathcal{C}}$ *unites* i if S is the smallest active component in $\hat{\mathcal{C}}$ containing both s_i and t_i . Terminals are labeled *satisfied*, *alive*, or *dead*; initially, all terminals are alive. If a component S uniting i is formed, the terminals s_i and t_i are marked satisfied if they were both alive immediately before the formation of S . (As satisfied terminals are no longer willing to pay for the growth of a component S containing them, we adjust $\text{Potential}(S)$ if necessary; the procedure `ProcessHistory` handles the necessary bookkeeping.) Once we form a component $S \in \hat{\mathcal{C}}$, we “grow” it by increasing an auxiliary dual variable $y(S)$; simultaneously, we decrease $\text{Potential}(S)$ to pay for this growth. Once $\text{Potential}(S)$ becomes 0, a

component is labeled inactive. (Recall that all components are active when they are formed.) When a component becomes inactive, all of its unsatisfied terminals are marked dead. (Intuitively, it is now more effective to pay the associated penalties than to try to connect them to their partners.) For any terminal s_i (respectively t_i), we use $\text{History}(s_i)$ (respectively $\text{History}(t_i)$) to denote the set of components $S \in \hat{\mathcal{C}}$ such that S contains s_i (t_i) and s_i (t_i) was alive after S was formed.

Finally, we note that the auxiliary variables $y(S)$ do not exist in our dual LP formulation; instead, we have variables $y_{i,S}$. We ensure that at the end of the algorithm, $y(S) = \sum_{i: S \in \mathcal{S}_i} y_{i,S}$. In order to determine how $y(S)$ is split among the variables $y_{i,S}$ we maintain an associated variable $\text{Uncharged}(S)$, the uncharged growth of S . The procedure `ProcessHistory` ensures that $\text{Uncharged}(S)$ (and hence $y(S)$) is split appropriately among the dual variables $y_{i,S}$.

We initialize the set of active components to be the set of terminals, and set $\text{Potential}(s_i) = \text{Potential}(t_i) = \pi_i/2$ for each $1 \leq i \leq h$.

The following proposition is entirely straightforward, as we increase $y(S)$ and $\text{Uncharged}(S)$ together:

Proposition 6.2.1. *For each $S \in \hat{\mathcal{C}}$, as soon as S is marked inactive or S becomes part of a larger component, we have $\text{Uncharged}(S) = y(S)$.*

The procedure `ProcessHistory` ensures that $\text{Uncharged}(S)$ (and hence $y(S)$) is split appropriately among the dual variables $y_{i,S}$. We omit a proof of the proposition below:

Proposition 6.2.2. *When the main While loop of Primal-Dual Forest terminates, for each pair of terminals (s_i, t_i) , we have called $\text{ProcessHistory}(s_i)$ and $\text{ProcessHistory}(t_i)$.*

Lemma 6.2.3. *For each component $S \in \hat{\mathcal{C}}$, $y(S) = \sum_{i: S \in \mathcal{S}_i} y_{i,S}$. For each dead terminal s_i , we have $\sum_{S \in \text{History}(s_i)} y_{i,S} = \pi_i/2$; similarly, for each dead t_i , $\sum_{S \in \text{History}(t_i)} y_{i,S} = \pi_i/2$.*

Proof. It is easy to verify these statements by checking that the algorithm maintains the invariant that for each component $S \in \hat{\mathcal{C}}$, $\text{Potential}(S) + \sum_{S' \in \hat{\mathcal{C}}: S' \subseteq S} \text{Uncharged}(S') = \sum_{\text{Alive } i \in S} \pi_i/2$. \square

Lemma 6.2.4. *The variables $y_{i,S}$ from the algorithm Primal-Dual Forest correspond to a feasible dual solution for the LP Dual-PCSF.*

Primal-Dual Forest

$F \leftarrow \emptyset, \mathcal{C} \leftarrow \{\{s_i\}: 1 \leq i \leq h\} \cup \{\{t_i\}: 1 \leq i \leq h\}$

For all $1 \leq i \leq h$:

Potential($\{s_i\}$) = Potential($\{t_i\}$) = $\pi_i/2$

While there is an active component: $\langle\langle \text{Begin Main Loop} \rangle\rangle$

Increase y_S by Δ for each $S \in \mathcal{C}$ until an edge goes tight,
or a component uses all its potential.

For each $S \in \mathcal{C}$:

Potential(S) \leftarrow Potential(S) $- \Delta$; Uncharged(S) \leftarrow Uncharged(S) $+ \Delta$

If (edge e connecting S_1, S_2 goes tight)

$F \leftarrow F \cup \{e\}$

$\mathcal{C} \leftarrow (\mathcal{C} \setminus \{S_1, S_2\}) \cup \{S_1 \cup S_2\}$

$y(S_1 \cup S_2) \leftarrow 0$; Uncharged($S_1 \cup S_2$) $\leftarrow 0$

Potential($S_1 \cup S_2$) \leftarrow Potential(S_1) + Potential(S_2)

For each i such that $S_1 \cup S_2$ unites i :

If (s_i is alive), Potential($S_1 \cup S_2$) \leftarrow Potential($S_1 \cup S_2$) $-$ ProcessHistory(s_i)

If (t_i is alive), Potential($S_1 \cup S_2$) \leftarrow Potential($S_1 \cup S_2$) $-$ ProcessHistory(t_i)

If (s_i AND t_i are alive), Mark s_i, t_i as satisfied

Else, Mark s_i, t_i as dead

If (component S uses all its potential)

$\mathcal{C} \leftarrow \mathcal{C} \setminus \{S\}$ $\langle\langle \text{Mark } S \text{ as inactive} \rangle\rangle$

For each alive $s_i \in S$: $\langle\langle \text{Similarly for each alive } t_j \in S \rangle\rangle$

Mark s_i as dead

ProcessHistory(s_i)

End While $\langle\langle \text{End Main Loop} \rangle\rangle$

For each edge $e \in F$: $\langle\langle \text{Deletion Phase} \rangle\rangle$

Delete e if $F - e$ does not separate any pair of satisfied terminals

Proof. It is easy to verify that all constraints are satisfied: For any edge e , it is added to F once $\sum_{S:e \in \delta(S)} y(S) = c_e$, and subsequently there is no active component S such that $e \in \delta(S)$. As $y(S) = \sum_{i: S \in \mathcal{S}_i} y_{i,S}$, we satisfy the associated constraint. Further, for any pair s_i, t_i , the procedure ProcessHistory guarantees that $\sum_{S:s_i \in S, t_i \notin S} y_{i,S} \leq \pi_i/2$, and similarly $\sum_{S:s_i \notin S, t_i \in S} y_{i,S} \leq \pi_i/2$. \square

Theorem 6.2.5. *If F denotes the forest returned by the algorithm Primal-Dual Forest, then $\sum_{e \in F} c_e + \sum_{i \text{ separated by } F} \pi_i \leq 4\text{OPT}$, where OPT denotes the cost of an optimal prize-collecting Steiner forest.*

Proof. As OPT is upper bounded by the value of any feasible dual solution, it suffices to show

$$\sum_{e \in F} c_e + \sum_{i \text{ separated by } F} \pi_i \leq 4 \sum_S \sum_{i: S \in \mathcal{S}_i} y_{i,S}.$$

```

ProcessHistory( $s_i$ ):
  RemainingCharge( $s_i$ )  $\leftarrow \pi_i/2$ 
  For each  $S \in \text{History}(s_i)$ , in increasing order of size:
    If Uncharged( $S$ )  $\leq$  RemainingCharge( $s_i$ ):
       $\langle\langle \text{All uncharged growth of } S \text{ can be charged to } s_i \rangle\rangle$ 
      RemainingCharge( $s_i$ )  $\leftarrow$  RemainingCharge( $s_i$ )  $-$  Uncharged( $S$ )
       $y_{i,S} \leftarrow$  Uncharged( $S$ )
      Uncharged( $S$ )  $\leftarrow 0$ 
    Else:  $\langle\langle \text{Charge as much as possible to } s_i \rangle\rangle$ 
      Uncharged( $S$ )  $\leftarrow$  Uncharged( $S$ )  $-$  RemainingCharge( $s_i$ )
       $y_{i,S} \leftarrow$  RemainingCharge( $s_i$ )
      RemainingCharge( $s_i$ )  $\leftarrow 0$ 
  Return 0
Return RemainingCharge( $s_i$ )

```

For any i to be separated by the forest F , either s_i or t_i (or both) must have been marked dead, and thus from Lemma 6.2.3, we have $\pi_i \leq 2 \sum_{S \in \mathcal{S}_i} y_{i,S}$. Hence it suffices to prove

$$\sum_{e \in F} c_e + \sum_{i \text{ separated by } F} 2 \sum_{S \in \mathcal{S}_i} y_{i,S} \leq \left(2 \sum_S \sum_{i: S \in \mathcal{S}_i} y_{i,S} \right) + \left(2 \sum_i \sum_{S \in \mathcal{S}_i} y_{i,S} \right).$$

We prove $\sum_{e \in F} c_e \leq 2 \sum_S y(S)$, which, using the fact that $y(S) = \sum_{i: S \in \mathcal{S}_i} y_{i,S}$, implies the inequality above. One can now use the standard primal-dual proof technique of [2, 91]. Since an edge e is added to F only when it becomes tight, we have $c_e = \sum_{S: e \in \delta(S)} y(S)$; hence, the desired inequality is equivalent to:

$$\sum_S |\delta(S) \cap F| y(S) \leq 2 \sum_S y(S).$$

To verify this inequality, we check that in every iteration of the while loop, the *increase* in both the left- and right-hand sides satisfies the inequality. Since $y(S)$ increases only for components that are active in a given iteration, and we raise $y(S)$ uniformly by Δ for each $S \in \mathcal{C}$, this is equivalent to checking that in any iteration, $\sum_{\text{Active } S} |\delta(S) \cap F| \Delta \leq 2\Delta \cdot n_a$, where n_a denotes the number of components active in this iteration.

Construct an auxiliary graph H by beginning with $G(V, F)$, and shrinking each currently active and inactive component to a single vertex. Discard any isolated vertex of H corresponding to an inactive component. We now argue that the average degree in H of the vertices corresponding to active components is at most 2; this completes the proof. To bound the average degree of the

vertices corresponding to active components, we note that the average degree of all vertices is less than 2 (as H is a forest), and that each vertex corresponding to an inactive component has degree at least 2. If the latter is not true, there is an inactive component which is a leaf of H , incident to a single edge e . But no inactive component separates a satisfied terminal from its partner, and so the edge e would have been deleted from F in the Deletion Phase of Primal-Dual Forest. \square

Running time: Primal-Dual Forest can be implemented in $O(nh + n^2 \log n)$ time where h is the number of pairs with strictly positive penalties and n is the number of nodes in the graph. The only additional difficulty in the algorithm as compared to the primal-dual algorithmic framework of Goemans and Williamson [91] for prize-collecting Steiner tree and related problems is the **ProcessHistory** step. When two active components merge, for each pair $s_i t_i$ united by this merge, we have to go over the sets in **History**(s_i) and **History**(t_i). The sets in $\hat{\mathcal{C}}$ form a laminar family on V and hence $|\hat{\mathcal{C}}| = O(n)$ and thus **ProcessHistory**(s_i) can be implemented in $O(n)$ time by considering each set in $\hat{\mathcal{C}}$, and similarly for **ProcessHistory**(t_i). Since a pair is united at most once, and each terminal is marked dead at most once, the total work involved in processing the histories is $O(nh)$. Priority queues can be used to keep track of the events corresponding to edges becoming tight and components becoming inactive; this is very similar to the implementation in [91] and the total work involved is $O(n^2 \log n)$ time.

6.3 The reduction to fixed treewidth: building spanners

Recall that the first step in building a spanner for the Steiner tree and forest problems was to construct a new plane graph G' in which (i) all terminals are on the outer face and (ii) the length of the outer face is $O(1) \cdot \text{OPT}$. This was done by splicing open an Euler tour of an $O(1)$ approximate solution in the original graph G , and converting the tour into the outer face.³ In the prize-collecting versions, however, we do not know which terminals to connect, and it is not possible to find an $O(1)$ -approximate solution in which *all* the terminals are connected to the root (for Steiner tree) or to their partners (for Steiner forest).

In Section 6.3.1, we prove Theorems 6.1.4 and 6.1.5, showing that we can find a solution of

³In the case of Steiner forest, this is done separately for distinct trees in the $O(1)$ -approximate forest.

cost $O(1/\varepsilon)\text{OPT}$ which connects “almost” all the terminals connected by any optimal solution. More precisely, the total penalty of terminals connected by an optimal solution but not by our $O(1/\varepsilon)$ -approximate solution is at most εOPT . In Section 6.5, we complete the construction of the spanner, using ideas from [17, 25].

6.3.1 Scaling penalties to capture important terminals

We prove Theorem 6.1.5, which implies Theorem 6.1.4, as prize-collecting Steiner tree is a special case of prize-collecting Steiner forest. Given an instance I of the prize-collecting Steiner forest on a graph $G(V, E)$, with c_e denoting the cost of edge $e \in E$ and π_i the penalty for not connecting s_i to t_i , we define a new instance I' as follows: The graph and edge cost functions are unchanged, but we scale the penalties so that the penalty for not connecting s_i to t_i is $\pi'_i = 2\pi_i/\varepsilon$.

Theorem 6.3.1. *Let F^* be any optimal solution to an instance I of prize-collecting Steiner forest, and let $\text{OPT} = \sum_{e \in F^*} c_e + \sum_{i \text{ separated by } F^*} \pi_i$. Let F' be the forest output by algorithm Primal-Dual Forest on the instance I' with penalties scaled as above. Let X denote the index set of the terminal pairs separated by F' but not by F^* . Then, $\sum_{e \in F'} c_e \leq 8\text{OPT}/\varepsilon$, and $\sum_{i \in X} \pi_i \leq \varepsilon\text{OPT}$.*

Proof. We first note that the cost of an optimal solution to I' is at most $2\text{OPT}/\varepsilon$; simply use the forest F^* , which pays $2/\varepsilon$ times as much penalty for every separated pair as it did in I . Thus, as Primal-Dual Forest is a 4-approximation, we have $\sum_{e \in F'} c_e \leq 8\text{OPT}/\varepsilon$.

To prove that the total penalty of pairs in X is small, consider a Steiner forest instance defined on these pairs: As F^* connects all the terminals in X to their partners, the cost of an optimal Steiner forest for X is at most OPT . Suppose, by way of contradiction, that $\sum_{i \in X} \pi_i > \varepsilon\text{OPT}$, and hence that $\sum_{i \in X} \pi'_i > 2\text{OPT}$. Now consider the dual of a natural LP for the Steiner Forest instance induced by X that is given in Figure 6.1.

Let $y_{i,S}$ be the feasible solution to Dual-PCSF returned by Primal-Dual Forest on instance I' . Now, construct a dual solution to the LP Dual-Steiner-Forest(X) as follows: For each set S separating some pair $s_i t_i$ with $i \in X$, set $z_S = \sum_{i \in X} y_{i,S}$. As $\sum_{S: e \in \delta(S)} \sum_{i: S \in \mathcal{S}_i} y_{i,S} \leq c_e$ from the feasibility of the solution to Dual-PCSF, we conclude that the dual variables z_S correspond to a feasible solution of Dual-Steiner-Forest(X).

$$\begin{array}{c}
\text{Dual-Steiner-Forest}(X) \\
\max \sum_{S \text{ separating some } i \in X} z_S \\
\sum_{S: e \in \delta(S)} z_S \leq c_e \quad (\forall e) \\
z_S \geq 0 \quad (\forall S)
\end{array}$$

Figure 6.1: Dual LP for Steiner Forest.

Thus, we have a feasible solution to $\text{Dual-Steiner-Forest}(X)$ of total value $\sum_S \sum_{i \in X: S \in \mathcal{S}_i} y_{i,S}$. But each $i \in X$ was not connected by F' , and so we must have marked either s_i or t_i as dead. Hence, from Lemma 6.2.3 $\sum_{S \in \mathcal{S}_i} y_{i,S} \geq \pi'_i/2$. That is, the value of our feasible dual solution is at least $\sum_{i \in X} \pi'_i/2 > \text{OPT}$. By weak duality, the length of any Steiner forest for X must be greater than OPT . But F^* is a Steiner forest for X of total length at most OPT , which is a contradiction. \square

We can now prove Theorem 6.1.5:

Proof of Theorem 6.1.5: Let F^* be an optimal solution to a given instance I of Steiner forest, and let $\text{OPT} = \sum_{e \in F^*} c_e + \sum_{i \text{ separated by } F^*} \pi_i$. Construct a forest F by running the Primal-Dual Forest algorithm on the scaled instance I' ; from Theorem 6.3.1 above, the total length of edges in F is at most $8\text{OPT}/\varepsilon$. If X denotes the terminal pairs separated by F but not by F^* , the penalty paid by F is at most $\sum_{i \text{ separated by } F^*} \pi_i + \sum_{i \in X} \pi_i \leq \text{OPT} + \varepsilon\text{OPT}$. Thus, F is a forest of total cost $O(8/\varepsilon + (1 + \varepsilon))\text{OPT}$.

It remains only to argue that there is a $(1 + \varepsilon)$ -approximate solution F' that connects a subset of the pairs connected by F . Let F_{-X}^* denote a solution formed from F^* by paying the penalty for any terminal pair in X ; clearly, the cost of F_{-X}^* is the cost of X added to $\sum_{i \in X} \pi_i$, which is at most $\text{OPT} + \varepsilon\text{OPT}$. \square

The first part of the spanner construction for the prize-collecting Steiner tree problem is now complete: As guaranteed by Theorem 6.1.4, find a tree T of cost $O(1/\varepsilon)\text{OPT}$ such that there exists a $(1 + \varepsilon)$ -approximate solution connecting only terminals in T . Now form an Euler tour of T by duplicating edges, splice along this tour, and make the tour the outer face of a new graph G' . Now

we have a graph in which all (relevant) terminals are on the outer face, and the total length of the outer face is $O(1/\varepsilon)\text{OPT}$. The rest of the spanner construction proceeds along the lines of [25]; see Section 6.5.

For the prize-collecting Steiner forest problem, however, more work is required. The forest F guaranteed by Theorem 6.1.5 may have many components, which cannot be treated in isolation. As in [17], we use a *prize-collecting* clustering scheme to merge some components of the forest. The intuition is that after this clustering, the remaining components are “far apart”, and hence can be treated separately. The prize-collecting clustering algorithm is as follows: Contract each tree T_i of F to a single vertex v_i , to obtain a new graph \hat{G} . We let v_1, v_2, \dots denote the vertices of \hat{G} corresponding to the contracted trees T_1, T_2, \dots of F ; note that \hat{G} additionally has the vertices of G not contained in any tree T_i . With each v_i , we associate a potential $\phi_{v_i} = \frac{1}{\varepsilon} \text{Length}(T_i)$. Now run the standard prize-collecting primal-dual algorithm as in [91], using potentials ϕ_{v_i} . Initially, each vertex is an active component, with potential ϕ_{v_i} ; in each step, the algorithm decreases the potentials of all active components uniformly until either a component runs out of potential, or an edge becomes “tight”. In the former case, the component is marked as inactive. In the latter case, the two components adjacent to the edge are merged, and their potentials combined. This is similar to the algorithm **Primal-Dual Forest** in Section 6.2, but simpler, as we do not have the additional accounting necessary to handle terminal pairs that only wish to connect to each other. As in **Primal-Dual Forest**, the algorithm maintains a collection of dual variables $y_{v_i, S}$ for each v_i and set $S \subseteq V(\hat{G})$. A complete description of this algorithm is given in [17].

The first stage of the clustering algorithm terminates when all components are marked inactive. Let F_1 denote the forest of tight edges selected by the algorithm after the first stage. In the second stage, we delete any edge e from F_1 if it is the unique edge incident to an inactive component. Let F_2 denote the set of edges remaining.

Lemma 6.3.2 ([17]). *The total length of all edges in F_2 is at most $2 \sum_i \phi_{v_i} = \frac{2}{\varepsilon} \cdot \text{Length}(F)$, which is $O(\frac{1}{\varepsilon^2})\text{OPT}$.*

Let $\mathcal{T}^1, \mathcal{T}^2, \dots$ be the trees comprising the forest F_2 . We will now argue that these trees are sufficiently “far apart”, and so we can treat them separately; we formalize this intuition in the rest of this subsection. Recall that from Theorem 6.1.5, there is a $(1 + \varepsilon)$ -approximate solution

(in the original graph G) that does not connect (that is, pays the penalty for) terminal pairs in distinct components of F , the forest returned by Primal-Dual Forest. Let F^* denote this solution; $\text{Cost}(F^*) \leq (1 + \varepsilon)\text{OPT}$. We construct a set of prize-collecting Steiner forest instances, one for each tree \mathcal{T}^j of F_2 . In instance I^j , we have $\pi_i^j = \pi_i$ for each pair (s_i, t_i) connected by \mathcal{T}^j , and $\pi_i^j = 0$ for all other pairs. Let OPT^j denote the cost of an optimal prize-collecting Steiner forest to instance I^j ; we prove the following theorem:

Theorem 6.3.3 (Following [17]). $\sum_j \text{OPT}^j \leq (1 + \varepsilon)\text{Cost}(F^*)$.

Given this theorem, we can separately solve each instance I^j ; it is easy to see that if we obtain a ρ -approximation to each instance, combining them yields a solution of cost at most $\rho(1 + \varepsilon)\text{Cost}(F^*) = \rho(1 + O(\varepsilon))\text{OPT}$. But for each instance I^j , the tree \mathcal{T}^j contains *all* terminal pairs with non-zero penalty, and hence we can splice open the tree and convert the corresponding Euler tour into the outer face to obtain a graph in which the length of the outer face is bounded, and all relevant terminals are on this outer face. This allows us to proceed with the spanner construction as described in Section 6.5.

Thus, it remains only to prove Theorem 6.3.3; this closely follows the work of [17], with some additional care needed because the cost of a forest includes both its length and penalty. The proof of Theorem 6.3.3 can be found in Section 6.4.

Finishing the Spanner: We can now complete the spanner construction using the following theorem, implicit in the work of [25]:

Theorem 6.3.4. *Let I be an instance of prize-collecting Steiner forest on a planar graph G . Let F^* be an optimal solution to I . Given a tree T spanning all terminals of I , for any fixed $\varepsilon > 0$, there is a polynomial time algorithm to find a planar graph $H \subseteq G$ such that: (i) $\text{Length}(H) \leq f(\varepsilon) \cdot \text{Length}(T)$ for some function f that depends (exponentially) on ε and (ii) there is a solution to instance I in the graph H of cost no more than $(1 + \varepsilon)\text{Cost}(F^*) + \varepsilon \cdot \text{Length}(T)$.*

The similar theorem stated in [25] for instances of the Steiner tree problem is slightly less general, though their proof technique can be used to show the theorem we state here. For completeness, we provide a proof in Section 6.5.

6.3.2 Completing the reduction

In Section 6.3.1, we constructed a forest F_2 by first running the algorithm Primal-Dual Forest on a modified instance with scaled penalties, and then running the prize-collecting clustering algorithm of [17]. We proved two useful properties of F_2 : In Lemma 6.3.2, we showed that $\text{Cost}(F_2) \leq (20/\varepsilon^2)\text{OPT}$, and in Theorem 6.3.3, we argued that we could separately solve a prize-collecting Steiner forest instance induced by each tree \mathcal{T}^j of F_2 without increasing the cost significantly. (Formally, we showed that $\sum_j \text{OPT}^j \leq (1 + \varepsilon)\text{OPT}$.)

Now, for each tree \mathcal{T}^j of F_2 , construct a spanner for the instance I^j . (Recall that \mathcal{T}^j spans all terminals in I^j .) Using a parameter $\varepsilon' = \varepsilon^3/20$, Theorem 6.3.4 guarantees a spanner H^j for I^j such that (i) $\text{Length}(H^j) = f'(\varepsilon) \cdot \text{Length}(\mathcal{T}^j)$ for some function f' depending only on ε , and (ii) $\text{OPT}(H_j) \leq (1 + \varepsilon')\text{OPT}^j + \varepsilon' \cdot \text{Length}(\mathcal{T}^j)$, where $\text{OPT}(H_j)$ denotes the cost of an optimal prize-collecting Steiner forest in H_j . Hence,

$$\begin{aligned} \sum_j \text{OPT}(H_j) &\leq (1 + \varepsilon') \sum_j \text{OPT}^j + \varepsilon' \sum_j \text{Length}(\mathcal{T}^j) \\ &\leq (1 + 2\varepsilon)\text{OPT} + \frac{\varepsilon^3}{20} \cdot \frac{20}{\varepsilon^2} \text{OPT} \\ &= (1 + 3\varepsilon)\text{OPT} \end{aligned}$$

where the second inequality follows from Lemma 6.3.2 and Theorem 6.3.3.

Thus, if we can obtain a ρ -approximation to each instance I^j in the graph H^j , we obtain a $\rho(1+3\varepsilon)$ -approximation to the original prize-collecting Steiner forest instance. We use the following theorem of [67].

Theorem 6.3.5 (Demaine, Hajiaghayi, Mohar [67]). *Let G be any planar graph, and let k be any integer such that $k \geq 2$. The edges of G can be partitioned into k sets such that contracting any one of the sets results in a graph of treewidth $O(k)$. Furthermore, this partition can be found in polynomial time.*

Proof of Theorem 6.1.1: Let H^j be the spanner for instance I^j as constructed above. Set $k = (1/\varepsilon') \cdot f'(\varepsilon)$, where $f'(\varepsilon)$ is the function such that $\text{Length}(H^j) \leq f'(\varepsilon)\text{Length}(\mathcal{T}^j)$.

Let E_1, \dots, E_k be the decomposition of the edges of H^j that is guaranteed by Theorem 6.3.5.

Let E_{i^*} be the set of edges that has minimum length among the sets E_1, \dots, E_k . We have

$$\text{Length}(E_{i^*}) \leq \frac{\text{Length}(H^j)}{k} \leq \frac{f'(\varepsilon) \cdot \text{Length}(T^j)}{k} = \varepsilon' \cdot \text{Length}(T^j).$$

Let $\widehat{H}^j = H^j / E_{i^*}$; that is, \widehat{H}^j is the graph obtained from H^j by contracting the edges in E_{i^*} . We assign new penalties to terminal pairs of \widehat{H}^j in the natural way: When we contract an edge uv into a single vertex w , we replace each terminal pair (u, x) with a new pair (w, x) with the same penalty as (u, x) , and we similarly replace each terminal pair (v, y) with a new pair (w, y) with the same penalty as (v, y) . Let $\text{OPT}(\widehat{H}^j)$ denote the cost of an optimal prize-collecting Steiner forest in \widehat{H}^j ; it is obvious that $\text{OPT}(\widehat{H}^j) \leq \text{OPT}(H^j)$.

Since \widehat{H}^j has treewidth at most k , if there is a ρ -approximation for prize-collecting Steiner forest in graphs of fixed treewidth, we can find a ρ -approximate forest \widehat{F}^j in \widehat{H}^j . We can then map \widehat{F}^j to a forest F^j in H^j using the edges in E_{i^*} . By construction,

$$\text{Cost}(F^j) \leq \text{Cost}(\widehat{F}^j) + \text{Length}(E_{i^*}) \leq \rho \text{OPT}(H^j) + \varepsilon' \cdot \text{Length}(T^j).$$

Combining such solution F^j for each H^j , we find a forest of total cost $\sum_j \rho \text{OPT}(H^j) + \varepsilon' \sum_j \text{Length}(T^j)$. Using equation (1), the first term is at most $\rho(1 + 3\varepsilon) \text{OPT}$, and from the choice of ε' , the second term is at most εOPT . \square

6.4 Proof of Theorem 6.3.3

We use the following two technical lemmas from [17]. A graph $\hat{H} \subseteq \hat{G}$ is said to *exhaust* a vertex $v_i \in V(\hat{G})$ if, for all S such that $y_{v_i, S} > 0$, \hat{H} contains at least one edge of $\delta(S)$. (Recall that v_i corresponds to the contracted tree T_i of F .)

Lemma 6.4.1 (Lemma 10 of [17]). *Let V' be the set of vertices of \hat{G} exhausted by a graph H . $\text{Length}(H) \geq \sum_{v_i \in V'} \phi_{v_i}$.*

Lemma 6.4.2 ([17]). *Let $\hat{H} \subseteq \hat{G}$ connect two vertices v_1, v_2 in distinct components of F_2 . Then, \hat{H} exhausts at least one of v_1, v_2 .*

We can now complete the proof of Theorem 6.3.3. Recall that F denotes the forest returned by Primal-Dual Forest, and F^* is an optimal prize-collecting Steiner forest that pays the penalty for

terminal pairs in distinct components of F . Trees in F were contracted to form vertices of \hat{G} ; we then ran the prize-collecting clustering algorithm to form trees $\mathcal{T}^1, \mathcal{T}^2, \dots$, making up the forest F_2 . We wish to show that if OPT^j denotes the cost of an optimal solution for the instance I^j (induced by terminal pairs in tree \mathcal{T}^j of F_2), then $\sum_j \text{OPT}^j \leq (1 + \varepsilon) \text{Cost}(F^*)$.

To prove this theorem, we construct a set \mathcal{D} of trees $\{T'_p\}$ such that each $T'_p \in \mathcal{D}$ only connects terminal pairs in a single component of F_2 . Further, $\sum_p \text{Length}(T'_p) \leq (1 + \varepsilon) \text{Length}(F^*)$, and every pair connected by F^* is connected by some T'_p . Such a set \mathcal{D} of trees clearly yields solutions to the instances I^j , proving the theorem.

We now construct the desired set \mathcal{D} . We begin by setting \mathcal{D} to be the collection of trees in F^* , and then modify it as follows. Let v_i be any vertex of \hat{G} exhausted by F^* ; prune from F^* all terminals in the tree T_i of F contracted to form v_i , and add the tree T_i to \mathcal{D} . When this process terminates, each tree in \mathcal{D} only connects pairs in a single component of F_2 . Suppose this were not true; all trees *added* to \mathcal{D} clearly satisfy this condition, so it only remains to consider trees originally in F^* (from which some terminals may have been pruned). But any tree T^* of F^* connecting two vertices of \hat{G} in distinct components of F_2 must exhaust one of these vertices (from Lemma 6.4.2), and hence the corresponding terminals should have been pruned from T^* , which yields a contradiction. It is also easy to see that any terminal pairs connected by F^* are also connected by some tree in \mathcal{D} .

To bound the cost of the trees in \mathcal{D} , we simply show that the length of the trees *added* to \mathcal{D} is at most $\varepsilon \cdot \text{Length}(F^*)$. Let v_i denote the vertex of \hat{G} corresponding to the tree T_i of F , and V' the set of vertices exhausted by F^* . The length of the added trees is simply $\sum_{v_i \in V'} \text{Length}(T_i)$. From Lemma 6.4.1, $\text{Length}(F^*) \geq \sum_{v_i \in V'} \phi_{v_i} = \sum_{v_i \in V'} \text{Length}(T_i) / \varepsilon$. Rearranging, we get $\sum_{v_i \in V'} \text{Length}(T_i) \leq \varepsilon \cdot \text{Length}(F^*)$.

6.5 Proof of Theorem 6.3.4

We begin by duplicating the edges of T , and introducing multiple copies of its non-leaf vertices in order to transform the Euler tour corresponding to T into a cycle. Let G' be the resulting graph. We then make this cycle the outer face Δ of G' .

Definition 6.5.1 (Definition 6.2, Borradaile *et al.* [25]). *A path P is ε -short in G' if for every pair*

of vertices x and y on P , the distance from x to y along P is at most $(1 + \varepsilon)$ times the distance from x to y in G' (i.e., $\text{dist}_P(x, y) \leq (1 + \varepsilon)\text{dist}_{G'}(x, y)$).

Strips. Let $\Delta[x, y]$ denote the subpath (in clockwise direction) of the outer face Δ from x to y . We find a pair of vertices x, y on Δ such that $\Delta[x, y]$ is a minimal subpath of Δ that is not ε -short in G' . Let N be a shortest path from x to y in G' . The subgraph enclosed by $\Delta[x, y] \cup N$ is a *strip*. We recursively decompose the subgraph of G' enclosed by $N \cup (\Delta - \Delta[x, y])$ into strips, if the graph is nontrivial.

Lemma 6.5.2 (Lemma 6.3, Borradaile *et al.* [25]). *The total length of all the boundary edges of all the strips is at most $(\varepsilon^{-1} + 1) \cdot \text{Length}(\Delta)$.*

Columns. Consider a strip, with north and south boundaries N and S (N is the shortest path we added when we created the strip). We select vertices s_0, s_1, \dots on S and paths C_0, C_1, \dots inside the strip as follows. The vertex s_0 is the left endpoint common to S and N , and column C_0 is the (empty) shortest path from s_0 to N . Now suppose that we have selected vertices s_0, s_1, \dots, s_{i-1} and columns C_0, C_1, \dots, C_{i-1} . The vertex s_i is the first vertex on S such that the distance from s_{i-1} to s_i on S is greater than ε times the distance from s_i to N in the strip, and the column C_i is the shortest path in the strip from s_i to N .

Lemma 6.5.3 (Lemma 6.4, Borradaile *et al.* [25]). *The sum of the lengths of the columns in a strip is at most $\text{Length}(S)/\varepsilon$, where S is the south boundary of the strip.*

Supercolumns. Let

$$k = \frac{1}{\varepsilon^2} \left(\frac{1}{\varepsilon} + 1 \right).$$

For each strip, we select a subset of the columns $\{C_0, C_1, \dots\}$ of the strip as follows. Let

$$\mathcal{C}_i = \{C_j \mid j \equiv i \pmod{k}\},$$

where $0 \leq i \leq k - 1$. Let i^* be the index that minimizes $\text{Length}(\mathcal{C}_i)$. We designate the columns in \mathcal{C}_{i^*} as the supercolumns of the strip.

Lemma 6.5.4 (Lemma 6.5, Borradaile *et al.* [25]). *The sum of the lengths of the supercolumns in a strip is at most $1/k$ times the sum of the lengths of the columns in the strip.*

Lemma 6.5.5. *The sum over all strips of the length of all the supercolumns is at most $\varepsilon \cdot \text{Length}(\Delta)$.*

Proof. By Lemma 6.5.2 and Lemma 6.5.3, the total length of the columns is at most

$$\frac{1}{\varepsilon} \left(\frac{1}{\varepsilon} + 1 \right) \text{Length}(\Delta).$$

By Lemma 6.5.4, the total length of the supercolumns is at most

$$\frac{1}{k} \cdot (\text{total length of columns}) \leq \frac{1}{k} \cdot \frac{1}{\varepsilon} \left(\frac{1}{\varepsilon} + 1 \right) \text{Length}(\Delta) = \varepsilon \cdot \text{Length}(\Delta).$$

□

Mortar Graph. The mortar graph MG is a subgraph of *the original graph* G consisting of the edges of the given tree T (that was doubled to form the outer face Δ of G'), the edges of the shortest paths that define the strips, and the edges of the supercolumns.

Lemma 6.5.6. *The length of the mortar graph MG is at most $(\frac{3}{\varepsilon} + \varepsilon) \cdot \text{Length}(\Delta)$.*

Proof. The total length of the strips is at most

$$\left(\frac{1}{\varepsilon} + 1 \right) \text{Length}(\Delta).$$

The total length of the supercolumns is at most $\varepsilon \cdot \text{Length}(\Delta)$. The length of T is precisely half the length of Δ , which consisted of two copies of each edge of T . Thus, the total length of MG is at most $(\frac{1}{\varepsilon} + 1.5 + \varepsilon) \cdot \text{Length}(\Delta)$ □

Proposition 6.5.7. *The mortar graph MG contains every vertex of T .*

Bricks. A brick consists of all edges of the original graph G that are (strictly) enclosed by the boundary of some face f of the mortar graph. (Note that if an edge e on the outer face of G is not a part of MG , it is “enclosed” by the outer face of MG .) For each face f of the mortar graph that encloses at least one edge, there is a corresponding brick.

Lemma 6.5.8 (Lemma 6.10, Borradaile *et al.* [25]). *The boundary ∂B of a brick B , in counter-clockwise order, is the concatenation of four paths W_B, S_B, E_B, N_B such that*

- (1) *The set of edges of $B - \partial B$ is non-empty.*
- (2) *Every vertex of T that is in B is on S_B or N_B .*
- (3) *N_B is 0-short in B , and every proper subpath of S_B is ε -short in B .*
- (4) *There exists a number $k' \leq k$ and vertices $s_0, s_1, \dots, s_{k'}$ ordered west to east on S_B such that, for each i and each vertex x on $S_B[s_i, s_{i+1})$, $\text{dist}_{S_B}(x, s_i) < \varepsilon \cdot \text{dist}_B(x, N_B)$.*

Definition 6.5.9 (Definition 10.3, Borradaile *et al.* [25]). *Let H be a subgraph of G such that P is a path in H . A joining vertex of H with P is a vertex of P that is the endpoint of an edge of $H - P$.*

Lemma 6.5.10 (Theorem 10.7, Borradaile *et al.* [25]). *Let B be a plane graph with boundary $W \cup S \cup E \cup N$, satisfying the brick properties of Lemma 6.5.8. Let F be a set of edges of B . There is a forest \tilde{F} of B with the following properties:*

- (1) *If two vertices of $N \cup S$ are connected in F then they are connected in \tilde{F} .*
- (2) *The number of joining vertices of F with both N and S is at most $\alpha(\varepsilon)$, where $\alpha(\varepsilon) = o(\varepsilon^{-5.5})$.*
- (3) *$\text{Length}(\tilde{F}) \leq (1 + c\varepsilon)\text{Length}(F)$, for some fixed constant c .*

Portals. Let $\theta = \theta(\varepsilon)$ be a parameter that depends polynomially on $1/\varepsilon$. For each brick B , we designate some vertices of ∂B as portals, evenly spaced around B as follows. Let $v_0 \in \partial B$ be the endpoint of an edge strictly enclosed by ∂B ; we designate v_0 as a portal. Now suppose we have designated v_0, v_1, \dots, v_{i-1} as portals. Let v_i be the first vertex on ∂B such that $\text{Length}(\partial B[v_{i-1}, v_i]) > \text{Length}(\partial B)/\theta$. We designate v_i as a portal, unless $v_0 \in V(\partial B(v_{i-1}, v_i])$, in which case we stop.

Lemma 6.5.11 (Lemma 7.1, Borradaile *et al.* [25]). *For any vertex x on ∂B , there is a portal y such that the x -to- y subpath of ∂B has length at most $\text{Length}(\partial B)/\theta$.*

Lemma 6.5.12 (Lemma 7.2, Borradaile *et al.* [25]). *There are at most θ portals on ∂B .*

Portal-connected graph. For any subgraph G'' of the mortar graph MG , we construct a planar graph $\mathcal{B}^+(G'')$ as follows. For each face f of G'' corresponding to a brick B , we embed a copy of B inside the face f , and, for each portal v of B , we connect the copy of v in the brick with the copy of v on f using a zero-length edge. We refer to these zero-length edges as *portal edges*, and we refer to $\mathcal{B}^+(MG)$ as the *portal-connected graph*. Finally, any new vertex receives penalty zero (these vertices are copies of vertices of the mortar graph).

Theorem 6.5.13. *Let F^* be an optimal Prize-Collecting Steiner forest in G . There exists a constant $\theta = \theta(\varepsilon)$ depending polynomially on $1/\varepsilon$ such that, for any choice of portals satisfying the condition in Lemma 6.5.11, the corresponding portal-connected graph $\mathcal{B}^+(MG)$ contains a forest \widehat{F} with the following properties:*

- (1) $\text{Length}(\widehat{F}) \leq (1 + c_1\varepsilon)\text{Length}(F^*) + c_2\varepsilon \cdot \text{Length}(\Delta)$, where c_1, c_2 are absolute constants
- (2) Any two vertices of MG in the same component of F^* are connected by \widehat{F} .

Proof. Let F^* be an optimal tree to the Prize-Collecting Steiner forest problem in G , and let S^* be the set of all vertices of the mortar graph MG that are in F^* . We will follow the proof of Theorem 3.2 in Borradaile *et al.* [25]; there are two main steps. First, we transform F^* into a solution in G that only has a few joining vertices in each brick. To convert this into the desired forest \widehat{F} in the portal-connected graph $\mathcal{B}^+(MG)$, we simply add edges connecting the joining vertices in each brick to the nearest portals. As there are not many joining vertices, we can connect them to the portals without significantly increasing the cost. We describe the process completely below.

First, we add the east and west boundaries of each brick; let F_1 be the union of F^* with the east and west boundaries (E_B and W_B) for each brick B . By Lemma 6.5.5, $\text{Length}(F_1) \leq \text{Length}(F^*) + \varepsilon \cdot \text{Length}(\Delta)$. Clearly, F_1 connects all vertices of S^* connected by F^* .

Next, we reduce the number of joining vertices on the north and south boundaries of each brick. Let $F_{1|B}$ be the subgraph of F_1 that is strictly embedded in a brick B of G . We replace $F_{1|B}$ with the forest $F_{2|B}$ that is guaranteed by Lemma 6.5.10. We have

$$\text{Length}(F_{2|B}) \leq (1 + c_1\varepsilon)\text{Length}(F_{1|B}).$$

Let N and S denote the north and south boundaries of the brick. Since any two vertices of $N \cup S$ that are connected in $F_{1|B}$ are also connected in $F_{2|B}$, it follows that $F_{2|B}$ connects all vertices of S^* connected by $F_{1|B}$.

We apply this procedure for each brick in order to get a subgraph F_2 . Since the bricks are disjoint,

$$\begin{aligned} \text{Length}(F_2) &\leq (1 + c_1\varepsilon)\text{Length}(F_1) \\ &\leq (1 + c_1\varepsilon)(\text{Length}(F^*) + \varepsilon \cdot \text{Length}(\Delta)) \\ &= (1 + c_1\varepsilon)\text{Length}(F^*) + (\varepsilon + c_1\varepsilon^2)\text{Length}(\Delta) \end{aligned}$$

Moreover, F_2 connects all vertices of S^* connected by F_1 .

Now we convert the forest $F_2 \subseteq G$ to a subgraph of $\mathcal{B}^+(MG)$. Note that every edge of G has at least one corresponding edge in $\mathcal{B}^+(MG)$ (an edge e of MG has three copies: one mortar edge, and one inside each of the bricks corresponding to the two faces of G' incident to e). For each edge e of F_2 , we select a corresponding edge of $\mathcal{B}^+(MG)$ as follows. If e is an edge of MG , we select the corresponding mortar edge of $\mathcal{B}^+(MG)$. Otherwise, we select the unique edge corresponding to e in $\mathcal{B}^+(MG)$. Let F_3 denote the resulting subgraph of $\mathcal{B}^+(MG)$. We have:

$$\text{Length}(F_3) = \text{Length}(F_2) \leq (1 + c_1\varepsilon)\text{Length}(F^*) + (\varepsilon + c_1\varepsilon^2)\text{Length}(\Delta).$$

Since F_3 does not connect the connected components of F_2 , we connect it using portal edges and mortar edges as follows. Consider a brick B , and let V_B denote the set of joining vertices of F_3 with $N_B \cup S_B$. For each vertex $v \in V_B$, let p_v be the portal vertex that is closest to v , let P_v be the shortest v -to- p_v path along ∂B , and let P'_v be the corresponding path of mortar edges. Let e_v be the portal edge corresponding to p_v . We add P_v , P'_v , and e_v to F_3 . We apply this procedure for

each brick in order to get the subgraph \widehat{F} . First, we bound the length of \widehat{F} .

$$\begin{aligned}
\text{Length}(\widehat{F}) &\leq \text{Length}(F_3) + \sum_B \sum_{v \in V_B} (\text{Length}(P_v) + \text{Length}(e_v) + \text{Length}(P'_v)) \\
&= \text{Length}(F_3) + 2 \sum_B \sum_{v \in V_B} \text{Length}(P_v) \\
&\quad (\text{Since } \text{Length}(e) = 0, \text{Length}(P'_v) = \text{Length}(P_v)) \\
&\leq \text{Length}(F_3) + 2 \sum_B \sum_{v \in V_B} \text{Length}(\partial B) / \theta(\varepsilon) \quad (\text{By Lemma 6.5.11}) \\
&\leq \text{Length}(F_3) + 2 \sum_B \alpha(\varepsilon) \text{Length}(\partial B) / \theta(\varepsilon) \quad (\text{By Lemma 6.5.10}) \\
&= \text{Length}(F_3) + \frac{2\alpha(\varepsilon)}{\theta(\varepsilon)} \cdot \sum_B \text{Length}(\delta(B)) \\
&\leq \text{Length}(F_3) + \frac{4\alpha(\varepsilon)}{\theta(\varepsilon)} \cdot \text{Length}(MG) \\
&\leq \text{Length}(F_3) + \frac{16\alpha(\varepsilon)}{\varepsilon\theta(\varepsilon)} \cdot \text{Length}(\Delta) \quad (\text{By Lemma 6.5.6})
\end{aligned}$$

Setting $\theta(\varepsilon) = 16\varepsilon^{-2}\alpha(\varepsilon)$ gives us

$$\begin{aligned}
\text{Length}(\widehat{F}) &\leq \text{Length}(F_3) + \varepsilon \cdot \text{Length}(\Delta) \\
&\leq (1 + c_1\varepsilon) \text{Length}(F^*) + (2 + c_1\varepsilon)\varepsilon \cdot \text{Length}(\Delta)
\end{aligned}$$

It remains only to show that \widehat{F} connects any two vertices of S^* connected by F_2 , and hence by F^* . Let x and y be two vertices of S^* connected by F_2 via an $x - y$ path P in F_2 . The definition of F_3 breaks P into disjoint paths. Consider one such path P_i that is not a subpath of MG . By construction, the endpoints of P_i are joining vertices. When we construct \widehat{F} from F_3 , we connect the endpoints of P_i to their corresponding vertices on MG via portal edges. Therefore there is an $x - y$ path in \widehat{F} . \square

Spanner. For each brick B and for each subset X of the portals of B , we find an optimal Steiner Tree for B and X . The *spanner* H consists of all edges of these Steiner Trees together with the edges of the mortar graph MG .

Lemma 6.5.14. *The total length of the spanner H is at most $(1 + 2^{1+\theta})(\frac{3}{\varepsilon} + \varepsilon)\text{Length}(\Delta)$.*

Proof. As shown in Lemma 4.1 of [25], the total length of all Steiner trees is at most $2^{1+\theta} \cdot \text{Length}(MG)$. Thus the length of H is at most $(1 + 2^{1+\theta}) \cdot \text{Length}(MG)$. Lemma 6.5.6 completes the proof. \square

Lemma 6.5.15. *The spanner H contains a prize-collecting Steiner forest F' of cost at most $(1 + c_1\varepsilon)\text{Length}(F^*) + c_2\varepsilon \cdot \text{Length}(\Delta) + \text{Penalty}(F^*)$, for some absolute constants c_1, c_2 .*

Proof. We will follow the proof of Lemma 4.2 in Borradaile *et al.* [25]. Let F^* be an optimal forest in G and let \hat{F} be the forest guaranteed by Theorem 6.5.13. For each brick B and for each connected component K of the intersection of \hat{F} with B , let X be the set of portals of B belonging to K ; we replace K with the optimal Steiner Tree for B and X contained in the spanner. Let \tilde{F} be the subgraph resulting from all these replacements. We have

$$\text{Length}(\tilde{F}) \leq \text{Length}(\hat{F}) \leq (1 + c_1\varepsilon)\text{Length}(F^*) + c_2\varepsilon \cdot \text{Length}(\Delta)$$

Moreover, since \hat{F} connects all vertices of MG connected by F^* , and all terminals are vertices of MG as they are connected by T , it follows that \tilde{F} also connects all terminals connected by F^* . Hence, $\text{Penalty}(\tilde{F}) \leq \text{Penalty}(F^*)$; as $\text{Cost}(\tilde{F}) = \text{Length}(\tilde{F}) + \text{Penalty}(\tilde{F})$, we obtain the lemma. \square

Theorem 6.3.4 now follows almost directly from Lemmas 6.5.14 and 6.5.15, as $\text{Length}(\Delta) = 2 \cdot \text{Length}(T)$; simply construct the spanner H using a modified parameter $\varepsilon' = \frac{\varepsilon}{\max\{c_1, 2c_2\}}$.

6.6 The Prize-collecting Traveling Salesperson problem

To obtain a **PTAS** for the Prize-collecting Traveling Salesperson problem, we need first to construct a spanner, and then to solve the problem in graphs of fixed treewidth. Here, we focus on the former step; the latter is relatively straightforward via dynamic programming. (See Section 6.7 for a similar dynamic program for the Prize-collecting Steiner Tree problem.)

Given an instance I of the Prize-collecting Traveling Salesperson problem, we fix an optimal solution, and use OPT to denote its cost, L to denote its length, and P its penalty; $\text{OPT} = L + P$. If we could find a tree T of length $O(f(\varepsilon)) \cdot \text{OPT}$ such that the total penalty of vertices in the optimal solution but not in T is at most εOPT , then we could construct a spanner using ideas from

Section 6.3 and Section 6.5. (A similar approach was previously used by Klein for the subset-TSP problem [125].) We now describe how to find such a tree T .

As in Section 6.3.1, we scale penalties suitably. Consider the prize-collecting *Steiner tree* instance on the same graph with the same penalties; clearly, an optimal solution has cost at most OPT . If π_i is the penalty for vertex v_i , let I' denote the prize-collecting Steiner tree instance in which the penalty for vertex v_i is $\pi'_i = 2\pi_i/\varepsilon$. When we run the primal-dual algorithm for prize-collecting Steiner tree on instance I' , we obtain a tree T of total cost at most $4\text{OPT}/\varepsilon$, as the original instance had an optimal solution of cost at most OPT . Thus, it remains only to argue that the total penalty of vertices in the optimal solution to the original instance of the prize-collecting Traveling Salesperson Problem, but not in T , is at most εOPT . Let X denote the set of such vertices.

We note that there exists a tree of length at most $L \leq \text{OPT}$ containing all the vertices of X (as they are visited by the optimal Traveling Salesperson tour). However, suppose for contradiction that the total penalty of the vertices in X is greater than εOPT , and hence the total scaled penalty of vertices in X is greater than 2OPT . Exactly as in the proof of Theorem 6.3.1, we could then obtain a feasible solution of cost greater than OPT to the dual of an LP for the instance of Steiner tree defined on the vertices of X ; this yields a contradiction.

6.7 Prize-collecting Steiner Tree in graphs of fixed treewidth

In this section, for any fixed integer $k \geq 2$, we give a polynomial-time algorithm to optimally solve the prize-collecting Steiner tree problem in graphs of treewidth at most $k - 1$.

A tree decomposition of a graph G is a pair $(\mathcal{T}, \mathcal{B})$, where $\mathcal{T} = (I, F)$ is a tree, and $\mathcal{B} = \{B_i \mid i \in I\}$ is a family of subsets of $V(G)$ such that

- (1) $\bigcup_{i \in I} B_i = V(G)$
- (2) for every edge $uv \in E(G)$, there exists an i such that $\{u, v\} \subseteq B_i$
- (3) for every vertex $v \in V(G)$, the set of nodes $\{i \in I \mid v \in B_i\}$ forms a connected subtree of \mathcal{T}

We refer to vertices of \mathcal{T} as nodes, and to each set of \mathcal{B} as a bag. The width of a tree decomposition $(\mathcal{T}, \mathcal{B})$ is the size of the largest bag B_i minus one. As shown in [22], for any fixed k , there is a

polynomial time algorithm (in fact a linear time algorithm) that constructs a tree decomposition of G of width at most k , or reports that G has treewidth greater than k . In the following, we assume that we have a tree decomposition for G of width at most $k - 1$, for some fixed k .

A tree decomposition $(\mathcal{T}, \mathcal{B})$ is *nice* if the tree \mathcal{T} is rooted and, for every node $i \in I$, either

- (1) i has no children (i is a leaf node)
- (2) i has exactly two children i_1, i_2 and $B_{i_1} = B_{i_2} = B_i$ (i is a join node)
- (3) i has a single child j and $B_i = B_j \cup \{v\}$ for some vertex $v \in V(G)$ (i is an introduce node)
- (4) i has a single child j and $B_i = B_j - \{v\}$ for some vertex $v \in V(G)$ (i is a forget node)

The following lemma is well-known and it is straightforward to prove.

Lemma 6.7.1. *There is a linear time algorithm that, given a tree decomposition for G , constructs a nice tree decomposition $(\mathcal{T}, \mathcal{B})$ of the same width. Moreover, the tree \mathcal{T} has $O(|V|)$ nodes.*

6.7.1 A dynamic program for Prize-collecting Steiner Tree

We solve the problem using dynamic programming on a nice tree decomposition $(\mathcal{T}, \mathcal{B})$ of width $k - 1$. For each node $i \in I$, let V_i be the set of all vertices appearing in the bags corresponding to the nodes of the subtree of \mathcal{T} rooted at i . Let G_i be the subgraph of G induced by V_i . We start with an informal overview of the algorithm. For the purposes of exposition, we assume that there is only one optimal prize-collecting Steiner tree T^* (if there are several solutions, we fix one of them). Additionally, we assume without loss of generality that the root vertex r is in the bag corresponding to the root node of \mathcal{T} . Now fix a node i of \mathcal{T} , and consider the graph G_i . Clearly, we would like to compute the subgraph F of T^* that lies in G_i . In order to do so, we will specify some information about this subgraph F . More precisely, we will specify the subgraph H of T^* that lies in $G[B_i]$, and a partition α of the vertices in H induced by the connected components of F , i.e., each part of α consists of all the vertices in H that are in the same connected component of F . (Intuitively, α tells us that we need to connect each part using a tree of G_i and all of these trees are guaranteed to be connected to the root outside G_i .) It follows from the optimality of T^* that F is a minimum cost subgraph of G_i satisfying

(c_1) $F[B_i] = H$

(c_2) the partition of $V(H)$ induced by the connected components of H is a refinement of α (if two vertices u, v are in the same connected component of H , then u and v are in the same part of α)

(c_3) the partition of $V(H)$ induced by the connected components of F is α

Let $c(i, H, \alpha)$ be the minimum cost of a subgraph F of G_i satisfying (c_1) – (c_3). We will compute $c(i, H, \alpha)$ for all valid tuples (i, H, α) using dynamic programming. The cost of the optimal prize-collecting Steiner tree is equal to $\min_{H, \alpha} c(r', H, \alpha)$, where r' is the root node of \mathcal{T} , and the minimum is over all pairs (H, α) such that H is a subgraph of $G[B_{r'}]$ containing r , and α has a single part containing the vertices of H . To see why this is true, consider a pair (H, α) such that H is a subgraph of $G[B_{r'}]$ containing r , and α consists of a single part containing the vertices of H . If T is a solution for the subproblem (r', H, α) then T is a tree that contains r , and hence T is a valid prize-collecting Steiner tree. Conversely, let T be a prize-collecting Steiner tree. Let $H = T[B_{r'}]$ and let α be the partition of $V(H)$ induced by T . Since T is a tree containing r , H contains r and α consists of a single part containing the vertices of H . Therefore T is a valid solution for the subproblem (r', H, α) .

Let i be a node of \mathcal{T} . Then i is a leaf node, a join node, an introduce node, or a forget node, and we consider each of these cases separately. Before we describe the recurrence for $c(i, H, \alpha)$, we introduce some useful terminology (borrowed from [17]).

We can view a partition α as an equivalence relation over the vertices, and we write $u \equiv_\alpha v$ if u and v are in the same part of α . Let α_1 and α_2 be two partitions of the same vertex set. We say that α_1 is *finer* than α_2 — or equivalently, that α_1 is a refinement of α_2 — if $u \equiv_{\alpha_1} v$ implies $u \equiv_{\alpha_2} v$. If α_1 is finer than α_2 , we say that α_2 is *coarser* than α_1 . We use $\alpha_1 \vee \alpha_2$ to denote the finest partition that is coarser than both α_1 and α_2 (there is a unique such partition).

Node i is a leaf node. Let β be the partition of $V(H)$ induced by the connected components of H . We have

$$c(i, H, \alpha) = \begin{cases} \text{Length}(H) + \text{Penalty}(B_i - V(H)) & \text{if } \alpha = \beta \\ \infty & \text{otherwise} \end{cases} \quad (6.1)$$

Proof of Equation 6.1: Since $G_i = G[B_i]$, H is the only subgraph satisfying (c_1) . If $\alpha \neq \beta$, there is no subgraph satisfying $(c_1) - (c_3)$. Otherwise, H is the only subgraph satisfying $(c_1) - (c_3)$ and its cost is $\text{Length}(H) + \text{Penalty}(H)$. \square

Node i is a join node. Let i_1 and i_2 be the children of i . We have

$$c(i, H, \alpha) = \min_{\alpha_1, \alpha_2} (c(i_1, H, \alpha_1) + c(i_2, H, \alpha_2) - \text{Length}(H)) \quad (6.2)$$

where the minimum is taken over all partitions α_1, α_2 of $V(H)$ such that $\alpha = \alpha_1 \vee \alpha_2$.

The intuition behind Equation 6.2 is the following. Let F, F_1, F_2 denote the restrictions of the optimal tree T^* to G_i, G_{i_1}, G_{i_2} (respectively). Then F is the union of F_1 and F_2 . Let $\alpha, \alpha_1, \alpha_2$ be the partitions of $B_i \cap V(T^*)$ induced by the connected components of F, F_1, F_2 . Since F_1 and F_2 intersect only at $B_i \cap V(T^*)$, $\alpha = \alpha_1 \vee \alpha_2$.

Proof of Equation 6.2: Let (α_1, α_2) be a pair that minimizes the right hand side of Equation 6.2.

Let F_ℓ be an optimal solution for the subproblem (i_ℓ, H, α_ℓ) , where $\ell = 1, 2$. Let $F = F_1 \cup F_2$. Now we claim that F is a solution for the subproblem (i, H, α) , i.e., it satisfies the conditions $(c_1) - (c_3)$. Clearly, F satisfies (c_1) and (c_2) . Now let u and v be two vertices in the same part of α . Since $\alpha = \alpha_1 \vee \alpha_2$, u and v are in the same part of α_ℓ for some ℓ , and thus u and v are in the same connected component of F_ℓ . Thus the partition of $V(H)$ induced by F is coarser than α . Since α is coarser than α_1 and α_2 , it follows that α is coarser than the partition of $V(H)$ induced by F as well. Therefore F satisfies (c_3) . Since $E(H) \subseteq E(F_1) \cap E(F_2)$,

$$\text{Length}(F) \leq \text{Length}(F_1) + \text{Length}(F_2) - \text{Length}(H).$$

Since $V(F) = V(F_1) \cup V(F_2)$ and $V(G_i) = V(G_{i_1}) \cup V(G_{i_2})$, we have

$$\text{Penalty}(V(G_i) - V(F)) \leq \text{Penalty}(V(G_{i_1}) - V(F_1)) + \text{Penalty}(V(G_{i_2}) - V(F_2)).$$

Thus

$$c(i, H, \alpha) \leq \text{Cost}(F) \leq \text{Cost}(F_1) + \text{Cost}(F_2) - \text{Length}(H) = c(i_1, H, \alpha_1) + c(i_2, H, \alpha_2) - \text{Length}(H).$$

Conversely, let F be an optimal solution for the subproblem (i, H, α) . Let F_ℓ be the restriction of F to G_{i_ℓ} , where $\ell = 1, 2$. Let α_ℓ be the partition induced by the connected components of F_ℓ .

Now we claim that F_ℓ is a solution for the subproblem (i_ℓ, H, α_ℓ) : by construction, F_ℓ satisfies $(c_1) - (c_3)$. Since F_1 and F_2 intersect only at $V(H)$, $\alpha = \alpha_1 \vee \alpha_2$. Therefore the right hand side of the equation is at most

$$\text{Cost}(F_1) + \text{Cost}(F_2) - \text{Length}(H) \leq \text{Cost}(F) = c(i, H, \alpha),$$

which completes the proof. \square

Node i is a forget node. Let j be the child of i , and let v be the vertex in $B_j - B_i$. Fix a subgraph H of $G[B_i]$, and a partition α of $V(H)$. Let S be a subset of the neighbors of v that are in B_i . Let $E(v, S)$ denote the edges with an endpoint in v and the other in S . Let $\alpha(v, S)$ be the partition of $V(H) \cup \{v\}$ obtained from α as follows: we merge each part of α that contains a vertex in S into a single part and add v to it; we add all remaining parts of α to $\alpha(v, S)$. We have

$$c(i, H, \alpha) = \min \left(c(j, H, \alpha), \min_{S \subseteq V(H) \cap \Gamma(v)} (c(j, H \cup \{v\} \cup E(v, S), \alpha(v, S))) \right) \quad (6.3)$$

where the second minimum is taken over all sets $S \subseteq V(H) \cap \Gamma(v)$ such that S has at most one vertex in each part of α .

The intuition behind Equation 6.3 is the following. Let F, F' denote the restriction of the optimal tree T^* to G_i, G_j (respectively). If T^* does not contain v , we have $F' = F$. Therefore we may assume that T^* contains v , and thus F' consists of F and the edges of $E(T^*) \cap E(G_j)$ that are incident to v . The edges of F' that are incident to v have at most one endpoint in each connected component of F . Thus each connected component of F' that does not contain v is a connected component of F , and the connected component of F' containing v consists of one or more connected components of F that connect to each other via the edges incident to v .

Proof of Equation 6.3: Suppose the minimum of the right hand side of the equation is achieved by an optimal solution F' for the subproblem (j, H, α) . Since v is not in H , F' does not contain v . Thus F' is a solution for the subproblem (i, H, α) , and therefore $c(i, H, \alpha)$ is at most the right hand side of Equation 6.3. Therefore we may assume that the minimum of the right hand side is achieved by an optimal solution F'_S for the subproblem $(j, H \cup \{v\} \cup E(v, S), \alpha(v, S))$. Let $F = F'_S - E(v, S) - \{v\}$. Now we claim that F is a solution for the subproblem (i, H, α) . By construction, F satisfies (c_1) and (c_2) . Therefore it suffices to show that F satisfies (c_3) .

Note that we may assume without loss of generality that F'_S is a forest. Now suppose that F'_S has an edge e whose endpoints are in different parts of α . Since e is not incident to v , it follows that e is in H . But the partition of $V(H)$ induced by the connected components of H is a refinement of α , which is a contradiction.

Let u and w be two vertices in the same connected component of F . It follows that the unique path of F'_S between u and w does not pass through v , and hence u and w are in the same part of α (since otherwise the path between u and w has an edge with both endpoints in different parts of α). Therefore the partition of $V(H)$ induced by the connected components of F is a refinement of α . Conversely, let u and w be two vertices contained in the same part of α . Since $\alpha(v, S)$ is coarser than α , u and w are in the same connected component of F'_S . Let P be the unique path in F'_S between u and w . If $P - v$ is a path, u and w are connected in F . Therefore we may assume that v is an internal vertex of P . Let u', w' be the two neighbors of v on P , where u' is on the subpath of P from u to v . Since there is a path between u and u' in F (namely, the subpath of P from u to u'), it follows from the previous argument that u and u' are in the same part of α . Similarly, w and w' are in the same part of α . Therefore S has two vertices in the same part of α , which is a contradiction. Thus α is a refinement of the partition of $V(H)$ induced by the connected components of F . It follows that F satisfies (c_3) as well, and hence $c(i, H, \alpha)$ is at most the right hand side of the equation.

Conversely, let F be an optimal solution for the subproblem (i, H, α) . Since F is also a solution for the subproblem (j, H, α) , it follows that $c(i, H, \alpha)$ is at least the right hand side of the equation.

□

Node i is an introduce node. Let j be the child of i , and let v be the vertex in $B_i - B_j$. Let S be the set of all neighbors u of v such that the edge uv is in H . For each partition α' of $V(H) - v$, we let $\alpha'(v, S)$ be the partition of $V(H)$ obtained from α' as follows: we merge each part of α' that contains a vertex in S into a single part and add v to it; we add all remaining parts of α' to $\alpha'(v, S)$. We have

$$c(i, H, \alpha) = \begin{cases} c(j, H, \alpha) + \text{Penalty}(v) & \text{if } v \notin V(H) \\ \min_{\alpha'} (c(j, H - v, \alpha') + \sum_{uv \in H} \text{Length}(uv)) & \text{otherwise} \end{cases} \quad (6.4)$$

where the minimum is taken over all partitions α' of $V(H) - \{v\}$ satisfying

(i) S has at most one vertex in each part of α'

(ii) $\alpha'(v, S) = \alpha$

(Note that there exists a partition α' that satisfies the conditions above.)

The intuition behind Equation 6.4 is the following. Let F, F' denote the restriction of the optimal tree T^* to G_i, G_j (respectively). If T^* does not contain v , we have $F = F'$. Therefore we may assume that T^* contains v , and thus F consists of F' together with the edges of $E(T^*) \cap E(G[B_i])$ that are incident to v . The edges of F that are incident to v have at most one endpoint in each connected component of F' . Thus each connected component of F that does not contain v is a connected component of F' , and the connected component of F containing v consists of one or more connected components of F' that connect to each other via the edges incident to v .

Proof of Equation 6.4: Suppose that v is not in H . Let F be an optimal solution for the subproblem (i, H, α) . Since v is not in H , F is a solution for the subproblem (j, H, α) as well, of cost

$$\begin{aligned} \text{Length}(F) + \text{Penalty}(V(G_j) - V(F)) &= \text{Length}(F) + \text{Penalty}(V(G_i) - V(F)) - \text{Penalty}(v) \\ &= c(i, H, \alpha) - \text{Penalty}(v) \end{aligned}$$

Thus

$$c(i, H, \alpha) \geq c(j, H, \alpha) + \text{Penalty}(v).$$

Conversely, let F be an optimal solution for the subproblem (j, H, α) . Then F is a solution for (i, H, α) of cost

$$\begin{aligned} \text{Length}(F) + \text{Penalty}(V(G_i) - V(F)) &= \text{Length}(F) + \text{Penalty}(V(G_j) - V(F)) + \text{Penalty}(v) \\ &= c(j, H, \alpha) + \text{Penalty}(v) \end{aligned}$$

Thus

$$c(i, H, \alpha) \leq c(j, H, \alpha) + \text{Penalty}(v).$$

Therefore we may assume that v is in H . Let α' be a partition of $V(H) - \{v\}$ satisfying the conditions above, and let F' be an optimal solution for the subproblem $(j, H - v, \alpha')$. Let $F = F' \cup E(v, S) \cup \{v\}$, where $E(v, S)$ is the set of all edges of H that are incident to v . Now we claim that F is a solution for the subproblem (i, H, α) . By construction, F satisfies (c_1) and (c_2) . Therefore it suffices to verify that F satisfies (c_3) .

Let u and w be two vertices in the same connected component of F . Suppose that u and w are connected in F' . Then u and w are in the same part of α' and, since α is coarser than α' , u and w are in the same part of α . Therefore we may assume that u and w are not connected in F' . Thus u and w are in different parts of α' , each of which contains a vertex in S . It follows that the two parts have merged into a single part of $\alpha'(v, S) = \alpha$, and hence u and w are in the same part of α . Conversely, let u and w be two vertices in the same part of α . If u and w are in the same part of α' , it follows that u and w are connected in F' . Therefore we may assume that u and w are in different parts P_1 and P_2 of α' , each of which contains a vertex of S . Let u' and w' be the two vertices of $P_1 \cap S$, $P_2 \cap S$. Then there exists a path in F' from u to u' , and a path from w to w' . These two paths together with the edges $u'v$, vw' form a connected subgraph of F . It follows that u and w are connected in F , and hence F satisfies (c_3) .

We have

$$\text{Length}(F) = \text{Length}(F') + \sum_{uv \in G} \text{Length}(uv).$$

Since $V(F) = V(F') \cup \{v\}$ and $V(G_i) = V(G_j) \cup \{v\}$,

$$\text{Penalty}(V(G_i) - V(F)) = \text{Penalty}(V(G_j) - V(F')).$$

Thus

$$c(i, H, \alpha) \leq c(j, H - v, \alpha') + \sum_{uv \in H} \text{Length}(uv).$$

Conversely, let F be an optimal solution for the subproblem (i, H, α) . Without loss of generality, F is a forest. Let $F' = F - v$, and let α' be the partition of $V(H) - \{v\}$ induced by F' . Since F is a forest, v has at most one neighbor in each part of α' . Now we claim that $\alpha'(v, S) = \alpha$. Let T be any connected component of F that does not contain v . It follows that T is a connected component of F' as well. Since the partition of $V(H)$ induced by the connected components of F

is equal to α , α' contains each part of α that does not intersect S . Now consider the connected component T of F that contains v , and let T_1, \dots, T_ℓ be the connected components of $T - v$. Since each T_j is a connected component of α' , it follows that the part of $\alpha'(v, S)$ containing v can be obtained by merging the parts of α' induced by T_1, \dots, T_ℓ into a single part, and adding v to it. Thus $\alpha'(v, S) = \alpha$.

Now we claim that F' is a solution for the subproblem $(j, H - v, \alpha')$. By construction, F' satisfies (c_1) and (c_2) . Additionally, it follows from the definition of α' that F' satisfies (c_3) . We have

$$\text{Length}(F) = \text{Length}(F') + \sum_{uv \in H} \text{Length}(uv).$$

As before,

$$\text{Penalty}(V(G_i) - V(F)) = \text{Penalty}(V(G_j) - V(F')).$$

Thus

$$c(i, H, \alpha) \geq c(j, H - v, \alpha') + \sum_{uv \in H} \text{Length}(uv).$$

□

Proof of Theorem 6.1.2: Let b_k be the number of partitions of a k -element set⁴, and let s_k be the number of subgraphs of a graph with k vertices. Since each bag has at most k vertices and \mathcal{T} has $O(|V|)$ nodes, there are $O(|V| \cdot b_k \cdot s_k)$ distinct subproblems. Additionally, we can evaluate each subproblem in $O(b_k^2)$ time once we have a solution for each of the subproblems it depends on. (The most expensive evaluation corresponds to a join node.) Therefore we can find an optimal prize-collecting Steiner tree in $O(b_k^3 \cdot s_k \cdot |V|)$ time. □

6.8 Concluding remarks

In this chapter, we consider several prize-collecting network design problems in planar graphs. Our main results are a **PTAS** for prize-collecting Steiner Tree and Traveling Salesperson problems. We also describe a reduction for the prize-collecting Steiner Forest problem in planar graphs to the

⁴The Bell number B_k is the number of partitions of a k -element set. To avoid confusion with the bags of the tree decomposition, we will use b_k to refer to the k -th Bell number.

problem in graphs of fixed treewidth; our reduction, coupled with a ρ approximation for the prize-collecting Steiner Forest problem in graphs of fixed treewidth, it gives an $(1 + \epsilon)\rho$ approximation for the problem in planar graphs.

The results for the prize-collecting Steiner Tree and Traveling Salesperson problems also extend to graphs of bounded genus, following previous ideas from [23, 67, 125]. These extensions are not included in this thesis.

The main problems left open by our work is whether there is a **PTAS** for related problems in planar graphs where the set of vertices to be connected or visited is not fixed. Some examples of such problems are k -MST, k -Stroll, and Orienteering. In order to illustrate some of the difficulties involved, we focus on the k -MST and k -Stroll problems. In the k -MST problem, the goal is to construct a minimum-cost tree containing at least k vertices, and in the k -Stroll problem, the goal is to find a shortest walk visiting at least k vertices.

The main difficulty appears to be in constructing an appropriate *spanner*, a subgraph of cost $O(f(\epsilon) \cdot \text{OPT})$ that contains a $(1 + \epsilon)$ -approximate optimal solution. For the prize-collecting problems, the spanner we construct is a subgraph H of good cost with the additional property that for *any* optimal solution, the total penalty of vertices in the solution but not in H is at most ϵOPT . Informally, H is a subgraph that contains “nearly all” the vertices of any optimal solution. However, for problems such as k -Stroll and k -MST, such a subgraph H does not always exist; there may be very many optimal and near-optimal solutions that are completely disjoint. Thus, one cannot find a subgraph H of good cost containing nearly all the vertices of *any* optimal solution; a different approach will be needed to argue that there exists *one* near-optimal solution contained in H .

Another open problem is whether we can achieve an improved approximation for the prize-collecting Steiner Forest problem in fixed treewidth graphs. Currently, the 2.54 approximation for general graphs is also the best approximation known for fixed treewidth graphs. The recent work of Bateni *et al.* [17] shows that it is **APX**-hard (even in graphs of treewidth 2), but it may be possible to achieve an approximation ratio that is better than 2.54. In particular, a dynamic programming approach might yield a 2-approximation. Theorem 6.1.1 implies that this would give the same ratio (up to a factor of $(1 + \epsilon)$) in planar graphs.

Chapter 7

Node-weighted Network Design in Planar and Minor-free Graphs

7.1 Introduction

In this chapter¹ we consider a class of problems that can be modeled as follows. Given an undirected graph $G = (V, E)$ find a subgraph H of *minimum weight/cost* such that H satisfies certain desired *connectivity* properties. A common cost model is to assign a non-negative weight $w(e)$ to each $e \in E$ and the weight/cost of H is simply the total weight of edges in it. A number of well-studied problems can be cast as special cases. Examples include polynomial-time solvable problems such as the Minimum Spanning Tree (MST) problem when H is required to connect all the nodes of G , and the NP-hard Steiner Tree problem where H is required to connect only a given subset $S \subseteq V$ of terminals. A substantial generalization of these problems is the Survivable Network Design (SNDP) problem which is defined as follows. In addition to the graph G , we are also given as input an integer requirement function $r(uv)$ for each (unordered) pair of nodes uv in G ; the goal is to find a minimum-weight subgraph H that contains $r(uv)$ edge-disjoint paths between u and v for each pair uv . This problem is called the edge-connectivity SNDP (EC-SNDP) to distinguish from more general problems such as Elem-SNDP and VC-SNDP that require the paths to be element and vertex disjoint, respectively. SNDP arises naturally in the design of fault-tolerant networks, and various special cases have been extensively studied. Algorithmic approaches for SNDP and related problems are based on solving a larger class of abstract network design problems such as covering proper and skew-supermodular cut-requirement functions that we describe formally later.

Node weights: The cost of a network is dependent on the application. In connectivity problems, as we remarked, a common model is the edge-weight model. A more general problem is obtained

¹This chapter is based on joint work with Chandra Chekuri and Ali Vakilian and it has appeared in [37]. Copyrights to the conference version of [37] are held by Springer.

when each node v of G has a weight $w(v)$ and the weight of H is the total weight of the nodes in H ². Node weights are relevant in several applications, in particular telecommunication networks, where they can model the cost of setting up routing and switching infrastructure at a given node. There have also been several recent applications in wireless network design [137, 140] where the weight function is closely related to that of node weights. We refer the reader to [65] for some additional applications of node weights to network formation games.

The node-weighted versions of network design problems often turn out to be strictly harder to approximate than their corresponding edge-weighted versions. For instance, the edge-weighted **Steiner Tree** problem admits a 1.39-approximation [28]; in contrast, Klein and Ravi [124] showed, via a simple reduction from the **Set Cover** problem, that the node-weighted **Steiner Tree** problem is hard to approximate to within an $\Omega(\log n)$ -factor unless $\mathbf{P} = \mathbf{NP}$, where n is the number of nodes in G . They also described a $(2 \log h)$ -approximation where h is the number of terminals. A more dramatic difference emerges if we consider **SNDP**. Jain gave a 2-approximation for the edge-weighted **EC-SNDP** problem [106]. The best known approximation for the node-weighted **EC-SNDP** problem is $O(k \log n)$ [136], where $k = \max_{uv} r(uv)$ is the maximum connectivity requirement. Nutov [136] gives evidence, via a reduction from the k -**Densest Subgraph** problem, that for the node-weighted problem a dependence on k in the approximation ratio is necessary.

Demaine, Hajiaghayi, and Klein [65] considered the approximability of the node-weighted **Steiner Tree** problem in planar graphs. In an interesting result, they adapted the well-known primal-dual algorithm for the edge-weighted problem [2, 88] to the node-weighted problem and showed that it gives a 6-approximation in planar graphs. Demaine *et al.* also showed that their algorithm works for a more general class of $\{0, 1\}$ -valued proper functions (first considered by Goemans and Williamson [88]) that includes several other problems such as the **Steiner Forest** problem; Their analysis also shows that one obtains a constant factor approximation (the algorithm is the same) for any minor-closed family of graphs where the constant depends on the family. In addition to their theoretical value, these results have the potential to be useful in practice since in many real-world networks the underlying graph G is either planar or has very few crossings.

²For many problems of interest, including **Steiner Tree** and **SNDP**, the version with weights on both edges and nodes can be reduced to the version with only node weights; sub-divide an edge e by placing a new node v_e and set the weight of v_e to be that of e .

Our results: In this chapter we consider node-weighted network design problems in planar graphs with higher connectivity requirements. In particular, we consider EC-SNDP and show that the insights in [65] can be used to develop improved approximation algorithms for this more general problem as well. However, the results require non-trivial technical work that we explain after we state the results. The algorithm works for any graph but the ratio is constant for planar graphs and more generally graphs from any minor-closed family; we articulate the precise dependence of the ratio on the family in later sections. Our main result is summarized in the following theorem.

Theorem 7.1.1. *There is an $O(k)$ -approximation for node-weighted EC-SNDP in planar graphs where k is the maximum requirement.*

The above theorem extends to a more general problem that we describe now. An integer valued set function $f : 2^V \rightarrow \mathbb{Z}_+$ on the vertex set of G is said to be proper if it satisfies the following conditions: (i) f is symmetric, that is, $f(S) = f(V - S)$ for all S , and (ii) f is maximal, that is, $f(A \cup B) \leq \max\{f(A), f(B)\}$ for any two disjoint sets A, B . Given a proper function f on V (by a value oracle) and a graph G on V , the f -covering problem is to find a subgraph H of minimum weight such that $|\delta_H(S)| \geq f(S)$ for all S ³. EC-SNDP is a special case of this problem [168]. We obtain an $O(k)$ -approximation for the node-weighted version of this problem in planar graphs where $k = \max_S f(S)$.

Overview of technical ideas and contribution: The two main algorithmic approaches for SNDP are the following. The first is the augmentation approach pioneered by Williamson *et al.* [168] in which the required network is built in k phases. At the end of the first $(i - 1)$ phases the connectivity of a pair uv is at least $\min\{r(uv), i - 1\}$. Thus the i 'th phase is required to increase the connectivity of some of the pairs by 1 by adding additional edges; the advantage of this approach is that we now work with a 0-1 covering problem. On the other hand the covering problem is no longer so simple. The function that we need to cover falls into the more general class of *uncrossable* functions: A requirement function $f : 2^V \rightarrow \{0, 1\}$ is uncrossable if for any sets $A, B \subseteq V$, $f(A) = f(B) = 1$ implies $f(A \cap B) = f(A \cup B) = 1$ or $f(A - B) = f(B - A) = 1$. Williamson *et al.* [168] showed that a primal-dual algorithm achieves a 2-approximation for the edge-weighted

³We work with node-induced subgraphs H of G in which case H may not contain all the nodes of a set $S \subset V$. In that case $\delta_H(S)$ denotes the edges of H with exactly one endpoint in S .

version of covering uncrossable functions. Nutov [136] gave an $O(\log n)$ -approximation for the node-weighted case. These results for uncrossable functions, when combined with the augmentation framework, give a $2k$ and an $O(k \log n)$ approximation for the edge-weighted and node-weighted versions of EC-SNDP in general graphs⁴. The second approach for SNDP is the powerful iterated rounding technique pioneered by Jain which led to a 2-approximation for EC-SNDP [106] and also for covering a certain class of skew-supermodular functions⁵. Iterated rounding does not quite apply to node-weighted problems for various technical reasons.

We follow the augmentation approach. Demaine *et al.* adapted the primal-dual algorithm for edge-weighted 0-1-proper functions to the node-weighted case. The novel technical ingredient in their analysis is to understand properties of *node-minimal* feasible solutions instead of edge-minimal feasible solutions. For the most part, problems captured by 0-1-proper functions are very similar to the Steiner Forest problem, a canonical problem in this class. In this setting it is possible to visualize and understand node-minimal solutions through connected components and basic reachability properties. In the augmentation approach for higher-connectivity, as we remarked, the problem in each phase is no longer that of covering a proper function but belongs to the richer class of covering uncrossable functions. The primal-dual analysis for this class of functions is more subtle and abstract [168] and proceeds via uncrossing arguments and laminar witness families.

Our main technical contribution is understanding properties of node-minimal feasible solutions for uncrossable functions. We refer the reader to Theorem 7.3.1 in Section 7.3 for the precise statement; the theorem holds for general graphs (not just planar graphs) and may have other applications. We remark on a crucial aspect of our algorithm and analysis. Why do our results only apply for covering proper functions and not the more general class of skew-supermodular functions? For the node-weighted problem of covering an arbitrary uncrossable function there is no natural covering LP relaxation. However, we observe that the particular uncrossable functions that arise in the augmentation framework for a proper function (including EC-SNDP) have certain additional connectivity properties that allow for an LP relaxation and the primal-dual approach.

⁴The approximation for the edge-weighted version can be improved to $2H_k$ by doing the augmentation in the reverse [89].

⁵A function $f : 2^V \rightarrow \mathbb{Z}$ is skew-supermodular if for all $A, B \subseteq V$, $f(A) + f(B) \leq \max\{f(A \cap B) + f(A \cup B), f(A - B) + f(B - A)\}$. A skew-supermodular function f with $f(A) \leq 1$ for all A gives rise to an uncrossable function.

We obtain a constant factor approximation in each phase and this results in an $O(k)$ -approximation overall where k is the maximum requirement.

As in [65] we use planarity only in one step of the analysis where we argue about the average degree of a certain graph that is a minor of the original graph; this is the reason that the algorithm and analysis generalize to any minor-closed family.

Other related work: We refer the reader to [92] for a survey on primal-dual based algorithms for network design, and to recent surveys [94, 129] for an overview of the known approximation results and references to related work.

Chapter outline: The rest of this chapter is organized as follows. Section 7.2 describes our algorithm based on the augmentation approach and the primal-dual algorithm for each phase of the augmentation. The analysis is done by assuming the main technical theorem on a node-minimal augmentation of the uncrossable requirement functions that arise in the augmentation framework. We state and prove this theorem in Section 7.3.

7.2 Algorithm for node-weighted EC-SNDP and proper functions

We start by defining the node-weighted EC-SNDP problem formally. The input consists of an undirected node-weighted graph $G = (V, E)$ (weight of node v is denoted by $w(v)$) and a requirement function $r(uv)$ for each pair of nodes. The goal is to find a minimum node-weighted subgraph H of G such that H contains $r(uv)$ edge-disjoint paths for each pair uv . We use k to denote the maximum requirement. A node u is called a *terminal* if there is some node v such that $r(uv) > 0$. Since any feasible solution has to contain all terminals, we can assume without loss of generality that the weight of every terminal is zero. We define a function $f : 2^V \rightarrow \mathbb{Z}_+$ where $f(S) = \max\{r(uv) \mid u \in S, v \notin S\}$. It is well-known that f is a proper function. By Menger's theorem, solving node-weighted EC-SNDP is equivalent to finding a minimum node-weight subgraph H such that $|\delta_H(S)| \geq f(S)$ for all $S \subset V$. (Recall that $\delta_H(S)$ is the set of all edges of H with exactly one endpoint in S .) Our algorithm and analysis extend to the problem of finding a node-weighted subgraph to cover a given proper function. For an arbitrary proper function f we call a node v a terminal if $f(\{v\}) > 0$; maximality of f implies that S contains a terminal if $f(S) > 0$.

Again, we can assume without loss of generality that terminals have zero weight, since they are included in any feasible solution.

We alert the reader that, in order to cover the function f , we need to pick a set of *edges*. But since the weights are (only) on the nodes, we pay for a set of nodes and we can use any of the edges in the graph induced by the nodes in order to cover the function. More precisely, our goal is to select a minimum-weight node-induced subgraph $H = G[X]$ that covers f , where X is a subset of nodes of G . We will always assume that X contains the terminals.

As we mentioned, our algorithm for covering f uses the augmentation framework introduced in [168]. Let $f_p : 2^V \rightarrow \mathbb{Z}$ be the function such that $f_p(S) = \min \{f(S), p\}$ for each set S . If f is a proper function then f_p is also a proper function. The algorithm performs k phases: for $1 \leq p \leq k$, at the end of phase p , the algorithm has a subgraph H_p that covers f_p . In phase p the algorithm starts with H_{p-1} that covers f_{p-1} and adds some additional nodes to obtain H_p that covers f_p . We can express the underlying optimization problem in phase p as follows.

It is convenient to assume that all of the vertices of H_{p-1} have zero weight; since we have already paid for the nodes, we can set their weight to zero at the beginning of phase p . Let $G'_p = (V, E(G) - E(H_{p-1}))$. (We emphasize that G'_p has all of the nodes of G and that the terminals and vertices of $V(H_{p-1})$ have zero weight.) Our goal is to select a minimum-weight subgraph H of G'_p that covers the following 0-1 function $h_p : 2^V \rightarrow \{0, 1\}$. For each set S , we have $h_p(S) = 1$ iff $f(S) \geq p$ and $|\delta_{H_{p-1}}(S)| = p - 1$. The function h_p is known to be an uncrossable function [168]; note that it may no longer be a proper function. We use a primal-dual algorithm to cover h_p in the graph G'_p . A 2-approximation exists for this covering problem for the edge-weighted problem and an $O(\log n)$ -approximation for the node-weighted case [136]. We show that the primal-dual algorithm achieves an $O(1)$ -approximation for the node-weighted case in planar graphs, however, we emphasize that it only applies for the specific uncrossable functions that arise from proper functions as above; in particular it is important that the chosen subgraphs at the end of each phase are node-induced. We describe and analyze the primal-dual algorithm below and point out the place where we need this restriction.

7.2.1 A primal-dual algorithm for the augmentation problem

In the following, we fix a phase p of the augmentation framework. Let $h = h_p$ and $G' = G'_p$. Recall that all of the terminals and the vertices selected in the first $p - 1$ phases have zero weight. In the following, we use $\Gamma_{G'}(S)$ to denote the set of all vertices v such that $v \notin S$ but there is an edge $uv \in E(G')$ such that $u \in S$. We use a primal-dual algorithm in order to select a subgraph H of G' that covers h . The primal and dual LPs that we use are described below. We remark that the primal LP has unbounded integrality gap for an arbitrary uncrossable function⁶. However, the function h that arises from a proper function f in the augmentation framework has additional properties that allow us to avoid such examples.

(Primal)	(Dual)
$\min \sum_{v \in V} x(v)w(v)$	$\max \sum_{S \subseteq V} y(S)h(S)$
s.t. $\sum_{v \in \Gamma_{G'}(S)} x(v) \geq h(S) \quad \forall S \subseteq V$	s.t. $\sum_{S: v \in \Gamma_{G'}(S)} y(S) \leq w(v) \quad \forall v \in V$
$x(v) \geq 0 \quad \forall v \in V$	$y(S) \geq 0 \quad \forall S \subseteq V$

We omit the constraint $x(v) \leq 1$ in the primal since h is a 0-1 function.

The primal-dual algorithm is a “standard” one in that it is the natural adaptation to the node-weighted setting (as done in [65]) of the primal-dual algorithm for edge-weighted network design formalized by Goemans and Williamson [88]. The algorithm selects a set $X \subseteq V(G')$ of nodes such that the graph $G'[X]$ covers h . Initially, X consists of all vertices that have zero weight. We also maintain a feasible dual solution \mathbf{y} that is implicitly initialized to zero. We proceed in iterations. Consider iteration i and let X_{i-1} be the set of nodes selected in the first $i - 1$ iterations; the set X_0 consists of all zero-weight vertices. A set S is *violated* with respect to X_{i-1} iff $h(S) = 1$ and $\delta_{G'[X_{i-1}]}(S) = \emptyset$. A set S is a *minimal violated set* with respect to X_{i-1} iff S is a violated set and no proper subset of S is violated. Let \mathcal{C}_i denote the collection of all minimal violated sets with respect to X_{i-1} . As shown in [168], no two minimal violated sets of an uncrossable function can

⁶A simple example is a function h such that there is a single set S such that $h(S) = 1$. Each vertex in S has weight 1, and each vertex in $V - S$ has weight 0. The optimum solution has value 1 since at least one node in S has to be picked but the optimum LP value is 0; note that the value is 0 even if we have integrality constraints.

intersect; further the collection of minimal violated sets for h arising from proper functions can be computed in polynomial time. Moreover, Lemma 7.2.1 below shows that the sets in \mathcal{C}_i are subsets of X_{i-1} . If \mathcal{C}_i is empty, $G'[X_{i-1}]$ covers h and we return $G'[X_{i-1}]$. Otherwise, we increase the dual variables $\{y(S)\}_{S \in \mathcal{C}_i}$ uniformly until a dual constraint for a vertex v becomes tight, i.e., we have $\sum_{S: v \in \Gamma_{G'}(S)} y(S) = w(v)$; we add v to X . Note that, since the components of \mathcal{C}_i are contained in X_{i-1} , for each minimal violated component $C \in \mathcal{C}_i$, none of the vertices in $\Gamma_{G'}(C)$ are in X_{i-1} and thus it is possible to increase the dual variables $\{y(S)\}_{S \in \mathcal{C}_i}$.

Finally we perform a *reverse-delete* step. Let X be the set of vertices selected by the primal-dual algorithm. We select a subset Y of X as follows. We start with $Y = X$. We order the vertices of Y in the reverse of the order in which they were selected by the primal-dual algorithm. Let v be the current vertex. If $G'[Y - v]$ is a feasible cover for h , we remove v from Y .

The primal-dual algorithm described above is not well-defined for an arbitrary uncrossable function h but the following property holds for those that arise from proper functions. Using the following lemma, we can show that the algorithm is well-defined and it outputs a cover of h in polynomial time.

Lemma 7.2.1. *Every minimal violated component $C \in \mathcal{C}_i$ is a subset of X_{i-1} .*

Proof. Consider $C \in \mathcal{C}_i$ and suppose $C \not\subseteq X_{i-1}$. Let $C' = C \cap X_{i-1}$. We observe that $f_p(C \setminus C') = 0$ since all the terminals are in X_{i-1} . Since f_p is maximal, we have $f_p(C) \leq \max\{f_p(C'), f_p(C \setminus C')\} = \max\{f_p(C'), 0\} = f_p(C')$. Since $C \in \mathcal{C}_i$, we have $f_p(C) = p$ and $|\delta_{G[X_{i-1}]}(C)| = p - 1$. Therefore $f_p(C') \geq f_p(C) = p$. Additionally, $\delta_{G[X_{i-1}]}(C) = \delta_{G[X_{i-1}]}(C')$, since $G[X_{i-1}]$ does not have any edges incident to vertices in $V \setminus X_{i-1}$. It follows that C' is violated with respect to X_{i-1} , which contradicts the minimality of C . \square

The following lemma follows from Lemma 7.2.1 and the fact that we can compute the minimal violated components in polynomial time [168]. The proof of the lemma is relatively standard once we have Lemma 7.2.1 and we omit it.

Lemma 7.2.2. *The primal-dual algorithm returns a feasible cover of h . Moreover, the algorithm can be implemented to run in polynomial time.*

Lemma 7.2.3. *The dual solution \mathbf{y} constructed by the primal-dual algorithm satisfies the primal complementary slackness conditions. More precisely, for each vertex $v \in X$, we have $\sum_{S:v \in \Gamma_{G'}(S)} y(S) = w(v)$.*

Proof. We can prove the lemma by induction on the number of iterations of the primal-dual algorithm. Initially, \mathbf{y} is zero and X_0 consists of all zero-weight vertices. Thus the complementary slackness conditions are satisfied at the beginning of the algorithm. Now consider iteration $i > 0$. By Lemma 7.2.1, none of the vertices in X_{i-1} are adjacent in G' to the minimal violated components of \mathcal{C}_i . Thus, at the end of iteration i , we have $\sum_{S:v \in \Gamma_{G'}(S)} y(S) = w(v)$ for each vertex $v \in X_{i-1}$. Additionally, the vertices in $X_i - X_{i-1}$ became tight in iteration i . Thus, at the end of iteration i , we have $\sum_{S:v \in \Gamma_{G'}(S)} y(S) = w(v)$ for each vertex $v \in X_i - X_{i-1}$ as well. \square

Now we turn our attention to the analysis of the primal-dual algorithm. In the following, we show that the algorithm achieves an $O(1)$ approximation for the augmentation problem when the graph G is from a minor-closed family of graphs \mathcal{G} ; the constant depends on the family \mathcal{G} .

Theorem 7.2.4. *If G is a graph from a minor-closed family of graphs \mathcal{G} , the weight of the set Y is $O(\text{OPT}_h)$, where OPT_h is the optimum solution to the LP relaxation for covering h .*

The dual variables are grown uniformly in each iteration and the standard primal-dual analysis [88] gives a condition under which the approximation ratio can be upper bounded. This is encapsulated in the lemma below.

Lemma 7.2.5. *Let $B_i = Y - X_{i-1}$. Suppose there exists a γ such that, for each iteration i of the primal-dual algorithm, $\sum_{C \in \mathcal{C}_i} |B_i \cap \Gamma_{G'}(C)| \leq \gamma |\mathcal{C}_i|$. Then the weight of Y is at most γOPT_h , where OPT_h is the value of an optimal solution to the LP relaxation.*

Proof. By Lemma 7.2.3, y satisfies the primal complementary slackness conditions. Therefore we have

$$\sum_{v \in Y} w(v) = \sum_{v \in Y} \sum_{S:v \in \Gamma_{G'}(S)} y(S) = \sum_{S \subseteq V} y(S) |Y \cap \Gamma_{G'}(S)|.$$

Note that, if we can show that $\sum_{S \subseteq V} y(S) |Y \cap \Gamma_{G'}(S)| \leq \gamma \sum_{S \subseteq V} y(S) h(S)$, it will follow that we have a γ -approximation: since y is feasible, $\sum_{S \subseteq V} y(S) h(S)$ is a lower bound on the fractional optimum, which in turn is a lower bound on the integral optimum.

We show by induction on the number of iterations of the primal-dual algorithm that

$$\sum_{S \subseteq V} y(S) |Y \cap \Gamma_{G'}(S)| \leq \gamma \sum_{S \subseteq V} y(S) h(S).$$

Note that $y(S) > 0$ only if $h(S) = 1$. Therefore $\sum_{S \subseteq V} y(S) h(S) = \sum_{S \subseteq V} y(S)$ and thus it suffices to prove that

$$\sum_{S \subseteq V} y(S) |Y \cap \Gamma_{G'}(S)| \leq \gamma \sum_{S \subseteq V} y(S).$$

Initially, all dual variables $y(S)$ are zero and therefore the inequality holds. Now consider iteration i of the primal-dual algorithm. Recall that X_{i-1} is the set of all vertices selected in the first $i - 1$ iterations of the primal-dual algorithm and \mathcal{C}_i is the set of all minimal violated sets with respect to X_{i-1} .

Let ϵ denote the amount by which we increased $y(S)$, where $S \in \mathcal{C}_i$, in iteration i . The left-hand side increases by $\sum_{C \in \mathcal{C}_i} \epsilon |Y \cap \Gamma_{G'}(C)|$, and the right-hand side increases by $\gamma \epsilon |\mathcal{C}_i|$. Therefore it suffices to show that

$$\sum_{C \in \mathcal{C}_i} |Y \cap \Gamma_{G'}(C)| \leq \gamma |\mathcal{C}_i|.$$

Recall that $B_i = Y - X_{i-1}$. By Lemma 7.2.1, for each component $C \in \mathcal{C}_i$, $\Gamma_{G'}(C) \cap X_{i-1}$ is empty and thus $\Gamma_{G'}(C) \cap Y = \Gamma_{G'}(C) \cap B_i$. Therefore we can rewrite the inequality above as:

$$\sum_{C \in \mathcal{C}_i} |B_i \cap \Gamma_{G'}(C)| \leq \gamma |\mathcal{C}_i|.$$

□

The content of the above lemma is the following. Consider the minimal violated sets in \mathcal{C}_i . The set $B_i = Y - X_{i-1}$ forms a *node-minimal* set that together with X_{i-1} covers h (minimality follows from the reverse delete step). We are interested in γ , the “average degree”⁷ of the components in \mathcal{C}_i , with respect to nodes in B_i . In general graphs γ can be $\Omega(n)$ in the worst case which does not give a useful bound. However, planar graphs are sparse. Thus one can bound the average degree

⁷Here we are abusing the term slightly and we refer to the ratio $\sum_{C \in \mathcal{C}_i} |B_i \cap \Gamma_{G'}(C)| / |\mathcal{C}_i|$ as the average degree of the components in \mathcal{C}_i . One can view the ratio as the average degree of the components if we shrink each of the components in \mathcal{C}_i to a single vertex and we remove parallel edges.

if one can bound the number of *nodes* in B_i that are adjacent to components in \mathcal{C}_i . This was done in [65] for 0-1 proper functions but the case of uncrossable functions is more involved and it is our main technical contribution. Theorem 7.3.1 is stated in a general and useful form and proved in Section 7.3. Assuming the theorem, we finish the analysis as follows. The following lemma upper bounds the number of nodes in B_i that are adjacent to components in \mathcal{C}_i .

Lemma 7.2.6. *Let B'_i be the set of all vertices $u \in B_i$ such that $u \in \Gamma_{G'}(C)$ for some component $C \in \mathcal{C}_i$. We have $|B'_i| \leq 4|\mathcal{C}_i|$.*

In order to take advantage of the fact that planar and minor-closed graphs are sparse, we need the following technical ingredient. The proof of Lemma 7.2.7 follows from the maximality of f_p and it is similar to the proof of Lemma 7.2.1.

Lemma 7.2.7. *For each component $C \in \mathcal{C}_i$, the graph $G[C]$ is connected.*

Proof. Suppose for contradiction that the graph $G[C]$ is disconnected. Let A_1, A_2, \dots, A_ℓ be the connected components of $G[C]$. Since the function f_p is proper there is an index j such that $f_p(A_j) \geq f_p(C)$. Since A_j is a connected component of $G[C]$, we have $|\delta_G(A_j)| \leq |\delta_G(C)|$. Since $G[X_{i-1}]$ is a subgraph of G , it follows that $\delta_{G[X_{i-1}]}(A_j) \subseteq \delta_{G[X_{i-1}]}(C)$. Since C is a violated component with respect to h in $G[X_{i-1}]$ we have $f_p(C) = p$ and $|\delta_{G[X_{i-1}]}(C)| = p - 1$. It follows from the discussion above that $f_p(A_j) \geq p$ and $|\delta_{G[X_{i-1}]}(C)| \leq p - 1$, and thus A_j is a violated set with respect to h in $G[X_{i-1}]$. But this contradicts the minimality of C . \square

In order to finish the average degree argument, we shrink each component $C \in \mathcal{C}_i$ into a single node and we use Lemma 7.2.6 and the fact that, for a graph K from a minor-closed family \mathcal{G} there is a constant c' that depends only on the family such that $|E(K)| \leq c'|V(K)|$.

Lemma 7.2.8. *Let $B_i = Y - X_{i-1}$. If G is a graph from a minor-closed family of graphs \mathcal{G} , we have $\sum_{C \in \mathcal{C}_i} |B_i \cap \Gamma_{G'}(C)| \leq c|\mathcal{C}_i|$, where c is a constant that depends only on the family \mathcal{G} .*

Proof. Let B'_i be the set of all vertices $u \in B_i$ such that $u \in \Gamma_{G'}(C)$ for some component $C \in \mathcal{C}_i$. We construct a minor K of G as follows; we start with G , remove nodes in $V - (B'_i \cup_{C \in \mathcal{C}_i} C)$ and then shrink each $C \in \mathcal{C}_i$ to a single node. We also remove parallel edges in order to get a simple graph. The resulting graph is indeed a minor of G , since for each $C \in \mathcal{C}_i$, we have seen that $G[C]$

is a connected component of G (and the components in \mathcal{C}_i are disjoint as we noted). The total number of nodes in K is equal to $|B'_i| + |\mathcal{C}_i| \leq 5|\mathcal{C}_i|$ by Lemma 7.2.6.

Note that $\sum_{C \in \mathcal{C}_i} |B_i \cap \Gamma_{G'}(C)|$ is equal to the number of edges of K connecting the vertices in B'_i to the vertices representing the shrunk components of \mathcal{C}_i . Therefore it suffices to upper bound the number of edges in K . Since K is from a minor-closed family \mathcal{G} , there is a constant c' that depends only on the family such that $|E(K)| \leq c'|V(K)| \leq 5c'|\mathcal{C}_i|$. \square

Theorem 7.2.4 follows from Lemma 7.2.5 and Lemma 7.2.8. Theorem 7.2.4 together with the augmentation framework gives an $O(k)$ -approximation for finding a minimum node-weighted subgraph to cover a proper function with maximum requirement k . The result for EC-SNDP is a special case of this result.

7.3 Proof of Theorem 7.3.1

Let $G = (V, E)$ be a graph. Let $h : 2^V \rightarrow \{0, 1\}$ be a requirement function. A set S is *violated* if $h(S) = 1$. A set C is a *minimal violated component* of h if C is violated and no proper subset of C is violated. Let H be a subgraph of G . The graph H is a *feasible cover* for h if, for any set $S \subseteq V$ such that $h(S) = 1$, there is at least one edge of H leaving S ; in other words, $|\delta_H(S)| \geq h(S)$. We say that H is a *node-minimal* feasible cover for h if, for any vertex $v \in V(H)$, $H - v$ is not a feasible cover for h .

Now we are ready to state our main theorem.

Theorem 7.3.1. *Let $h : 2^V \rightarrow \{0, 1\}$ be an uncrossable function. Let \mathcal{C} be the minimal violated components of h . Let H be a node-minimal feasible cover for h . Let X be the set of all vertices $v \in V(H)$ such that v is not in the union of the components in \mathcal{C} and there is an edge of H connecting v to a component of \mathcal{C} . Then $|X| \leq 4|\mathcal{C}|$.*

We devote the rest of this section to the proof of Theorem 7.3.1. A basic property of uncrossable functions [168] is stated below.

Lemma 7.3.2. *Let h be an uncrossable function. The minimal violated components of h are disjoint. Moreover, if S is a violated set and C is a minimal violated component, S and C do not properly intersect.*

We start with a high-level overview of the proof. The main idea is to pick a subset M of the edges of H such that M is an *edge-minimal* feasible cover for h . Such a minimal cover has nice properties that were pointed out and used in the analysis for edge-weighted problems [168]. More precisely, for each edge $e \in M$, we can pick a “witness set” that is a violated set such that e is the only edge of M that is leaving the set. Moreover, we can pick a family of witness sets, one for each edge of M , such that the family is laminar⁸. This laminar family can be used to upper bound the number of edges of M that are incident to the components of \mathcal{C} .

We are interested in analyzing a node-minimal cover H which is not necessarily edge-minimal; there can be a node u that is adjacent to components in \mathcal{C} but it is possible that an edge-minimal cover M does not contain any of the edges connecting u to components of \mathcal{C} . Thus we cannot use the witness family to count such vertices. We address this issue by counting them separately using a witness family for a *different* set of edges.

We now turn our attention to the formal proof of the theorem. We refer to the vertices in X as *critical* vertices. We refer to edges connecting a critical vertex to a component $C \in \mathcal{C}$ as *red edges*, and we refer to all other edges of H as *blue edges*.

We define two subsets of edges F_1 and F_2 as follows. We start with $F_1 = E(H)$ and we remove some of the edges as follows. We order the *blue edges* arbitrarily. We consider the blue edges in this order. Let e be the current edge. If $F_1 - e$ is a feasible solution for h , we remove e from F_1 . At the end of this process, each red edge is in F_1 and each blue edge in F_1 is necessary to cover h . We refer to critical vertices that are incident to at least one blue edge of F_1 as *regular* vertices; critical vertices that are not regular are referred to as *special* vertices. As we will see shortly, we can use the blue edges in F_1 to upper bound the number of regular vertices.

In order to count the special vertices, we pick a subset F_2 of F_1 as follows. We start with $F_2 = F_1$. We consider the *red edges* of F_2 in some order. Let e be the current edge. If $F_2 - e$ is a feasible cover, we remove e from F_2 . We can use the red edges in F_2 to upper bound the number of special vertices. Since H is a node-minimal cover for h , each special vertex is incident to at least one red edge of F_2 .

Note that F_2 is an edge-minimal feasible cover for h while F_1 is a feasible cover but is not

⁸A set family \mathcal{F} is laminar iff no two sets in \mathcal{F} properly intersect.

necessarily edge-minimal. The difficulty is in counting the regular vertices via F_1 . We consider the regular and special vertices separately. Theorem 7.3.1 follows from the following two lemmas.

Lemma 7.3.3. *The number of regular vertices is at most $2|C|$.*

Lemma 7.3.4. *The number of special vertices is at most $2|C|$.*

Our counting arguments are based on the laminar witness family approach of Williamson *et al.* More precisely, we define a witness set as follows.

Definition 7.3.5. *Let F be a set of edges. A set $S_e \subseteq V$ is an **F -witness set** for an edge e iff $h(S_e) = 1$ and $\delta_F(S_e) = \{e\}$.*

An F -witness set S_e is a violated set; from Lemma 7.3.2 it follows that for each component $C \in \mathcal{C}$, $C \subseteq S_e$ or $C \cap S_e = \emptyset$.

Recall that a family of sets \mathcal{L} is *laminar* if no two sets in \mathcal{L} properly intersect; differently said, for any two sets $A, B \in \mathcal{L}$, either A and B are disjoint or one is contained in the other. The following lemma follows from [168].

Lemma 7.3.6 ([168]). *Let F be a feasible cover for an uncrossable function h . Let $M \subseteq F$ be a subset of F such that, for each edge $e \in M$, $F - e$ is not a feasible cover for h . There is a laminar family $\mathcal{L} = \{S_e \mid e \in M\}$ such that S_e is an F -witness set for e .*

Our approach is to use laminar witness families for the blue edges of F_1 and the red edges of F_2 in order to count the regular and special vertices. Before we turn our attention to the counting arguments, we describe some properties of laminar witness families that we need.

We can associate a forest \mathcal{F} with a laminar set family \mathcal{L} as follows. The forest \mathcal{F} has a node ν_S for each set $S \in \mathcal{L}$. We add an edge between ν_A and ν_B iff A is the smallest set in \mathcal{L} that contains B . Let $\mathcal{L} = \{S_e \mid e \in M\}$ be a laminar F -witness family for a set $M \subseteq F$ of edges. Let \mathcal{T} be the tree associated with $\mathcal{L} \cup \{V\}$; we root \mathcal{T} at the node ν_V .

We define the following bijection between the edges of the tree \mathcal{T} and the edges of M . Let e be an edge of M and let S_e be the witness set for e . The node ν_{S_e} has a parent ν_A in \mathcal{T} , and we associate the edge $e \in M$ with the edge (ν_A, ν_{S_e}) of \mathcal{T} . We say that the edge e *corresponds* to the edge (ν_A, ν_{S_e}) . A node ν_S of \mathcal{T} *owns* a vertex $v \in V$ iff S is the smallest set in $\mathcal{L} \cup \{V\}$ that contains v .

Proposition 7.3.7. *Let $\mathcal{L} = \{S_e \mid e \in M\}$ be a laminar F -witness family for a set $M \subseteq F$ of edges. Let \mathcal{T} be the tree associated with $\mathcal{L} \cup \{V\}$. For each leaf ν_S of \mathcal{T} there is a distinct component $C \in \mathcal{C}$ such that $C \subseteq S$.*

Proof. The set S represented by ν_S is a violated set and therefore it contains a minimal violated component $C \in \mathcal{C}$. Since ν_S is a leaf, ν_S owns all the vertices in C . \square

The following simple observation plays a crucial role in our counting argument.

Proposition 7.3.8. *Let $\mathcal{L} = \{S_e \mid e \in M\}$ be a laminar F -witness family for a set $M \subseteq F$ of edges. Let \mathcal{T} be the tree associated with $\mathcal{L} \cup \{V\}$. Let ν_S be a node of \mathcal{T} and let e be an edge of $F \setminus M$. Either both endpoints of e are contained in S or neither endpoint of e is contained in S . In particular, the endpoints of e are owned by the same node of \mathcal{T} .*

Proof. If $S = V$, then S contains both endpoints of e . Therefore we may assume that S is an F -witness set for an edge $f \in M$. Since e is in $F - M$, it follows that $e \neq f$. Since f is the only edge of M leaving S , it follows that e is not leaving S . Thus either both endpoints of e are in S or both endpoints of e are outside of S . \square

Corollary 7.3.9. *Let $\mathcal{L} = \{S_e \mid e \in M\}$ be a laminar F -witness family for a set $M \subseteq F$ of edges. Let \mathcal{T} be the tree associated with $\mathcal{L} \cup \{V\}$. Let e be an edge of $F \setminus M$. Then both endpoints of e are owned by the same node of \mathcal{T} .*

The following lemma was proved in [168].

Lemma 7.3.10 ([168]). *Let $\mathcal{L} = \{S_e \mid e \in M\}$ be a laminar F -witness family for a set $M \subseteq F$ of edges. Let \mathcal{T} be the tree associated with $\mathcal{L} \cup \{V\}$. Let e be an edge of M and let (ν_A, ν_{S_e}) be the edge of \mathcal{T} corresponding to e , where S_e is the witness set for e and ν_A is the parent of ν_{S_e} . Then ν_A owns one endpoint of e and ν_{S_e} owns the other endpoint of e .*

Counting argument for regular vertices. Let $\mathcal{L}_{F_1} = \{S_e \mid e \text{ is a blue edge in } F_1\}$ be a laminar F_1 -witness family for the blue edges in F_1 that is guaranteed by Lemma 7.3.6. Let \mathcal{T}_{F_1} be the tree associated with the family $\mathcal{L}_{F_1} \cup \{V\}$; we view \mathcal{T}_{F_1} as a rooted tree whose root is the node corresponding to V .

Recall that each regular vertex u is incident to a red edge ur ; the edge ur is in F_1 , since F_1 contains all the red edges. Additionally, u is incident to a blue edge $ub \in F_1$. Since r is contained in a minimal component of \mathcal{C} , it follows from Proposition 7.3.8 that the node of \mathcal{T}_{F_1} that owns u also owns a component $C_u \in \mathcal{C}$. Our approach is to charge each regular vertex u in its subtree; more precisely, we charge u to a component $C \in \mathcal{C}$ that is owned by a node in the subtree rooted at the node that owns u and C_u .

We charge each regular vertex u as follows. Recall that there is a blue edge $ub \in F_1$ that is incident to u . Let ν_A and ν_B be the nodes of \mathcal{T}_{F_1} that own u and b , respectively. By Lemma 7.3.10, one of ν_A, ν_B is the parent of the other.

Suppose that ν_A is the parent of ν_B . Since each leaf owns a component of \mathcal{C} (from Proposition 7.3.7), there is a descendant of ν_B (possibly ν_B itself) that owns a component of \mathcal{C} . Let ν_S be the closest such descendant, i.e., a descendant whose distance to ν_B is minimized. (If there are several descendants whose distance to ν_B is minimum, we pick one of them arbitrarily.) We charge u to one of the components of \mathcal{C} that ν_S owns; we refer to this charge as a *subtree charge* (since u is charged in a subtree rooted at a child of the node ν_A that owns u). Since a regular vertex v and its component C_v are owned by the same node of the tree, the components C_v serve as sentinels that ensure that there is at most one subtree charge to each component of \mathcal{C} .

Suppose that ν_A is a child of ν_B . We charge u to the component C_u ; we refer to this charge as a *parent charge* (since the charge corresponds to the tree edge connecting the node ν_A that owns C to its parent). Since each node has at most one parent edge, there is at most one parent charge to each component of \mathcal{C} .

Proposition 7.3.11. *There is at most one subtree charge to each component $C \in \mathcal{C}$.*

Proof. Suppose for contradiction that C incurs two subtree charges from two nodes u_1 and u_2 . Let ν_{A_1} and ν_{A_2} be the nodes of \mathcal{T}_{F_1} that own u_1 and u_2 , respectively. Since the node that owns C is a descendant of ν_{A_1} and ν_{A_2} , one of ν_{A_1}, ν_{A_2} is an ancestor of the other. Moreover, ν_{A_1} and ν_{A_2} cannot be the same node, since otherwise u_1 and u_2 would have been charged in different subtrees of $\nu_{A_1} = \nu_{A_2}$. Without loss of generality, ν_{A_1} is a proper ancestor of ν_{A_2} . By Proposition 7.3.8, ν_{A_2} owns a component $C' \in \mathcal{C}$ such that there is a red edge of F_1 connecting u_2 to a component $C' \in \mathcal{C}$. Since ν_{A_2} is closer to ν_{A_1} than the node that owns C is, we have the desired contradiction. \square

Proposition 7.3.12. *There is at most one parent charge to each component $C \in \mathcal{C}$.*

Proof. Let ν_A be the node of \mathcal{T}_{F_1} that owns C . Since a parent charge to C corresponds to the edge of \mathcal{T}_{F_1} connecting ν_A to its parent, C is charged at most once. \square

Proof of Lemma 7.3.3: Each component of \mathcal{C} is charged at most twice and thus the number of regular vertices is at most $2|\mathcal{C}|$. \square

Counting argument for special vertices. Recall that F_2 is an edge-minimal cover of h . Moreover, a critical vertex v is special only if there is an edge $e \in F_2$ (in fact a red edge) such that e connects v to a minimal violated component C . Thus, the total number of special vertices is upper bounded by $\sum_{C \in \mathcal{C}} |\delta_{F_2}(C)|$. Williamson *et al.* [168] show that for any edge-minimal cover of an uncrossable function this is upper bounded by $2|\mathcal{C}|$. Thus we can upper bound the number of special vertices by $2|\mathcal{C}|$ which proves Lemma 7.3.4. We remark that some of the regular vertices are counted in this step as well, but this can only help us.

7.4 Concluding remarks

In this chapter, we consider the node-weighted EC-SNDP problem in planar and minor-closed families of graphs. Our main result is an $O(k)$ approximation for the problem, which improves the previous $O(k \log n)$ approximation of Nutov [136].

In subsequent work that is not part of this thesis, we considered the more general Elem-SNDP and VC-SNDP problems that require the paths to be element and vertex disjoint, respectively. We gave a matching $O(k)$ approximation for the node-weighted Elem-SNDP problem in minor-free graphs, which improves a previous $O(k \log n)$ approximation of Nutov [136]. This result, together with a reduction from VC-SNDP to Elem-SNDP of Chuzhoy and Khanna [56], gives an $O(k^4 \log n)$ approximation for VC-SNDP in node-weighted minor-free graphs; this result is again an $O(\log n)$ factor improvement over the best approximation known for general graphs [136]. A notable special case of VC-SNDP is the single-source problem in which we have a root s and the requirement of a pair uv is non-zero only if $s \in \{u, v\}$. We believe that our algorithm for Elem-SNDP, together with a decomposition of Nutov [136], gives an $O(k^2)$ approximation for the single-source VC-SNDP problem in node-weighted planar graphs.

References

- [1] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. *Proc. of ACM STOC*, 573–581, 2005.
- [2] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing* 24(3):440–456, 1995.
- [3] C. Alpert and A. Kahng. Recent developments in netlist partitioning: A survey. *Integration: the VLSI Journal* 19(1-2):1–81, 1995.
- [4] M. Andrews. Approximation algorithms for the edge-disjoint paths problem via Raacke decompositions. *Proc. of IEEE FOCS*, 277–286, 2010.
- [5] M. Andrews, J. Chuzhoy, V. Guruswami, S. Khanna, K. Talwar, and L. Zhang. Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica* 30(5):485–520, 2010.
- [6] M. Andrews, J. Chuzhoy, S. Khanna, and L. Zhang. Hardness of the undirected edge-disjoint paths problem with congestion. *Proc. of IEEE FOCS*, 226–241, 2005.
- [7] A. Archer, M. Bateni, M. Hajiaghayi, and H. Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *Proc. of IEEE FOCS*, 427–436, 2009.
- [8] S. Arora. *Probabilistic checking of proofs and hardness of approximation problems*. Ph.D. thesis, University of California, Berkeley, 1994.
- [9] S. Arora. Polynomial time approximation schemes for Euclidean Traveling Salesman and other geometric problems. *Journal of the ACM* 45(5):753–782, 1998.
- [10] S. Arora and G. Karakostas. A $2 + \varepsilon$ approximation algorithm for the k -MST problem. *Mathematical Programming* 107(3):491–504, 2006.
- [11] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k -trees and prize-collecting salesmen. *SIAM Journal on Computing* 28(1):254–262, 1998.
- [12] Y. Azar and O. Regev. Combinatorial algorithms for the unsplittable flow problem. *Algorithmica* 44(1):49–66, 2006.
- [13] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM* 41(1):153–180, 1994.

- [14] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. *Proc. of ACM STOC*, 166–174, 2004.
- [15] M. H. Bateni, C. Chekuri, A. Ene, M. T. Hajiaghayi, N. Korula, and D. Marx. Prize-collecting Steiner problems on planar graphs. *Proc. of ACM-SIAM SODA*, 1028–1049, 2011.
- [16] M. H. Bateni, M. T. Hajiaghayi, and D. Marx. Prize-collecting network design on planar graphs. *CoRR* abs/1006.4339, 2010.
- [17] M. H. Bateni, M. T. Hajiaghayi, and D. Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM* 58(5):21, 2011.
- [18] P. Berman and V. Ramaiyer. Improved approximations for the Steiner tree problem. *Journal of Algorithms* 17(3):381–408, 1994.
- [19] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting Traveling Salesman Problem. *Mathematical Programming* 59(1):413–420, 1993.
- [20] G. Birkhoff. Rings of sets. *Duke Mathematical Journal* 3:443–454, 1937.
- [21] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for Orienteering and discounted-reward TSP. *SIAM Journal on Computing* 37(2):653–670, 2008.
- [22] H. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *Proc. of ACM STOC*, p. 234, 1993.
- [23] G. Borradaile, E. Demaine, and S. Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Proc. of STACS*, 171–182, 2009.
- [24] G. Borradaile, P. Klein, and C. Mathieu. A polynomial-time approximation scheme for Euclidean Steiner forest. *Proc. of IEEE FOCS*, 115–124, 2008.
- [25] G. Borradaile, P. Klein, and C. Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Trans. Algorithms* 5(3):1–31, 2009.
- [26] A. Z. Broder, A. M. Frieze, and E. Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM Journal on Computing* 23(5):976–989, 1994.
- [27] N. Buchbinder, J. Naor, and R. Schwartz. Simplex partitioning via exponential clocks and the multiway cut problem. *Proc. of ACM STOC*, 535–544, 2013.
- [28] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. An improved LP-based approximation for Steiner tree. *Proc. of ACM STOC*, 583–592, 2010.
- [29] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica* 47(1):53–78, 2007.
- [30] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. *Proc. of IEEE FOCS*, 36–45, 2003.
- [31] C. Chekuri and J. Chuzhoy. Large-treewidth graph decompositions and applications. *Proc. of ACM STOC*, 2013.

- [32] C. Chekuri and J. Chuzhoy. Polynomial bounds for the grid-minor theorem. *Arxiv preprint ArXiv:1305.6577*, 2013.
- [33] C. Chekuri and A. Ene. Approximation algorithms for submodular multiway partition. *Proc. of IEEE FOCS*, 2011.
- [34] C. Chekuri and A. Ene. Submodular cost allocation problem and applications. *Proc. of ICALP*, 354–366, 2011.
- [35] C. Chekuri and A. Ene. The all-or-nothing flow problem in directed graphs with symmetric demand pairs. Manuscript, 2013.
- [36] C. Chekuri and A. Ene. Poly-logarithmic approximation for maximum node disjoint paths with constant congestion. *Proc. of ACM-SIAM SODA*, 2013.
- [37] C. Chekuri, A. Ene, and A. Vakilian. Node-weighted network design in planar and minor-closed families of graphs. *Proc. of ICALP*, 206–217, 2012.
- [38] C. Chekuri, S. Guha, and J. Naor. The Steiner k -cut problem. *SIAM Journal on Discrete Mathematics* 20(1):261–271, 2006.
- [39] C. Chekuri, S. Kannan, A. Raja, and P. Viswanath. Multicommodity flows and cuts in polymatroidal networks. *Proc. of ITCS*, 399–408, 2012.
- [40] C. Chekuri, S. Khanna, and F. B. Shepherd. The all-or-nothing multicommodity flow problem. *Proc. of ACM STOC*, 156–165, 2004.
- [41] C. Chekuri, S. Khanna, and F. B. Shepherd. Edge-disjoint paths in planar graphs. *Proc. of IEEE FOCS*, 71–80, 2004.
- [42] C. Chekuri, S. Khanna, and F. B. Shepherd. Multicommodity flow, well-linked terminals, and routing problems. *Proc. of ACM STOC*, 183–192, 2005.
- [43] C. Chekuri, S. Khanna, and F. B. Shepherd. Well-linked terminals for node-capacitated routing problems. Manuscript, 2005.
- [44] C. Chekuri, S. Khanna, and F. B. Shepherd. An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing* 2(7):137–146, 2006.
- [45] C. Chekuri, S. Khanna, and F. B. Shepherd. Edge-disjoint paths in planar graphs with constant congestion. *SIAM Journal on Computing* 39:281–301, 2009.
- [46] C. Chekuri, S. Khanna, and F. B. Shepherd. A note on multiflows and treewidth. *Algorithmica* 54(3):400–412, 2009.
- [47] C. Chekuri and N. Korula. A graph reduction step preserving element-connectivity and applications. *Proc. of ICALP* 254–265, 2009.
- [48] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms* 8(3):23, 2012.
- [49] C. Chekuri, M. Mydlarz, and F. B. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms* 3(3):27, 2007.

- [50] J. Cheriyan and M. Salavatipour. Packing element-disjoint steiner trees. *ACM Transactions on Algorithms* 3(4):47, 2007.
- [51] M. Chlebík and J. Chlebíková. Approximation hardness of the Steiner tree problem on graphs. *Proc. of SWAT*, 170–179, 2002.
- [52] F. Chudak, T. Roughgarden, and D. Williamson. Approximate k -MSTs and k -Steiner trees via the primal-dual method and Lagrangean relaxation. *Mathematical Programming* 100(2):411–421, 2004.
- [53] J. Chuzhoy. Routing in undirected graphs with constant congestion. *Proc. of ACM STOC*, 2012.
- [54] J. Chuzhoy, V. Guruswami, S. Khanna, and K. Talwar. Hardness of routing with congestion in directed graphs. *Proc. of ACM STOC*, 165–178, 2007.
- [55] J. Chuzhoy and S. Khanna. Polynomial flow-cut gaps and hardness of directed cut problems. *Journal of the ACM* 56(2):6, 2009.
- [56] J. Chuzhoy and S. Khanna. An $O(k^3 \log n)$ -approximation algorithm for vertex-connectivity survivable network design. *Theory of Computing* 8:401–413, 2012.
- [57] J. Chuzhoy and S. Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. *Proc. of IEEE FOCS*, 2012.
- [58] V. Chvátal. Tough graphs and hamiltonian circuits. *Discrete Mathematics* 5(3):215–228, 1973.
- [59] G. Călinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint. *IPCO*, 182–196, 2007.
- [60] G. Călinescu, H. J. Karloff, and Y. Rabani. An improved approximation algorithm for multiway cut. *Journal of Computer and System Sciences* 60(3):564–574, 2000.
- [61] W. Cunningham. On submodular function minimization. *Combinatorica* 5(3):185–192. Springer, 1985.
- [62] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing* 23(4):864–894, 1994.
- [63] A. DeLong, A. Osokin, H. N. Isack, and Y. Boykov. Fast approximate energy minimization with label costs. *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2173–2180, 2010.
- [64] E. Demaine and M. Hajiaghayi. Approximation schemes for planar graph problems. *Encyclopedia of Algorithms*, 2008.
- [65] E. Demaine, M. Hajiaghayi, and P. Klein. Node-weighted Steiner tree and group Steiner tree in planar graphs. *Proc. of ICALP*, 328–340, 2009.
- [66] E. Demaine and M. T. Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica* 28:19–36. Springer Berlin / Heidelberg, 2008. (<http://dx.doi.org/10.1007/s00493-008-2140-4>).

- [67] E. D. Demaine, M. T. Hajiaghayi, and B. Mohar. Approximation algorithms via contraction decomposition. *Proc. of ACM-SIAM SODA*, 278–287, 2007.
- [68] S. Dughmi. Submodular functions: Extensions, distributions, and algorithms. A survey. *CoRR* abs/0912.0322, 2009.
- [69] J. Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial structures and their applications* 69–87, 1970.
- [70] A. Ene and J. Vondrák. A lattice matching problem and the hardness of submodular cost allocation. Manuscript, 2013.
- [71] A. Ene, J. Vondrák, and Y. Wu. Local distribution and the symmetry gap: Approximability of multiway partitioning problems. *Proc. of ACM-SIAM SODA*, 2013.
- [72] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing* 5(4):691–703, 1976.
- [73] U. Feige, M. Hajiaghayi, and J. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing* 38:629–657, 2008.
- [74] L. Fleischer and S. Iwata. Improved algorithms for submodular function minimization and submodular flow. *Proc. of ACM STOC*, 107–116, 2000.
- [75] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science* 10(2):111–121, 1980.
- [76] A. Frank. On connectivity properties of Eulerian digraphs. *Annals of Discrete Mathematics* 41:179–194, 1989.
- [77] A. Frieze. Edge-disjoint paths in expander graphs. *SIAM Journal on Computing* 30(6):1790–1801, 2001.
- [78] S. Fujishige. *Submodular functions and optimization*. Elsevier Science, 2005.
- [79] T. Fukunaga. Computing minimum multiway cuts in hypergraphs from hypertree packings. *Proc. of IPCO*, 15–28, 2010.
- [80] M. Fürer and B. Raghavachari. Approximating the minimum-degree Steiner tree to within one of optimal. *Journal of Algorithms* 17(3):409–423, 1994.
- [81] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [82] N. Garg. A 3-approximation for the minimum tree spanning k vertices. *Proc. of IEEE FOCS*, 302–309, 1996.
- [83] N. Garg. Saving an epsilon: a 2-approximation for the k -MST problem in graphs. *Proc. of ACM STOC*, 396–402, 2005.
- [84] N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica* 18(1):3–20, 1997.

- [85] N. Garg, V. V. Vazirani, and M. Yannakakis. Multiway cuts in node weighted graphs. *Journal of Algorithms* 50(1):49–61, 2004.
- [86] D. Ge, Y. Ye, and J. Zhang. The fixed-hub single allocation problem: A geometric rounding approach. Manuscript, 2007.
- [87] G. Goel, C. Karande, P. Tripathi, and L. Wang. Approximability of combinatorial problems with multi-agent submodular cost functions. *Proc. of IEEE FOCS*, 755–764, 2009.
- [88] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing* 24:296, 1995.
- [89] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, E. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. *Proc. of ACM-SIAM SODA*, 223–232, 1994.
- [90] M. X. Goemans, N. J. A. Harvey, S. Iwata, and V. S. Mirrokni. Approximating submodular functions everywhere. *Proc. of ACM-SIAM SODA*, 535–544, 2009.
- [91] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing* 24(2):296–317, 1995.
- [92] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. *Approximation algorithms for NP-hard problems*, 144–191, 1996. PWS Publishing Co.
- [93] O. Goldschmidt and D. Hochbaum. A polynomial algorithm for the k -cut problem for fixed k . *Mathematics of Operations Research* 24–37, 1994.
- [94] A. Gupta and J. Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science* 16(1):3–20, 2011.
- [95] V. Guruswami, S. Khanna, R. Rajaraman, F. B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences* 67(3):473–496, 2003.
- [96] S. Gutner. Elementary approximation algorithms for prize collecting Steiner tree problems. *Information Processing Letters* 107(1):39–44, 2008.
- [97] M. Hajiaghayi and K. Jain. The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema. *Proc. of ACM-SIAM SODA*, 631–640, 2006.
- [98] M. T. Hajiaghayi, R. Khandekar, G. Kortsarz, and Z. Nutov. Prize-collecting Steiner network problems. *ACM Transactions on Algorithms* 9(1):2, 2012.
- [99] J. Håstad. Some optimal inapproximability results. *Journal of the ACM* 48(4):798–859, 2001.
- [100] H. Hind and O. Oellermann. Menger-type results for three or more vertices. *Congressus Numerantium* 179–204, 1996.
- [101] D. S. Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996.

- [102] S. Hougardy and H. Prömel. A 1.598 approximation algorithm for the Steiner problem in graphs. *Proc. of ACM-SIAM SODA*, 448–453, 1999.
- [103] S. Iwata and K. Nagano. Submodular function minimization under covering constraints. *Proc. of IEEE FOCS*, 671–680, 2009.
- [104] S. Iwata and J. Orlin. A simple combinatorial algorithm for submodular function minimization. *Proc. of ACM-SIAM SODA*, 1230–1237, 2009.
- [105] B. Jackson. Some remarks on arc-connectivity, vertex splitting, and orientation in graphs and digraphs. *Journal of Graph Theory* 12(3):429–436, 1988.
- [106] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica* 21(1):39–60, 2001.
- [107] D. Johnson, M. Minkoff, and S. Phillips. The prize collecting Steiner tree problem: theory and practice. *Proc. of ACM-SIAM SODA*, 760–769, 2000.
- [108] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B* 82(1):138–154, 2001.
- [109] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Excluding a grid minor in digraphs. Manuscript, 2001.
- [110] D. R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research* 24(2):383–413, 1999.
- [111] D. R. Karger, P. N. Klein, C. Stein, M. Thorup, and N. E. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. *Mathematics of Operations Research* 29(3):436–461, 2004.
- [112] M. Karpinski and A. Zelikovsky. New approximation algorithms for the Steiner tree problems. *Journal of Combinatorial Optimization* 1(1):47–65, 1997.
- [113] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 85–103, 1972.
- [114] K. Kawarabayashi and Y. Kobayashi. The edge disjoint paths problem in Eulerian graphs and 4-edge-connected graphs. *Proc. of ACM-SIAM SODA*, 345–353, 2010.
- [115] K. Kawarabayashi and Y. Kobayashi. Breaking $O(n^{1/2})$ -approximation algorithms for the edge-disjoint paths problem with congestion two. *Proc. of ACM STOC*, 81–88, 2011.
- [116] R. Khandekar, S. Rao, and U. Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM* 56(4):19, 2009.
- [117] J. Kleinberg. Approximation algorithms for disjoint paths problems. Ph.D. thesis, MIT, 1996.
- [118] J. Kleinberg. An approximation algorithm for the disjoint paths problem in even-degree planar graphs. *Proc. of IEEE FOCS*, 627–636, 2005.
- [119] J. Kleinberg and R. Rubinfeld. Short paths in expander graphs. *Proc. of IEEE FOCS*, 86–95, 1996.

- [120] J. Kleinberg and E. Tardos. Disjoint paths in densely embedded graphs. *Proc. of IEEE FOCS*, 52–61, 1995.
- [121] J. Kleinberg and E. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. *Journal of Computers and System Sciences* 57(1):61–73, 1998.
- [122] J. M. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields. *Journal of the ACM* 49(5):616–639, 2002.
- [123] P. Klein, S. Plotkin, S. Rao, and E. Tardos. Approximation algorithms for Steiner and directed multicuts. *Journal of Algorithms* 22(2):241–269, 1997.
- [124] P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms* 19(1):104–115, 1995.
- [125] P. N. Klein. A subset spanner for planar graphs: with application to subset TSP. *Proc. of ACM STOC*, 749–756, 2006.
- [126] P. N. Klein, S. A. Plotkin, and S. Rao. Excluded minors, network decomposition, and multicommodity flow. *Proc. of ACM STOC*, 682–690, 1993.
- [127] S. Kolliopoulos and C. Stein. Approximating disjoint-path problems using packing integer programs. *Mathematical Programming* 99(1):63–87, 2004.
- [128] P. Kolman and C. Scheideler. Improved bounds for the unsplittable flow problem. *Journal of Algorithms* 61(1):20–44, 2006.
- [129] G. Kortsarz and Z. Nutov. Approximating minimum cost connectivity problems. *Handbook on Approximation Algorithms and Metaheuristics*, 2007. Chapman and Hall/CRC.
- [130] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM* 46(6):787–832, 1999.
- [131] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica* 15(2):215–245, 1995.
- [132] L. Lovász. Submodular functions and convexity. *Mathematical programming: the state of the art* 235–257, 1983.
- [133] W. Mader. A reduction method for edge-connectivity in graphs. *Annals of Discrete Mathematics* 3:145–164, 1978.
- [134] C. Nagarajan, Y. Sharma, and D. Williamson. Approximation algorithms for prize-collecting network design problems with general connectivity requirements. *Proc. of WAOA*, 174–187, 2008.
- [135] G. Naves and A. Sebo. Multiflow feasibility: An annotated tableau. *Research Trends in Combinatorial Optimization* p. 261, 2008.
- [136] Z. Nutov. Approximating Steiner networks with node-weights. *SIAM Journal of Computing* 39(7):3001–3022, 2010.

- [137] Z. Nutov. Approximating Steiner network activation problems. *Proc. of LATIN*, 2012.
- [138] K. Okumoto, T. Fukunaga, and H. Nagamochi. Divide-and-conquer algorithms for partitioning hypergraphs and submodular systems. *Algorithmica* 1–20, 2010.
- [139] L. Orecchia, L. Schulman, U. Vazirani, and N. Vishnoi. On partitioning graphs via single commodity flows. *Proc. of ACM STOC*, 461–470, 2008.
- [140] D. Panigrahi. Survivable network design problems in wireless networks. *Proc. of ACM-SIAM SODA*, 2011.
- [141] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences* 43(3):425–440, 1991.
- [142] M. Queyranne. A combinatorial algorithm for minimizing symmetric submodular functions. *Proc. of ACM-SIAM SODA*, 98–101, 1995.
- [143] M. Queyranne. On optimum size-constrained set partitions. Combinatorial Optimization Workshop, AUSSOIS, 1999.
- [144] H. Racke. Minimizing congestion in general networks. *Proc. of IEEE FOCS*, 43–52, 2002.
- [145] P. Raghavan and C. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica* 7(4):365–374, 1987.
- [146] S. Rao and S. Zhou. Edge disjoint paths in moderately connected graphs. *SIAM Journal on Computing* 39(5):1856–1887, 2010.
- [147] R. Ravi, R. Sundaram, M. Marathe, D. Rosenkrantz, and S. Ravi. Spanning trees: Short or small. *SIAM Journal on Discrete Mathematics* 9(2):178–200, 1996.
- [148] B. Reed. Introducing directed tree width. *Electronic Notes in Discrete Mathematics* 3:222–229, 1999.
- [149] N. Robertson and P. Seymour. Graph minors V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B* 41(1):92–114, 1986.
- [150] N. Robertson and P. Seymour. Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B* 63(1):65–110, 1995.
- [151] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B* 62(2):323–348, 1994.
- [152] G. Robins and A. Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM Journal on Discrete Mathematics* 19(1):122–134, 2006.
- [153] T. Schaefer. The complexity of satisfiability problems. *Proc. of ACM STOC*, 216–226, 1978.
- [154] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B* 80(2):346–355. Elsevier, 2000.
- [155] A. Schrijver. *Combinatorial optimization*. Springer, 2003.

- [156] L. Seguin-Charbonneau and F. B. Shepherd. Maximum edge-disjoint paths in planar graphs with congestion 2. *Proc. of IEEE FOCS*, 200–209, 2011.
- [157] Y. Sharma, C. Swamy, and D. Williamson. Approximation algorithms for prize collecting forest problems with submodular penalty functions. *Proc. of ACM-SIAM SODA*, 1275–1284, 2007.
- [158] M. Singh and L. Lau. Approximating minimum bounded degree spanning trees to within one of optimal. *Proc. of ACM STOC*, 661–670, 2007.
- [159] A. Srinivasan. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. *Proc. of IEEE FOCS*, 416–425, 1997.
- [160] Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *Proc. of IEEE FOCS*, 697–706, 2008.
- [161] Z. Svitkina and E. Tardos. Min-max multiway cut. *Proc. of APPROX-RANDOM*, 207–218, 2004.
- [162] Z. Svitkina and E. Tardos. Facility location with hierarchical facility costs. *ACM Transactions on Algorithms* 6(2):1–22, 2010.
- [163] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonica* 24(6):573–577, 1980.
- [164] V. V. Vazirani. *Approximation algorithms*. Springer, 2004.
- [165] J. Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. *Proc. of ACM STOC*, 67–74, 2008.
- [166] J. Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing* 42(1):265–304, 2013.
- [167] D. B. West. *Introduction to graph theory*. Prentice Hall, 2001.
- [168] D. Williamson, M. Goemans, M. Mihail, and V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica* 15(3):435–454, 1995.
- [169] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.
- [170] S. Win. On a connection between the existence of k -trees and the toughness of a graph. *Graphs and Combinatorics* 5(1):201–205, 1989.
- [171] M. Xiao. Finding minimum 3-way cuts in hypergraphs. *Information Processing Letters* 110(14-15):554–558, 2010.
- [172] A. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica* 9(5):463–470, 1993.
- [173] L. Zhao, H. Nagamochi, and T. Ibaraki. Greedy splitting algorithms for approximating multiway partition problems. *Mathematical Programming* 102(1):167–183, 2005.