APPROXIMATION ALGORITHMS FOR NETWORK DESIGN AND ORIENTEERING

BY

NITISH JOHN KORULA

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, 2010

Urbana, Illinois

Doctoral Committee:

Associate Professor Chandra Chekuri, Chair
Associate Professor Jeff Erickson
Associate Professor Sariel Har-Peled
Associate Professor Anupam Gupta, Carnegie Mellon University

Abstract

This thesis presents approximation algorithms for some $\mathcal{NP}$-Hard combinatorial optimization problems on graphs and networks; in particular, we study problems related to Network Design. Under the widely-believed complexity-theoretic assumption that $\mathcal{P} \neq \mathcal{NP}$, there are no efficient (i.e., polynomial-time) algorithms that solve these problems exactly. Hence, if one desires efficient algorithms for such problems, it is necessary to consider *approximate* solutions: An approximation algorithm for an $\mathcal{NP}$-Hard problem is a polynomial time algorithm which, for any instance of the problem, finds a solution whose value is guaranteed to be within a multiplicative factor ρ of the value of an optimal solution to that instance. We attempt to design algorithms for which this factor ρ, referred to as the approximation ratio of the algorithm, is as small as possible.

The field of Network Design comprises a large class of problems that deal with constructing networks of low cost and/or high capacity, routing data through existing networks, and many related issues. In this thesis, we focus chiefly on designing *fault-tolerant* networks. Two vertices u, v in a network are said to be *k-edge-connected* if deleting any set of $k-1$ edges leaves u and v connected; similarly, they are *k-vertex connected* if deleting any set of $k-1$ other vertices or edges leaves u and v connected. We focus on building networks that are highly connected, meaning that even if a small number of edges and nodes fail, the remaining nodes will still be able to communicate. A brief description of some of our results is given below.

We study the problem of building 2-vertex-connected networks that are large and have low cost. Given an n-node graph with costs on its edges and any integer k, we give an $O(\log n \log k)$ approximation for the problem of finding a minimum-cost 2-vertex-connected subgraph containing at least k nodes. We also give an algorithm of similar approximation ratio for maximizing the number of nodes in a 2-vertex-connected subgraph subject to a budget constraint on the total

cost of its edges. Our algorithms are based on a pruning process that, given a 2-vertex-connected graph, finds a 2-vertex-connected subgraph of any desired size and of *density* comparable to the input graph, where the density of a graph is the ratio of its cost to the number of vertices it contains. This pruning algorithm is simple and efficient, and is likely to find additional applications.

Recent breakthroughs on vertex-connectivity have made use of algorithms for *element-connectivity* problems. We develop an algorithm that, given a graph with some vertices marked as terminals, significantly simplifies the graph while preserving the pairwise element-connectivity of all terminals; in fact, the resulting graph is bipartite. We believe that our simplification/reduction algorithm will be a useful tool in many settings. We illustrate its applicability by giving algorithms to find many trees that each span a given terminal set, while being disjoint on edges and non-terminal vertices; such problems have applications in VLSI design and other areas. We also use this reduction algorithm to analyze simple algorithms for single-sink network design problems with high vertex-connectivity requirements; we give an $O(k \log n)$-approximation for the problem of k-connecting a given set of terminals to a common sink. We study similar problems in which different types of links, of varying capacities and costs, can be used to connect nodes; assuming there are economies of scale, we give algorithms to construct low-cost networks with sufficient capacity or bandwidth to simultaneously support flow from each terminal to the common sink along many vertex-disjoint paths.

We further investigate capacitated network design, where edges may have arbitrary costs and capacities. Given a connectivity requirement R_{uv} for each pair of vertices u, v, the goal is to find a low-cost network which, for each uv, can support a flow of R_{uv} units of traffic between u and v. We study several special cases of this problem, giving both algorithmic and hardness results.

In addition to Network Design, we consider certain Traveling Salesperson-like problems, where the goal is to find short walks that visit many distinct vertices. We give a $(2+\varepsilon)$-approximation for Orienteering in undirected graphs, achieving the best known approximation ratio, and the first approximation algorithm for Orienteering in directed graphs. We also give improved algorithms for Orienteering with time windows, in which vertices must be visited between specified release times and deadlines, and other related problems. These problems are motivated by applications in the fields of vehicle routing, delivery and transportation of goods, and robot path planning.

To my parents, for more reasons than I can list.

Acknowledgments

This thesis would not have been possible without the support of many people. Most of all, I would like to thank my advisor, Chandra Chekuri; I was extremely fortunate that he joined the University of Illinois at the perfect time. Chandra's advice and encouragement have been invaluable to me over the last four years. The door to his office was always open to me, and I rarely left the office without feeling that I had learned something new or gained an interesting perspective. I am more grateful than I can express for all that I learned from him about research, teaching and communicating, and everything else.

Thanks to Jeff Erickson, Sariel Har-Peled and Manoj Prabhakaran for advice, support, and four years worth of jokes over lunch; I learned an incredible amount from their wonderful classes and seminars, and from any number of random conversations. Thanks in particular to Sariel for working with me when I was a beginning graduate student, and for having more confidence in my abilities than I did myself. Also thanks to Anupam Gupta, for his encouragement and perhaps the single most valuable comment at my preliminary proposal defense. Two other faculty members at UIUC I am particularly grateful to are Lenny Pitt and Doug West: I was lucky to be Lenny's Teaching Assistant for three semesters, and to have the privilege — and pleasure — of designing an introductory course with him. His passion for education is both contagious and inspiring. Doug has been a great mentor to me, and to almost all the graduate students he met; his comments and suggestions have made me a better writer and speaker. Doug always found time to discuss problems and conjectures related to my research, and invariably gave me good advice and pointers; his suggestions helped with algorithms in both Chapters 3 and 4 of this thesis.

I am most grateful to my sister Nisha, who taught me how to read and how to enjoy reading; she set the stage for what I hope will be a lifetime of learning. Thanks to my whole family for their love and support, and for encouraging me to pursue everything I was interested in.

I have to thank so many of my friends, at UIUC and elsewhere, and for so many reasons: Ditch and Beep, for being great neighbors and friends. Amit and Nitish, for three years of sharing an apartment, cooking, and weekly trips to Bombay Grill. Sanketh and Ruchika, for arguments, rants, quiet conversations and good friendship. Jayabrata and Komal, for laughter and good food (at least most of the time). Lachu, you kept me sane and happy through some difficult times; thanks for always being such a big part of my life. Madhu, for teaching me so much about myself and everything else, and for making me so happy; reading your old emails always makes me smile. Ranji, your remarkable gift for quiet good friendship made so much of a difference to my four years at BITS and all the time after.

Life and research at UIUC would have been much less interesting and enjoyable without the wonderful theory grad students, some of whom I was fortunate enough to collaborate with: Thanks to Erin, Dan Cranston, Kevin, and Mike for being good friends and great mentors. And thanks to Maji, Ben, Sungjin, Alina, Amir, Kyle, Dan Schreiber, and Nirman, for making the last few years as much fun as the first.

I would like to thank Prof. Sundar S Balasubramaniam of BITS, Pilani, for introducing me to the beautiful world of algorithms, and supervising my first research projects. I had two productive and fascinating internships with the Market Algorithms research group at Google; thanks to Martin, Jon, Vahab, Muthu, Cliff and Monika for the great experience. I am grateful to Nikhil Bansal and Viswanath Nagarajan for inviting me to visit IBM Research, and for the successful collaborations that followed; I also thank Sanjeev Khanna and Deeparnab Chakrabarty for the collaborations on Capacitated Network Design that led to Chapter 6 of this thesis. Finally, I am grateful for financial support from Chandra Chekuri's NSF Grant CCF 07-28782 and a Dissertation Completion Fellowship awarded by the University of Illinois Graduate College.

Table of Contents

List of Tables

List of Figures

List of Symbols

$G(V, E)$	A graph G, with vertex set V and edge set E.
n, m	Typically the number of vertices and edges in a graph, respectively. That is, given a graph $G(V, E)$, we use n to denote $\|V\|$ and m to denote $\|E\|$.
OPT	The value of an optimal solution to the problem instance being considered.
$\lambda_G(u, v)$	The edge-connectivity between vertices u and v in a given graph G.
$\kappa_G(u, v)$	The vertex-connectivity between vertices u and v in a given graph G.
$\kappa'_G(u, v)$	The element-connectivity between two *terminal* vertices u and v in a given graph G.

Chapter 1

Introduction

This thesis describes approximation algorithms for certain $\mathcal{NP}$-Hard combinatorial optimization problems on graphs and networks. In an optimization problem, the goal is to find a best (or *optimal*) solution from a set of feasible solutions to the problem; for instance, one may wish to find a solution of minimum cost or maximum profit. In discrete or combinatorial optimization problems, the set of potential solutions is finite; however, the set may be of exponential size or larger, and it is impractical and inefficient to examine each solution in turn in order to find the best. Thus, there is a need for efficient methods or algorithms to solve such problems.

An optimization problem is in the complexity class $\mathcal{P}$ (for polynomial time) if there is an algorithm to solve it (i.e., find an optimal solution) whose running time grows polynomially with the size of the input.[1] It is generally accepted that the class $\mathcal{P}$ corresponds to the set of problems that can be effectively solved in practice [70, 74]; among the great successes of theoretical computer science have been finding polynomial-time algorithms for problems such as solving linear programs, primality testing, and computing shortest paths, maximum matchings, and minimum-cost flows in graphs.

Many natural problems, however, are not known to be in $\mathcal{P}$; in this thesis, we consider problems in the class $\mathcal{NP}$, which contains $\mathcal{P}$. Informally, an optimization problem is in $\mathcal{NP}$ if there is a *non-deterministic* algorithm to solve it whose running time grows polynomially with the input size.[2] Some of the most difficult problems in $\mathcal{NP}$ arise in a variety of contexts and have numerous applications. It is widely believed that $\mathcal{P} \neq \mathcal{NP}$; if this is true, there are no efficient algorithms to find optimal solutions to such $\mathcal{NP}$-Hard problems. This gives rise to an extremely interesting set of

[1] Typically, the set of feasible solutions is represented *implicitly*; as the set of solutions may be exponentially large, an algorithm that examines each possible solution may not run in polynomial time. See Section 1.2.1 for a more formal definition of the complexity class $\mathcal{PO}$ of optimization problems.

[2] See Section 1.2.1 for a precise definition of $\mathcal{NP}$-Optimization problems.

questions: Given the fundamental nature and wide range of applications of these problems, solving them is of both significant theoretical and practical interest. However, assuming $\mathcal{P} \neq \mathcal{NP}$, there are no polynomial-time algorithms that are guaranteed to find optimal solutions to all instances of these problem.

To solve such problems, then, we must either use algorithms with super-polynomial running times, or relax the requirement that we always find optimal solutions. Both approaches have been studied extensively and have had numerous successes. For some problems, algorithms that have exponential running time can be used to solve instances that are not too large. More often, though, there is a need for efficient algorithms, even if they are not guaranteed to find optimal solutions. In practice, heuristic algorithms are frequently used; these algorithms commonly have low running times, and perform well on many inputs/instances. They may not be entirely satisfactory from a theoretical perspective, though, as there are rarely formal guarantees on their performance, and there may be instances on which their performance is very poor. In this thesis, we design *approximation algorithms*, which are polynomial-time algorithms guaranteed to return solutions that are "close" to optimal; we elaborate on this in Section 1.2.

In Section 1.1 below, we describe some of the motivating applications and background for problems considered in this thesis, and in Section 1.2, we review some basic definitions related to approximation algorithms. Finally, we describe the contributions and organization of this thesis in Section 1.3.

1.1 Connectivity and Network Design

This thesis focuses on Network Design and related problems in graphs.[3] Graphs naturally model a large number of systems/environments, with weight functions on edges encoding distances, costs, capacities, or other relevant parameters. In particular, for Network Design problems, vertices often represent nodes in a computer or communications network, with edges representing existing or potential links between these nodes. A common goal of problems in this class is to find a small/low-cost set of edges that satisfies a certain connectivity or other structural requirement. For example,

[3]See [153] for a summary of basic graph theory and associated terminology. We consider problems on both directed and undirected graphs, though the latter are our primary focus in this thesis.

the well-known MINIMUM SPANNING TREE problem asks one to find a minimum-cost set of edges that connects all vertices of a given graph. More generally, Network Design encompasses a large class of problems that deal with building low-cost networks, routing data or traffic through existing networks, and other related questions.

Connectivity and Network Design problems play an important role in combinatorial optimization and algorithms both for their theoretical appeal and their usefulness in real-world applications. Many of these problems, such as the well-known STEINER TREE problem, are $\mathcal{NP}$-hard and there has been a large and rich literature on approximation algorithms to solve them. A number of elegant and powerful techniques and results have been developed over the years (see [73, 152]). In particular, the primal-dual method [3, 92] and iterated rounding [109] have led to some remarkable results.

In most of the Network Design problems considered in this thesis, the goal is to construct networks that are fault-tolerant, or have some redundancy. Both edges and nodes of a network can fail, and this is not uncommon; on several occasions in recent years, accidental cuts of undersea cables due to earthquakes and damage from ships led to significant loss of Internet connectivity in the Middle East, India, and East Asia. Power outages are another common cause of the failure of network components. More mundane considerations, such as high latency due to congestion, may also make an edge or node effectively unusable for a short time. Thus, it is desirable to build networks which allow users to communicate even in the presence of edge or vertex failures. This goal can sometimes be achieved by duplicating large parts of the network, but such a solution may incur significant expense; instead, we give efficient algorithms to design low-cost networks with the desired robustness.

Two vertices of a graph are said to be *k-edge-connected* if the failure of any $k-1$ *edges* leaves them still connected, and *k-vertex-connected* if the failure of any $k-1$ *other vertices or edges* leaves them connected.[4] Note that if two vertices are k-vertex-connected, they must also be k-edge-connected. It is easier to design networks that tolerate only edge-failures; there has been much work [90, 91, 154, 109] on such problems, which are now well understood. In this thesis, we explore two directions where fewer results were previously known: First, we construct networks

[4] As is standard in the literature, we use k-connected to mean k-vertex-connected. Where there is any possible ambiguity, however, we explicitly use k-vertex-connected.

resilient to both edge and vertex failures, and second, we consider problems where edges may have widely differing *capacities*, measuring (for instance) the amount of network traffic the edges can support. We discuss these directions in turn below.

Problems requiring the construction of graphs with high vertex-connectivity (that is, resilient to vertex failures) are typically more difficult than their edge-connectivity counterparts, as vertex-connectivity exhibits less structure than edge-connectivity. Algorithmic techniques that are successful for edge-connectivity problems often do not apply to their vertex-connectivity variants, and until recently, few results were known for vertex-connectivity problems. To help bridge the gap between edge- and vertex-connectivity, Jain *et al.* [111] introduced the intermediate notion of *element connectivity*. Given a graph $G(V, E)$ with the vertex set partitioned into a set of terminals $T \subseteq V$ and non-terminals $V \setminus T$, the *element-connectivity* between a pair of terminals u, v is defined to be the minimum number of edges or *non-terminal* vertices that must be deleted to separate u from v. (An *element* is an edge or a non-terminal vertex. Thus, the element-connectivity between terminals u and v could be equivalently defined as the maximum number of element-disjoint paths between u and v.) That is, to determine the edge-connectivity of a pair of terminals, one is only allowed to delete edges, and to determine the vertex connectivity, one can delete edges or vertices. To determine the intermediate element-connectivity, one can delete edges or non-terminal vertices. Following the work of Chuzhoy and Khanna [68, 69], and the concurrent work described in Chapters 4 and 5 of this thesis, it was realized that understanding element-connectivity was extremely useful in solving vertex-connectivity problems; this has led to many new algorithms for constructing networks with high vertex-connectivity.

In the construction of real networks, one may often have available equipment with different discrete *capacities*. In addition to connecting nodes, one may wish to allow a large amount of traffic between them. Thus, when building a link between two nodes, one must decide how much bandwidth it should provide; high-capacity optical fiber links are more expensive than low-capacity cables, but there are often economies of scale. Network designers typically wish to build low-cost networks that can route the desired amount of traffic between various pairs of nodes. There are two classes of capacitated network design problems commonly studied in the literature: In both cases, it is assumed that for each pair of nodes u, v, a traffic requirement R_{uv} is specified. In the first class

of *multi-commodity flow*-type problems, the goal is to create a network that can *simultaneously* send R_{uv} units of traffic between every pair of nodes u, v. In the second class of problems, related to the connectivity questions described above, the goal is to construct a network such that, for any pair of vertices u, v, the network can send R_{uv} units of traffic between u and v. We extend our results of Chapter 5 to provide the first approximation algorithms for certain fault-tolerant multi-commodity flow problems, assuming the cost of constructing high-capacity links exhibits economies of scale[5]. In Chapter 6, we consider the second class of problems, with no assumptions on costs and capacities. We give both algorithms and hardness results for various special cases, yielding new insights into the approximability of these problems.

Besides the Network Design problems discussed above, we consider certain path planning/vehicle routing problems related to the well-known Traveling Salesperson Problem (TSP). Here, the vertices of a graph represent locations in a (usually metric) space, with the length of an edge representing the distance between its endpoints, or the time it takes to travel between them; we typically wish to find a short path/tour for a vehicle that visits many locations. Though the motivating applications are different from those of the Network Design problems we consider, there are several underlying similarities between the problems: Finding a low-cost set of edges that connects many vertices and forms a path resembles finding a low-cost network connecting many vertices, and there are many algorithmic techniques and ideas common to both sets of problems. We discuss this connection futher in Chapter 3.

1.2 Preliminaries: Approximation Algorithms

An approximation algorithm for an optimization problem may not return an optimal solution, but is guaranteed to return a feasible solution that is near-optimal; we measure the quality of an approximation algorithm $\mathcal{A}$ by the (worst-case) ratio between the value of an optimal solution and that of the solution returned by $\mathcal{A}$.[6] For a minimization problem, we say that algorithm $\mathcal{A}$ is a ρ-approximation algorithm if, for any instance of the problem, the value of the solution

[5] More precisely, assuming the cost vs. capacity function is concave/sub-additive.

[6] For all problems considered in this thesis, the objective value of any feasible solution is non-negative; hence, this ratio is always positive.

returned by $\mathcal{A}$ is at most ρ times the value of an optimal solution to that instance. Similarly, for a maximization problem, algorithm $\mathcal{A}$ is said to be a ρ-approximation algorithm if, for any instance, $\mathcal{A}$ returns a solution of value at least $1/\alpha$ times that of an optimal solution. A ρ-approximation algorithm is said to have *approximation ratio* ρ. Note that an optimal algorithm for a problem has approximation ratio equal to 1; any algorithm that does not always return optimal solutions has approximation ratio greater than 1. See Section 1.2.1 for more precise definitions of optimization problems, approximation ratios, etc. Throughout this thesis, we use OPT to denote the value of an optimal solution to the given problem instance.

Some optimization problems, such as KNAPSACK, have Fully Polynomial Time Approximation Schemes (polynomial-time algorithms with approximation ratio $1 + \varepsilon$, for any $\varepsilon > 0$); by contrast, unless $\mathcal{P} = \mathcal{NP}$, the MAX-CLIQUE problem cannot be approximated within a ratio of $|V|^{1-\varepsilon}$, for any $\varepsilon > 0$. In between these two extremes – the one problem very easy to solve, both theoretically and practically, and the other essentially inapproximable – lies a vast landscape of optimization problems. The problems we study in this thesis similarly exhibit varying degrees of approximability: some admit algorithms of small constant approximation ratios, while others are inapproximable to within poly-logarithmic or higher factors. A rich and extensive mathematical theory has been developed to understand and classify such optimization problems, to devise approximation algorithms and prove intractability. For an overview of the field of approximation algorithms, see the recent books [152, 73, 18].

In addition to their theoretical interest, approximation algorithms have immense practical applicability. Though a constant-factor (or logarithmic, or even worse) approximation ratio may seem of limited use, this ratio is only a worst-case guarantee. Often, we merely prove weak upper bounds on an approximation ratio, and the actual performance of the algorithm may be considerably better than this ratio. Even when the bound is tight, this may be due to contrived or pathological examples that are unlikely to arise in real applications. In practice, these approximation algorithms may produce solutions within a few percentage points of the optimal solution. Perhaps more importantly, though, approximation algorithms frequently provide significant insight into the combinatorial structure of a problem, or a class of problems; this insight helps one design algorithms and heuristics tuned to specific applications.

1.2.1 Optimization Problems and Approximation Algorithms

An optimization problem Π is formally defined by a quadruple $(\mathcal{I}_\Pi, \mathcal{S}_\Pi, m_\Pi, \text{goal}_\Pi)$ such that:

- $\mathcal{I}_\Pi$ is the (usually infinite) set of *instances* of problem Π.
- $\mathcal{S}_\Pi$ is a function that, for each instance $I \in \mathcal{I}_\Pi$, defines a set of *feasible solutions* $\mathcal{S}_\Pi(I)$ for I.
- m_Π is an objective/measure function that for each instance $I \in \mathcal{I}_\Pi$ and for each feasible solution $S \in \mathcal{S}_\Pi(I)$, defines the (non-negative) value $m_\Pi(I, S)$ of this solution.
- goal_Π is either min or max, specifying whether the problem Π is a minimization or a maximization problem.

An optimization problem Π is said to be an $\mathcal{NPO}$ (for $\mathcal{NP}$ Optimization) problem if it satisfies the following conditions:

- For any instance I and any feasible solution $S \in \mathcal{S}_\Pi(I)$, the solution S is polynomially bounded in the size of I. That is, there exists a polynomial p such that $|S| \leq p(|I|)$. (Here, $|I|, |S|$ denote the lengths of descriptions of I and S respectively.)
- There is a polynomial-time algorithm to determine if $S \in \mathcal{S}_\Pi(I)$. That is, there exists a polynomial-time computable boolean function f such that $f(I, S)$ is true if $S \in \mathcal{S}_\Pi(I)$ and false otherwise.
- The measure function m_Π is computable in polynomial time.

An algorithm to (exactly) solve a minimization (respectively, maximization) problem Π is one that, for any instance $I \in \mathcal{I}_\Pi$, returns a solution $S \in \mathcal{S}_\Pi(I)$ minimizing (respectively, maximizing) the value $m_\Pi(I, S)$. (That is, for any $S' \in \mathcal{S}_\Pi(I)$, we have $m_\Pi(I, S) \leq m_\Pi(I, S')$ for a minimization problem, and $m_\Pi(I, S) \geq m_\Pi(I, S')$ for a maximization problem.) We use $\text{OPT}(I)$ to denote such an optimal solution S. An algorithm $\mathcal{A}$ is said to be a polynomial-time algorithm for a problem Π if there exists a constant $c > 0$ such that, for any instance $I \in \mathcal{I}_\Pi$, $\mathcal{A}$ runs in time $O(|I|^c)$, where $|I|$ denotes the size of the representation of I.

A problem $\Pi \in \mathcal{NPO}$ is in the class $\mathcal{PO}$ if there is a polynomial-time algorithm to solve it exactly.

For any optimization problem Π, an algorithm $\mathcal{A}$ is a polynomial-time approximation algorithm for Π if it is a polynomial-time algorithm that, for any instance $I \in \mathcal{I}_\Pi$, returns a solution $S \in \mathcal{S}_\Pi(I)$ which is "close" to the optimal solution $\mathrm{OPT}(I)$ in $\mathcal{S}_\Pi(I)$. We say that an algorithm $\mathcal{A}$ for a minimization problem Π has approximation ratio (at most) ρ if, for any instance $I \in \mathcal{I}_\Pi$, $\mathcal{A}$ returns a solution $S \in \mathcal{S}_\Pi(I)$ such that for any other feasible solution S', we have $m_\Pi(I, S) \leq \rho \cdot m_\Pi(I, S')$. Equivalently, A returns a solution S such that $m_\Pi(I, S) \leq \rho \cdot m_\Pi(\mathrm{OPT}(I)$. Similarly, algorithm $\mathcal{A}$ for a maximization problem Π has approximation ratio (at most) ρ if, for any instance $I \in \mathcal{I}_\Pi$, $\mathcal{A}$ returns a solution $S \in \mathcal{S}_\Pi(I)$ such that for any other feasible solution S', we have $m_\Pi(I, S) \geq \frac{1}{\rho} \cdot m_\Pi(I, S')$. An algorithm with approximation ratio ρ is said to be a ρ-approximation algorithm.

Hardness of Approximation

Informally, an $\mathcal{NPO}$ problem Π is said to be ρ-hard to approximate if, under a suitable complexity-theoretic assumption (typically, assuming $\mathcal{P} \neq \mathcal{NP}$), there is no polynomial-time approximation algorithm with approximation ratio at most ρ. A seminal such result is that of Arora *et al.* [16], which proved constant-factor hardness of approximation for the Max-3-SAT problem. In particular, [16] showed that it is $\mathcal{NP}$-Hard to distinguish between instances of Max-3-SAT for which there exists an assignment satisfying all clauses, and instances for which no assignment satisfies more then δ fraction of the clauses, for some constant $\delta < 1$. Many remarkable results on hardness of approximation followed this line of work.

Typically, a problem is shown to be hard to approximate in one of two ways, briefly described below.

1. Using a reduction from an $\mathcal{NP}$-Complete problem: For a minimization problem Π, suppose there exists an $\mathcal{NP}$-Complete decision problem $\mathcal{D}$, a polynomial-time computable function f mapping instances of $\mathcal{D}$ to instances of Π, and two constants c_1 and $c_2 > c_1$ such that the following conditions hold.

    - The function f maps any YES instances of $\mathcal{D}$ to an instance of Π with optimal value at most c_1.

- The function f maps NO instances of $\mathcal{D}$ to instances of Π with optimal value at least c_2.

Then, it follows that unless $\mathcal{P} = \mathcal{NP}$, there is no approximation algorithm for Π with approximation ratio better than c_2/c_1. (If there were such an algorithm for Π, one could use it with the function f to obtain a polyomial-time algorithm to solve the $\mathcal{NP}$-Complete problem $\mathcal{D}$.) Note that if c_1 and c_2 are functions of the input size instead of constants, one can obtain a non-constant hardness of approximation for Π.

2. Using an *approximation-preserving* reduction from an $\mathcal{NPO}$ problem that is known to be hard to approximate: For example, suppose there exist two $\mathcal{NPO}$ minimization problems Π and Π', polynomial-time computable functions f and g, and two constants c_1, c_2 such that the following conditions hold.

   - The function f maps each instance $I \in \mathcal{I}_\Pi$ to an instance $I' \in \mathcal{I}_{\Pi'}$, and for each instance $I \in \mathcal{I}_\Pi$, $\mathrm{OPT}(f(I)) \leq c_1 \mathrm{OPT}(I)$.
   - For each instance $I' \in \mathcal{I}_{\Pi'}$ such that $I' = f(I)$ for some $I \in \mathcal{I}_\Pi$, the function g maps each feasible solution in $\mathcal{S}_{\Pi'}(I')$ to a feasible solution in $\mathcal{S}_{\Pi(I)}$. Further, for each solution $S' \in \mathcal{S}_{\Pi'}(I')$, we have $m_\Pi(g(S')) - m_\Pi(\mathrm{OPT}(I)) \leq c_2 \left[m_{\Pi'}(S') - m_{\Pi'}(\mathrm{OPT}(I'))\right]$.

   Then, it follows that if there is no polynomial-time ρ-approximation algorithm for Π, there is no polynomial-time $\left(1 + \frac{\rho-1}{c_1 \cdot c_2}\right)$-approximation algorithm for Π'. (If there were such an algorithm for Π', one could use it with the functions f and g to obtain a polynomial-time ρ-approximation for Π.) A reduction as described above is known as an L-reduction; one can use this or other approximation-preserving reductions for both minimization and maximization problems to prove a desired hardness of approximation result.

See [12, 103, 18] for definitions, several inapproximability results and discussion of related work.

1.3 Thesis Contributions and Organization

As described above, this thesis focuses on Network Design problems and on certain related TSP-like vehicle routing problems. In particular, we deal with constructing fault-tolerant networks; we

also consider issues pertinent to practical networks such as budget constraints, links of differing capacities, etc.

In Chapter 2, we consider ORIENTEERING and related problems, where the goal is to find short paths that visit many locations in a given metric space; there are obvious applications to various vehicle routing and delivery problems, robot path planning, etc. In Chapter 3, we begin our study of fault-tolerant network design, giving algorithms to design large, low-cost, 2-connected graphs (that is, graphs that allow communication even after the failure of a single edge or vertex). Going beyond 2-connectivity (that is, allowing more failures) requires several new technical ideas; among these is the concept of *element-connectivity* [79].[7] In Chapter 4, we give an algorithm to drastically simplify graphs while preserving the element-connectivity between pairs of terminal vertices. We demonstrate the usefulness of this algorithm by showing how to find many element-disjoint trees in a graph; each such tree can route traffic between its terminal vertices, and hence even if some trees fail, the terminals can communicate through the remaining trees. In Chapter 5, we give a simple and efficient algorithm to k-vertex-connect terminals to a common root vertex, meaning that the deletion of any $k-1$ vertices or edges still leaves the remaining terminals connected to the root. We also describe how the algorithm can be generalized to construct a low-cost network that can simultaneously support a fault-tolerant flow of traffic from each terminal to the root. Finally, in Chapter 6, we consider more general Capacitated Network Design problems.

Though there are several ideas and techniques common to many of the problems we consider, the chapters are largely self-contained, and so can, for the most part, be read in any order. The reader may wish to read the Reduction Lemma of Chapter 4 (in particular, Sections 4.1 and 4.2) before Chapter 5 on vertex-connectivity. Certain technical sections which may be skipped on first reading are indicated as such in the text.

1.3.1 ORIENTEERING and Related Problems

In the ORIENTEERING problem, defined by [93], the input is an edge-weighted (directed or undirected) graph $G(V, E)$, with given start and end vertices $s, t \in V$, and a non-negative time limit B. The weight on an edge represents its length, or the time taken to travel between its endpoints.

[7] See Section 1.1 or Section 1.3.3 for a definition of element-connectivity.

The goal is to find an $s-t$ walk of total length at most B that maximizes the number of distinct vertices visited by the walk. We also study the related k-Stroll problem, in which the input is similar, but we are given an integer k instead of the time limit B; the goal in this problem is to find the shortest $s-t$ walk that visits at least k distinct vertices. These problems are closely related to each other (the constraint and objective are interchanged), and related to other well known problems such as the Traveling Salesperson Problem (TSP), which asks for the shortest tour that visits all vertices and returns to the start vertex. In fact, TSP is the special case of k-Stroll when $k = |V|$ and $s = t$. TSP, Orienteering, k-Stroll and related problems have a large number of applications related to vehicle routing, transportation and distribution of goods, etc.; see [150] for a detailed discussion of vehicle routing and applications. Other motivations come from robot path planning, or the scheduling of jobs performed at different locations. Given the numerous applications for the natural problems of Orienteering and k-Stroll, they have been studied extensively [93, 11, 60, 31, 25, 88, 42, 57, 135].

In Chapter 2, we describe approximation algorithms for Orienteering in both directed and undirected graphs. For undirected graphs, we give a $(2+\varepsilon)$-approximation algorithm; this is the best approximation ratio currently known. In directed graphs, the problem is significantly harder; we gave an $O(\log^2 \mathrm{OPT})$-approximation, where OPT denotes the value of an optimal solution. This was the first non-trivial approximation algorithm for Orienteering in directed graphs. Our algorithmic techniques also apply to k-Stroll and other related problems, which we discuss briefly in Chapter 2.

In addition to the basic Orienteering problem, we consider the more general problem of Orienteering with Time Windows (Orient-TW). In this problem, one is additionally given a time window $[R(v), D(v)]$ for each vertex v; as before, one has to find an $s-t$ walk of length at most the given time limit B, but now the goal is to maximize the number of vertices visited within their time windows. This problem also has several applications in the field of vehicle routing, particularly related to the delivery of goods and scheduling of work. Various special cases of Orient-TW have been studied [151, 26, 25, 56, 57, 85], and it was known that in both directed and undirected graphs, an α-approximation for the basic Orienteering problem yields an $O(\alpha \log^2 \mathrm{OPT})$-approximation for Orient-TW [25]. It is natural to conjecture that an $O(\log \mathrm{OPT})$-approximation is possible, as

this can be achieved in quasi-polynomial time. We make progress towards resolving this conjecture by showing that there is an $O(\alpha \max\{\log \mathrm{OPT}, \log L\})$-approximation, where L denotes the ratio between the lengths of the longest and shortest time windows.

1.3.2 Constructing and Pruning 2-Connected Graphs

In the k-MST problem, one is given an edge-weighted graph G and an integer k, abd the goal is to find a minimum-cost connected subgraph of G that contains at least k vertices. (Without loss of generality, this subgraph is a tree; hence the name k-MST.) There is an approximation-preserving reduction from the well-known Steiner Tree problem to k-MST. In another closely related problem, referred to as Max-Prize Tree, one is given the edge-weighted graph and a budget B; the goal is to find a connected subgraph of cost at most B that contains as many vertices as possible. Problems such as k-MST and Max-Prize Tree that are related to finding large, cheap, connected graphs arise naturally in several applications. Algorithms for these problems also find many other uses, such as in the algorithms for Orienteering and k-Stroll described in Chapter 2 and [42, 31]. Hence, there has been a long sequence of results on k-MST, Max-Prize Tree and applications [20, 32, 87, 15, 88, 42, 112, 31].

Algorithms for k-MST and Max-Prize Tree are useful in Network Design applications where one may want to build low-cost networks that provide connectivity to many clients, but there are constraints such as a budget on the network cost, or a minimum quota on the number of clients. However, networks with a tree structure are very vulnerable to failure; the loss of *any* single edge will break the network into disconnected pieces that cannot communicate. Thus in Chapter 3, we consider the natural generalization of k-MST to higher connectivity: In the k-2VC problem, we are given an edge-weighted graph G and an integer k; the goal is to find a minimum cost k-vertex subgraph of G that is 2-vertex-connected. We give the first approximation algorithm for the k-2VC problem in Chapter 3, an $O(\log n \log k)$-approximation. We also consider the Budget-2VC problem, in which we are given an edge-weighted graph G and a budget B; the goal is to find a 2-vertex-connected subgraph H of G with total edge cost at most B that maximizes the number of vertices in H. We describe a bi-criteria approximation for Budget-2VC that gives an $O(\frac{1}{\varepsilon} \log^2 n)$ approximation, while violating the budget by a factor of at most $3 + \varepsilon$.

Our algorithms for k-2VC and BUDGET-2VC both use the main technical tool of Chapter 3, an algorithm to *prune* 2-connected graphs while (approximately) preserving their *density*, defined as the ratio of the cost of a subgraph to the number of vertices it contains. Roughly speaking, given a 2-connected graph H, this algorithm finds a 2-connected subgraph of any desired size, at the cost of only a logarithmic increase in density. This algorithm and other technical results on 2-connected graphs we prove in Chapter 3 are independently interesting, and likely to find other applications beyond k-2VC and BUDGET-2VC.

1.3.3 A Graph Reduction Step Preserving Element-Connectivity

In a number of graph problems, one focuses on a specified subset of the vertices, often called *terminals*. For instance, in the well-known STEINER TREE problem, the input is an edge-weighted graph $G(V, E)$, together with a set of terminals $T \subseteq V$; the goal is to find a minimum-cost tree that connects all the terminals. Such a tree (referred to as a *Steiner* tree) may use some non-terminal vertices, but these vertices are not required to be in the tree. In Network Design applications, some nodes (such as communication hubs, military command centers, or offices of government and emergency service departments) may be more important than others; hence, one may wish to construct networks in which a given set of terminals are highly connected, but non-terminals are not. Thus, if a few network components fail, non-terminals may lose connectivity, but the important terminal vertices will remain able to communicate.

Recall the definition of element-connectivity from Section 1.1: Given a graph $G(V, E)$ with terminal set $T \subseteq V$ and non-terminals $V \setminus T$, the element-connectivity between terminals u, v is the minimum number of edges or non-terminal vertices that must be deleted to separate u from v. Thus, in a graph with high element-connectivity, the terminals can communicate even if edges or non-terminals fail. Element-connectivity problems are of interest both because they are natural, and because algorithms for them (such as in [79]) are useful building blocks: Chuzhoy and Khanna [68, 69] recently gave the first non-trivial algorithms for natural vertex-connectivity problems by reducing them to element-connectivity problems.

In Chapter 4, we give an algorithm for simplifying (also called reducing) graphs while preserving the pairwise element-connectivity of *all* terminals. Repeated applications of this simplification

step yield a bipartite graph, with partite sets T and $V \setminus T$. As bipartite graphs have highly restricted structure, it is often easy to design algorithms or prove theorems for this setting; the reduction step implies that such results typically extend easily to general graphs. Thus, we obtain a general template for problems such as finding element-disjoint structures in arbitrary graphs: Begin by applying the simplification step repeatedly, and obtain a bipartite graph. Find the desired structures in this bipartite graph, and this automatically yields the corresponding structures in the original graph.

A similar reduction step for preserving edge-connectivity in graphs due to Mader has seen a very large number of applications; see [129, 81, 123, 110, 59, 125, 124, 114] for a list of pointers. We believe that our reduction step preserving element-connectivity, which has already found applications in [63, 114], will also find many uses. In Chapter 4, we show an application to packing element-disjoint Steiner trees and forests. We also use this reduction step to give an extremely short and simple proof of the main result of Chapter 5; our previous proof, and that of a similar result in [68], both required several pages of technically involved work.

1.3.4 Single Sink Network Design with Vertex-Connectivity Requirements

In the Survivable Network Design Problem, the input is an undirected graph $G(V, E)$ with edge costs, and an integer *connectivity requirement* R_{uv} for every pair of vertices u, v. The goal is to find a minimum-cost subgraph $H \subseteq G$ satisfying the connectivity requirements, meaning that there should be R_{uv} disjoint paths between u and v for each pair uv. In the EC-SNDP problem, the requirement is that the paths must be edge-disjoint, while in the harder VC-SNDP problem, the paths are required to be vertex-disjoint. Equivalently, a feasible solution H for EC-SNDP has the property that for any pair u, v, deleting any $R_{uv} - 1$ edges cannot separate u from v; similarly, a feasible solution for VC-SNDP has the property that deleting any $R_{uv} - 1$ vertices or edges cannot separate u from v.

SNDP captures a large number of natural and interesting problems: The special case when $R_{uv} = 1$ for all pairs uv is simply the well known Minimum Spanning Tree problem, that can be solved optimally in near-linear time. On the other hand, if there is a set of terminals T such that $R_{uv} = 1$ for any pair of terminals uv and 0 otherwise, we obtain the $\mathcal{APX}$-Hard Steiner

TREE problem. SNDP with connectivity requirements greater than 1 has obvious applications to designing robust networks, and this is our focus in Chapter 5.

As we have discussed already, edge-connectivity problems tend to be easier than their vertex-connectivity counterparts; the EC-SNDP problem has been well studied, with a series of papers [90, 91, 154, 109] culminating in the seminal 2-approximation of Jain [109]. Until recently, however, the only non-trivial algorithms known for the VC-SNDP problem were for the case when the highest connectivity requirement $\max_{uv}\{R_{uv}\}$ is at most 2. Thus, when studying higher connectivity requirements (to construct networks resilient to more than one failure), it is natural to focus first on special cases of the problem.

In Chapter 5, we study *Single Sink* vertex-connectivity problems. Our main focus is the SS-k-CONNECTIVITY problem: Here, instead of dealing with arbitrary pairwise connectivity requirements, we assume that there is a specified sink/root vertex r, and a given collection of terminals; the goal is to find a minimum-cost subgraph in which each terminal is k-vertex-connected to the sink. (Equivalently, this is a subgraph in which each terminal can send flow along k vertex-disjoint paths to the root.) This problem was first introduced by Chakraborty, Chuzhoy and Khanna [39], who gave an $O(k^{O(k^2)} \log^4 n)$-approximation for the problem, for any $k \geq 1$. We give an extremely simple and efficient $O(k \log n)$-approximation for the SS-k-CONNECTIVITY problem[8]; our proof is based on the graph reduction step described in Chapter 4. We also include an earlier direct proof of a slightly weaker approximation ratio for this algorithm; this proof is interesting in its own right, and forms the basis for some subsequent work.

Besides the basic SS-k-CONNECTIVITY problem, we consider some more general *capacitated* single-sink Network Design problems in Chapter 5. Here, potential edges one can select have discrete capacities and costs; high capacity edges cost more than low-capacity ones, though economies of scale apply. Now, the goal is to build a low-cost network with enough capacity to *simultaneously* support flow from each terminal to the root along k disjoint paths. Such problems, referred to by the names RENT-OR-BUY and BUY-AT-BULK, were previously studied only in the case when $k = 1$ [146, 7, 43, 9, 99]. We give a poly-logarithmic approximation for the single-sink BUY-AT-BULK problem when $k = 2$; we also show that an algorithm of Charikar and Karagiazova [40] proposed

[8]This follows a similar result due to Chuzhoy and Khanna [68]; see Chapter 5 for further discussion.

for the $k = 1$ case gives an $O(2^{O(\sqrt{\log n})})$-approximation for any fixed $k \geq 1$. These are the first approximation algorithms for Buy-at-Bulk and related problems with connectivity requirement greater than 1, and they have not yet been improved upon. For more precise details of the various different Buy-at-Bulk models and accurate statements of our results, we refer the reader to Chapter 5.

1.3.5 Capacitated Network Design

In the Capacitated version of the Survivable Network Design Problem, the input is a graph $G(V, E)$ with an integer capacity $u(e)$ and cost $c(e)$ for each edge $e \in E$, along with a connectivity requirement R_{uv} for each pair of vertices uv. The goal is to find a minimum-cost subgraph $H \subseteq G$ in which, for each pair uv, R_{uv} units of flow can be sent between u and v. (Equivalently, the capacity of a minimum cut in H between u and v should be at least R_{uv}. Note that the subgraph H does not need to simultaneously support traffic between every pair of nodes, unlike the requirements for the Rent-or-Buy and Buy-at-Bulk problems described above.) It is easy to see that the special case of Capacitated-SNDP in which all edges have the same capacity is equivalent to the basic Survivable Network Design Problem described in Section 1.3.4. However, Capacitated-SNDP more accurately captures design problems that arise in constructing real fault-tolerant networks, as it is fairly common to have equipment with different discrete capacities.

Though Jain [109] gave a 2-approximation for EC-SNDP, the capacitated version is much harder to approximate. The approximation algorithms literature on this problem has been very limited [91, 37]; even the best algorithms previously known have very weak approximation ratios in general. In Chapter 6, we give new results for several special cases of the problem, shedding light on its approximability. We show that when all requirements R_{uv} are uniform, or nearly uniform, there is an $O(\log n)$-approximation; we obtain this bound by rounding a strengthened LP for the problem. We also consider the problem when one can buy multiple *copies* of edges: That is, for any integer $k > 0$, one can pay $k \cdot c(e)$ for edge e and obtain capacity $k \cdot u(e)$ between its endpoints. We show an $O(\log n)$-approximation in this setting, and prove that the problem is $\Omega(\log \log n)$-hard to approximate even in the single-sink case.

Without the ability to buy copies of edges, however, Capacitated-SNDP appears to be very

difficult to approximate, and we prove several hardness results. In fact, even in the case when just a single pair uv has requirement $R_{uv} > 0$, it appears hard to obtain good approximations. This is in stark contrast to the unit-capacity case, where the single-pair problem can be solved optimally even in *directed* graphs via minimum-cost flow algorithms in polynomial time. We prove that the capacitated version of this single-pair problem is essentially inapproximable in directed graphs; there is no $2^{\log^{(1-\delta)} n}$-approximation for any $\delta > 0$ unless $\mathcal{NP} \subseteq DTIME(n^{\texttt{polylog}(n)})$. For undirected graphs, we show that even the strengthened LP referred to earlier has integrality gap $\Omega(n)$ for the single-pair problem; we also show that the single-pair problem is $\Omega(\log \log n)$-hard to approximate.

Chapter 2

The Orienteering Problem

2.1 Introduction [1]

The Traveling Salesperson Problem (TSP) and its variants have been an important driving force for the development of new algorithmic and optimization techniques. This is due to several reasons. First, the problems have many practical applications. Second, they are often simple to state and intuitively appealing. Third, for historical reasons, TSP has been a focus for trying new ideas. See [128, 100] for detailed discussion on various aspects of TSP. In this chapter, we consider some TSP variants in which the goal is to find a tour or a walk that maximizes the number of nodes visited, subject to a strict time limit (also called budget) requirement. The main problem of interest is the ORIENTEERING problem, introduced by [93][2], which we define formally below. The input to the problem consists of an edge-weighted graph $G = (V, E)$ (directed or undirected), two vertices $s, t \in V$ and a non-negative time limit B. The goal is to find an s-t walk of total length at most B so as to maximize the number of *distinct* vertices visited by the walk. Note that a vertex may be visited multiple times by the walk, but is only counted once in the objective function. (Alternatively, we could work with the metric completion of the given graph.) One could consider weighted versions of ORIENTEERING, where the goal is to maximize the sum of the weights of visited vertices; using standard scaling techniques (see Section 2.2), one can reduce the weighted version to the unweighted problem at the loss of a factor of $(1 + o(1))$ in the approximation ratio. Hence, we focus on the unweighted version throughout this chapter. We use OPT to denote the number of distinct vertices visited by an optimal solution; OPT can be as large as n, the number

[1] This chapter is based on joint work with Chandra Chekuri and Martin Pál, and has appeared in [55, 51]. Copyrights to the conference and journal versions of [55] are held by the Society for Industrial and Applied Mathematics (SIAM) and the Association for Computing Machinery (ACM) respectively.

[2] The problems we describe are referred to by several different names in the literature, one of which is prize-collecting TSP.

of vertices in the graph, but may be much smaller.

We also study a more general problem, referred to as ORIENTEERING with *time-windows*. In this problem, we are additionally given a time-window (or interval) $[R(v), D(v)]$ for each vertex v. A vertex is counted as visited only if the walk visits v at some time $t \in [R(v), D(v)]$. (If a vertex v is reached before $R(v)$, we may choose to "wait" at v until $R(v)$, so the walk can obtain credit for v, and then resume the walk. The time spent "waiting" is included in the length of the walk.) For ease of notation, we use ORIENT-TW to refer to the problem of ORIENTEERING with time-windows. A problem of intermediate complexity is the one in which $R(v) = 0$ for all v. We refer to this problem as ORIENTEERING with deadlines (ORIENT-DEADLINE); it has also been called the Deadline-TSP problem by [25]. The problem where vertices have release times but not deadlines (that is, $D(v) = \infty$ for all v) is equivalent to ORIENT-DEADLINE.[3]

One of the main motivations for budgeted/time-limited TSP problems comes from real world applications under the umbrella of vehicle routing; a large amount of literature on this topic can be found in operations research. Problems in this area arise in transportation, distribution of goods, scheduling of work, etc.; the book [150] discusses various aspects of vehicle routing. Another motivation for these problems comes from robot motion planning where typically, the planning problem is modeled as a Markov decision process. However there are situations where this does not capture the desired behaviour and it is more appropriate to consider ORIENTEERING-type objective functions in which the reward at a site expires after the first visit; see [31], which discussed this issue and introduced the discounted-reward TSP problem. In addition to the practical motivation, budgeted TSP problems are of theoretical interest.

ORIENTEERING is NP-hard via a straightforward reduction from TSP and we focus on approximation algorithms; it is also known to be APX-hard to approximate [31]. The first non-trivial approximation algorithm for ORIENTEERING was due to Arkin, Mitchell and Narasimhan [11], who gave a $(2 + \varepsilon)$ approximation for points in the Euclidean plane. For ORIENTEERING in arbitrary metric spaces (this is equivalent to ORIENTEERING in undirected graphs), Blum *et al.* [31] gave the first approximation algorithm with a ratio of 4; this was shortly improved to a ratio of 3 by

[3]To see that these problems are equivalent, note that an $s - t$ walk of length at most B that visits vertex v in the time window $[R(v), \infty]$ for vertex v is the reversal of a $t - s$ walk of length at most B that visits v in $[0, B - R(v)]$, and vice versa. See [25] for the formal reduction between the two problems.

Bansal *et al.* [25]. Subsequently, Chen and Har-Peled [60] obtained a PTAS for Orienteering in fixed-dimensional Euclidean space. The basic insights for approximating Orienteering were obtained by Blum *et al.* in [31], where a related problem called the Minimum-Excess problem was defined. It was shown in [31] that an approximation for the Min-Excess problem implies an approximation for Orienteering. Further, the Min-Excess problem can be approximated using algorithms for the k-Stroll problem. In the k-Stroll problem, the goal is to find a minimum length walk from s to t that visits at least k vertices. Note that the k-Stroll problem and the Orienteering problem are equivalent in terms of exact solvability but an approximation for one does not immediately imply an approximation for the other. Still, the clever reduction of [31] (via the intermediate Min-Excess problem) shows that an approximation algorithm for k-Stroll implies a corresponding approximation algorithm (losing a small constant factor in the approximation ratio) for Orienteering. The results in [31, 25] are based on existing approximation algorithms for k-Stroll [88, 42] in undirected graphs. In directed graphs, no non-trivial algorithm was known for the k-Stroll problem[4] and the best previously known approximation ratio for Orienteering was $O(\sqrt{\mathrm{OPT}})$. A different approach was taken for the directed Orienteering problem by Chekuri and Pál [57]; the authors use a recursive greedy algorithm to obtain a $O(\log \mathrm{OPT})$ approximation for Orienteering and for several generalizations, but unfortunately the running time is *quasi-polynomial* in the input size.

In this chapter, we obtain improved algorithms for Orienteering and related problems in both undirected and directed graphs. Our main results are encapsulated by the following theorems.

Theorem 2.1. *For any fixed $\varepsilon > 0$, there is an algorithm with running time $n^{O(1/\varepsilon^2)}$ achieving a $(2+\varepsilon)$-approximation for* Orienteering *in undirected graphs.*

Theorem 2.2. *There is an $O(\log^2 \mathrm{OPT})$-approximation for* Orienteering *in directed graphs.*[5]

Orienteering with Time Windows: Orient-Deadline and Orient-TW are more difficult problems than Orienteering; in fact Orient-TW is NP-hard even on the line [151]. The recursive greedy algorithm of [57] mentioned previously applies to Orienteering even when the reward

[4]Very recently, a poly-logarithmic approximation was given by [27] for k-Stroll in directed graphs; see the discussion of related work at the end of this section.

[5]A similar result was obtained concurrently and independently by [135]. See related work for more details.

function is a given monotone submodular set function[6] f on V, and the objective is to maximize $f(S)$ where S is the set of vertices visited by the walk. Several non-trivial problems, including Orient-TW, can be captured by using different submodular functions. Thus, the algorithm from [57] provides an $O(\log \mathrm{OPT})$ approximation for Orient-TW in directed graphs, but it runs in *quasi-polynomial* time. We make the following natural conjecture:

Conjecture 2.3. *There is a polynomial time $O(\log \mathrm{OPT})$ approximation for* Orient-TW *in directed (and undirected) graphs.*

Even in undirected graphs the best ratio known previously for Orient-TW was $O(\log^2 \mathrm{OPT})$. Our primary motivation is to close the gap between the ratios achievable in polynomial and quasi-polynomial time respectively. We remark that the quasi-polynomial time algorithm in [57] is quite different from all the other polynomial time algorithms for Orient-TW, which use algorithms for Orienteering as a black box; it does not appear easy to find a polynomial time equivalent to this quasi-polynomial time algorithm. In this chapter we make some progress in closing the gap, while also obtaining some new insights. An important aspect of our approach is to understand the complexity of the problem in terms of the maximum and minimum time-window lengths. Let $L(v) = D(v) - R(v)$ be the length of the time-window of v. Let $L_{\max} = \max_v L(v)$ and $L_{\min} = \min_v L(v)$. Our results depend on the ratio $L = L_{\max}/L_{\min}$[7]; our main result in this setting is the following theorem:

Theorem 2.4. *In directed and undirected graphs, there is an $O(\alpha \max\{\log \mathrm{OPT}, \log L\})$ approximation for* Orient-TW*, where α denotes the approximation ratio for* Orienteering.

Our results for Orient-TW are stated in more detail in Section 2.5; note that for polynomially-bounded instances, Theorem 2.4 implies an $O(\log n)$ approximation. We define the parameter L following the work of Frederickson and Wittman [85]; they showed that a constant factor approximation is achievable in undirected graphs if all time-windows are of the same length (that is, $L = 1$) and the end points of the walk are not specified. We believe that L is a natural parameter

[6] A function $f : 2^V \to \mathbb{R}^+$ is a montone submodular set function if f satisfies the following properties: (i) $f(\emptyset) = 0$, $f(A) \leq f(B)$ for all $A \subseteq B$ and (ii) $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq V$.

[7] If $L_{\min} = 0$, consider the set of vertices which have zero-length time-windows. If this includes a significant fraction of the vertices of an optimal solution, use dynamic programming to get a $O(1)$-approximation. Otherwise, we can ignore these vertices and assume $L_{\min} > 0$ without losing a significant fraction of the optimal reward.

	$\mathbb{R}^d$	Undirected Graphs	Directed Graphs
ORIENTEERING	$1+\varepsilon$	$2+\varepsilon$	$O(\log^2 \mathrm{OPT})$
ORIENT-DEADLINE	$*$	$O(\log \mathrm{OPT})$†	$O(\log^3 \mathrm{OPT})$
ORIENT-TW	$*$	$O(\log^2 \mathrm{OPT})$† $O(\max\{\log \mathrm{OPT}, \log L\})$	$O(\log^4 \mathrm{OPT})$ $O(\log^2 \mathrm{OPT} \cdot \max\{\log \mathrm{OPT}, \log L\})$

Table 2.1: The best currrently known polynomial-time approximation ratios for ORIENTEERING and ORIENT-TW. The $1+\varepsilon$-approximation for ORIENTEERING in Euclidean Space ($\mathbb{R}^d$ for fixed dimension d) is due to [60]; the other entries for Euclidean space marked '$*$' have the same approximation ratio as the more general undirected graph problems. The two entries marked '†' in undirected graphs are due to [25]; all remaining entries are from this thesis. The quasi-polynomial time algorithm of [57] gives an $O(\log \mathrm{OPT})$-approximation for *all* problems in this table.

to consider in the context of time-windows. In many practical settings L is likely to be small, and hence, algorithms whose performance depends on L may be better than those that depend on other parameters. In [25] an $O(\log D_{\max})$ approximation is given for ORIENT-TW in undirected graphs where $D_{\max} = \max_v D(v)$ and only the start vertex s is specified (here it is assumed that all the input is integer valued). We believe that $L_{\max}$ is a better measure than $D_{\max}$ for ORIENT-TW; $L_{\max} \leq D_{\max}$ for all instances, and $L_{\max}$ is considerably smaller in many instances. Further, our algorithm applies to directed graphs while the algorithm in [25] is applicable only for undirected graphs. Finally, our algorithm is for the point-to-point version while the one in [25] does not guarantee that the walk ends at t.

We believe that our results for ORIENT-TW, though obtained using relatively simple ideas, are interesting, useful and shed more light on the complexity of the problem. In particular, some of these ideas may lead to an $O(\log n)$ approximation for the time-window problem in undirected graphs even when L is not poly-bounded.

Table 2.1 above summarizes the best known approximation ratios for ORIENTEERING and ORIENT-TW. We now give a high level description of our technical ideas.

Overview of Algorithmic Ideas: For ORIENTEERING we follow the basic framework of [31], which reduces ORIENTEERING to k-STROLL via the MIN-EXCESS problem (formally defined in Section 2.2). We thus focus on the k-STROLL problem.

In undirected graphs, Chaudhuri *et al.* [42] give a $(2+\varepsilon)$-approximation for the k-STROLL problem. To improve the 3-approximation for ORIENTEERING via the method of [31], one needs a

2-approximation for the k-Stroll problem with some additional properties. Unfortunately it does not appear that even current advanced techniques can be adapted to obtain such a result (see [88] for a more technical discussion of this issue). We get around this difficulty by giving a *bi-criteria* approximation for k-Stroll. For k-Stroll, let L be the length of an optimal path, and D be the shortest path in the graph from s to t. (Thus, the excess of the optimal path is $L - D$.) Our main technical result for k-Stroll is a polynomial-time algorithm that, for any given $\varepsilon \geq 0$, finds an s-t walk of length at most $\max\{1.5D, 2L - D\}$ that contains at least $(1-\varepsilon)k$ vertices. For this, we prove various structural properties of near optimal k-Stroll solutions via the algorithm of [42], which in turn relies on the algorithm of [15] for k-MST. We also obtain a bi-criteria algorithm for Min-Excess.

For directed graphs, no non-trivial approximation algorithm was known for the k-Stroll problem. In [57] the $O(\log \mathrm{OPT})$ approximation for Orienteering is used to obtain an $O(\log^2 k)$ approximation for the k-TSP problem in quasi-polynomial time: In the k-TSP problem, the goal is to find a walk containing at least k vertices that begins *and ends* at a given vertex s; that is, k-TSP is the special case of k-Stroll where $t = s$. Once again we focus on a bi-criteria approximation for k-Stroll and obtain a solution of length $3\mathrm{OPT}$ that visits $\Omega(k/\log^2 k)$ nodes. Our algorithm for k-Stroll is based on an algorithm for k-TSP for which we give an $O(\log^3 k)$ approximation — for this we use simple ideas inspired by the algorithms for asymmetric traveling salesperson problem (ATSP) [86, 116] and an earlier poly-logarithmic approximation algorithm for k-MST [20].

For Orient-TW, we scale time window lengths so $L_{\min} = 1$; our main insight is that (with a constant-factor loss in approximation ratio), the problem can be reduced to *either* a collection of Orient-Deadline instances (for which we use an $O(\log \mathrm{OPT})$-approximation), or an instance in which all release times and deadlines are integral and in which the longest window has length $L = L_{\max}/L_{\min}$. In the latter case, we note that windows of length at most L can be partitioned into $O(\log L)$ smaller windows whose lengths are powers of 2, such that a window of length 2^i begins and ends at a multiple of 2^i. This allows us to decompose the instance into $O(\log L)$ instances of Orienteering.

2.1.1 Related Work

We have already mentioned several papers on ORIENTEERING and similar problems; we now describe some related work not previously discussed. We first consider undirected graphs. The ORIENTEERING problem was formally defined by Golden, Levy and Vohra in [93]. Goemans and Willimson [90] considered the PRIZE-COLLECTING STEINER TREE and TSP problems (these are special cases of the more general version defined in [24]); in these problems the objective is to minimize the cost of the tree (or tour) plus a penalty for not visiting nodes. They used primal-dual methods to obtain a 2-approximation. This influential algorithm was used to obtain constant factor approximation algorithms for the k-MST, k-TSP and k-STROLL problems [32, 87, 15, 88, 42], improving upon an earlier poly-logarithmic approximation [20]. As we mentioned already, the algorithms for k-STROLL yield algorithms for ORIENTEERING [31]. ORIENT-TW was shown to be NP-hard even when the graph is a path [151]; for the path, Bar-Yehuda, Even and Shahar [26] give an $O(\log \mathrm{OPT})$ approximation. The best known approximation for general undirected graphs is $O(\log^2 \mathrm{OPT})$, given by [25]; the ratio improves to $O(\log \mathrm{OPT})$ for the case of deadlines only [25]. A constant factor approximation can be obtained if the number of distinct time windows is fixed [56].

In directed graphs, the problems are less understood. For example, though the k-STROLL problem is only known to be APX-hard, no non-trivial approximation was known until a few years subsequent to the work described in this chapter; recently, Bateni and Chuzhoy [27] gave a $\min\{O(\log^2 n \log k/\log\log n), O(\log^4 k)\}$-approximation for k-STROLL in directed graphs. In [57], Chekuri and Pál showed that a simple recursive greedy algorithm that runs in quasi-polynomial time gives an $O(\log \mathrm{OPT})$ approximation for ORIENTEERING and for ORIENT-TW. The algorithm also applies to the problem where the objective function is any given submodular function on the vertices visited by the walk; several more complex problems can be captured by this generalization. Motivated by the lack of algorithms for the k-STROLL problem, Chekuri and Pál [58] also studied the Asymmetric Traveling Salesperson Path problem (ATSPP). ATSPP is the special case of k-STROLL with $k = n$. Although closely related to the well studied ATSP problem, an approximation algorithm for ATSPP does not follow directly from that for ATSP. Chekuri and Pál [58] give an $O(\log n)$ approximation for ATSPP, matching the best ratio that was known for ATSP until

very recently, when Asadpour *et al.* [17] gave an $O(\log n / \log\log n)$-approximation for ATSP by improving the upper bound on the integrality gap of the well-known Held-Karp LP relaxation for ATSP [104].

Concurrently with and independent from the work described in this chapter, Nagarajan and Ravi [135] obtained an $O(\log^2 n)$ approximation for ORIENTEERING in directed graphs. They also use a bi-criteria approach for the k-STROLL problem and obtain results essentially similar to those in this chapter for directed graph problems, including rooted k-TSP. However their algorithm for (bi-criteria) k-STROLL is based on an LP approach while we use a simple combinatorial greedy merging algorithm. Our ratios depend only on OPT or k while theirs depend also on n. On the other hand, the LP approach has some interesting features; in particular, following the recent improved upper bound on the integrality gap of the Held-Karp LP relaxation [17], the approximation ratio of their algorithm improved to $O(\log^2 n / \log\log n)$. (See [135] for more details.)

Chapter Outline

We begin by presenting some useful technical results and notation in Section 2.2: We first show that using standard scaling techniques, one can reduce a weighted version of ORIENTEERING to the unweighted version we focus on subsequently, at a small loss in the approximation ratio. We then describe a reduction from ORIENTEERING to k-STROLL, originally due to [31]; we modify this reduction so that *bi-criteria* approximations for k-STROLL can be used to obtain approximation algorithms for ORIENTEERING, and generalize it to also apply to directed graphs.

In Section 2.3, we give a bi-criteria approximation algorithm for k-STROLL in undirected graphs, with some additional guarantees on the solution returned; this, combined with the reduction of Section 2.2, yields a $(2+\varepsilon)$ approximation for undirected ORIENTEERING. Next, in Section 2.4, we give a bi-criteria approximation algorithm for k-STROLL in directed graphs. This gives an $O(\log^2 \mathrm{OPT})$-approximation for directed ORIENTEERING, along with similar results for related problems.

Finally, in Section 2.5, we reduce ORIENT-TW to ORIENTEERING: We show that in both undirected and directed graphs, an α-approximation algorithm for ORIENTEERING can be used to obtain an $O(\alpha \max\{\log \mathrm{OPT}, \log L\})$-approximation for ORIENT-TW, where L denotes the ratio

between the minimum and maximum time-window lengths. For instances in which L is polynomially bounded, this reduction, combined with our results from Sections 2.3 and 2.4, yields an $O(\log n)$-approximation for ORIENT-TW in undirected graphs, and an $O(\log^3 n)$-approximation for ORIENT-TW in directed graphs.

2.2 Preliminaries and Notation

In the *weighted* version of ORIENTEERING, the input consists of a graph G with a length on each edge, two vertices $s, t \in V$, a non-negative time-limit B, and a *weight* $w(v)$ on every vertex v. The goal is to find an $s-t$ walk of length at most B that maximizes the sum of the weights of vertices visited by the walk. (If a vertex is visited multiple times, its weight is only counted once towards the objective function.) We show below that the weighted version of ORIENTEERING can be effectively reduced to the unweighted version. This uses standard scaling and rounding techniques; we include the proof here for completeness.

Lemma 2.5. *If there is a polynomial-time ρ-approximation algorithm for* ORIENTEERING*, there is a polynomial-time algorithm which returns a ρ-approximate solution to instances of weighted* ORIENTEERING *when all vertex weights are integers between* 1 *and* n^2*.*

Proof. Given an input graph G, for each vertex $v \in V(G)$, let $w(v)$ denote the weight of v. Construct a new graph G' by replacing each vertex $v \in V(G)$ with a clique of $w(v)$ copies of v, and setting the length of the edges connecting copies of v to be 0. It is easy to see that for any vertices $s, t \in V(G)$ and any time limit B, there is an $s-t$ walk in G of length B and total weight W if and only if there is a walk in G' from a copy of s to a copy of t that visits W distinct vertices and has length B. The total number of vertices in G' is at most n^3, and we can thus use a polynomial-time ρ-approximation for ORIENTEERING to find a ρ-approximate solution to the original instance of weighted ORIENTEERING in G. □

Lemma 2.6. *If there is a polynomial-time ρ-approximation for* ORIENTEERING*, then there is a polynomial-time $\rho(1 + o(1))$-approximation for weighted* ORIENTEERING*.*

Proof. We assume without loss of generality that the input graph G is complete, and that the edge lengths satisfy the triangle inequality; if not, we can ensure this by working with the metric

completion of G. For any path P, we use $w(P)$ to denote the total weight of vertices visited by P. We use OPT to denote the optimal walk in G, and $w(\text{OPT})$ to denote the total weight of vertices it visits.

We now describe an algorithm that effectively reduces the input to an instance of weighted ORIENTEERING in which all weights are integers between 1 and n^2. Guess (that is, try each of the n possibilities for) the maximum-weight vertex u visited by the optimal walk. Let G' denote the graph in which we delete from G all vertices of weight greater than $w(u)$ and all vertices of weight less than $\frac{w(u)}{n^2}$. Consider the walk OPT' in G' obtained by shortcutting OPT to skip over deleted vertices. Its length is at most the specified time limit B, and the weight $w(\text{OPT}')$ of vertices it visits is at least $w(\text{OPT}) - n\frac{w(u)}{n^2} = w(\text{OPT}) - \frac{w(u)}{n} \geq (1 - 1/n)w(\text{OPT})$.

Now, for each vertex v, set $w'(v) = \lfloor \frac{n^2}{w(u)} \cdot w(v) \rfloor$; note that $w'(v)$ is an integer between 1 and n^2. Consider the path OPT'; its modified weight $w'(\text{OPT}')$ is at least $\left(\frac{n^2}{w(u)}\left(1 - \frac{1}{n}\right) \cdot w(\text{OPT})\right) - n$; the former term comes from multiplying the weights by $n^2/w(u)$, and the latter from the floor operation, as OPT' visits at most n vertices. As $w(\text{OPT}) \geq w(u)$, this modified weight is at least $\left(\frac{n^2}{w(u)}\left(1 - \frac{1}{n}\right) \cdot w(\text{OPT})\right) - n\frac{\text{OPT}}{w(u)} = \left(\frac{n^2}{w(u)}\left(1 - \frac{2}{n}\right) \cdot w(\text{OPT})\right)$.

From Lemma 2.5 above, we can obtain a ρ-approximate solution P to the instance of weighted ORIENTEERING on the graph G' with vertex weights w'. That is, we find a walk P such that $w'(P) \geq \frac{1}{\rho} w'(\text{OPT}') \geq \frac{1}{\rho}\left(\frac{n^2}{w(u)}\left(1 - \frac{2}{n}\right) \cdot w(\text{OPT})\right)$. But for each vertex v, $w(v) \geq \frac{w(u)}{n^2} \cdot w'(v)$. Therefore, $w(P) \geq \frac{1}{\rho}\left(1 - \frac{2}{n}\right) w(\text{OPT})$; this gives the desired result. □

In [31], ORIENTEERING was reduced to the k-STROLL problem; for completeness, we provide a brief description of this reduction. We adapt some of their technical lemmas for our setting. Recall that in the k-STROLL problem, we are given a graph $G(V, E)$, two vertices $s, t \in V$, and a target integer k; the goal is to find a minimum-length walk from s to t that visits at least k vertices.

Given a (directed or undirected) graph G, for any path P that visits vertices u, v (with u occurring before v on the path), we define $d^P(u, v)$ to be the distance along the path from u to v, and $d(u, v)$ to be the shortest distance in G from u to v. We define $excess^P(u, v)$ (the excess of P from u to v) to be $d^P(u, v) - d(u, v)$. We simplify notation in the case that $u = s$, the start vertex of the path P: we write $d^P(v) = d^P(s, v)$, $d(v) = d(s, v)$, and $excess^P(v) = excess^P(s, v)$.

If P is a path from s to t, the *excess of path* P is defined to be $excess^P(t)$. That is, the excess of a path is the difference between the length of the path and the distance between its endpoints. (Equivalently, $length(P) = d(t) + excess^P(t)$.) In the MIN-EXCESS path problem, we are given a graph $G = (V, E)$, two vertices $s, t \in V$, and an integer k; our goal is to find an s-t path of minimum excess that visits at least k vertices. The path that minimizes excess clearly also has minimum total length, but the situation is slightly different for approximation. If x is the excess of the optimal path, an α-approximation for the minimum-excess problem has length at most $d(t) + \alpha x \leq \alpha(d(t) + x)$, and so it gives us an α-approximation for the minimum-length (i.e. the k-STROLL) problem; the converse is not necessarily true. Below, we reduce the MIN-EXCESS problem to k-STROLL, and then reduce ORIENTEERING to MIN-EXCESS.

2.2.1 From k-STROLL to ORIENTEERING, via MIN-EXCESS:

We first describe the algorithm due to [31] for the MIN-EXCESS problem, given one for the k-STROLL problem. If an optimal path P visits vertices in increasing order of their distance from s, we say that it is *monotonic*. The best monotonic path can be found via dynamic programming. In general, however, P may be far from monotonic; in this case, we break it up into continuous segments that are either monotonic, or have large excess. An optimal path in monotonic sections can be found by dynamic programming, and we use an algorithm for k-STROLL in the large-excess sections. Intuitively, in these large-excess sections, the length of the path is comparable to its excess; therefore, a good approximation for k-STROLL in these sections yields a good approximation for the MIN-EXCESS problem. We formalize this intuition below.

For each real r, define $f(r)$ as the number of edges on the optimal path P with one endpoint at distance from s less than r, and the other endpoint at distance at least r from s. We partition the real line into maximal intervals such that in each interval, either $f(r) = 1$ or $f(r) > 1$. (See Figure 2.1 below, essentially similar to that of [31].) Let b_i denote the left endpoint of the ith interval: An interval from b_i to b_{i+1} is of *Type 1* (corresponding to a monotonic segment) if, for each r between b_i and b_{i+1}, $f(r) = 1$. The remaining intervals are of *Type 2* (corresponding to segments with large excess).

For each interval i, from vertex u (at distance b_i from s) to vertex v (at distance b_{i+1} from

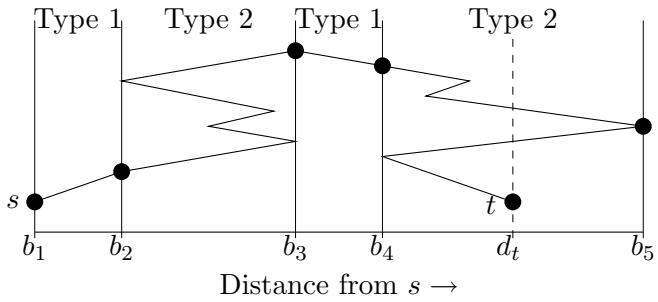


Figure 2.1: A breakdown of a path P into Type-1 (monotonic) and Type-2 (large-excess) intervals. The solid vertical lines indicate segment boundaries, with dots corresponding to s and t, and the first and last vertex of each segment.

s), we define $ex(i)$ as the *increase* in excess that P incurs while going from u to v. (That is, $ex(i) = excess^P(v) - excess^P(u)$.) Also, we let ℓ_i be the length of P contained in interval i, and d_i be the length of the shortest path from u to v contained entirely in interval i. From our definitions, the overall excess of the optimal path P is given by $excess^P(t) = \sum_i ex(i)$. In [31], it is shown that in undirected graphs, for any Type-2 interval i, $\ell_i \geq 3(b_{i+1} - b_i)$. (For the last interval, we instead obtain $\ell_i \geq 3(d(t) - b_i)$.) To see that this is true, note from Figure 2.1 that ℓ_i is at least the integral of $f(d)$ for each d between b_i and b_{i+1}. Since i is an interval of Type 2, $f(d) \geq 2$; further, one can observe using a parity argument that $f(d) \geq 3$, since if P crosses distance d only twice, it must end at distance less than d. For the results of [31], it suffices to prove that the global excess, $excess(P)$, is at least $\frac{2}{3}\sum_{i \text{ of Type 2}} \ell_i$, which follows from the previous argument. We need to refine this slightly in the following lemma by bounding the local excess in each interval, instead of the global excess.

Lemma 2.7. *For any Type-2 interval i of path P in an undirected graph, $ex(i) \geq \max\{\ell_i - d_i, \frac{2\ell_i}{3}\}$.*

Proof. We have:

$$\begin{aligned} ex(i) &= \big(d^P(v) - d(v)\big) - \big(d^P(u) - d(u)\big) \\ &= \big(d^P(v) - d^P(u)\big) - (d(v) - d(u)) \\ &= \ell_i - (b_{i+1} - b_i). \end{aligned}$$

(In the case of the last segment, containing t, the last equality should be $\ell_i - (d(t) - b_i)$.) For any Type-2 segment, $\ell_i \geq 3(b_{i+1} - b_i)$ (or $3(d_t - b_i)$), so we have $ex(i) \geq \frac{2\ell_i}{3}$. Also, the shortest-path

distance d_i from u to v contained in interval i is at least $b_{i+1} - b_i$. Therefore, $ex(i) \geq \ell_i - d_i$. □

We now briefly describe the dynamic-programming algorithm of [31] for MIN-EXCESS: A vertex v belongs to interval i if its distance from s is greater than b_i and at most b_{i+1}. (Note that v may be any vertex of G, not necessarily one on an optimal path P.) For each interval that might be in an optimal solution, and for each reward that might be collected in this interval, find a short path using vertices of this interval that collects at least the desired reward. For each interval, find paths assuming that it is both a Type-1 interval and a Type-2 interval. In the former case, the optimal path is monotonic, so we can easily find it using a dynamic programming subroutine. In the latter case, we use an approximation algorithm for k-STROLL to find a short path that collects at least the desired reward. Having found a good solution for each possible interval, one can "guess" the intervals of the optimal solution and stitch them together using a master dynamic program. Thus, the following lemma is proved in [31]; we provide a proof here for completeness.

Lemma 2.8 ([31])**.** *In undirected graphs, a β-approximation to the k-*STROLL *problem implies a $(\frac{3\beta}{2} - \frac{1}{2})$-approximation to the* MIN-EXCESS *problem.*

Proof. Let L denote the total length of an optimal path P, L_1 denote the total length of P in Type-1 intervals, and L_2 the total length in Type-2 intervals. Recall that for every Type-1 interval, we can find the optimal path using dynamic programming, and for every Type-2 interval, we use our approximation algorithm for k-STROLL to find a short path that collects the reward we desire. The path P' we find has total length at most $L_1 + \beta L_2$. The excess of the optimal path P is $L - d(t)$, while the excess of our path P' is at most $L_1 + \beta L_2 - d(t) = L - d(t) + (\beta - 1)L_2$. From Lemma 2.7, $\frac{2L_2}{3} \leq \mathit{excess}(P)$. Hence, the excess of P' is at most $\mathit{excess}(P) + \frac{3}{2}(\beta - 1)\mathit{excess}(P)$. □

Using very similar arguments, we can prove an analogous result for directed graphs. First, we need the equivalent of Lemma 2.7. In directed graphs, for each real r, we let $f(r)$ be the number of arcs a on the optimal path P such that the tail of a is at distance less than r from s, and the head of a is at distance at least r from s. All other definitions are identical to those in the undirected case. Now, we can only observe that $f(d)$ is at least 2 for all d in Type 2 intervals. (As before, a parity argument implies that path P must cross distance d at least 3 times, but on one of these

occasions, the tail of the arc will have distance at least d from s, while the head has distance less than d. Hence, this does not contribute to $f(d)$.) It is now easy to prove the required lemma:

Lemma 2.9. *For any Type-2 interval i of path P in a directed graph, $ex(i) \geq \max\{\ell_i - d_i, \frac{\ell_i}{2}\}$.*

Proof. Exactly as in Lemma 2.7, we obtain $ex(i) = l_i - (b_{i+1} - b_i)$. Again, the shortest path distance d_i from the first vertex of the interval to the last is at least $b_{i+1} - b_i$, and so $ex(i) \geq l_i - d_i$. However, we can now only conclude that $l_i \geq 2(b_{i+1} - b_i)$. Therefore, $ex(i) \geq l_i/2$. □

We now reduce Min-Excess to k-Stroll in directed graphs; the proof is similar to that of Lemma 2.8 in undirected graphs.

Lemma 2.10. *In directed graphs, a β-approximation to the k-Stroll problem implies a $(2\beta - 1)$-approximation to the* Min-Excess *problem.*

Proof. Let L denote the total length of an optimal path P, L_1 denote the total length of P in Type-1 intervals, and L_2 the total length in Type-2 intervals. The path P' we find has total length at most $L_1 + \beta L_2$. The excess of the optimal path P is $L - d(t)$, while the excess of our path P' is at most $L_1 + \beta L_2 - d(t) = L - d(t) + (\beta - 1)L_2$. From Lemma 2.7, $\frac{L_2}{2} \leq excess(P)$. Hence, the excess of P' is at most $excess(P) + 2(\beta - 1)excess(P)$. □

We now reduce Orienteering to the Min-Excess problem. The following lemma, due to [25], applies to both directed and undirected graphs.

Lemma 2.11 ([25])**.** *A γ-approximation to the* Min-Excess *problem implies a $\lceil\gamma\rceil$-approximation for* Orienteering.

Proof. Consider an optimal path P that visist OPT vertices, and break it into $h = \lceil\gamma\rceil$ consecutive segments $P_1, P_2, \ldots P_h$, each containing OPT$/h$ vertices. Guess (that is, try all possible choices for) the first and last vertex of each segment. For each i, let s_i, t_i be the first and last vertices of segment P_i, and let ex_i, the local excess of P_i, be the difference between the length of P_i and the shortest-path distance from s_i to t_i. Let P_j be the segment with least excess; $h \cdot ex_j \leq \sum_{i=1}^{h} ex_i = d^P(s,t) - \sum_{i=1}^{h} d(s_i, t_i)$. Now, use the Min-Excess approximation algorithm to find a new s_j-t_j path P'_j that visits at least OPT$/h$ vertices, and with excess at most $\gamma \leq h$ times that of P_j.

Finally, construct a path P' by going directly from s to s_j, follow P'_j from s_j to t_j, and then go directly from t_j to t. The total length of this path is at most $\sum_{i=1}^{h} d(s_i, t_i) + h \cdot ex_j \leq d^P(s,t) = \mathit{length}(P)$. Therefore, we have an $s-t$ path of length at most the given time limit, that visits at least $\mathrm{OPT}/\lceil\gamma\rceil$ vertices. □

The way in which our algorithms differ from those of [31] and [25] is that we use *bi-criteria approximations* for k-Stroll. We say that an algorithm is an (α, β)-approximation to the k-Stroll problem if, given a graph G, vertices $s, t \in V(G)$, and a target integer k, it finds a path which visits at least k/α vertices, and has length at most β times the length of an optimal path that visits k vertices.

Lemmas 2.10 and 2.11 can be easily extended to show that an (α, β)-approximation to the k-Stroll algorithm for directed graphs gives an $(\alpha\lceil 2\beta - 1\rceil)$-approximation for the Orienteering problem in directed graphs. In Section 2.4, we use this fact, with a $(O(\log^2 k), 3)$-approximation for the k-Stroll problem in directed graphs, to get an $O(\log^2 \mathrm{OPT})$-approximation for directed Orienteering.[8] For undirected graphs, one might try to use Lemmas 2.8 and 2.11 with a $(1+\varepsilon, 2)$-approximation for the k-Stroll problem, but this leads to a $((1+\varepsilon)\times\lceil 2.5\rceil) = (3+\varepsilon)$ approximation for Orienteering. To obtain the desired ratio of $(2+\varepsilon)$, we need a refined analysis to take advantage of the particular bi-criteria algorithm that we develop for k-Stroll; the details are explained in Section 2.3.

2.3 A $(2+\varepsilon)$-Approximation for Undirected Orienteering

In the k-Stroll problem, given a metric graph G, with 2 specified vertices s and t, and a target integer k, we wish to find an s-t path of minimum length that visits at least k vertices. Let L be the length of an optimal such path, and D the shortest-path distance in G from s to t. In this section, we describe a *bi-criteria* approximation algorithm for the k-Stroll problem, as guaranteed by the following theorem:

Theorem 2.12. *For any $\varepsilon > 0$, there is an algorithm with running time $O(n^{O(1/\varepsilon^2)})$ that, given a graph G, two vertices s and t and a target integer k, finds an s-t walk of length at most*

[8] When we use the k-Stroll algorithm as a subroutine, we call it with $k \leq \mathrm{OPT}$, where OPT is the number of vertices visited by an optimum Orienteering solution.

$\max\{1.5D, 2L - D\}$ *that visits at least* $(1 - \varepsilon)k$ *vertices, where* L *is the length of the optimal* s*-*t *path that visits* k *vertices and* D *is the shortest-path distance from* s *to* t*.*

We prove Theorem 2.12 in Section 2.3.2; first, in Section 2.3.1, we describe the desired $(2 + \varepsilon)$-approximation for Orienteering in undirected graphs, assuming Theorem 2.12.

2.3.1 From k-Stroll to Min-Excess

We solve the Minimum-Excess problem using essentially the algorithm of [31]; as explained in Section 2.2.1, the key difference is that instead of calling the k-Stroll algorithm of [42] as a subroutine, we use the algorithm of Theorem 2.12 that returns a bi-criteria approximation. In addition, the analysis is slightly different, making use of the fact that our algorithm returns a path of length at most $\max\{1.5D, 2L - D\}$. In the arguments below, we fix an optimum path P, and chiefly follow the notation of [31].

Theorem 2.13. *For any fixed* $\varepsilon > 0$*, there is a polynomial-time algorithm to find an* s*-*t *path visiting at least* $(1 - \varepsilon)k$ *vertices, with excess at most twice that of an optimal path* P *visiting* k *vertices.*

Proof. As described in Section 2.2, the algorithm uses dynamic programming similar to that in [31] with our bi-criteria k-Stroll algorithm of Theorem 2.12 in place of an approximate k-Stroll algorithm. Let P' be the path returned by our algorithm. Roughly speaking, P' will be at least as good as a path obtained by replacing the segment of P in each of its intervals by a path that the algorithm finds in that interval. In Type-1 intervals the algorithm finds an optimum path because it is monotonic. In Type-2 intervals we have a bi-criteria approximation that gives a $(1 - \varepsilon)$ approximation for the number of vertices visited. This implies that P' contains at least

$(1-\varepsilon)k$ vertices. To bound the excess, we sum up the lengths of the replacement paths to obtain:

$$\begin{aligned}
length(P') &\leq \sum_{i \text{ of Type 1}} \ell_i + \sum_{i \text{ of Type 2}} \max\{1.5d_i, 2\ell_i - d_i\} \\
&\leq \sum_i \ell_i + \sum_{i \text{ of Type 2}} \max\{0.5\ell_i, \ell_i - d_i\} \\
&\leq \sum_i \ell_i + \sum_{i \text{ of Type 2}} ex(i) \\
&\leq length(P) + excess^P(t) \\
&= d(t) + 2excess^P(t)
\end{aligned}$$

where the second inequality comes from rearranging terms and the fact that $d_i \leq \ell_i$, and the third inequality follows from Lemma 2.7. Therefore, the excess of P' is at most twice that of P, the optimal path. □

For completeness, we restate Lemma 2.11, modified for a bi-criteria excess approximation: An (α, β)-approximation to the Min-Excess problem gives an $\alpha\lceil\beta\rceil$-approximation to the Orienteering problem.

Proof of Theorem 2.1. For any constant $\varepsilon > 0$, to obtain a $(2+\varepsilon)$-approximation for the undirected Orienteering problem, first find ε' such that $2+\varepsilon = \frac{2}{1-\varepsilon'}$. Theorem 2.13 implies that there is a $(\frac{1}{1-\varepsilon'}, 2)$-bi-criteria approximation algorithm for the Min-Excess problem that runs in $n^{O(1/\varepsilon^2)}$ time. Now, we use (the bi-criteria version of) Lemma 2.11 to get a $\frac{2}{1-\varepsilon'} = (2+\varepsilon)$-approximation for Orienteering in undirected graphs. □

It now remains only to prove Theorem 2.12, to which we devote the rest of this section.

2.3.2 The Proof of Theorem 2.12

Given graph G, vertices s, t, and integer k, for any fixed $\varepsilon > 0$, we wish to find an s-t path that visits at least $(1 - O(\varepsilon))k$ vertices, and has total length at most $\max\{1.5D, 2L - D\}$. Our starting point is the following theorem on k-Stroll, proved by [42]:

Theorem 2.14 ([42])**.** *Given a graph G,two vertices s and t and a target integer k, let L be the length of an optimal path from s to t visiting k vertices. For any $\delta > 0$, there is a polynomial-time*

algorithm to find a tree of length at most $(1+\delta)L$ *and containing at least* k *vertices, including both* s *and* t*.*

The algorithm of [42] *guesses* $O(1/\delta)$ vertices $s = w_1, w_2, w_3, \ldots, w_{m-1}, w_m = t$ such that an optimal path P visits the guessed vertices in this order, and for any i, the distance from w_i to w_{i+1} along P is $\leq \delta L$. It then uses the k-MST algorithm of [15] with the given set of guessed vertices to obtain a tree satisfying Theorem 2.14; this tree is also guaranteed to contain all the guessed vertices. We can assume that all edges of the tree have length at most δL; longer edges can be subdivided without adding more than $O(1/\delta)$ vertices.

Our bi-criteria approximation algorithm for k-Stroll begins by setting $\delta = \varepsilon^2$, and using the algorithm of Theorem 2.14 to obtain a k-vertex tree T containing s and t. We are guaranteed that $length(T) \leq (1+\delta)L$ (recall that L is the length of a shortest s-t path P visiting k vertices). Let $P^T_{s,t}$ be the path in T from s to t; we can double all edges of T not on $P^T_{s,t}$ to obtain a path P_T from s to t that visits at least k vertices. The length of the path P_T is $2length(T) - length(P^T_{s,t}) \leq 2length(T) - D$.

If either of the following conditions holds, the path P_T visits k vertices and has length at most $\max\{1.5D, 2L - D\}$, which is the desired result:

- The total length of T is at most $5D/4$. (In this case, P_T has length at most $3D/2$.)
- $length(P^T_{s,t}) \geq D + 2\delta L$. (In this case, P_T has length at most $2(1+\delta)L - (D+2\delta L) = 2L - D$.)

We refer to these as the *easy doubling conditions*. Our aim will be to show that if neither of the easy doubling conditions applies, we can use T to find a new tree T' containing s and t, with length at most L, and with at least $(1 - O(\varepsilon))k$ vertices. Then, by doubling the edges of T' that are not on the s-t path (in T'), we obtain a path of length at most $2L - D$ that visits at least $(1 - O(\varepsilon))k$ vertices.

Below, we describe the structure the tree T must have if neither of the easy doubling conditions holds, and in the following subsection, how to use this information to obtain the tree T'.

Structure of the Tree

If neither of the easy doubling conditions holds, then since D is at most $4/5$ of the length of T, and the length of $P_{s,t}^T$ is less than $D + 2\delta L$, the total length of the edges of $T \setminus P_{s,t}^T$ is greater than $(1/5 - 2\delta)L$. In this section, we describe how to construct the desired tree T' by removing a small piece of $T \setminus P_{s,t}^T$.

Say that a set of edges S in $T \setminus P_{s,t}^T$ is an *isolated* component if the total length of S is less than εL, and S is a connected component of $T \setminus P_{s,t}^T$.

Proposition 2.15. *We can greedily decompose the edge set of $T \setminus P_{s,t}^T$ into $\Omega(1/\varepsilon)$ disjoint pieces such that:*

- *Each piece is either a connected subgraph of or the union of isolated components of $T \setminus P_{s,t}^T$.*
- *Each piece has length in $[\varepsilon L, 3\varepsilon L)$, unless it is the union of* all *isolated components of $T \setminus P_{s,t}^T$ and has length less than εL.*

Proof. Consider the following greedy algorithm: root T at s, and consider a deepest node v in $T \setminus P_{s,t}^T$ such that the total length of edges in the subtree rooted at v is at least εL. If the total length of all edges in the subtree is at most $2\varepsilon L$, this forms a piece that is connected and has the desired size. Otherwise, (arbitrarily) select enough children of v such that the total size of all their subtrees, together with their edges to v, is between εL and $2\varepsilon L$. (Since the subtree rooted at each child has size $< \varepsilon L$ and each edge has length $\leq \delta L \ll \varepsilon L$, this is always possible.) Again, this forms a piece that is connected and has the required size.

Now delete the edges of the piece just found from T, and recurse. When no more such pieces can be found, we may be left with parts of length $< \varepsilon L$ hanging off the s-t path. For any such part that has a further piece hanging off it, connect it to that piece, increasing its length to less than $3\varepsilon L$. The remaining parts are isolated components, and unless their total size is less than εL, it is easy to combine them arbitrarily into groups with total length in $[\varepsilon L, 3\varepsilon L]$. □

Let $\mathcal{T}$ be the tree formed as follows: We have one vertex s' for $P_{s,t}^T$ and one vertex for each of the pieces of Proposition 2.15. (Thus, $\mathcal{T}$ has $\Omega(1/\varepsilon)$ vertices.) There is an edge between vertices $v_1, v_2 \in V(\mathcal{T})$ corresponding to edge sets S_1, S_2 iff S_1 contains the parent edge in T of a minimum-depth edge in S_2, or vice versa. (In the special case that $v_1 = s'$, and the minimum-depth edge in

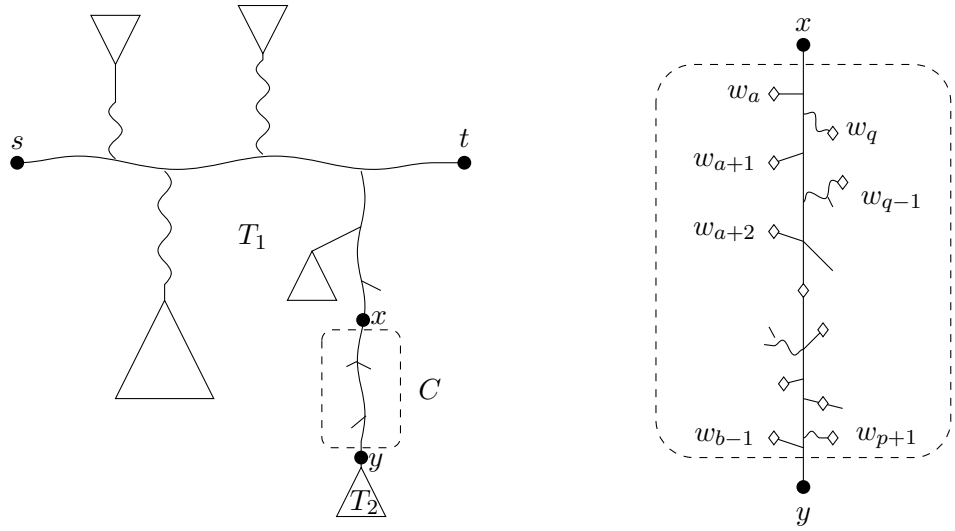


Figure 2.2: To the left is the tree T; a constant fraction of its length is not on $P^T_{s,t}$. We break these parts into pieces; the path-like piece C of degree 2, with fewer than $32\varepsilon k$ vertices, is shown in the box with the dashed lines. The right shows C in more detail, with vertices x and y at the head and foot of the spine, and guessed vertices shown as diamonds.

S_2 is incident in T to s, we add the edge between $v_1 = s'$ and v_2.) Note that any piece containing isolated components becomes a leaf of $\mathcal{T}$ adjacent to s'.

Proposition 2.16. *The tree $\mathcal{T}$ contains a vertex of degree 1 or 2 that corresponds to a piece with length in $[\varepsilon L, 3\varepsilon L)$, and containing at most $32\varepsilon k$ vertices of the original tree T that are not contained in other pieces.*

Proof. The number of vertices in $\mathcal{T}$ (not including s'), is at least $\frac{(1/5-2\delta)L}{3\varepsilon L} = \frac{1}{15\varepsilon} - \frac{2\varepsilon}{3} \geq \frac{1}{16\varepsilon}$. At least one more than half these vertices have degree 1 or 2, since $\mathcal{T}$ is a tree. If the union of all isolated components has size less than εL, we discard the vertex corresponding to this piece; we are left with at least $1/(32\varepsilon)$ vertices of degree 1 or 2. If each of them corresponds to a piece that has more than $32\varepsilon k$ vertices not in other pieces, the total number of vertices they contain is more than k, which is a contradiction. □

If $\mathcal{T}$ has a leaf that corresponds to a piece with at most $32\varepsilon k$ vertices, we delete this piece from T, giving us a tree T' with length at most $(1+\delta)L - \varepsilon L < L$, with at least $(1-32\varepsilon)k$ vertices. Doubling the edges of T' not on its s-t path, we obtain an s-t walk that visits $(1-32\varepsilon)k$ vertices and has length at most $2L - D$, and we are done.

If there does not exist such a leaf, we can find a vertex of degree 2 in $\mathcal{T}$, corresponding to

a connected subgraph/piece C of $T \setminus P_{s,t}^T$, with length ℓ in $[\varepsilon L, 3\varepsilon L)$, and at most $32\varepsilon k$ vertices. Deleting C from T gives us two trees T_1 and T_2; let T_1 be the tree containing s and t. We can reconnect the trees using the shortest path between them. If the length of this path is at most $\ell - \delta L$, we have a new tree T' with length at most L, and containing at least $(1 - 32\varepsilon)k$ vertices. In this case, as before, we are done.

Therefore, we now assume that the shortest path in G that connects T_1 and T_2 has length greater than $\ell - \delta L$, and use this fact repeatedly. (Recall that the total length of piece C is ℓ.) One consequence of this fact is that the piece C is *path-like*. That is, if x and y are the two vertices of $T - C$ with edges to C, the length of the path in C from x to y is more than $\ell - \delta L$; we refer to this path from x to y as the *spine* of the piece. (See Figure 2.2.) It follows that the total length of edges in C that are not on the spine is less than δL. We also refer to the vertex $x \in T_1$ adjacent to C as the *head* of the spine, and $y \in T_2$ adjacent to C as the *foot* of the spine. Finally, we say that for any vertices $p, q \in C$, the *distance along the spine* between vertices p and q is the length of those edges on the path between p and q that lie on the spine.

We assume for the moment that T_2 contains at least one vertex that was guessed by the algorithm of Theorem 2.14. Consider the highest-numbered guessed vertex w_p in T_2; where is the next guessed vertex w_{p+1}? It is not in T_2 by definition, nor in T_1 because the shortest path from T_2 to T_1 has length at least $\ell - \delta L$, and the edge $w_p w_{p+1}$ has length $\leq \delta L$. Therefore, it must be in C. Similarly, since $\delta L \ll l - \delta L$, the guessed vertices $w_{p+2}, w_{p+3}, \ldots$ must be in C. (In fact, there must be at least $\frac{\ell - \delta L}{\delta L} = \Omega(1/\varepsilon)$ such consecutive guessed vertices in C.) Let w_q be the highest-numbered of these consecutive guessed vertices in C.

By an identical argument, if w_b is the lowest-numbered guessed vertex in T_2, $w_{b-1}, w_{b-2}, \ldots$ must be in C. Let w_a be the lowest-numbered of these consecutive guessed vertices, so that the vertices $w_a, w_{a+1}, \ldots w_{b-2}, w_{b-1}$ are all in C.

Remark 2.17. *If T_2 does not contain any guessed vertices, the procedure above is to be modified by finding the guessed vertex w nearest the foot of the spine. Remove from C the path from w to the foot, and those branches off the spine adjacent to this path; add these edges to the tree T_2. Now, T_2 contains a guessed vertex and we may continue; this does not change our proof in any significant detail.*

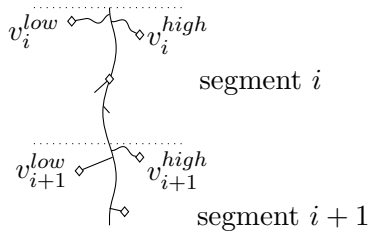


Figure 2.3: Two consecutive segments.

We now break up the piece C into *segments* as follows: Starting from x, the head of the spine, we cut C at distance $10\delta L$ along the spine from x. We repeat this process until the foot of the spine, obtaining at least $\frac{\ell-\delta L}{10\delta L} \geq \frac{1}{10\varepsilon} - \frac{1}{10}$ segments. We discard the segment nearest x and the two segments nearest y, and number the remaining segments from 1 to r consecutively from the head; we have at least $\frac{1}{10\varepsilon} - \frac{1}{10} - 3 \geq \frac{1}{15\varepsilon}$ segments remaining. For each segment, we refer to the end nearer x (the head of the spine) as the *top* of the segment, and the end nearer y as the *bottom* of the segment.

We now restrict our attention to guessed vertices in the range w_a to w_{b-1} and w_{p+1} through w_q. For each segment i, define v_i^{low} to be the lowest-numbered guessed vertex in segments i through r, and v_i^{high} to be the highest-numbered guessed vertex in segments i through r. (See Figure 2.3 above.)

Lemma 2.18. *For each i:*

1. *v_i^{low} occurs before v_{i+1}^{low} in the optimal path, and v_i^{high} occurs after v_{i+1}^{high} in the optimal path.*
2. *the distance along the spine from the top of segment i to each of v_i^{low} and v_i^{high} is at most $2\delta L$.*
3. *the distance between v_i^{low} and v_{i+1}^{low}, is at least $7\delta L$; the distance between v_i^{high} and v_{i+1}^{high} is at least $7\delta L$.*

Proof. We prove the statements for v_i^{low} and v_{i+1}^{low}; those for v_i^{high} and v_{i+1}^{high} are symmetric. Our proofs repeatedly use the fact (referred to earlier) that the shortest path from x to y does not save more than δL over ℓ, the length of C.

First, we claim that each segment contains some guessed vertex between w_a and w_{b-1}. Suppose some segment i did not; let c be the first index greater than or equal to a such that w_c is not above

segment i in the tree. (Since w_a is above segment i, and w_b below it, we can always find such an index c.) Therefore, w_{c-1} is above segment i, and w_c below it. We can now delete segment i, and connect the tree up using the edge between w_{c-1} and w_c; this edge has length at most δL. But this gives us a path from x to y of length at most $\ell - 10\delta L + \delta L$, which is a contradiction.

Now, let v_i^{low} be the guessed vertex w_j; we claim that it is in segment i. Consider the location of the guessed vertex w_{j-1}. By definition, it is not in segments i through r; it must then be in segments 1 through $i-1$. If w_j were not in segment i, we could delete segment i (decreasing the length by $10\delta L$) and connect x and y again via the edge between w_j and w_{j-1}, which has length at most δL. Again, this gives us a path that is shorter by at least $9\delta L$, leading to a contradiction. Therefore, for all i, v_i^{low} is in segment i.

Because the lowest-numbered guessed vertex in segments i through r is in segment i, it has a lower number than the lowest-numbered guessed vertex in segments $i+1$ through r. That is, v_i^{low} occurs before v_{i+1}^{low} on the optimal path, which is the first part of the lemma.

We next prove that for all i, the distance along the spine from v_i^{low} to the top of segment i is at most $2\delta L$. If this is not true, we could delete the edges of the spine from v_i^{low} to the top of segment i, and connect v_i^{low} to the previous guessed vertex, which must be in segment $i-1$. The deletion decreases the length by at least $2\delta L$, and the newly added edge costs at most δL, giving us a net saving of at least δL; as before, this is a contradiction.

The final part of the lemma now follows, because we can delete the edges of the spine from v_i^{low} to the bottom of the segment (decreasing our length by at least $8\delta L$), and if the distance from v_i^{low} to v_{i+1}^{low} were less than $7\delta L$, we would save at least δL, giving a contradiction. □

Now, for each segment i, define $gain(i)$ to be the sum of the reward collected by the optimal path between v_i^{low} and v_{i+1}^{low} and the reward collected by the optimal path between v_{i+1}^{high} and v_i^{high}. Since these parts of the path are disjoint, $\sum_i gain(i) \leq k$, and there are at least $\frac{1}{15\varepsilon}$ such segments, there must exist some i such that $gain(i) \leq 15\varepsilon k$. By enumerating over all possibilities, we can find such an i.

Contracting the Graph

We assume we have found a segment numbered i such that $gain(i) \leq 15\varepsilon k$. Consider the new graph H formed from G by contracting together the 4 vertices v_i^{low}, v_i^{high}, v_{i+1}^{low} and v_{i+1}^{high} of G to form a new vertex v'; we prove the following proposition.

Proposition 2.19. *The graph H has a path of length at most $L - 14\delta L$ that visits at least $(1-15\varepsilon)k$ vertices.*

Proof. Consider the optimal path P in G, and modify it to find a path P_H in H by shortcutting the portion of the path between v_i^{low} and v_{i+1}^{low}, and the portion of the path between v_{i+1}^{high} and v_i^{high}. Since $gain(i) \leq 15\varepsilon k$, the new path P_H visits at least $(1 - 15\varepsilon)k$ vertices. Further, since the shortest-path distance from v_i^{low} to v_{i+1}^{low} and the shortest-path distance from v_i^{high} to v_{i+1}^{high} are each $\geq 7\delta L$, path P_H has length at most $L - 14\delta L$. □

Using the algorithm of [15], we can find a tree T_H in H of total length at most $L - 13\delta L$ with at least $(1 - 15\varepsilon)k$ vertices. This tree T_H may not correspond to a tree of G (if it uses the new vertex v'). However, we claim that we can find a tree T_i in G of length at most $13\delta L$, that includes each of v_i^{low}, v_i^{high}, v_{i+1}^{low}, v_{i+1}^{high}. We can combine the two trees T_H and T_i to form a tree T' of G, with total length L.

Proposition 2.20. *There is a tree T_i in G containing v_i^{low}, v_i^{high}, v_{i+1}^{low} and v_{i+1}^{high}, of total length at most $13\delta L$.*

Proof. We use all of segment i, and enough of segment $i + 1$ to reach v_{i+1}^{low} and v_{i+1}^{high}. The edges of segment i along the spine have length $\leq 10\delta L$, v_{i+1}^{low} and v_{i+1}^{high} each have distance along the spine at most $2\delta L$ from the top of segment $i + 1$ (by Lemma 2.18). Finally, the total length of *all* the edges in the piece C not on the spine is at most δL. Therefore, to connect all of v_i^{low}, v_i^{high}, v_{i+1}^{low} and v_{i+1}^{high}, we must use edges of total length at most $(10 + 2 + 1)\delta L = 13\delta L$. □

We can now complete the proof of Theorem 2.12:

Proof of Theorem 2.12. Set $\varepsilon' = \varepsilon/32$ and run the algorithm of [42] with $\delta = \varepsilon'^2$ to obtain a k-vertex tree T of length at most $(1 + \delta)L$. If either of the easy doubling conditions holds, we

can double all the edges of T not on its s-t path to obtain a new s-t walk visiting k vertices, with length at most $\max\{1.5D, 2L - D\}$.

If neither of the easy doubling conditions holds, use T to obtain T' containing s and t, with length at most L and at least $(1 - 32\varepsilon')k$ vertices. Doubling edges of T' not on its s-t path, we find a new s-t path visiting $(1 - 32\varepsilon')k = (1 - \varepsilon)k$ vertices, of length at most $2L - D$. □

2.4 Orienteering in Directed Graphs

We give an algorithm for Orienteering in directed graphs, based on a bi-criteria approximation for the (rooted) k-TSP problem: Given a graph G, a start vertex s, and an integer k, find a cycle in G of minimum length that contains s and visits k vertices. We assume that G always contains such a cycle; let OPT be the length of a shortest such cycle. We assume knowledge of the value of OPT, and that G is complete, with the arc lengths satisfying the asymmetric triangle inequality.

Our algorithm finds a cycle in G containing s that visits at least $k/2$ vertices, and has length at most $O(\log^2 k) \cdot \text{OPT}$. The algorithm gradually builds up a collection of strongly connected components. Each vertex starts as a separate component, and subsequently components are merged to form larger components. The main idea of the algorithm is to find low density cycles that visit multiple components, and use such cycles to merge components. (The density of a cycle C is defined as its length divided by the number of vertices that it visits; there is a polynomial-time algorithm to find a minimum-density cycle in directed graphs.) While merging components, we keep the invariant that each component is strongly connected and *Eulerian*, that is, each arc of the component can be visited exactly once by a single closed walk.

We note that this technique is similar to the algorithms of [86, 116] for ATSP; however, the difficulty is that a k-TSP solution need not visit all vertices of G and the algorithm is unaware of the vertices to visit. We deal with this using two tricks. First, we force progress by only merging components of similar size, hence ensuring that each vertex only participates in a logarithmic number of merges — when merging two trees or lists, one can charge the cost of merging to the smaller side, however when merging multiple components via a cycle, there is no useful notion of a smaller side. Second, we are more careful about picking representatives for each component; picking an arbitrary representative vertex from a component does not work. A variant that does work is to

BUILDCOMPONENTS:
for (each i in $\{0, 1, \ldots, \lfloor \log_2(k/4 \log_2 k) \rfloor\}$) do:
  For each component in tier i
    (Arbitrarily) assign each vertex a distinct color in $\{1, \ldots, 2^{i+1} - 1\}$.
  Let $\{\mathcal{V}_j^i \mid j = 1, \ldots, 2^{i+1} - 1\}$ be the resulting color classes.
  Let H_j^i be the subgraph of G induced by the vertex set $\mathcal{V}_j^i$.
  While (there is a cycle C of density at most $\alpha \cdot 2^i$ in some graph H_j^i)
    Let $v_1, \ldots, v_l$ be the vertices of H_j^i visited by C
    Let v_p belong to component $\mathcal{C}_p$, $1 \leq p \leq l$
    (Two vertices of H_j^i never share a component, so $\mathcal{C}_1, \ldots, \mathcal{C}_l$ are distinct.)
    Form a new component $\mathcal{C}$ by merging $\mathcal{C}_1, \ldots, \mathcal{C}_l$ using C
    ($\mathcal{C}$ must belong to a higher tier)
    Remove all vertices of $\mathcal{C}$ from the graphs $H_{j'}^i$ for $j' \in \{1, \ldots, 2^{i+1} - 1\}$.

contract each component to a single vertex, however, this loses an additional logarithmic factor in the approximation ratio since an edge in a contracted vertex may have to be traversed a logarithmic number of times in creating a cycle in the original graph. To avoid this, our algorithm ensures components are Eulerian. One option is to pick a representative from a component *randomly* and one can view our coloring scheme described below as a derandomization.

We begin by pre-processing the graph to remove any vertex v such that the sum of the distances from s to v and v to s is greater than OPT; such a vertex v obviously cannot be in an optimum solution. Each remaining vertex initially forms a component of size 1. As components combine, their sizes increase; we use $|X|$ to denote the size of a component X, i.e. the number of vertices in it. We assign the components into tiers by size; components of size $|X|$ will be assigned to tier $\lfloor \log_2 |X| \rfloor$. Thus, a tier i component has at least 2^i and fewer than 2^{i+1} vertices; initially, each vertex is a component of tier 0. For ease of notation, we use α to denote the quantity $4 \log k \cdot \mathrm{OPT}/k$.

In the main phase of the algorithm, we will iteratively push components into higher tiers, until we have enough vertices in large components, that is, components of size at least $k/4 \log k$. The procedure BUILDCOMPONENTS (above) implements this phase. Once we have amassed at least $k/2$ vertices belonging to large components, we finish by attaching a number of these components to the root s via direct arcs. Before providing the details of the final phase of the algorithm, we establish some properties of the algorithm BUILDCOMPONENTS.

Lemma 2.21. *Throughout the algorithm, all components are strongly connected and Eulerian. If*

any component X was formed by combining components of tier i, the sum of the lengths of arcs in X is at most $(i+1)\alpha|X|$.

Proof. Whenever a component is formed, the newly added arcs form a cycle in G. It follows immediately that every component is strongly connected and Eulerian. We prove the bound on arc lengths by induction.

Let C be the low-density cycle found on vertices $v_1, v_2, \ldots v_l$ that connects components of tier i to form the new component X. Let $\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_l$ be the components of tier i that are combined to form X. Because the density of C is at most $\alpha 2^i$, the total length of the arcs in C is at most $\alpha 2^i l$. However, each tier i component has at least 2^i vertices, and so $|X| \geq 2^i l$. Therefore, the total length of arcs in C is at most $\alpha|X|$.

Now, consider any component $\mathcal{C}_h$ of tier i; it was formed by combining components of tier at most $i-1$, and so, by the induction hypothesis, the total length of all arcs in component $\mathcal{C}_h$ is at most $i\alpha|\mathcal{C}_h|$. Therefore, the total length of all arcs in all the components combined to form X is $i\alpha \sum_{h=1}^{l} |\mathcal{C}_h| = i\alpha|X|$. Together with the newly added arcs of C, which have weight at most $\alpha|X|$, the total weight of all arcs in component X is at most $(i+1)\alpha|X|$. □

Let O be a fixed optimum cycle, and let $o_1, \ldots, o_k$ be the vertices it visits.

Lemma 2.22. *At the end of iteration i of* BuildComponents*, at most $\frac{k}{2\log k}$ vertices of O remain in components of tier i.*

Proof. Suppose that more than $\frac{k}{2\log k}$ vertices of O remain in tier i at the end of the ith iteration. We show a low-density cycle in one of the graphs H_j^i, contradicting the fact that the while loop terminated because it could not find any low-density cycle: Consider the color classes $\mathcal{V}_j^i$ for $j \in \{1, \ldots, 2^{i+1}-1\}$. By the pigeonhole principle, one of these classes has to contain more than $k/(2\log k \cdot 2^{i+1})$ vertices of O.[9] We can "shortcut" the cycle O by visiting only these vertices; this new cycle has cost at most OPT and visits at least two vertices. Therefore, it has density less than $(2^{i+2} \cdot \mathrm{OPT} \log k)/k$, which is $2^i \cdot \alpha$. Hence, the while loop would not have terminated. □

We call a component *large*, if it has at least $k/4\log k$ vertices. Since we lose at most $\frac{k}{2\log k}$

[9]The largest value of i used is such that $k/2\log k \cdot 2^{i+1} \geq 1$, so there are always at least 2 vertices in this color class.

vertices of O in each iteration, and there are fewer than $\log k$ iterations, we must have at least $k/2$ vertices of O in large components after the final iteration.

Theorem 2.23. *There is an $O(n^4)$-time algorithm that, given a directed graph G and a vertex s, finds a cycle with $k/2$ vertices rooted at s, of length $O(\log^2 k)\mathrm{OPT}$, where OPT is the length of an optimum k-TSP tour rooted at s.*

Proof. Run the algorithm BuildComponents, and consider the large components; at least $k/2$ vertices are contained in these components. Greedily select large components until their total size is at least $k/2$; we have selected at most $2\lfloor(\log k)\rfloor$ components. For each component, pick a representative vertex v arbitrarily, and add arcs from s to v and v to s; because of our pre-processing step (deleting vertices far from s), the sum of the lengths of newly added arcs for each representative is at most OPT. Therefore, the total length of newly added arcs (over all components) is at most $2\log k \cdot \mathrm{OPT}$. The large components selected, together with the newly added arcs, form a connected Eulerian component H, containing s. Let $k' \geq k/2$ be the number of vertices of H. From Lemma 2.21, we know that the sum of the lengths of arcs in H (not counting the newly added arcs) is at most $(\log k - 1)\alpha k'$. With the newly aded arcs, the total length of arcs of H is at most $4\log^2 k \cdot \mathrm{OPT} \times k'/k$. Since H is Eulerian, there is a cycle of at most this length that visits each of the k' vertices of H.

If, from this cycle, we pick a segment of $k/2$ consecutive vertices uniformly at random, the expected length of this segment will be $2\log^2 k\mathrm{OPT}$. Hence, the shortest segment containing $k/2$ vertices has length at most $2\log^2 k \cdot \mathrm{OPT}$. Concatenate this with the arc from s to the first vertex of this segment (paying at most OPT), and the arc (again of cost $\leq \mathrm{OPT}$) from the last vertex to s; this gives us a cycle that visits at least $k/2$ vertices, and has cost less than $3\log^2 k \cdot \mathrm{OPT}$.

The running time of this algorithm is dominated by the time to find minimum-density cycles, each of which takes $O(nm)$ time [4], where n and m are the number of vertices and edges respectively. The algorithm makes $O(n)$ calls to the cycle-finding algorithm which implies the desired $O(n^4)$ bound. □

By using the algorithm from Theorem 2.23 greedily $\log k$ times, we obtain the following corollary.

Corollary 2.24. *There is an $O(\log^3 k)$ approximation for the rooted k-TSP problem in directed*

graphs.

Theorem 2.25. *There is an $O(n^4)$-time algorithm that, given a directed graph G and nodes s, t, finds an s-t path of length* 3OPT *containing $\Omega(k/\log^2 k)$ vertices, where* OPT *is the length of an optimal k-*Stroll *from s to t.*

Proof. We pre-process the graph as before, deleting any vertex v if the sum of the distance from s to v and the distance from v to t is greater than OPT. In the remaining graph, we consider two cases: If the distance from t to s is at most OPT, we leave the graph unmodified. Otherwise, we add a 'dummy' arc from t to s of length OPT. Now, there is a cycle through s that visits at least k vertices, and has length at most 2OPT. We use the previous theorem to find a cycle through s that visits $k/2$ vertices and has length less than $6\log^2 k$OPT. Now, break this cycle up into consecutive segments, each containing $\lfloor k/(12\log^2 k)\rfloor$ vertices (except possibly the last, which may contain more). One of these segments has length less than OPT; it follows that this part cannot use the newly added dummy arc. We obtain a path from s to t by beginning at s and taking the shortest path to the first vertex in this segment; this has length at most OPT. We then follow the cycle until the last vertex of this segment (again paying at most OPT), and then take the shortest path from the last vertex of the segment to t. The total length of this path is at most 3OPT, and it visits at least $\lfloor k/(12\log^2 k)\rfloor$ vertices. □

We can now complete the proof of Theorem 2.2, showing that there is an $O(\log^2 \text{OPT})$ approximation for Orienteering in directed graphs.

Proof of Theorem 2.2. As mentioned in Section 2.2, Lemmas 2.10 and 2.11 can be extended to show that an (α, β)-bi-criteria approximation to the directed k-Stroll problem can be used to get an $(\alpha \cdot \lceil 2\beta - 1\rceil)$-approximation to the Orienteering problem on directed graphs. Theorem 2.25 gives us a $(O(\log^2 k, 3)$-approximation to the the directed k-Stroll problem, which implies that there is a polynomial-time $O(\log^2 \text{OPT})$-approximation algorithm for the directed Orienteering problem. □

2.5 Orienteering with Time Windows

Much of the prior work on Orient-TW, following [25], can be cast in the following general framework: Use combinatorial methods to reduce the problem to a collection of sub-problems where the time-windows can be ignored. Each sub-problem has a subset of vertices V', start and end vertices $s', t' \in V'$, and a time-interval I in which we must travel from s' to t', visiting as many vertices of V' within their time windows as possible. However, the sub-problem is constructed such that the time-window for every vertex in V' entirely contains the interval I. Therefore, the sub-problem is really an instance of Orienteering (without time-windows). An approximation algorithm for Orienteering can be used to solve each sub-problem, and these solutions can be pasted together using dynamic programming. The next subsection describes this framework in more detail.

We use the same general framework; as a consequence, our results apply to both directed and undirected graphs; while solving a sub-problem we use either the algorithm for Orienteering on directed graphs, or the algorithm for undirected graphs. Better algorithms for either of these problems would immediately translate into better algorithms for Orient-TW.

Subsequently, we use α to denote the approximation ratio for Orienteering, and state our results in terms of α; from the previous sections, α is $O(1)$ for undirected graphs and $O(\log^2 \mathrm{OPT})$ for directed graphs.

Recall that $L_{\max}$ and $L_{\min}$ are the lengths of the longest and shortest time time-windows respectively, and L is the ratio $\frac{L_{\max}}{L_{\min}}$. We first provide two algorithms with the following guarantees:

- $O(\alpha \log L_{\max})$, if the release time and deadline of every vertex are integers.
- $O(\alpha \log \mathrm{OPT})$, if $L \leq 2$.

We note that the first algorithm is already an improvement over the $O(\log D_{\max})$-approximation of [25], and as mentioned in the introduction, our algorithm has the advantages that it can also be used in directed graphs, and is for the point-to-point version of the problem. The second algorithm immediately leads to an $O(\alpha \cdot \log \mathrm{OPT} \times \log L)$-approximation for the general time-window problem, which is already an improvement on $O(\alpha \log^2 \mathrm{OPT})$ when the ratio L is small. However, we can combine the first and second algorithms to obtain an $O(\alpha \max\{\log \mathrm{OPT}, \log L\})$-approximation for Orient-TW.

Throughout this section, we use $R(v)$ and $D(v)$ to denote (respectively) the release time and deadline of a vertex v. We also use the word *interval* to denote a time window; $I(v)$ denotes the interval $[R(v), D(v)]$. Typically, we use 'time-window' when we are interested in the start and end points of a window, and 'interval' when we think of a window as an interval along the 'time axis'. For any instance X of Orient-TW, we let $\mathrm{OPT}(X)$ denote the reward collected by an optimal solution for X. When the instance is clear from context, we use OPT to denote this optimal reward.

2.5.1 The General Framework

As described at the beginning of this section, the general method to solve Orient-TW is to reduce the problem to a set of sub-problems without time-windows. Given an instance of Orient-TW on a graph $G(V, E)$, suppose $\{V_1, V_2, \ldots V_m\}$ partition V, and we can associate times R_i and D_i with each V_i such that each of the following conditions holds:

- For each $v \in V_i$, $R(v) \leq R_i$ and $D(v) \geq D_i$.
- For $1 \leq i < m$, $D_i < R_{i+1}$.
- An optimal solution visits any vertex in V_i during $[R_i, D_i]$.

Then, we can solve an instance of Orienteering in each V_i separately, and combine the solutions using dynamic programming. The approximation ratio for such "composite" solutions would be the same as the approximation ratio for Orienteering. We refer to an instance of Orient-TW in which we can construct such a partition of the vertex set (and solve the sub-problems separately) as a *modular instance*. Later, we describe a dynamic program that can solve modular instances.

Unfortunately, given an arbitrary instance of Orient-TW, it is unlikely to be a modular instance. Therefore, we define restricted versions of a given instance:

Definition 2.26. *Let A and B be instances of* Orient-TW *on the same underlying graph (with the same edge-weights), and let $I_A(v)$ and $I_B(v)$ denote the intervals for vertex v in instances A and B respectively. We say that B is a* restricted version *of A if, for every vertex v, $I_B(v)$ is a sub-interval of $I_A(v)$.*

Clearly, a walk that gathers a certain reward in a restricted version of an instance will gather at least that reward in the original instance. We attempt to solve ORIENT-TW by constructing a set of restricted versions that are easier to work with. Typically, the construction is such that the reward of an optimal solution in at least one of the restricted versions is a significant fraction of the reward of an optimal solution in the original instance. Hence, an approximation to the optimal solution in the 'best' restricted version leads us to an approximation for the original instance.

This idea leads us to the next proposition, the proof of which is straightforward, and hence omitted.

Proposition 2.27. *Let A be an instance of* ORIENT-TW *on a graph $G(V, E)$. If $B_1, B_2, \ldots B_\beta$ are restricted versions of A, and for all vertices $v \in V$, $I_A(v) = \bigcup_{1 \leq i \leq \beta} I_{B_i}(v)$, there is some B_j such that* $\mathrm{OPT}(B_j) \geq \frac{\mathrm{OPT}(A)}{\beta}$.

The restricted versions we construct will usually be modular instances of ORIENT-TW. Therefore, the general algorithm for ORIENT-TW is:

1. Construct a set of β restricted versions of the given instance; each restricted version is a modular instance.

2. Pick the best restricted version (enumerate over all choices), find an appropriate partition, and use an α-approximation for ORIENTEERING together with dynamic programming to solve that instance.

It follows from the previous discussion that this gives a $(\alpha \times \beta)$-approximation for ORIENT-TW. We next describe how to solve modular instances of ORIENT-TW.

A dynamic program for modular instances

Recall that a modular instance is an instance of ORIENT-TW on a graph $G(V, E)$ in which the vertex set V can be partitioned into $V_1, V_2, \ldots V_m$, such that an optimal solution visits vertices of V_i after time R_i and before D_i. For any vertex $v \in V_i$, $R(v) \leq R_i$ and $D(v) \geq D_i$. Further, vertices of V_i are visited before vertices of V_j, for all $j > i$.

To solve a modular instance, for each V_i we could 'guess' the first and last vertex visited by an optimal solution, and guess the times at which this solution visits the first and last vertex. If α is

the approximation ratio of an algorithm for orienteering, we find a path in each V_i that collects an α-fraction of the optimal reward, and combine these solutions.

More formally, one could use the following dynamic program: For any $u, v \in V_i$, consider the graph induced by V_i, and let $\mathrm{OPT}(u, v, t)$ denote the optimal reward collected by any walk from u to v of length at most t (ignoring time-windows). Now, define $\Pi_i(v, T)$ for $v \in V_i, R_i \leq T \leq D_i$ as the optimal reward collected by any walk in G that begins at s at time 0, and ends at v by time T. Given $\mathrm{OPT}(u, v, t)$, the following recurrence allows us to easily compute $\Pi_i(v, T)$:

$$\Pi_i(v, T) = \max_{u \in V_i, w \in V_{i-1}, t \leq T - R_i} \mathrm{OPT}(u, v, t) + \Pi_{i-1}(w, T - t - d(w, u)).$$

Of course, we cannot exactly compute $\mathrm{OPT}(u, v, t)$; instead, we use an α-approximation algorithm for Orienteering to compute an approximation to $\mathrm{OPT}(u, v, t)$ for all $u, v \in V_i$, $t \leq D_i - R_i$. This gives an α-approximation to $\Pi_i(v, T)$ using the recurrence above.

Unfortunately, the running time of this algorithm depends polynomially on T; this leads to a pseudo-polynomial algorithm. To obtain a polynomial-time algorithm, we use a standard technique of dynamic programming based on reward instead of time (see [31, 57]). Using standard scaling tricks for maximization problems, one can reduce the problem with arbitrary rewards on the vertices to the problem where the reward on each vertex is 1; the resulting loss in approximation can be made $(1 + o(1))$. Thus, the maximum reward is n.

To construct a dynamic program based on reward instead of time, we wish to find, for each $u, v \in V_i$ and each $k_i \in [0, |V_i|]$, an optimal (shortest) walk from u to v that collects reward at least k_i. However, we cannot do this exactly. One could try to find an *approximately* shortest $u - v$ walk collecting reward k_i, but this does not lead to a good solution overall: taking slightly too much time early on can have bad consequences for later groups V_j. Instead, we "guess" the length (using binary search over the maximum walk length in G) of an optimal walk that obtains reward k_i, and for each guess use the α-approximate Orienteering algorithm. This guarantees that if there is a u-v walk of length B that collects reward k_i, then we find a u-v walk of length at most B that collects reward at least k_i/α. Finally, to obtain the desired approximation for the entire instance, we stitch together the solutions from each V_i using a dynamic program very similar to the one described above based on time.

2.5.2 The Algorithms

Using the framework described above, we now develop algorithms which achieve approximation ratios depending on the lengths of the time-windows. We first consider instances where all time-windows have integral end-points, and then instances for which the ratio $L = \frac{L_{\max}}{L_{\min}}$ is bounded. Finally, we combine these ideas to obtain an $O(\alpha \max\{\log \mathrm{OPT}, \log L\})$-approximation for all instances of ORIENT-TW.

An $O(\alpha \log L_{\max})$-approximation when Interval Endpoints are Integral

We now focus on instances of ORIENT-TW in which, for all vertices v, $R(v)$ and $D(v)$ are integers. Our algorithm is based on the following simple lemma:

Lemma 2.28. *Any interval of length $M > 1$ with integral endpoints can be partitioned into at most $2\lfloor \log M \rfloor$ disjoint sub-intervals, such that the length of any sub-interval is a power of 2, and any sub-interval of length 2^i begins at a multiple of 2^i. Further, there are at most 2 sub-intervals of each length.*

Proof. Use induction on the length of the interval. The lemma is clearly true for intervals of length 2 or 3. Otherwise, use at most 2 sub-intervals of length 1 at the beginning and end of the given interval, so that the *residual* interval (after the sub-intervals of size 1 are deleted) begins and ends at an even integer. To cover the residual interval, divide all integers in the (residual) problem by 2, and apply the induction hypothesis; we use at most $2 + (2\lfloor \log M/2 \rfloor) \leq 2\lfloor \log M \rfloor$ sub-intervals in total. It is easy to see that we use at most 2 sub-intervals of each length; intervals of length 2^i are used at the $(i+1)$th level of recursion. □

For ease of notation, we let ℓ denote $\lfloor \log L_{\max} \rfloor$ for the rest of this sub-section, and assume for ease of exposition that $L_{\max} \geq 2$. Given an instance of ORIENT-TW, for each vertex v with interval $I(v)$, we use Lemma 2.28 to partition $I(v)$ into at most 2ℓ sub-intervals. We label the sub-intervals of $I(v)$ as follows: For each $1 \leq i \leq \ell$, the first sub-interval of length 2^i is labeled $I_i^1(v)$ and the second sub-interval $I_i^2(v)$. (Note that there may be no sub-intervals of length 2^i.)

We now construct a set of at most 2ℓ restricted versions of the given instance. We call these restricted versions $B_1^1, B_2^1, \ldots B_\ell^1$ and $B_1^2, B_2^2, \ldots B_\ell^2$, such that the interval for vertex v in B_i^b is

$I_i^b(v)$. If $I_i^b(v)$ was not an interval used in the partition of $I(v)$, v is not present in the restricted version. (Equivalently, it has reward 0 or an empty time-window.)

Consider an arbitrary restricted instance B_i^b. All vertices in this instance of ORIENT-TW have intervals of length 2^i, and all time-windows begin at an integer that is a multiple of 2^i. Hence, any 2 vertices either have time-windows that are identical, or entirely disjoint. This means that B_i^b is a modular instance, so we can break it into sub-problems, and use an α-approximation to ORIENTEERING in the sub-problems to obtain an α-approximation for the restricted instance.

By Proposition 2.27, one of the restricted versions has an optimal solution that collects reward at least $\frac{\text{OPT}}{2\ell}$. Using an α-approximation for this restricted version gives us an $\alpha \times 2\ell = O(\alpha \log L_{\max})$-approximation for ORIENT-TW when all interval endpoints are integers.

An $O(\alpha \log \text{OPT})$-approximation when $L \leq 2$

For an instance of ORIENT-TW when $L = \frac{L_{\max}}{L_{\min}} \leq 2$, we begin by scaling all release times, deadlines, and edge lengths so that $L_{\min} = 1$ (and so $L_{\max} \leq 2$). Note that even if all release times and deadlines were integral prior to scaling, they may not be integral in the scaled version; after scaling, all interval lengths are in $[1, 2]$.

For each vertex v, we partition $I(v) = [R(v), D(v)]$ into 3 sub-intervals: $I_1(v) = [R(v), a]$, $I_2(v) = [a, b]$, and $I_3(v) = [b, D(v)]$, where $a = \lfloor R(v) + 1 \rfloor$ (that is, the next integer strictly greater than the release time) and $b = \lceil D(v) - 1 \rceil$ (the greatest integer strictly less than the deadline). The figure below illustrates the partitioning of intervals. Note that $I_2(v)$ may be a point, and in this case, we ignore such a sub-interval.

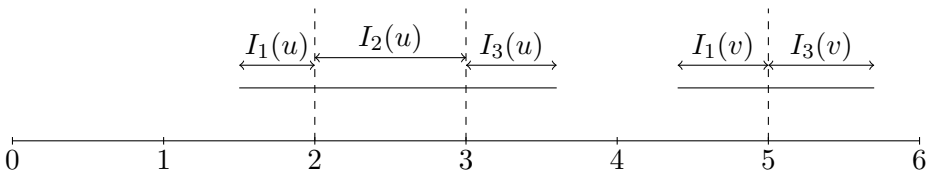


Figure 2.4: The partitioning of 2 intervals into sub-intervals. Note that on the right, $I_2(v)$ is empty.

We now construct 3 restricted versions of the given instance — B_1, B_2, and B_3 — such that the interval for any vertex v in B_i is simply $I_i(v)$. By Proposition 2.27, one of these has an optimal

solution that collects at least a third of the reward collected by an optimal solution to the original instance. Suppose this is B_2. All time-windows have length exactly 1, and start and end-points are integers. Therefore, B_2 is a modular instance, and we can get an α-approximation to the optimal solution in B_2; this gives a 3α-approximation to the original instance.

Dealing with B_1 and B_3 is not nearly as easy; they are not quite modular. Every interval in B_1 has length at most 1, and ends at an integer; for B_3, intervals have length at most 1 and start at an integer. We illustrate how to approximate a solution for B_3 within a factor of $O(\alpha \log \mathrm{OPT})$; the algorithm for B_1 is identical except that release times and deadlines are to be interchanged.

For B_3, we can partition the vertex set into $V_1, V_2, \ldots V_m$, such that all vertices in V_i have the same (integral) release time, and any vertex in V_i is visited before any vertex in V_j for $j > i$. Figure 2.5 shows such a partition. The deadlines for vertices in V_i may be all distinct. However, we can solve an instance of ORIENT-DEADLINE in each V_i separately, and paste the solutions together using dynamic programming. The solution we obtain will collect at least $\Omega(1/\alpha \log \mathrm{OPT})$ of the reward of an optimal solution for B_3, since there is a $O(\log \mathrm{OPT})$-approximation for ORIENT-DEADLINE ([25]). Therefore, this gives us a $3 \times O(\alpha \log \mathrm{OPT}) = O(\alpha \log \mathrm{OPT})$-approximation to the original instance.

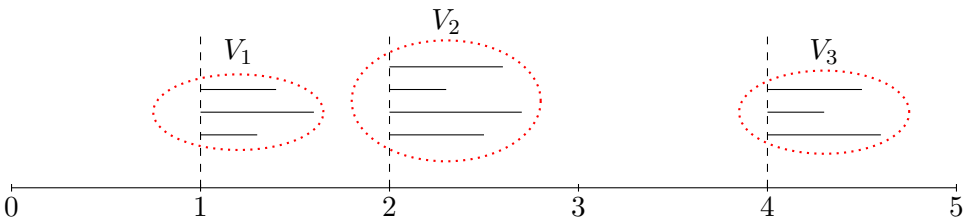


Figure 2.5: In B_3, all time-windows start at an integer and have length at most 1. Each setof vertices whose windows have a common beginning corresponds to a sub-problem that is an instance of orienteering with deadlines.

Similarly, we can use the $O(\log \mathrm{OPT})$-approximation for ORIENTEERING with release times to obtain an $O(\alpha \log \mathrm{OPT})$-approximation for B_1 . Therefore, when $L \leq 2$, we have an $O(\alpha \log \mathrm{OPT})$-approximation for ORIENT-TW.

Putting the pieces together

An arbitrary instance of ORIENT-TW may have $L > 2$, and interval end-points may not be integers. However, we can combine the algorithms from the two preceding sections to deal with such instances. We begin by scaling release times, deadlines, and edge lengths such that the shortest interval has length 1; the longest interval now has length $L = \frac{L_{\max}}{L_{\min}}$, where $L_{\max}$ and $L_{\min}$ are the lengths of the longest and shortest intervals in the original instance.

We now construct 3 restricted versions of the scaled instance: B_1, B_2, and B_3. For any vertex v with interval $[R(v), D(v)]$ in the scaled instance, we construct 3 sub-intervals. If the interval for v has length less than 2, we set $I_1(v) = [R(v), D(v)]$, and $I_2(v) = I_3(v) = \emptyset$. Otherwise, $I_1(v) = [R(v), a]$, $I_2(v) = [a, b]$, and $I_3(v) = [b, D(v)]$, where $a = \lceil R(v) + 1 \rceil$ and $b = \lfloor D(v) - 1 \rfloor$. As before, the interval for v in the instance B_i is $I_i(v)$.

One of the restricted versions collects at least a third of the reward of the original instance. Suppose this is B_1 or B_3. All intervals in B_1 and B_3 have length between 1 and 2 by our construction. Therefore, we can use the $O(\alpha \log \mathrm{OPT})$-approximation algorithm from section 2.5.2 to collect at least $\Omega(1/\alpha \log \mathrm{OPT})$ of the reward of an optimal solution to the original instance. It now remains only to consider the case that B_2 collects more than a third of the reward. In B_2, the end-points of all time-windows are integral, and the longest interval has length less than L. We can now use the algorithm of section 2.5.2 to obtain an $O(\alpha \log L)$-approximation.

Therefore, our combined algorithm is an $O(\alpha \max\{\log \mathrm{OPT}, \log L\})$-approximation for ORIENT-TW, proving Theorem 2.4.

2.5.3 Towards a Better Approximation, and Arbitrary Endpoints

In the previous sub-section, we obtained an approximation ratio of $O(\alpha \max\{\log \mathrm{OPT}, \log L\})$; we would like to improve this ratio to $O(\alpha \log \mathrm{OPT})$. Unfortunately, it does not seem easy to do this directly. A natural question, then, would be to obtain a ratio of $O(\alpha \log L)$; this is equivalent to an $O(\alpha)$ approximation for the case when $L \leq 2$. However, this is is no easier than finding an $O(\alpha \log \mathrm{OPT})$-approximation for arbitrary instances of ORIENT-TW, as we show in the next proposition.

Proposition 2.29. *An $O(\alpha)$ approximation algorithm for* Orient-TW *with $L \leq 2$ implies an $O(\alpha \log \mathrm{OPT})$-approximation for general instances of* Orient-TW.

Proof. We show that an $O(\alpha)$ approximation when $L \leq 2$ implies an $O(\alpha)$ approximation for Orient-Deadline. It follows from an algorithm of [25] that we can then obtain an $O(\alpha \log \mathrm{OPT})$-approximation for Orient-TW.

Given an arbitrary instance of Orient-Deadline on graph $G(V, E)$, we add a new start vertex s' to G. Connect s' to s with an edge of length $D_{\max} = \max_v D(v)$. The release time of every vertex is 0, but all deadlines are increased by $D_{\max}$. Observe that all vertices have time-windows of length between $D_{\max}$ and $2D_{\max}$, so $L \leq 2$. It is easy to see that given any walk beginning at s in the original instance, we can find an equivalent walk beginning at s' in the modified instance that visits a vertex in its time-window iff the original walk visited a vertex before its deadline in the given instance, and vice versa. Therefore, an $O(\alpha)$ approximation for the modified instance of Orient-TW gives an $O(\alpha)$ approximation for the original instance of Orient-Deadline. □

We *can*, however, obtain an $O(\alpha)$-approximation for Orient-TW when $L \leq 2$ if we remove the restriction that the walk must start and end at s and t, the specified endpoints. The algorithm of [85] for the case of $L = 1$ can be adapted relatively easily to give an $O(\alpha)$ approximation for $L \leq 2$. For completeness, we sketch the algorithm here.

We construct 5 restricted versions $B_1, \ldots B_5$, of a given instance A. For every vertex v, we create at most 5 sub-intervals of $I(v)$ by breaking it at every multiple of 0.5. (For instance $[3.7, 5.6]$ would be broken up into $[3.7, 4], [4, 4.5], [4.5, 5], [5, 5.5], [5.5, 5.6]$. Note that some intervals may have fewer than 5 sub-intervals.) The interval for v in $B_1(v)$ is the first sub-interval, and the interval in $B_5(v)$ is the last sub-interval, regardless of whether $I(v)$ has 5 sub-intervals. B_2, B_3, and B_4 each use one of any remaining sub-intervals.

B_2, B_3, and B_4 are modular instances, so if one of them is the best restricted version of A, we can use an α-approximation for orienteering to get reward at least $\frac{\mathrm{OPT}(A)}{5\alpha}$. Exactly as in Section 2.5.2, B_1 and B_5 are not quite modular instances; in B_1, all deadlines are half-integral but release times are arbitrary, and in B_5, all release times are half-integral, but deadlines are arbitrary.

Suppose that B_1 is the best restricted version. The key insight is that if the optimal walk in B_1 collects a certain reward starting at s at time 0, there is a walk in B_2 starting at s at time 0.5 that

collects *the same* reward. (This is the substance of Theorem 1 of [85].) Therefore, if B_1 is the best restricted version, we find an α-approximation to the best walk in B_2 starting at s at time 0.5; we are guaranteed that this walk collects reward at least $\frac{\mathrm{OPT}(A)}{5\alpha}$. Note that this walk may not reach the destination vertex t by the time limit, since we start 0.5 time units late. Similarly, if B_5 is the best restricted version, we can find a walk in B_4 that collects reward $\frac{\mathrm{OPT}(A)}{5\alpha}$ while beginning at s at time -0.5. (To avoid negative times, we can begin the walk at s' at time 0, where s' is the first vertex visited by the original walk after time 0.) This walk is guaranteed to reach t by the time limit, but does not necessarily begin at s.

Therefore, this algorithm is an $O(\alpha)$-approximation when $L \leq 2$, or an $O(\alpha \log L)$-approximation for general instances of Orient-TW. We note that one *cannot* use this with Proposition 2.29 to get an $O(\alpha \log \mathrm{OPT})$-approximation for the variant of Orient-TW where start/end vertices are not specified: The dynamic program for modular instances crucially uses the fact that we can specify both endpoints for the sub-problems.

2.6 Concluding Remarks

In this chapter, we gave an improved constant-factor approximation algorithm for Orienteering in undirected graphs, and the first polynomial-time approximation algorithm for Orienteering in directed graphs. These results were based on new algorithms for related problems such as k-Stroll and k-TSP. Also, the algorithms presented in this chapter can be combined with previously known techniques [31, 25, 56] to obtain the following results:

- A $(4+\varepsilon)$ approximation for the Max-Prize Tree problem in undirected graphs: Find a tree rooted at a given vertex s of total length at most B that maximizes the number of vertices in the tree. This improves the 6-approximation in [31, 25].
- A $(3+\varepsilon)$ approximation for Orient-TW when there are a fixed number of time windows; this improves a ratio of 4 from [56].

We also considered Orient-TW, and gave an $O(\alpha \max\{\log \mathrm{OPT}, \log L\})$ approximation in both directed and undirected graphs, where α denotes the approximation ratio for Orienteering. Particularly in undirected graphs, this goes some way to proving Conjecture 2.3, that there is an

$O(\log \mathrm{OPT})$-approximation for Orient-TW. Specifically, our results imply that for hard instances of Orient-TW, any near-optimal solution must visit many vertices of *widely* differing time lengths. We believe that our insights will aid in resolving the conjecture.

One problem with most current approaches to Orient-TW is that the algorithms for Orient-TW use the Orienteering algorithm in a black-box fashion. For directed graphs the current ratio for Orienteering is $O(\log^2 \mathrm{OPT})$ and hence the ratio for Orient-TW is worse by additional logarithmic factors. Can one avoid using Orienteering as a black-box? We note that the quasi-polynomial time algorithm of [57] deals with time windows directly, and hence has the same approximation ratio of $O(\log \mathrm{OPT})$ for both the basic orienteering and time-window problems in directed graphs.

Besides obtaining improved algorithms, it would be of interest to show that Orienteering and related problems are hard to approximate. Blum *et al.* [31] showed that Orienteering is $\mathcal{APX}$-Hard; in particular, they proved that it is $\mathcal{NP}$-Hard to obtain an approximation ratio better than $1481/1480 \approx 1.0007$. Few additional hardness of approximation results are known. In fact, no super-constant hardness is known even for Orient-TW in directed graphs, where the best current approximation ratio is $O(\log^4 \mathrm{OPT})$! We believe it should be possible to narrow this large gap between the upper and lower bounds on the approximability of Orient-TW.

Finally, we list a few additional open problems:

1. Is there a 2-approximation for Orienteering in undirected graphs? In addition to matching the known ratios for k-MST and k-TSP [88], this may lead to a more efficient algorithm than the one presented in this chapter.

2. Is there an $O(1)$ approximation for Orienteering in directed graphs?

3. What is the approximability of k-Stroll in directed graphs? Until this year, when Bateni and Chuzhoy [27] gave a $\min\{O(\log^2 n \log k / \log\log n), O(\log^4 k)\}$-approximation, only bi-criteria approximation algorithms were known. A natural question is whether their techniques or others can be extended to obtain an $O(\log^2 n)$ or $O(\log^2 k)$-approximation, matching the approximation ratio for Orienteering.

4. Our $(2+\varepsilon)$ approximation for ORIENTEERING gives a $(4+\varepsilon)$ approximation to the MAX-PRIZE TREE problem defined above ; a 3 approximation for the unrooted version follows from [42]. Can the approximation factor for the rooted version be improved to 3, or $(2+\varepsilon)$?

5. For many applications, each vertex has multiple disjoint time-windows, and we receive credit for a vertex if we visit it within any of its windows. If each vertex has at most k windows, a naïve algorithm loses an extra factor of k beyond the ratio for ORIENT-TW, but no better approximation is known. Any non-trivial result would be of interest. We note that the quasi-polynomial time $O(\log \mathrm{OPT})$-approximation of [57] obtains this same approximation ratio even for the problem where each vertex has multiple disjoint time windows.

Chapter 3

Finding 2-Connnected Subgraphs of a Prescribed Size

3.1 Introduction [1]

Problems related to finding low-cost connected subgraphs containing many vertices arise naturally in connectivity and network design, and have numerous applications. Perhaps the best known example is Minimum Spanning Tree, where the goal is to find a minimum-cost connected subgraph containing all the vertices of a given graph. Another such problem is the k-MST problem, introduced by Ravi *et al.* [142]: Given an edge-weighted graph G and an integer k, the goal is to find a minimum-cost subgraph (without loss of generality, a tree) of G that contains at least k vertices. It is not hard to see that the k-MST problem is at least as hard as the Steiner Tree problem; moreover an α-approximation for the k-MST problem implies an α-approximation for Steiner Tree. The k-MST problem has attracted considerable attention in the approximation algorithms literature and its study has led to several new algorithmic ideas and applications [20, 87, 15, 88, 42, 31]. Closely related to the k-MST problem is the budgeted or Max-Prize Tree problem [112, 31]; here we are given G and a budget B, and the goal is to find a subgraph H of G of total cost no more than B, that maximizes the number of vertices (or terminals) in H. Interestingly, it is only recently that the rooted version of Max-Prize Tree was shown to have an $O(1)$-approximation [31], although an $O(1)$ approximation was known for the k-MST problem much earlier [32].

Algorithms for k-MST and Max-Prize Tree are extremely useful in the design of algorithms for more complex problems. As merely one example, the best algorithms for the k-Stroll problem, defined in Chapter 2, are derived from approximation algorithms for k-MST. (See Theorem 2.14 in Section 2.3.2, for instance.) This approach also leads to improved algorithms for the Min-Excess and Orienteering problems, as discussed in Chapter 2. In addition to their theoretical interest,

[1] This chapter is based on joint work with Chandra Chekuri, and has appeared in [54].

problems such as k-MST and MAX-PRIZE TREE are partly motivated by applications in network design and related areas where one may want to build low-cost networks including (or servicing) many clients, but there are constraints such as a budget on the network cost, or a minimum quota on the number of clients. However, a network with a tree-like topology is far from robust; the failure of *any* edge will leave it disconnected. Thus, if fault-tolerance is desired, there is a need for large, low-cost networks with higher connectivity.

Recently, Lau *et al.* [126] considered the natural generalization of k-MST to higher connectivity. In particular they defined the (k, λ)-Subgraph problem to be the following: Find a minimum-cost subgraph of the given graph G that contains at least k vertices and is λ-edge connected. We use the notation k-λEC to refer to this problem. In [126, 127] a poly-logarithmic approximation was derived for the k-2EC problem. In this chapter, we consider the vertex-connectivity generalizations of the k-MST and MAX-PRIZE TREE problems. We define the k-λVC problem as follows: Given an integer k and a graph G with edge costs, find the minimum-cost λ-vertex-connected subgraph of G that contains at least k vertices. Similarly, in the BUDGET-λVC problem, given a budget B and a graph G with edge costs, the goal is to find a λ-vertex-connected subgraph of G of cost at most B, that maximizes the number of vertices it contains. In particular we focus on $\lambda = 2$ and develop approximation algorithms for both the k-2VC and BUDGET-2VC problems. We note that the k-λEC problem reduces to the k-λVC problem in an approximation preserving fashion, though the opposite reduction is not known. The k-λEC and k-λVC problems are NP-hard and also APX-hard for any $\lambda \geq 1$. Moreover, Lau *et al.* [126] give evidence that, for large λ, the k-λEC problem is likely to be harder to approximate by relating it to the approximability of the DENSE k-SUBGRAPH problem [77].

How do we solve these problems? The k-MST problem required several algorithmic innovations which eventally led to the current best approximation ratio of 2 [88]. The main technical tool which underlies $O(1)$ approximations for k-MST [32, 87, 66, 88] is a special property that holds for a LP relaxation of the PRIZE-COLLECTING STEINER TREE problem [92] which is a Lagrangian relaxation of the k-MST problem. Unfortunately, one cannot use these ideas (at least directly) for more general problems such as k-2VC (or the k-STEINER FOREST problem [101]) since the LP relaxation for the prize-collecting variant is not known to satisfy the above mentioned property.

We therefore rely on alternative techniques that take a more basic approach.

Our algorithms for k-2VC and BUDGET-2VC use the same high-level idea and rely on the notion of *density*: the density of a subgraph is the ratio of its cost to the number of vertices it contains. The algorithms greedily combine subgraphs of low density until the union of these subgraphs has the desired number of vertices or has cost equal to the budget. They fail only if we find a subgraph H of good density, but that is far too large. One needs, then, a way to *prune* H to find a smaller subgraph of comparable density. Our main structural result for pruning 2-connected graphs is the following[2]:

Theorem 3.1. *Let G be a 2-connected edge-weighted graph with density ρ, and a designated vertex $r \in V(G)$ such that every vertex of G has 2 vertex-disjoint paths to r of total weight/cost at most L. There is a polynomial-time algorithm that, given any integer $k \leq |V(G)|$, finds a 2-connected k-vertex subgraph H of G containing r, of total cost at most $O(\log k)\rho k + 2L$.*

Intuitively, the algorithm of Theorem 3.1 allows us to find a subgraph of *any* desired size, at the cost of only a logarithmic increase in density. Further, it allows us to require any vertex r to be in the subgraph, and also applies if we are given a *terminal* set S, and the output subgraph must contain k terminals. (In this case, the density of a subgraph is the ratio of its cost to the number of terminals it contains.) In addition, it applies if the terminals/vertices have arbitrary weights, and the density of a subgraph is the ratio of its cost to the sum of the weights of its terminals. All our algorithms apply to these weighted instances, but for simplicity of exposition, we discuss the more restricted unweighted versions throughout. We observe that pruning a tree (a 1-connected graph) is easy and one loses only a constant factor in the density; the theorem above allows one to prune 2-connected graphs. A technical ingredient that we develop is the following theorem: we believe that Theorems 3.1 and 3.2 are interesting in their own right and will find other applications besides algorithms for k-2VC and BUDGET-2VC.

Theorem 3.2. *Let G be a 2-vertex-connected graph with edge costs and let $S \subseteq V$ be a set of terminals. Then, there is a simple cycle C containing at least 2 terminals (a non-trivial cycle) such that the density of C is at most the density of G. Moreover, such a cycle can be found in polynomial time.*

[2]In fact, we prove the slightly stronger Theorem 3.13; see Section 3.4.

Using the above theorem and an LP approach we obtain the following.

Corollary 3.3. *Given a graph $G(V, E)$ with edge costs and a set of ℓ terminals $S \subseteq V$, there is an $O(\log \ell)$ approximation for the problem of finding a minimum-density non-trivial cycle.*

Note that Theorem 3.2 and Corollary 3.3 are of interest because we seek a cycle with at least *two* terminals. A minimum-density cycle containing only one terminal can be found by using the well-known min-mean cycle algorithm in directed graphs [4]. We remark, however, that although we suspect that the problem of finding a minimum-density non-trivial cycle is NP-hard, we currently do not have a proof. Theorem 3.2 shows that the problem is equivalent to the DENS-2VC problem, defined in the next section.

Armed with these useful structural results, we give approximation algorithms for both the k-2VC and BUDGET-2VC problems. Our results in fact hold for the more general versions of these problems where the input also specifies a subset $S \subseteq V$ of *terminals* and the goal is to find subgraphs with the desired number of terminals, or to maximize the number of terminals.[3]

Theorem 3.4. *There is an $O(\log \ell \cdot \log k)$ approximation for the k-2VC problem, where ℓ is the number of terminals.*

Corollary 3.5. *There is an $O(\log \ell \cdot \log k)$ approximation for the k-2EC problem, where ℓ is the number of terminals.*

Theorem 3.6. *There is a polynomial time bi-criteria approximation algorithm for* BUDGET-2VC *that, for any $0 < \varepsilon \leq 1$, outputs a subgraph of edge-weight $(3 + \varepsilon)B$ containing $\Omega(\frac{\varepsilon \cdot \mathrm{OPT}}{\log n \log \mathrm{OPT}})$ terminals, where* OPT *is the number of terminals in an optimum solution of cost B.*[4]

As mentioned before, the k-2EC problem was introduced by Lau *et al.* and an $O(\log^3 k)$ approximation was claimed for this problem in [126]. However, the algorithm and proof in [126] are incorrect. In independent work from ours, the authors of [126] later obtained a different algorithm for k-2EC that yields an $O(\log n \log k)$ approximation [127]; however, their algorithm does not

[3]For k-2EC and k-λEC, the problem with specified terminal set S can be reduced to the problem where every vertex in V is a terminal. Such a reduction does not seem possible for the k-2VC and k-λVC, so we work directly with the terminal version.

[4]For the *rooted* version of BUDGET-2VC (see Section 3.5), we obtain a subgraph of weight $(2 + \varepsilon)B$ with this number of terminals.

generalize to k-2VC. We give a more detailed comparison of the differences between their approach and ours in the next subsection. Subsequent to the work described in this chapter, Salavatipour and Safari [145] gave a constant-factor approximation for the k-λEC problem when the input graph is complete, with metric edge costs. More recently, Gupta, Krishnaswamy and Ravi [98] gave a weaker $O(\log^3 n)$ approximation algorithm for k-2EC based on tree-embedding techniques; this algorithm also does not appear to generalize to k-2VC.

3.1.1 Overview of Technical Ideas

For this section, we focus on the rooted version of k-2VC: the goal is to find a min-cost subgraph that 2-connects at least k terminals to a specified root vertex r. It is relatively straightforward to reduce k-2VC to its rooted version (see Section 3.5 for details). We draw inspiration from algorithmic ideas that led to poly-logarithmic approximations for the k-MST problem. As described above, our approach focuses on the idea of low-density subgraphs.

For a subgraph H that contains r, let $k(H)$ be the number of terminals that are 2-connected to r in H. Then the *density* of H is simply the ratio of the cost of H to $k(H)$. The Dens-2VC problem is to find a 2-connected subgraph of minimum density. An $O(\log \ell)$ approximation for the Dens-2VC problem (where ℓ is the number of terminals) can be derived in a somewhat standard way by using a bucketing and scaling trick on a linear programming relaxation for the problem. We exploit the known bound of 2 on the integrality gap of a natural LP for the Survivable Network Design Problem with vertex connectivity requirements in $\{0, 1, 2\}$ [79]. The bucketing and scaling trick has seen several uses in the past and has recently been highlighted in several applications [48, 47, 46].

Our algorithm for k-2VC uses a greedy approach at the high level. We start with an empty subgraph G' and use the approximation algorithm for Dens-2VC in an iterative fashion to greedily add terminals to G' until at least $k' \geq k$ terminals are in G'. This approach would yield an $O(\log \ell \log k)$ approximation if $k' = O(k)$. However, the last iteration of the Dens-2VC algorithm may add many more terminals than desired with the result that $k' \gg k$. In this case we cannot bound the cost of the solution obtained by the algorithm. To overcome this problem, one can try to *prune* the subgraph H added in the last iteration to only have the desired number of terminals. For the k-MST problem, H is a tree and pruning is quite easy. We remark that this yields a

rather straightforward $O(\log n \log k)$ approximation for k-MST and could have been discovered much before a more clever analysis given in [20].

Our main technical contribution is Theorem 3.1, to give a pruning step for the k-2VC problem. To accomplish this, we use two algorithmic ideas. The first is encapsulated in the cycle finding algorithm of Theorem 3.2. Second, we use this cycle finding algorithm to repeatedly merge subgraphs until we get the desired number of terminals in one subgraph; this latter step requires care. The cycle merging scheme is inspired by a similar approach from the work of Lau *et al.* [126] on the k-2EC problem and in our previous work [55] on directed ORIENTEERING. These ideas yield an $O(\log \ell \cdot \log^2 k)$ approximation. We give a modified cycle-merging algorithm with a more sophisticated and non-trivial analysis to obtain an improved $O(\log \ell \cdot \log k)$ approximation.

Some remarks are in order to compare our work to that of [126] on the k-2EC problem. The combinatorial algorithm in [126] is based on finding a low-density cycle or a related structure called a bi-cycle. The algorithm in [126] to find such a structure is incorrect. Further, the cycles are contracted along the way which limits the approach to the k-2EC problem (contracting a cycle in a 2-vertex-connected graph may make the resulting graph no longer 2-vertex-connected). In our algorithm we do not contract cycles and instead introduce dummy terminals with weights to capture the number of terminals in an already formed component. This requires us to address the minimum-density non-trivial simple cycle problem which we do via Theorem 3.2 and Corollary 3.3. In independent work, Lau *et al.* [127] obtain a new and correct $O(\log n \log k)$-approximation for k-2EC. They also follow the same approach that we do in using the LP for finding dense subgraphs followed by the pruning step. However, in the pruning step they use a very different approach; they use the sophisticated idea of nowhere-zero 6-flows [147]. Although the use of this idea is elegant, the approach works only for the k-2EC problem, while our approach is less complex and leads to an algorithm for the more general k-2VC problem.

Chapter Outline

We begin in Section 3.2 by giving an $O(\log \ell)$-approximation for the DENS-2VC problem, in which the goal is to find a minimum-density 2-connected subgraph of a given graph. Then, in Section 3.3, we prove Theorem 3.2 and Corollary 3.3 on finding good-density cycles; we first show the existence

of such cycles, and then give an efficient algorithm to find them. Using Theorem 3.2, we can prove our main technical result on pruning, Theorem 3.1, in Section 3.4. Finally, in Section 3.5, we show how to combine the pruning algorithm of Theorem 3.1 with the algorithm for DENS-2VC to obtain good algorithms for k-2VC and BUDGET-2VC.

3.2 An $O(\log \ell)$-Approximation for the DENS-2VC Problem

Recall that the DENS-2VC problem was defined as follows: Given a graph $G(V, E)$ with edge-costs, a set $T \subseteq V$ of terminals, and a root $r \in V(G)$, find a subgraph H of minimum density, in which every terminal of H is 2-connected to r. (Here, the density of H is defined as the cost of H divided by the number of terminals it contains, not including r.) In this section, we prove the following lemma:

Lemma 3.7. *There is an $O(\log \ell)$-approximation algorithm for the* DENS-2VC *problem, where ℓ is the number of terminals in the given instance.*

Our proof of this lemma uses an LP based approach and a bucketing and scaling trick (see [46, 48, 47] for applications of this idea), and a constant-factor bound on the integrality gap of an LP for SNDP with vertex-connectivity requirements in $\{0, 1, 2\}$ due to Fleischer, Jain and Williamson [79].

We define **LP-dens** as the following LP relaxation of DENS-2VC. For each terminal t, the variable y_t indicates whether or not t is chosen in the solution. (By normalizing $\sum_t y_t$ to 1, and minimizing the sum of edge costs, we minimize the density.) $\mathcal{C}_t$ is the set of all simple cycles containing t and the root r; for any $C \in \mathcal{C}_t$, f_C indicates how much 'flow' is sent from v to r through C. (Note that a pair of vertex-disjoint paths is a cycle; the flow along a cycle is 1 if we can 2-connect t to r using the edges of the cycle.) The variable x_e indicates whether the edge e is used by the solution.

$$\min \sum_{e \in E} c(e) x_e$$

$$\sum_{t \in T} y_t = 1$$

$$\sum_{C \in \mathcal{C}_t} f_C \geq y_t \qquad (\forall t \in T)$$

$$\sum_{C \in \mathcal{C}_t | e \in C} f_C \leq x_e \qquad (\forall t \in T, e \in E)$$

$$x_e, f_C, y_t \geq 0 \qquad (\forall e \in E, t \in T, C \in \mathcal{C}_t)$$

It is not hard to see that an optimal solution to **LP-dens** has cost at most the density of an optimal solution to Dens-2VC: Given a feasible solution H to Dens-2VC in which a set of terminals $T' \subseteq T$ is connected to the root, set $y_t = 1/|T'|$ for each terminal $t \in T'$, set $x_e = 1/|T'|$ for each edge e in $E(H)$, and for each $t \in T'$, pick two disjoint paths from t to the root and set $f_C = 1/|T'|$ on the cycle C formed by these two paths. We now show how to obtain an integral solution of density at most $O(\log \ell)\mathrm{OPT}_{LP}$, where OPT_{LP} is the cost of an optimal solution to **LP-dens**. The linear program **LP-dens** has an exponential number of variables but only a polynomial number of non-trivial constraints; hence, it can be solved in polynomial time. Fix an optimal solution to **LP-dens** of cost OPT_{LP}, and for each $0 \leq i < 2 \log \ell$ (for ease of notation, assume $\log \ell$ is an integer), let Y_i be the set of terminals t such that $2^{-(i+1)} < y_t \leq 2^{-i}$. Since $\sum_{t \in T} y_t = 1$, there is some index i such that $\sum_{t \in Y_i} y_t \geq \frac{1}{2 \log \ell}$. Since every terminal $t \in Y_i$ has $y_t \leq 2^{-i}$, the number of terminals in Y_i is at least $\frac{2^{i-1}}{\log \ell}$. We claim that there is a subgraph H of G with cost at most $O(2^{i+2}\mathrm{OPT}_{LP})$, in which every terminal of Y_i is 2-connected to the root. If this is true, the density of H is at most $O(\log \ell \cdot \mathrm{OPT}_{LP})$, and hence we have an $O(\log \ell)$-approximation for the Dens-2VC problem.

To prove our claim about the cost of the subgraph H in which every terminal of Y_i is 2-connected to r, consider scaling up the given optimum solution of **LP-dens** by a factor of 2^{i+1}. For each terminal $t \in Y_i$, the flow from t to r in this scaled solution[5] is at least 1, and the cost of the scaled

[5]This is a mild abuse of the term 'solution', since after scaling, $\sum_{t \in T} y_t = 2^{i+1}$.

solution is 2^{i+1}OPT_{LP}.

In [79], the authors describe a linear program LP_1 to find a minimum-cost subgraph in which a given set of terminals is 2-connected to the root, and show that this linear program has an integrality gap of 2. The variables x_e in the 'scaled solution' to **LP-dens** correspond to a feasible solution of LP_1 with Y_i as the set of terminals; the integrality gap of 2 implies that there is a subgraph H in which every terminal of Y_i is 2-connected to the root, with cost at most 2^{i+2}OPT_{LP}.

Therefore, the algorithm for Dens-2VC is:

1. Find an optimal fractional solution to **LP-dens**.
2. Find a set of terminals Y_i such that $\sum_{t \in Y_i} y_t \geq \frac{1}{2 \log \ell}$.
3. Find a min-cost subgraph H in which every terminal in Y_i is 2-connected to r using the algorithm of [79]. H has density at most $O(\log \ell)$ times the optimal solution to Dens-2VC.

3.3 Finding Low-Density Non-Trivial Cycles

A cycle $C \subseteq G$ is *non-trivial* if it contains at least 2 terminals. We define the min-density non-trivial cycle problem: Given a graph $G(V, E)$, with $S \subseteq V$ marked as terminals, edge costs and terminal weights, find a minimum-density cycle that contains at least 2 terminals. Note that if we remove the requirement that the cycle be non-trivial (that is, it contains at least 2 terminals), the problem reduces to the min-mean cycle problem in directed graphs, and can be solved exactly in polynomial time (see [4]). Algorithms for the min-density non-trivial cycle problem are a useful tool for solving the k-2VC and k-2EC problems. In this section, we give an $O(\log \ell)$-approximation algorithm for the minimum-density non-trivial cycle problem.

First, we prove Theorem 3.2, that a 2-connected graph with edge costs and terminal weights contains a simple non-trivial cycle, with density no more than the average density of the graph. We give two algorithms to find such a cycle; the first, described in Section 3.3.1, is simpler, but the running time is not polynomial. A more technical proof that leads to a strongly polynomial-time algorithm is described in Section 3.3.2; we recommend this proof be skipped on a first reading.

3.3.1 An Algorithm to Find Cycles of Average Density

To find a non-trivial cycle of density at most that of the 2-connected input graph G, we will start with an arbitrary non-trivial cycle, and successively find cycles of better density until we obtain a cycle with density at most Density(G). The following lemma shows that if a cycle C has an ear with density less than Density(C), we can use this ear to find a cycle of lower density.

Lemma 3.8. *Let C be a non-trivial cycle, and H an ear incident to C at u and v, such that $\frac{cost(H)}{weight(H-\{u,v\})} < Density(C)$. Let S_1 and S_2 be the two internally disjoint paths between u and v in C. Then $H \cup S_1$ and $H \cup S_2$ are both simple cycles and one of these is non-trivial and has density less than Density(C).*

Proof. C has at least 2 terminals, so it has finite density; H must then have at least 1 terminal. Let c_1, c_2 and c_H be, respectively, the sum of the costs of the edges in S_1, S_2 and H, and let w_1, w_2 and w_H be the sum of the weights of the terminals in S_1, S_2 and $H - \{u, v\}$.

Assume without loss of generality that S_1 has density at most that of S_2. (That is, $c_1/w_1 \leq c_2/w_2$.)[6] S_1 must contain at least one terminal, and so $H \cup S_1$ is a simple non-trivial cycle. The statement Density($H \cup S_1$) < Density(C) is equivalent to $(c_H + c_1)(w_1 + w_2) < (c_1 + c_2)(w_H + w_1)$.

$$\begin{aligned}
(c_H + c_1)(w_1 + w_2) &= c_1 w_1 + c_1 w_2 + c_H(w_1 + w_2) \\
&\leq c_1 w_1 + c_2 w_1 + c_H(w_1 + w_2) && (\text{Density}(S_1) \leq \text{Density}(S_2)) \\
&< c_1 w_1 + c_2 w_1 + (c_1 + c_2) w_H && (c_H/w_H < \text{Density}(C)) \\
&= (c_1 + c_2)(w_H + w_1)
\end{aligned}$$

Therefore, $H \cup S_1$ is a simple cycle containing at least 2 terminals of density less than Density(C). □

Lemma 3.9. *Given a cycle C in a 2-connected graph G, let G' be the graph formed from G by contracting C to a single vertex v. If H is a connected component of $G' - v$, $H \cup \{v\}$ is 2-connected in G'.*

[6] It is possible that one of S_1 and S_2 has cost 0 and weight 0. In this case, let S_1 be the component with non-zero weight.

Proof. Let H be an arbitrary connected component of $G' - v$, and let $H' = H \cup \{v\}$. To prove that H' is 2-connected, we first observe that v is 2-connected to any vertex $x \in H$. (Any set that separates x from v in H' separates x from the cycle C in G.)

It now follows that for all vertices $x, y \in V(H)$, x and y are 2-connected in H'. Suppose deleting some vertex u separates x from y. The vertex u cannot be v, since H is a connected component of $G' - v$. But if $u \neq v$, v and x are in the same component of $H' - u$, since v is 2-connected to x in H'. Similarly, v and y are in the same component of $H' - u$, and so deleting u does not separate x from y. □

We now show that given any 2-connected graph G, we can find a non-trivial cycle of density no more than that of G.

Theorem 3.10. *Let G be a 2-connected graph with at least 2 terminals. G contains a simple non-trivial cycle X such that Density$(X) \leq$ Density(G).*

Proof. Let C be an arbitrary non-trivial simple cycle; such a cycle always exists since G is 2-connected and has at least 2 terminals. If Density$(C) >$ Density(G), we give an algorithm that finds a new non-trivial cycle C' such that Density$(C') <$ Density(C). Repeating this process, we obtain cycles of successively better densities until eventually finding a non-trivial cycle X of density at most Density(G).

Let G' be the graph formed by contracting the given cycle C to a single vertex v. In G', v is not a terminal, and so has weight 0. Consider the 2-connected components of G' (from Lemma 3.9, each such component is formed by adding v to a connected component of $G' - v$), and pick the one of minimum density. If H is this component, Density$(H) <$ Density(G) by an averaging argument.

H contains at least 1 terminal. If it contains 2 or more terminals, recursively find a non-trivial cycle C' in H such that Density$(C') \leq$ Density$(H) <$ Density(C). If C' exists in the given graph G, it has the desired properties, and we are done. Otherwise, C' contains v, and the edges of C' form an ear of C in the original graph G. The density of this ear is less than the density of C, so we can apply Lemma 3.8 to obtain a non-trivial cycle in G that has density less than Density(C).

Finally, if H has exactly 1 terminal u, find any 2 vertex-disjoint paths using edges of H from u to distinct vertices in the cycle C. (Since G is 2-connected, there always exist such paths.) The

cost of these paths is at most $cost(H)$, and concatenating these 2 paths corresponds to an ear of C in G. The density of this ear is less than Density(C); again, we use Lemma 3.8 to obtain a cycle in G with the desired properties. □

We remark again that the algorithm of Theorem 3.10 does not lead to a polynomial-time algorithm, even if all edge costs and terminal weights are polynomially bounded. In Section 3.3.2, we describe a strongly polynomial-time algorithm that, given a graph G, finds a non-trivial cycle of density at most that of G. Note that neither of these algorithms may directly give a good approximation to the min-density non-trivial cycle problem, because the optimal non-trivial cycle may have density much less than that of G. However, we can use Theorem 3.10 to prove the following theorem:

Theorem 3.11. *There is an α-approximation to the (unrooted)* Dens-2VC *problem if and only if there is an α-approximation to the problem of finding a minimum-density non-trivial cycle.*

Proof. Assume we have a $\gamma(\ell)$-approximation for the Dens-2VC problem; we use it to find a low-density non-trivial cycle. Solve the Dens-2VC problem on the given graph; since the optimal cycle is a 2-connected graph, our solution H to the Dens-2VC problem has density at most $\gamma(\ell)$ times the density of this cycle. Find a non-trivial cycle in H of density at most that of H; it has density at most $\gamma(\ell)$ times that of an optimal non-trivial cycle.

Note that any instance of the (unrooted) Dens-2VC problem has an optimal solution that is a non-trivial cycle. (Consider any optimal solution H of density ρ; by Theorem 3.2, H contains a non-trivial cycle of density at most ρ. This cycle is a valid solution to the Dens-2VC problem.) Therefore, a $\beta(\ell)$-approximation for the min-density non-trivial cycle problem gives a $\beta(\ell)$-approximation for the Dens-2VC problem. □

Theorem 3.11 and Lemma 3.7 imply an $O(\log \ell)$-approximation for the minimum-density non-trivial cycle problem; this proves Corollary 3.3.

We say that a graph $G(V, E)$ is minimally 2-connected on its terminals if for every edge $e \in E$, some pair of terminals is not 2-connected in the graph $G - e$. Section 3.3.2 shows that in any graph which is minimally 2-connected on its terminals, every cycle is non-trivial. Therefore, the problem of finding a minimum-density non-trivial cycle in such graphs is just that of finding a

minimum-density cycle, which can be solved exactly in polynomial time. However, as we explain at the end of the section, this does not directly lead to an efficient algorithm for arbitrary graphs.

3.3.2 A Strongly Polynomial-Time Algorithm to Find Cycles of Average Density

In this section, we describe a strongly polynomial-time algorithm which, given a 2-connected graph $G(V, E)$ with edge costs and terminal weights, finds a non-trivial cycle of density at most that of G.

We begin with several definitions: Let C be a cycle in a graph G, and G' be the graph formed by deleting C from G. Let $H_1, H_2, \ldots H_m$ be the connected components of G'; we refer to these as *earrings* of C.[7] For each H_i, let the vertices of C incident to it be called its *clasps*. From the definition of an earring, for any pair of clasps of H_i, there is a path between them whose internal vertices are all in H_i.

We say that a vertex of C is an *anchor* if it is the clasp of some earring. (An anchor may be a clasp of multiple earrings.) A *segment* S of C is a path contained in C, such that the endpoints of S are both anchors, and no internal vertex of S is an anchor. (Note that the endpoints of S might be clasps of the same earring, or of distinct earrings.) It is easy to see that the segments partition the edge set of C. By deleting a segment, we refer to deleting its edges and internal vertices. Observe that if S is deleted from G, the only vertices of $G - S$ that lose an edge are the endpoints of S. A segment is *safe* if the graph $G - S$ is 2-connected.

Arbitrarily pick a vertex o of C as the *origin*, and consecutively number the vertices of C clockwise around the cycle as $o = c_0, c_1, c_2, \ldots, c_r = o$. The first clasp of an earring H is its lowest numbered clasp, and the last clasp is its highest numbered clasp. (If the origin is a clasp of H, it is considered the first clasp, not the last.) The *arc* of an earring is the subgraph of C found by traversing clockwise from its first clasp c_p to its last clasp c_q; the length of this arc is $q - p$. (That is, the length of an arc is the number of edges it contains.) Note that if an arc contains the origin, it must be the first vertex of the arc. Figure 3.1 illustrates several of these definitions.

Theorem 3.12. *Let H be an earring of minimum arc length. Every segment contained in the arc*

[7] If H_i were simply a path, it would be an ear of C, but H_i may be more complex.

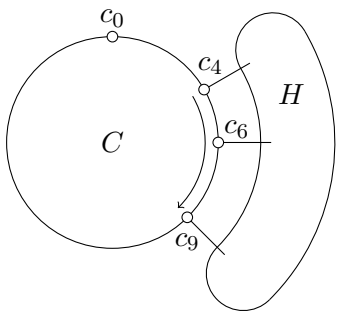


Figure 3.1: H is an earring of G, with clasps c_4, c_6, c_9; c_4 is its first clasp, and c_9 its last clasp. The arrow indicates the arc of H.

of H is safe.

Proof. Let $\mathcal{H}$ be the set of earrings with arc identical to that of H. Since they have the same arc, we refer to this as the arc of $\mathcal{H}$, or the *critical arc*. Let the first clasp of every earring in $\mathcal{H}$ be c_a, and the last clasp of each earring in $\mathcal{H}$ be c_b. Because the earrings in $\mathcal{H}$ have arcs of minimum length, any earring $H' \notin \mathcal{H}$ has a clasp c_x that is not in the critical arc. (That is, $c_x < c_a$ or $c_x > c_b$.)

We must show that every segment contained in the critical arc is safe; recall that a segment S is safe if the graph $G - S$ is 2-connected. Given an arbitrary segment S in the critical arc, let c_p and c_q $(p < q)$ be the anchors that are its endpoints. We prove that there are always 2 internally vertex-disjoint paths between c_p and c_q in $G - S$; this suffices to show 2-connectivity.

We consider several cases, depending on the earrings that contain c_p and c_q. Figure 3.2 illustrates these cases. If c_p and c_q are contained in the same earring H', it is easy to find two vertex-disjoint paths between them in $G - S$. The first path is clockwise from q to p in the cycle C. The second path is entirely contained in the earring H' (an earring is connected in $G - C$, so we can always find such a path.)

Otherwise, c_p and c_q are clasps of distinct earrings. We consider three cases: Both c_p and c_q are clasps of earrings in $\mathcal{H}$, one is (but not both), or neither is.

1. We first consider that both c_p and c_q are clasps of earrings in $\mathcal{H}$. Let c_p be a clasp of H_1, and c_q a clasp of H_2. The first path is from c_q to c_a through H_2, and then clockwise along the critical arc from c_a to c_p. The second path is from c_q to c_b clockwise along the critical path,

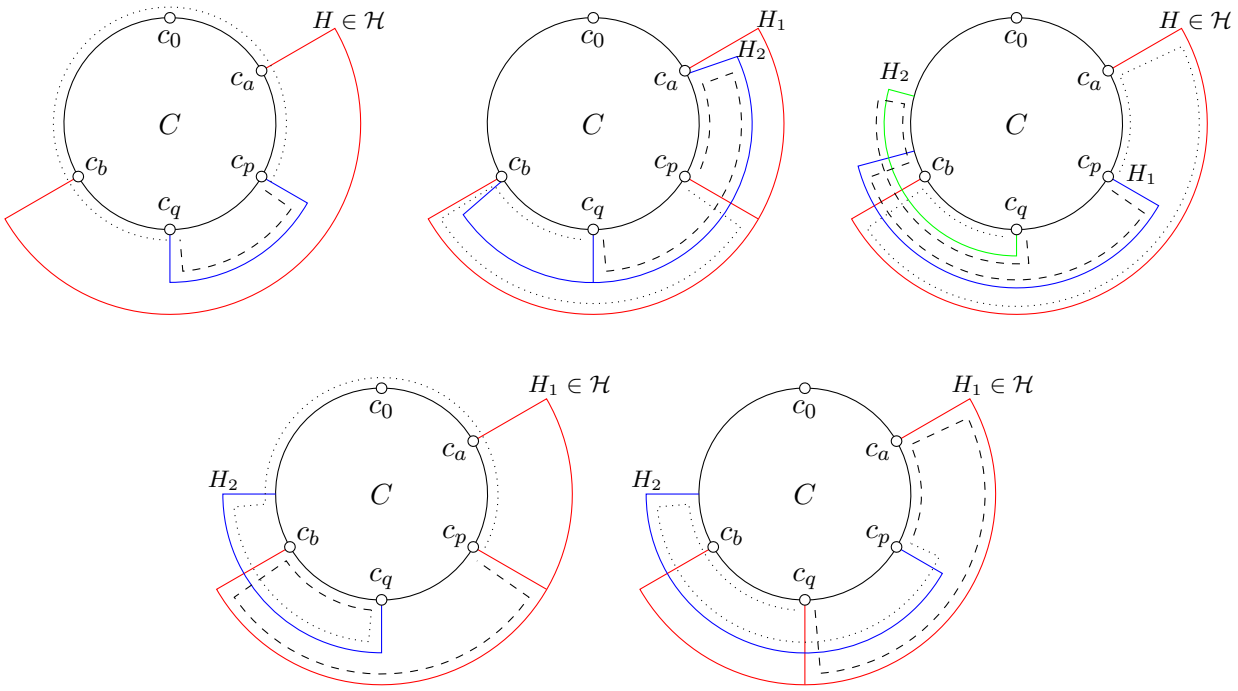


Figure 3.2: The various cases of Theorem 3.12 are illustrated in the order presented. In each case, one of the 2 vertex-disjoint paths from c_p to c_q is indicated with dashed lines, and the other with dotted lines.

and then c_b to c_p through H_1. It is easy to see that these paths are internally vertex-disjoint.

2. Now, suppose neither c_p nor c_q is a clasp of an earring in $\mathcal{H}$. Let c_p be a clasp of H_1, and c_q be a clasp of H_2. The first path we find follows the critical arc clockwise from c_q to c_b (the last clasp of the critical arc), from c_b to c_a through $H \in \mathcal{H}$, and again clockwise through the critical arc from c_a to c_p. Internal vertices of this path are all in H or on the critical arc. Let $c_{p'}$ be a clasp of H_1 not on the critical arc, and $c_{q'}$ be a last clasp of H_2 not on the critical arc. The second path goes from c_p to $c_{p'}$ through H_1, from p' to q' through the cycle C outside the critical arc, and from $c_{q'}$ to c_q through H_2. Internal vertices of this path are in H_1, H_2, or in C, but not part of the critical arc (since each of $c_{p'}$ and $c_{q'}$ are outside the critical arc). Therefore, we have 2 vertex-disjoint paths from c_p to c_q.

3. Finally, we consider the case that exactly one of c_p, c_q is a clasp of an earring in $\mathcal{H}$. Suppose c_p is a clasp of $H_1 \in \mathcal{H}$, and c_q is a clasp of $H_2 \notin \mathcal{H}$; the other case (where $H_1 \notin \mathcal{H}$ and

$H_2 \in \mathcal{H}$ is symmetric, and omitted, though figure 3.2 illustrates the paths.) Let q' be the index of a clasp of H_2 outside the critical arc. The first path is from c_q to c_b through the critical arc, and then from c_b to c_p through H_1. The second path is from c_q to $c_{q'}$ through H_2, and from $c_{q'}$ to c_p clockwise through C. Note that the last part of this path enters the critical arc at c_a, and continues through the arc until c_p. Internal vertices of the first path that are in C are on the critical arc, but have index greater than q. Internal vertices of the second path that belong to C are either not in the critical arc, or have index between c_a and c_p. Therefore, the two paths are internally vertex-disjoint.

□

We now describe our algorithm to find a non-trivial cycle of good density, proving Theorem 3.2: *Let G be a 2-connected graph with edge-costs and terminal weights, and at least 2 terminals. There is a polynomial-time algorithm to find a non-trivial cycle X in G such that Density$(X) \leq$ Density(G).*

Proof of Theorem 3.2. Let G be a graph with ℓ terminals and density ρ; we describe a polynomial-time algorithm that either finds a cycle in G of density less than ρ, or a proper subgraph G' of G that contains all ℓ terminals. In the latter case, we can recurse on G' until we eventually find a cycle of density at most ρ.

We first find, in $O(n^3)$ time, a minimum-density cycle C in G. By Theorem 3.10, C has density at most ρ, because the minimum-density *non-trivial* cycle has at most this density. If C contains at least 2 terminals, we are done. Otherwise, C contains exactly one terminal v. Since G contains at least 2 terminals, there must exist at least one earring of C.

Let v be the origin of this cycle C, and H an earring of minimum arc length. By Theorem 3.12, every segment in the arc of H is safe. Let S be such a segment; since v was selected as the origin, v is not an internal vertex of S. As v is the only terminal of C, S contains no terminals, and therefore, the graph $G' = G - S$ is 2-connected, and contains all ℓ terminals of G. □

The proof above also shows that if G is minimally 2-connected on its terminals (that is, G has no 2-connected proper subgraph containing all its terminals), every cycle of G is non-trivial. (If a cycle contains 0 or 1 terminals, it has a safe segment containing no terminals, which can be deleted; this gives a contradiction.) Therefore, given a graph that *is* minimally 2-connected on its terminals,

finding a minimum-density non-trivial cycle is equivalent to finding a minimum-density cycle, and so can be solved exactly in polynomial time. This suggests a natural algorithm for the problem: Given a graph that is not minimally 2-connected on its terminals, delete edges and vertices until the graph is minimally 2-connected on the terminals, and then find a minimum-density cycle. As shown above, this gives a cycle of density no more than that of the input graph, but this may not be the minimum-density cycle of the original graph. For instance, there exist instances where the minimum-density cycle uses edges of a safe segment S that might be deleted by this algorithm.

3.4 Pruning 2-Connected Graphs of Good Density

In this section, we prove Theorem 3.13. Theorem 3.1, stated in Section 3.1, is simply the special case of this theorem with $H = G$, and in which every vertex is a terminal.

Theorem 3.13. *Let G be a 2-connected edge-weighted graph with a designated vertex $r \in V(G)$ such that every vertex of G has 2 vertex-disjoint paths to r of total weight/cost at most L. Let $H \subseteq G$ be a 2-connected subgraph of G, with a given set $S \subseteq V(H)$ of terminals; let $\rho = cost(H)/|S|$ be the density of H. There is a polynomial-time algorithm that, given any integer $k \leq |V(H)|$, finds a 2-connected k-vertex subgraph H' of G containing r, of total cost at most $O(\log k)\rho k + 2L$.*

Let $\ell = |S|$ be the number of terminals in H, and $cost(H)$ its total cost; $\rho = \frac{cost(H)}{\ell}$ is the density of H. We describe an algorithm that finds a subgraph H' of G containing at least k terminals, each of which is 2-connected to the root, and of total edge cost $O(\log k)\rho k + 2L$. (Note that H' may not be a subgraph of H, as we are not even guaranteed that the root is in H.)

We can assume $\ell > (8 \log k) \cdot k$, or the following trivial solution suffices: Take the entire graph H, pick two distinct vertices $u, v \in V(H)$, and connect each of u and v to the root using 2 disjoint paths. The main phase of our algorithm proceeds by maintaining a set of 2-connected subgraphs that we call *clusters*, and repeatedly finding low-density cycles that merge clusters of similar weight to form larger clusters. (The weight of a cluster X, denoted by w_X, is (roughly) the number of terminals it contains.) Clusters are grouped into *tiers* by weight; tier i contains clusters with weight at least 2^i and less than 2^{i+1}. Initially, each terminal is a separate cluster in tier 0. We say a cluster is *large* if it has weight at least k, and *small* otherwise. The algorithm stops when most terminals

are in large clusters.

We now describe the algorithm MergeClusters (see below). To simplify notation, let α be the quantity $2\lceil \log k \rceil \rho$. We say that a cycle is *good* if it has density at most α; that is, good cycles have density at most $O(\log k)$ times the density of the input graph H.

```
MergeClusters:
For (each i in {0, 1, ..., (⌈log_2 k⌉ − 1)}) do:
  If (i = 0):
      Every terminal has weight 1
  Else:
      Mark all vertices as non-terminals
      For (each small 2-connected cluster X in tier i) do:
          Add a (dummy) terminal v_X to H of weight w_X
          Add (dummy) edges of cost 0 from v_X to two (arbitrary) distinct vertices of X
  While (H has a non-trivial cycle C of density at most α in H):
      Let X_1, X_2, ... X_q be the small clusters that contain a terminal or an edge of C.
      (Note that the terminals in C belong to a subset of {X_1, ... X_q}.)
      Form a new cluster Y (of a higher tier) by merging the clusters X_1, ... X_q
      w_Y ← Σ_{j=1}^{q} w_{X_j}
      If (i = 0):
          Mark all terminals in Y as non-terminals
      Else:
          Delete all (dummy) terminals in Y and the associated (dummy) edges.
```

We briefly remark on some salient features of this algorithm and our analysis before presenting the details of the proofs.

1. In iteration i, the terminals correspond to tier i clusters. Clusters are 2-connected subgraphs of H, and by using cycles to merge clusters, we preserve 2-connectivity as the clusters become larger.

2. When a cycle C is used to merge clusters, all small clusters that contain an edge of C (regardless of their tier) are merged to form the new cluster. Therefore, at any stage of the algorithm, all currently small clusters are edge-disjoint. Large clusters, on the other hand, are *frozen*; even if they intersect a good cycle C, they are not merged with other clusters on C. Thus, at any time, an edge may be in multiple large clusters and up to one small cluster.

3. In iteration i of MergeClusters, the density of a cycle C is only determined by its cost and the weight of terminals in C corresponding to tier i clusters. Though small clusters of

other (lower or higher) tiers might be merged using C, we do *not* use their weight to pay for the edges of C.

4. The ith iteration terminates when no good cycles can be found using the remaining tier i clusters. At this point, there may be some terminals remaining that correspond to clusters which are not merged to form clusters of higher tiers. However, our choice of α (which defines the density of good cycles) is such that we can bound the number of terminals that are "left behind" in this fashion. Therefore, when the algorithm terminates, most terminals are in large clusters.

By bounding the density of large clusters, we can find a solution to the rooted k-2VC problem of bounded density. Because we always use cycles of low density to merge clusters, an analysis similar to that presented in Chapter 2 for directed Orienteering (see also the work of [126]) shows that every large cluster has density at most $O(\log^2 k)\rho$. (Note the similarity between MergeClusters and the algorithm BuildComponents of Section 2.4, between Lemmas 2.22 and 3.17, and between Lemmas 2.21 and 3.20.) We first present this analysis, though it does not suffice to prove Theorem 3.1. A more careful analysis shows that there is at least one large cluster of density at most $O(\log k)\rho$; this allows us to prove the desired theorem.

We now formally prove that MergeClusters has the desired behavior. First, we present a series of claims which, together, show that when the algorithm terminates, most terminals are in large clusters, and all clusters are 2-connected.

Remark 3.14. *Throughout the algorithm, the graph H is always 2-connected. The weight of a cluster is at most the number of terminals it contains.*

Proof. The only structural changes to H are when new vertices are added as terminals; they are added with edges to two distinct vertices of H. This preserves 2-connectivity, as does deleting these terminals with the associated edges.

To see that the second claim is true, observe that if a terminal contributes weight to a cluster, it is contained in that cluster. A terminal can be in multiple clusters, but it contributes to the weight of exactly one cluster. □

We use the following simple proposition in proofs of 2-connectivity; the proof is straightforward, and hence omitted.

Proposition 3.15. *Let $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$ be 2-connected subgraphs of a graph $G(V, E)$ such that $|V_1 \cap V_2| \geq 2$. Then the graph $H_1 \cup H_2 = (V_1 \cup V_2, E_1 \cup E_2)$ is 2-connected.*

Lemma 3.16. *The clusters formed by* MergeClusters *are all 2-connected.*

Proof. Let Y be a cluster formed by using a cycle C to merge clusters $X_1, X_2, \ldots X_q$. The edges of the cycle C form a 2-connected subgraph of H, and we assume that each X_j is 2-connected by induction. Further, C contains at least 2 vertices of each X_j: if C contains an edge of X_j, this follows immediately, and if it contains a (dummy) terminal, it must contain at least the two vertices of X_j incident to this terminal.[8] Therefore, we can use induction and Proposition 3.15 above: We assume $C \cup \{X_l\}_{l=1}^{j}$ is 2-connected by induction, and C contains 2 vertices of X_{j+1}, so $C \cup \{X_l\}_{l=1}^{j+1}$ is 2-connected.

Note that we have shown $Y = C \cup \{X_j\}_{j=1}^{q}$ is 2-connected, but C (and hence Y) might contain dummy terminals and the corresponding dummy edges. However, each such terminal with the 2 associated edges is a ear of Y; deleting them leaves Y 2-connected. More formally, if u, v are the other endpoints of the edges incident to the dummy terminal in X_j, there are at least 2 disjoint paths remaining between u and v even after deleting the dummy edges, as X_j was 2-connected prior to the introduction of the dummy terminal. □

Lemma 3.17. *The total weight of small clusters in tier i that are not merged to form clusters of higher tiers is at most $\frac{\ell}{2\lceil \log k \rceil}$.*

Proof. Assume this were not true; this means that MergeClusters could find no more cycles of density at most α using the remaining small tier i clusters. But the total cost of all the edges is at most $cost(H)$, and the sum of terminal weights is at least $\frac{\ell}{2\lceil \log k \rceil}$; this implies that the density of the graph (using the remaining terminals) is at most $2\lceil \log k \rceil \cdot \frac{cost(H)}{\ell} = \alpha$. But by Theorem 3.10, the graph must then contain a good non-trivial cycle, and so the while loop would not have terminated. □

[8] A cluster X_j may be a singleton vertex (for instance, if we are in tier 0), but such a vertex does not affect 2-connectivity.

Corollary 3.18. *When the algorithm* MergeClusters *terminates, the total weight of large clusters is at least* $\ell/2 > (4 \log k) \cdot k$.

Proof. Each terminal not in a large cluster contributes to the weight of a cluster that was not merged with others to form a cluster of a higher tier. The previous lemma shows that the total weight of such clusters in any tier is at most $\frac{\ell}{2\lceil \log k \rceil}$; since there are $\lceil \log k \rceil$ tiers, the total number of terminals not in large clusters is less than $\lceil \log k \rceil \cdot \frac{\ell}{2\lceil \log k \rceil} = \ell/2$. □

So far, we have shown that most terminals reach large clusters, all of which are 2-connected, but we have not argued about the density of these clusters. The next lemma says that if we can find a large cluster of good density, we can find a solution to the k-2VC problem of good density.

Lemma 3.19. *Let* Y *be a large cluster formed by* MergeClusters. *If* Y *has density at most* δ, *we can find a* 2*-connected graph* H' *with at least* k *terminals and containing the root, of total cost at most* $2\delta k + 2L$.

Proof. Let $X_1, X_2, \ldots X_q$ be the clusters merged to form Y in order around the cycle C that merged them; each X_j was a small cluster, of weight at most k. A simple averaging argument shows that there is a consecutive segment of X_js with total weight between k and $2k$, such that the cost of the edges of C connecting these clusters, together with the costs of the clusters themselves, is at most $2\delta k$. Let X_a be the "first" cluster of this segment, and X_b the "last". Let v and w be arbitrary terminals of X_a and X_b respectively. Connect each of v and w to the root r using 2 vertex-disjoint paths; the cost of this step is at most $2L$. (We assumed that every terminal could be 2-connected to r using disjoint paths of cost at most L.) The graph H' thus constructed has at least k terminals, and total cost at most $2\delta k + 2L$.

We show that every vertex z of H' is 2-connected to r; this, together with the straightforward fact that r is not a cut-vertex of H', completes our proof. Let z be an arbitrary vertex of H'; suppose there is a cut-vertex x which, when deleted, separates z from r. Both v and w are 2-connected to r, and therefore neither is in the same component as z in $H' - x$. However, we describe 2 vertex-disjoint paths P_v and P_w in Y' from z to v and w respectively; deleting x cannot separate z from both v and w, which gives a contradiction. The paths P_v and P_w are easy to find; let X_j be the cluster containing z. The cycle C contains a path from vertex $z_1 \in X_j$ to $v' \in X_a$,

and another (vertex-disjoint) path from $z_2 \in X_j$ to $w' \in X_b$. Concatenating these paths with paths from v' to v in X_a and w' to w in X_b gives us vertex-disjoint paths P_1 from z_1 to v and P_2 from z_2 to w. Since X_j is 2-connected, we can find vertex-disjoint paths from z to z_1 and z_2, which gives us the desired paths P_v and P_w.[9] □

We now present the two analyses of density referred to earlier. The key difference between the weaker and tighter analysis is in the way we bound edge costs. In the former, each large cluster pays for its edges separately, using the fact that all cycles used have density at most $\alpha = O(\log k)\rho$. In the latter, we crucially use the fact that small clusters which share edges are merged. Roughly speaking, because small clusters are edge-disjoint, the average density of small clusters must be comparable to the density of the input graph H. Once an edge is in a large cluster, we can no longer use the edge-disjointness argument. We must pay for these edges separately, but we can bound this cost.

First, the following lemma allows us to show that every large cluster has density at most $O(\log^2 k)\rho$. Note that the intuition behind this proof is extremely similar to that of Lemma 2.21, bounding the length of a solution to k-Stroll in directed graphs.

Lemma 3.20. *For any cluster Y formed by* MergeClusters *during iteration i, the total cost of edges in Y is at most $(i+1) \cdot \alpha w_Y$.*

Proof. We prove this lemma by induction on the number of vertices in a cluster. Let $\mathcal{S}$ be the set of clusters merged using a cycle C to form Y. Let $\mathcal{S}_1$ be the set of clusters in $\mathcal{S}$ of tier i, and $\mathcal{S}_2$ be $\mathcal{S} - \mathcal{S}_1$. ($\mathcal{S}_2$ contains clusters of tiers less or greater than i that contained an edge of C.)

The cost of edges in Y is at most the sum of: the cost of C, the cost of $\mathcal{S}_1$, and the cost of $\mathcal{S}_2$. Since all clusters in $\mathcal{S}_2$ have been formed during iteration i or earlier, and are smaller than Y, we can use induction to show that the cost of edges in $\mathcal{S}_2$ is at most $(i+1)\alpha \sum_{X \in \mathcal{S}_2} w_X$. All clusters in $\mathcal{S}_1$ are of tier i, and so must have been formed before iteration i (any cluster formed during iteration i is of a strictly greater tier), so we use induction to bound the cost of edges in $\mathcal{S}_1$ by $i\alpha \sum_{X \in \mathcal{S}_1} w_X$.

[9] The vertex z may not be in any cluster X_j. In this case, P_v is formed by using edges of C from z to $v' \in X_a$, and then a path from v' to v; P_w is formed similarly.

Finally, because C was a good-density cycle, and only clusters of tier i contribute to calculating the density of C, the cost of C is at most $\alpha \sum_{X \in \mathcal{S}_1} w_X$. Therefore, the total cost of edges in Y is at most $(i+1)\alpha \sum_{X \in \mathcal{S}} w_X = (i+1)\alpha w_Y$. □

Let Y be an arbitrary large cluster; since we have only $\lceil \log k \rceil$ tiers, the previous lemma implies that the cost of Y is at most $\lceil \log k \rceil \cdot \alpha w_Y = O(\log^2 k)\rho w_Y$. That is, the density of Y is at most $O(\log^2 k)\rho$, and we can use this fact together with Lemma 3.19 to find a solution to the rooted k-2VC problem of cost at most $O(\log^2 k)\rho k + 2L$. This completes the 'weaker' analysis, but this does not suffice to prove Theorem 3.1; to prove the theorem, we would need to use a large cluster Y of density $O(\log k)\rho$, instead of $O(\log^2 k)\rho$.

For the purpose of the more careful analysis, implicitly construct a forest $\mathcal{F}$ on the clusters formed by MergeClusters. Initially, the vertex set of $\mathcal{F}$ is just S, the set of terminals, and $\mathcal{F}$ has no edges. Every time a cluster Y is formed by merging $X_1, X_2, \ldots X_q$, we add a corresponding vertex Y to the forest $\mathcal{F}$, and add edges from Y to each of $X_1, \ldots X_q$; Y is the parent of $X_1, \ldots X_q$. We also associate a cost with each vertex in $\mathcal{F}$; the cost of the vertex Y is the cost of the cycle used to form Y from $X_1, \ldots X_q$. We thus build up trees as the algorithm proceeds; the root of any tree corresponds to a cluster that has not yet become part of a bigger cluster. The leaves of the trees correspond to vertices of H; they all have cost 0. Also, any large cluster Y formed by the algorithm is at the root of its tree; we refer to this tree as T_Y.

For each large cluster Y after MergeClusters terminates, say that Y is of type i if Y was formed during iteration i of MergeClusters. We now define the *final-stage* clusters of Y: They are the clusters formed during iteration i that became part of Y. (We include Y itself in the list of final-stage clusters; even though Y was formed in iteration i of MergeClusters, it may contain other final-stage clusters. For instance, during iteration i, we may merge several tier i clusters to form a cluster X of tier $j > i$. Then, if we find a good-density cycle C that contains an edge of X, X will merge with the other clusters of C.) The *penultimate* clusters of Y are those clusters that exist just before the beginning of iteration i and become a part of Y. Equivalently, the penultimate clusters are those formed before iteration i that are the immediate children in T_Y of final-stage clusters. Figure 3.3 illustrates the definitions of final-stage and penultimate clusters. Such a tree could be formed if, in iteration $i-1$, 4 clusters of this tier merged to form D, a cluster of tier

$i+1$. Subsequently, in iteration i, clusters H and J merge to form F. We next find a good cycle containing E and G; F contains an edge of this cycle, so these three clusters are merged to form B. Note that the cost of this cycle is paid for by the weights of E and G only; F is a tier $i+1$ cluster, and so its weight is not included in the density calculation. Finally, we find a good cycle paid for by A and C; since B and D share edges with this cycle, they all merge to form the large cluster Y.

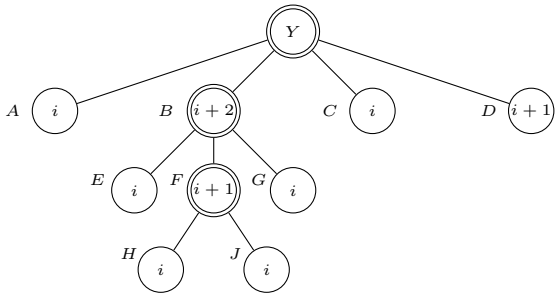


Figure 3.3: A part of the Tree T_Y corresponding to Y, a large cluster of type i. The number in each vertex indicates the tier of the corresponding cluster. Only final-stage and penultimate clusters are shown: final-stage clusters are indicated with a double circle; all other clusters are penultimate.

An edge of a large cluster Y is said to be a *final edge* if it is used in a cycle C that produces a final-stage cluster of Y. All other edges of Y are called *penultimate edges*; note that any penultimate edge is in some penultimate cluster of Y. We define the *final cost* of Y to be the sum of the costs of its final edges, and its *penultimate cost* to be the sum of the costs of its penultimate edges; clearly, the cost of Y is the sum of its final and penultimate costs. We bound the final costs and penultimate costs separately.

Recall that an edge is a final edge of a large cluster Y if it is used by MergeClusters to form a cycle C in the final iteration during which Y is formed. The reason we can bound the cost of final edges is that the cost of any such cycle is at most α times the weight of clusters contained in the cycle, and a cluster does not contribute to the weight of more than one cycle in an iteration. (This is also the essence of Lemma 3.20.) We formalize this intuition in the next lemma.

Lemma 3.21. *The final cost of any large cluster Y is at most αw_Y, where w_Y is the weight of Y.*

Proof. Let Y be an arbitrary large cluster. In the construction of the tree T_Y, we associated with each vertex of T_Y the cost of the cycle used to form the corresponding cluster. To bound the total

final cost of Y, we must bound the sum of the costs of vertices of T_Y associated with final-stage clusters. The weight of Y, w_Y is at least the sum of the weights of the penultimate tier i clusters that become a part of Y. Therefore, it suffices to show that the sum of the costs of vertices of T_Y associated with final-stage clusters is at most α times the sum of the weights of Y's penultimate tier i clusters. (Note that a tier i cluster must have been formed prior to iteration i, and hence it cannot itself be a final-stage cluster.)

A cycle was used to construct a final-stage cluster X only if its cost was at most α times the sum of weights of the penultimate tier i clusters that become a part of X. (Larger clusters may become a part of X, but they do not contribute weight to the density calculation.) Therefore, if X is a vertex of T_Y corresponding to a final-stage cluster, the cost of X is at most α times the sum of the weights of its tier i immediate children in T_Y. But T_Y is a tree, and so no vertex corresponding to an penultimate tier i cluster has more than one parent. That is, the weight of a penultimate cluster pays for only one final-stage cluster. Therefore, the sum of the costs of vertices associated with final-stage clusters is at most α times the sum of the weights of Y's penultimate tier i clusters, and so the final cost of Y is at most αw_Y. □

Lemma 3.22. *If Y_1 and Y_2 are distinct large clusters of the same type, no edge is a penultimate edge of both Y_1 and Y_2.*

Proof. Suppose, by way of contradiction, that some edge e is a penultimate edge of both Y_1 and Y_2, which are large clusters of type i. Let X_1 (respectively X_2) be a penultimate cluster of Y_1 (resp. Y_2) containing e. As penultimate clusters, both X_1 and X_2 are formed before iteration i. But until iteration i, neither is part of a large cluster, and two small clusters cannot share an edge without being merged. Therefore, X_1 and X_2 must have been merged, so they cannot belong to distinct large clusters, giving the desired contradiction. □

Theorem 3.23. *After* MergeClusters *terminates, at least one large cluster has density at most $O(\log k)\rho$.*

Proof. We define the *penultimate density* of a large cluster to be the ratio of its penultimate cost to its weight.

Consider the total penultimate costs of all large clusters: For any i, each edge $e \in E(H)$ can be a penultimate edge of at most 1 large cluster of type i. This implies that each edge can be a penultimate edge of at most $\lceil \log k \rceil$ clusters. Therefore, the sum of penultimate costs of all large clusters is at most $\lceil \log k \rceil cost(H)$. Further, the total weight of all large clusters is at least $\ell/2$. Therefore, the (weighted) average penultimate density of large clusters is at most $2\lceil \log k \rceil \frac{cost(H)}{\ell} = 2\lceil \log k \rceil \rho$, and hence there exists a large cluster Y of penultimate density at most $2\lceil \log k \rceil \rho$.

The penultimate cost of Y is, therefore, at most $2\lceil \log k \rceil \rho w_Y$, and from Lemma 3.21, the final cost of Y is at most αw_Y. Therefore, the density of Y is at most $\alpha + 2\lceil \log k \rceil \rho = O(\log k)\rho$. □

Theorem 3.23 and Lemma 3.19 together imply that we can find a graph $H' \subseteq G$ with at least k terminals and containing the root, of cost at most $O(\log k)\rho k + 2L$. This completes our proof of Theorem 3.13.

3.5 The Algorithms for the k-2VC and Budget-2VC Problems

We work with graphs in which some vertices are designated as *terminals*. Recall that the goal of the k-2VC problem is to find a minimum-cost 2-connected subgraph on at least k terminals. In the rooted k-2VC problem, we wish to find a min-cost subgraph on at least k terminals in which every terminal is 2-connected to the specified root r. The (unrooted) k-2VC problem can be reduced to the rooted version by *guessing* 2 vertices u, v that are in an optimal solution, creating a new root vertex r, and connecting it with 0-cost edges to u and v. It is not hard to show that any solution to the rooted problem in the modified graph can be converted to a solution to the unrooted problem by adding 2 minimum-cost vertex-disjoint paths between u and v. (Since u and v are in the optimal solution, the cost of these added paths cannot be more than OPT.) Similarly, one can reduce Budget-2VC to its rooted version. However, note that adding a min-cost set of paths between the guessed vertices u and v might require us to pay an additional amount of B, so to obtain a solution for the unrooted problem of cost $(3+\varepsilon)B$, we must find a solution for the rooted instance of cost $(2+\varepsilon)B$.

Note that the relationship between k-2VC and Budget-2VC is similar to that between k-

STROLL and ORIENTEERING, defined in Chapter 2; they are equivalent from the viewpoint of exact optimization, but this is not true from an approximation perspective. Still, we solve both k-2VC and BUDGET-2VC via the $O(\log \ell)$-approximation for the DENS-2VC problem, given in Lemma 3.7. We first describe our algorithm for the k-2VC problem. Let OPT be the cost of an optimal solution to the k-2VC instance. We assume knowledge of OPT; this can be dispensed with using standard methods. We pre-process the graph by deleting any terminal that does not have 2 vertex-disjoint paths to the root r of total cost at most OPT. The high-level description of the algorithm for the rooted k-2VC problem is given below.

$k' \leftarrow k$, $\quad G'$ is the empty graph.
While $(k' > 0)$:
  Use the approximation algorithm for DENS-2VC to find a subgraph H in G.
  If $(k(H) \leq k')$:
    $G' \leftarrow G' \cup H$, $\quad k' \leftarrow k' - k(H)$.
    Mark all terminals in H as non-terminals.
  Else:
    Prune H to obtain H' that contains k' terminals.
    $G' = G' \cup H'$, $\quad k' \leftarrow 0$.
Output G'.

At the beginning of any iteration of the while loop, the graph contains a solution to the DENS-2VC problem of density at most $\frac{\text{OPT}}{k'}$. Therefore, from Lemma 3.7, the graph H returned always has density at most $O(\log \ell)\frac{\text{OPT}}{k'}$. If $k(H) \leq k'$, we add H to G' and decrement k'; we refer to this as the *augmentation* step. Otherwise, we have a graph H of good density, but with too many terminals. In this case, we prune H to find a graph with the required number of terminals; this is the *pruning step*. A simple set-cover type argument shows the following lemma:

Lemma 3.24. *If, at every augmentation step, we add a graph of density at most $O(\log \ell)\frac{\text{OPT}}{k'}$ (where k' is the number of additional terminals that must be selected), the total cost of all the augmentation steps is at most $O(\log \ell \cdot \log k)\text{OPT}$.*

Therefore, it remains only to bound the cost of the graph H' added in the pruning step, and Theorem 3.13, proved in Section 3.4, is precisely what is needed. We can now prove our main result for the k-2VC problem, Theorem 3.4.

Proof of Theorem 3.4. Let OPT be the cost of an optimal solution to the (rooted) k-2VC

problem on an graph G. By Lemma 3.24, the total cost of the augmentation steps of our greedy algorithm is $O(\log \ell \cdot \log k)\mathrm{OPT}$. To bound the cost of the pruning step, let k' be the number of additional terminals that must be covered just prior to this step. The algorithm for the DENS-2VC problem returns a graph $H \subseteq G$ with $k(H) > k'$ terminals, and density at most $O(\log \ell)\frac{\mathrm{OPT}}{k'}$. As a result of our pre-processing step, every vertex has 2 vertex-disjoint paths in G to r of total cost at most OPT. Now, we use Theorem 3.13 to prune H and find a graph $H' \subseteq G$ with k' terminals and cost at most $O(\log k)\mathrm{Density}(H)k' + 2\mathrm{OPT} \leq O(\log \ell \cdot \log k)\mathrm{OPT} + 2\mathrm{OPT}$. Therefore, the total cost of our solution is $O(\log \ell \cdot \log k)\mathrm{OPT}$. □

We now describe the similar algorithm for the BUDGET-2VC problem. Given budget B, pre-process the graph as before by deleting vertices that do not have 2 vertex-disjoint paths to r of total cost at most B. Let OPT denote the number of vertices in the optimal solution, and $k = \mathrm{OPT}/c \log \ell \log \mathrm{OPT}$, for some constant $c = O(1/\varepsilon)$. We run the same greedy algorithm, using the $O(\log \ell)$-approximation for the DENS-2VC problem. Note that at each stage, the graph contains a solution to DENS-2VC of density at most $B/(\mathrm{OPT} - k) < 2B/\mathrm{OPT}$. Therefore, we have the following lemma:

Lemma 3.25. *If, at every augmentation step of the algorithm for* BUDGET-2VC*, we add a graph of density at most* $O(\log \ell)(2B/\mathrm{OPT})$*, the total cost of all augmentation steps is at most* $O(B/\log \mathrm{OPT}) \leq \varepsilon B$*.*

Again, to prove Theorem 3.6, giving a bi-criteria approximation for BUDGET-2VC, we only have to bound the cost of the pruning step.

Proof of Theorem 3.6. From the previous lemma, the total cost of the augmentation steps is at most εB. The graph H returned by the DENS-2VC algorithm has density at most $O(\log \ell \cdot B/\mathrm{OPT})$, and $k(H) > k'$ terminals. Now, from Theorem 3.1, we can prune H to find a graph H' containing k' terminals and cost at most $O(\log k' \log \ell \cdot B/\mathrm{OPT}) \cdot k' + 2B$. As $k' \leq k = \mathrm{OPT}/(c \log \ell \log \mathrm{OPT})$, a suitable choice of c ensures that the total cost of the pruning step is at most $\varepsilon B + 2B$. □

3.6 Concluding Remarks

In this chapter, we considered problems related to building large, low-cost, 2-connected graphs. These problems have applications to building low-cost fault-tolerant networks, and are generalizations of well-known problems such as k-MST that have numerous theoretical and practical uses. In particular, we gave an $O(\log \ell \log k)$-approximation for the k-2VC problem, and a bi-criteria approximation for Budget-2VC that 2-connects $\Omega(\mathrm{OPT}/\log k \log \ell)$ vertices while violating the cost by a constant factor.

We also showed that any 2-connected graph of density ρ with some vertices marked as terminals contains a non-trivial cycle with density at most ρ, and gave an algorithm to find such a cycle. Further, we found an $O(\log \ell)$-approximation for the problem of finding a minimum-density non-trivial cycle. However, we do not know the complexity of this problem; it may be possible to find a minimum-density non-trivial cycle *exactly* in polynomial time! If not, it should be possible to show that the problem is $\mathcal{NP}$-Hard, and it would be of interest to find a constant-factor approximation.

We list two additional open problems:

- Can the approximation ratio for the k-2VC problem be improved from the current $O(\log \ell \log k)$ to $O(\log n)$ or better? Removing the dependence on ℓ to obtain even $O(\log^2 k)$ could be interesting. If not, can one improve the approximation ratio for the easier k-2EC problem?

- In some applications, it may be of interest to design networks resilient to more than one failure, i.e., networks that are more than 2-connected. Can we obtain approximation algorithms for the k-λVC or k-λEC problems for $\lambda > 2$? Until recently, few results were known for problems where vertex-connectivity is required to be greater than 2, but there has been more progress with higher edge-connectivity requirements. Lau *et al.* [126] showed that for large λ, these problems are related to Dense k-Subgraph, and hence there are unlikely to be poly-logarithmic approximations. However, it may be possible to obtain approximation ratios that are polynomial in both λ and $\log n$.

Chapter 4

Element-Connectivity and Packing Disjoint Steiner Trees and Forests

4.1 Introduction [1]

Menger [132] proved the following fundamental min-max relation on vertex-connectivity: Given an undirected graph $G(V, E)$ and two nodes u, v, the maximum number of vertex-disjoint paths in G between u and v is equal to the minimum number of vertices and edges whose deletion separates u from v. Similarly, the maximum number of edge-disjoint paths between u and v is equal to the minimum number of edges whose deletion separates u from v; this is a special case of the well-known Max-flow/Min-cut theorem.

Hind and Oellermann [105] considered the natural generalization of Menger's theorem to more than two vertices: Given a graph $G(V, E)$ and a set of vertices $T \subseteq V$, what is the maximum number of vertex-disjoint *trees* connecting all vertices of T? As all trees are required to connect T, this question is meaningful if one asks for trees that share no edges or vertices of $V \setminus T$. For ease of notation, we use the single term *elements* to refer to the edges of E and vertices of $V \setminus T$. Thus, the question of Hind and Oellermann can be rephrased as follows: What is the maximum number of *element-disjoint* trees connecting T? Here, the natural upper bound is the minimum number of elements whose deletion separates T; analogous to the definitions of vertex-connectivity and edge-connectivity, this number is referred to as the *element-connectivity* of T. One might conjecture that a min-max relation analogous to Menger's theorem holds, i.e., that the number of element-disjoint trees spanning T is equal to the element-connectivity of T. Similarly, one might conjecture that the maximum number of *edge-disjoint* trees spanning T is equal to the edge-connectivity of T.

However, neither of these conjectures is true even when $|T| = 3$: Consider K_4, the complete

[1] This chapter is based on joint work with Chandra Chekuri, and has appeared in [53]. The original publication is available at `www.springerlink.com`, and the copyright is held by Springer-Verlag.

graph on four vertices, and let T be any set of 3 vertices. It is easy to see that the edge-connectivity and element-connectivity of T are 3, but there are only 2 edge-disjoint or element-disjoint trees connecting T.

Though the conjectures are not true when $|T| > 2$, Hind and Oellermann [105] showed that when $|T|$ is small, the number of element-disjoint trees spanning T is close to the element-connectivity of T; they used a graph *reduction step* to obtain this result. Subsequently, Cheriyan and Salavatipour [63] independently studied this question, calling this the problem of packing element-disjoint Steiner trees[2]; crucially using the graph reduction step, they showed that if k is the element-connectivity of T, there always exist $\Omega(k/\log|T|)$ element-disjoint Steiner trees. Moreover, this bound is tight (up to constant factors) in the worst case. In contrast, if we seek edge-disjoint Steiner trees then Lau [125] has shown that if T is $24k$ edge-connected in G, there are k edge-disjoint trees each of which spans T. Recently, Wu and West [156] improved this result to show that if T is $6.5k$ edge-connected, there are k edge-disjoint trees spanning T.

Element-Connectivity and Network Design

Motivated by problems independent from the question of Hind and Oellermann discussed above, Jain *et al.* [111] reintroduced element-connectivity as a connectivity measure intermediate to edge and vertex connectivities. Formally, given a graph $G(V, E)$, with its vertex set partitioned into a set T of terminals and a set $V \setminus T$ of non-terminals, we define the element-connectivity between two terminals u, v, denoted by $\kappa'_G(u, v)$, as the minimum number of elements (i.e. edges and non-terminals) whose deletion separates u from v. Note that one could equivalently define $\kappa'_G(u, v)$ as the maximum number of element-disjoint paths between u and v. We use $\kappa'_G(T)$ to denote the minimum number of elements whose deletion separates some terminals from others. It is easy to see that the element-connectivity between two terminals is at least their vertex connectivity and at most their edge-connectivity; that is, for any $u, v \in T$, we have $\kappa_G(u, v) \leq \kappa'_G(u, v) \leq \lambda_G(u, v)$, where $\kappa_G(u, v)$ and $\lambda_G(u, v)$ denote the vertex- and edge-connectivities between u and v in G respectively. Element-connectivity is also intermediate between edge- and vertex-connectivity more generally; in some respects it resembles the former, and in other ways resembles the latter. For

[2] A Steiner tree is simply a tree connecting all vertices of T.

example, $\kappa'(u,w) \geq \min(\kappa'(u,v), \kappa'(v,w))$ for any three terminals u, v, w; this triangle inequality holds for edge-connectivity but does not for vertex-connectivity. As discussed in Chapter 1, edge-connectivity problems in undirected graphs often appear to be "easier" than their vertex-connectivity counterparts. Vertex-connectivity exhibits less structure than edge-connectivity and this often translates into significant differences in the algorithmic and computational difficulty of the corresponding problems. Element-connectivity was introduced to help bridge this gap.

In particular, a problem of interest to Jain *et al.* [111] was the Survivable Network Design Problem (referred to as SNDP; see Section 1.3.4 for a description of this problem), a central problem in the design of robust networks. The edge-connectivity version of SNDP (EC-SNDP) was well understood, while the vertex-connectivity version (VC-SNDP) appeared intractable. Algorithmic techniques that had been used for EC-SNDP were successfully applied to the element-connectivity version [111, 79], leading to a 2-approximation for this problem, matching the best result known for EC-SNDP. These element-connectivity problems were partly motivated by applications: In designing some networks, there may be some important nodes (the terminals) which are highly reliable and must be able to communicate between themselves, and other nodes which are less reliable. Networks with high element-connectivity are robust against failure of the unreliable non-terminals and edges. In addition to the practical applications, Network Design problems with element-connectivity requirements were of interest because it was hoped that solving them would lead to new ideas that would be useful in understanding the harder vertex-connectivity problems. This approach has since proved very successful, with recent breakthroughs on VC-SNDP [68, 53, 69] being based on algorithms and ideas for the corresponding element-connectivity problems.

The preceding discussion above suggests that it is fruitful to study element-connectivity as a way to generalize edge-connectivity and attack problems on vertex-connectivity. We begin this chapter with a structural result on element-connectivity, generalizing the graph reduction step introduced by Hind and Oellermann [105] (and rediscovered by Cheriyan and Salavatipour [63]). We then use this result to obtain algorithms for finding element-disjoint Steiner trees and *forests*. Later, in Chapter 5, we use this reduction step to give a simple and elegant analysis of certain single-sink network design problems with *vertex*-connectivity requirements.

4.1.1 A Graph Reduction Step Preserving Element-Connectivity

The well-known *splitting-off* operation introduced by Lovász [129] is a standard tool in the study of (primarily) edge-connectivity problems. Given an undirected multi-graph G and two edges su and sv incident to s, the splitting-off operation replaces su and sv by the single edge uv. Lovász proved the following theorem on splitting-off to preserve *global* edge-connectivity.

Theorem 4.1 (Lovász)**.** *Let $G = (V \cup \{s\}, E)$ be an undirected multi-graph in which V is k-edge-connected for some $k \geq 2$ and degree of s is even. Then for every edge su there is another edge sv such that V is k-edge-connected after splitting-off su and sv.*

Mader strengthened the above theorem to show the existence of a pair of edges incident to s that when split-off preserve the *local* edge-connectivity of the graph.

Theorem 4.2 (Mader [130])**.** *Let $G = (V \cup \{s\}, E)$ be an undirected multi-graph, where $deg(s) \neq 3$ and s is not incident to a cut edge of G. Then s has two neighbours u and v such that the graph G' obtained from G by replacing su and sv by uv satisfies $\lambda_{G'}(x, y) = \lambda_G(x, y)$ for all $x, y \in V \setminus \{s\}$.*

Generalization to directed graphs are also known [130, 80, 108]. The splitting-off theorems have numerous applications in graph theory and combinatorial optimization; see [129, 81, 123, 110, 59, 125, 124, 114] for various pointers. Although splitting-off techniques can sometimes be used in the study of vertex-connectivity, their use is limited and no generally applicable theorem akin to Theorem 4.2 is known. On the other hand, Hind and Oellermann [105] proved an analogous reduction theorem on preserving global element connectivity. Below, we use $\kappa'_G(S)$ to denote $\min_{u,v \in S} \kappa'_G(u, v)$ and G/pq to denote the graph obtained from G by contracting vertices p, q.

Theorem 4.3 (Hind & Oellermann [105])**.** *Let $G = (V, E)$ be an undirected graph and $T \subseteq V$ be a terminal-set such that $\kappa'_G(T) \geq k$. Let (p, q) be* any *edge where $p, q \in V \setminus T$. Then $\kappa'_{G_1}(T) \geq k$ or $\kappa'_{G_2}(T) \geq k$ where $G_1 = G - pq$ and $G_2 = G/pq$.*

We generalize this theorem to show that either deleting an edge or contracting its endpoints preserves the *local* element-connectivity of every pair of terminals.

Reduction Lemma. *Let $G = (V, E)$ be an undirected graph and $T \subseteq V$ be a terminal-set. Let (p, q) be* any *edge where $p, q \in V \setminus T$ and let $G_1 = G - pq$ and $G_2 = G/pq$. Then one of the following holds: (i) $\forall u, v \in T$, $\kappa'_{G_1}(u, v) = \kappa'_G(u, v)$ (ii) $\forall u, v \in T$, $\kappa'_{G_2}(u, v) = \kappa'_G(u, v)$.*

It is easy to see that there is a polynomial-time algorithm to determine whether deleting or contracting an edge pq preserves the element-connectivity of every pair of terminals: First delete the edge pq to obtain the graph $G_1 = G - pq$, and test whether $\kappa'_{G_1}(u,v) = \kappa'_G(u,v)$ for every pair of terminals u, v. This requires $\binom{|T|}{2}$ min-cut computations. If the element-connectivity of every pair is preserved, we are done; if not, the Reduction Lemma guarantees that contracting the edge pq preserves each pairwise element-connectivity. One can design a more efficient algorithm that requires only $|T| - 1$ min-cut computations by exploiting the "triangle inequality" $\kappa'(u,w) \geq \min(\kappa'(u,v), \kappa'(v,w))$; we omit details from this thesis. It would be interesting to determine if still more efficient algorithms are possible.

Remark 4.4. *The Reduction Lemma, applied repeatedly, transforms a graph into another graph in which the non-terminals form a stable set. Moreover, the reduced graph is a minor of the original graph.*

Remark 4.5. *In studying element-connectivity, we often assume without loss of generality that there are no edges between terminals (by subdividing each such edge) and hence $\kappa'(u,v)$ is the maximum number of non-terminal disjoint u-v paths. Using the Reduction Lemma as described in the previous remark, we can thus obtain graphs in which both the terminals and non-terminals are stable sets; hence, these graphs are bipartite.*

Theorem 4.3 has been useful in the study of element-connectivity, and found applications in [63, 114]. The stronger Reduction Lemma, which preserves *local* connectivity, increases its applicability; as described above, we demonstrate applications in this chapter and the next to packing Steiner trees and forests, and to network design.

4.1.2 Overview of Results and Technical Ideas

Our main technical result is the Reduction Lemma, which simplifies graphs while preserving the element-connectivity of every pair of terminals. As remarked above, repeated applications yield a *bipartite* graph while preserving element connectivity. As bipartite graphs have highly restricted structure, it is often easier to design algorithms in this setting. We believe the Reduction Lemma will find additional applications besides those listed here, and will be a useful tool in simplifying

existing proofs, as shown by the following example: Chuzhoy and Khanna [68] gave a beautiful decomposition result for k-connectivity which is independently interesting from a graph theoretic point of view. The proof in [68] that such a decomposition exists is long and complicated, although it is based on only elementary operations. In Chapter 5, we use the Reduction Lemma to give an alternate proof which is both simple and extremely short.

In this chapter, we consider applications of the Reduction Lemma to packing element-disjoint Steiner trees and forests. In the element-disjoint Steiner forest packing problem, the input consists of a graph $G = (V, E)$ and disjoint terminal sets $T_1, T_2, \ldots, T_h$, and the goal is to find a maximum number of element-disjoint forests such that in each forest, T_i is connected for every $1 \leq i \leq h$. (Each such forest is referred to as a *Steiner forest*.) Clearly, an upper bound on the number of such forests is $\min_i\{\kappa'_G(T_i)\}$. It is natural to conjecture that if each $\kappa'_G(T_i) \geq k$, there exist $\Omega(k/\log|T|)$ element-disjoint Steiner forests, where $T = \bigcup_i T_i$; this would match the bound of Cheriyan and Salavatipour [63] for packing element-disjoint Steiner trees. However, previously known techniques do not seem to apply to the problem of packing forests, and [63] posed this as an open question. Our extension of the Reduction Lemma to preserve the local element-connectivity of every pair of terminals was primarily motivated by this question. For general graphs, we prove that there exist $\Omega(k/(\log|T|\log h))$ element disjoint forests; this can also be viewed as an $O(\log|T|\log h)$ approximation for the problem. Our algorithm begins by applying the Reduction Lemma to obtain a bipartite graph. Cheriyan and Salavatipour [63] took a similar approach for the problem of packing Steiner trees, using Theorem 4.3. Once they find a bipartite graph, they use a random coloring approach. To pack Steiner forests, however, one cannot apply the random coloring approach directly — in fact, we give an example at the end of Section 4.3 to show that it does not work. Instead we decompose the graph into highly connected subgraphs and then apply the random coloring approach in each subgraph separately.

We also study the packing problem in planar graphs and graphs of fixed genus, and prove substantially stronger results. Here too, the first step is to use the Reduction Lemma (recall that the reduced graph is a minor of the original graph and hence is also planar). After the reduction step, we employ a very different approach from the one for general graphs. Our main insight is that planarity restricts the ability of non-terminals to provide high element-connectivity to the

terminals. We formalize this intuition by showing that there are some two terminals u, v that have $\Omega(k)$ parallel edges between them which allows us to contract them and recurse. Using these ideas, for planar graphs we prove that there exist $\lceil k/5 \rceil - 1$ disjoint forests. Our method also extends to give an $\Omega(k)$ bound for graphs of a fixed genus, and we conjecture that one can find $\Omega(k)$ disjoint forests in graphs excluding a fixed minor; we give evidence for this by proving it for packing Steiner trees in graphs of fixed treewidth. Note that these bounds also imply corresponding approximation algorithms for maximizing the number of disjoint forests. These are the first non-trivial bounds for packing element-disjoint Steiner forests in general graphs or planar graphs. We note that a ρ-approximation for packing element-disjoint Steiner forests in planar graphs yields a 2ρ-approximation for the problem of packing *edge-disjoint* Steiner forests in planar graphs; see the end of Section 4.4 for details. Thus, it follows that we can find $\lceil k/10 \rceil - 1$ edge-disjoint Steiner forests; previously, the only algorithm known for packing edge-disjoint Steiner forests in planar graphs was the 32-approximation of Lau [125, 124].[3] Our proof is also simple; this simplicity comes from thinking about element-connectivity (using the Reduction Lemma) instead of edge-connectivity! Our proof further gives the strong property that for any planar graph, all the non-terminals in each forest have degree 2.

4.1.3 Related Work

There has been much interest in the recent past on algorithms for (integer) packing of disjoint Steiner trees in both the edge and element-connectivity settings [123, 110, 125, 124, 62, 63, 59]. See [94] for applications of Steiner tree packing to VLSI design. An outstanding open problem is Kriesell's conjecture which states that if the terminal set T is $2k$-edge-connected then there are k-edge-disjoint Steiner trees each of which spans T; this would generalize a classical theorem of Nash-Williams and Tutte on edge-disjoint spanning trees. Lau made substantial progress [125] and proved that $24k$-edge-connectivity suffices for k edge-disjoint Steiner trees; this was recently improved by Wu and West [156] to show that $6.5k$-edge-connectivity suffices. Recall that given several terminal sets $T_1, T_2, \ldots T_h \subseteq V(G)$, a *Steiner forest* is a forest such that each T_i $(1 \leq i \leq h)$ is contained in a single component of the forest. Lau extended his results in [124] to show that if

[3] Note that the algorithms of [125, 124] do not use planarity, and hence apply to general graphs.

each set T_i is $32k$ edge-connected in the graph G, then G contains k edge-disjoint Steiner forests. We remark that Mader's splitting-off theorem plays an important role in Lau's work. The element-disjoint Steiner tree packing problem was first considered by Hind and Oellermann. As mentioned earlier, Cheriyan and Salavatipour [63] gave a nearly tight bound for this problem: They gave an $O(\log n)$ approximation algorithm, and had previously shown that the problem of packing element-disjoint Steiner trees is hard to approximate to within a factor of $\Omega(\log n)$ [62]. The algorithm of [63] relies crucially on Theorem 4.3 followed by a simple randomized coloring algorithm whose analysis extends a similar algorithm for computing the domatic number of a graph [76]. In [71] the random coloring idea was shown to apply more generally in the context of packing bases of an arbitrary monotone submodular function; in addition, a derandomization was provided in [71] via the use of min-wise independent permutations.

Our work on packing Steiner forests in planar graphs was inspired by a question by Joseph Cheriyan [61]. Independent of our work, Aazami, Cheriyan and Jampani [1] proved that if a terminal set T is k-element-connected in a planar graph then there exist $k/2 - 1$ element-disjoint Steiner trees, and moreover this is tight. They also prove that it is NP-hard to obtain a $(1/2 + \varepsilon)$ approximation for this problem. Our bound for packing Steiner trees in planar graphs is slightly weaker than theirs; however, our algorithms and proofs are simple and intuitive, and generalize to packing Steiner forests. Their algorithm uses Theorem 4.3, followed by a reduction to a theorem of Frank, Király and Kriesell [83] that uses Edmonds' matroid partition theorem. One could attempt to pack Steiner forests using their approach (with the stronger Reduction Lemma in place of Theorem 4.3), but the theorem of [83] does not have a natural generalization for Steiner forests. The techniques of both [1] and this chapter extend to graphs of small genus or treewidth, and the ideas of [1] further apply to packing element-disjoint Steiner trees in graphs excluding a fixed minor; we discuss this in more detail in Section 4.4.

The problems of packing disjoint Steiner trees and forests (or other combinatorial structures) can be generalized to their capacitated variants. For example, each non-terminal could have a *capacity*, and the goal is to find a largest set of Steiner trees or forests such that the number of trees or forests containing a non-terminal is at most its capacity. Thus, the problem of packing element-disjoint Steiner trees or forests is simply the special case when the capacity of each non-terminal is 1. A

slightly different problem is obtained when one tries to to find a *fractional* packing: Now, one must assign a non-negative weight to each Steiner tree or forest such that the total weight of the trees or forests containing a non-terminal is at most its capacity; the goal is to maximize the total weight of all the trees or forests. Such fractional packing problems are of interest both as relaxations of the standard (integer) packing problems and in their own right; for instance, the fractional Steiner tree packing problem has applications in broadcasting over networks. Carr and Vempala [38] and Jain, Mahdian and Salavatipour [110] studied the connection between fractionally packing combinatorial structures and finding a minimum-cost such structure. Specifically, [110] showed that there is a ρ-approximation for fractionally packing Steiner trees iff there is a ρ-approximation for finding a minimum-cost Steiner tree. Călinescu, Chekuri, and Vondrák [71] state this result more generally: There is a ρ-approximation for fractionally packing a combinatorial structure iff there is a ρ-approximation for finding a minimum-cost structure. This follows from strong duality and the equivalence of separation and optimization, as the separation oracle for the dual of the fractional packing problem is the problem of finding a minimum-cost structure. See [71] for a clear presentation of these results.

Chapter Outline

In Section 4.2, we prove the Reduction Lemma. We then use it to obtain a polylogarithmic approximation for packing element-disjoint Steiner forests in Section 4.3. In Section 4.4, we obtain improved approximations for packing Steiner trees and forests in planar graphs and graphs of low genus. We conjecture that these results should extend to graphs excluding a fixed minor, and provide evidence for this by giving algorithms for graphs of low treewidth in Section 4.5.

4.2 The Reduction Lemma

Let $G(V, E)$ be a graph, with a given set $T \subseteq V(G)$ of terminals. For ease of notation, we subsequently refer to terminals as *black* vertices, and non-terminals (also called Steiner vertices) as *white*. The elements of G are white vertices and edges; two paths are *element-disjoint* if they have no white vertices or edges in common. Recall that the element-connectivity of two black vertices u

and v, denoted by $\kappa'_G(u, v)$, is the maximum number of element-disjoint (that is, disjoint in edges and white vertices) paths between u and v in G. We omit the subscript G when it is clear from the context.

For this section, to simplify the proof, we will assume that G has no edges between black vertices; any such edge can be subdivided, with a white vertex inserted between the two black vertices. It is easy to see that two paths are element-disjoint in the original graph iff they are element-disjoint in the modified graph. Thus, we can say that paths are element disjoint if they share no white vertices, or that u and v are k-element-connected if the smallest set of white vertices whose deletion separates u from v has size k.

Recall that our lemma generalizes Theorem 4.3 on preserving global connectivity. We remark that our proof is based on a cutset argument unlike the path-based proofs in [105, 63] for the global case.

Reduction Lemma. *Given $G(V, E)$ and T, let $pq \in E(G)$ be any edge such that p and q are both white. Let $G_1 = G - pq$ and $G_2 = G/pq$ be the graphs formed from G by deleting and contracting pq respectively. Then, (i) $\forall u, v \in T, \kappa'_{G_1}(u, v) = \kappa'_G(u, v)$ or (ii) $\forall u, v \in T, \kappa'_{G_2}(u, v) = \kappa'_G(u, v)$.*

Proof. Consider an arbitrary edge pq. Deleting or contracting an edge can reduce the element-connectivity of a pair by at most 1. Suppose the lemma were not true; there must be pairs s, t and x, y of black vertices such that $\kappa'_{G_1}(s, t) = \kappa'_G(s, t) - 1$ and $\kappa'_{G_2}(x, y) = \kappa'_G(x, y) - 1$. The pairs have to be distinct since it cannot be the case that $\kappa'_{G_1}(u, v) = \kappa'_{G_2}(u, v) = \kappa'_G(u, v) - 1$ for any pair u, v. (To see this, if one of the $\kappa'_G(u, v)$ u-v paths uses pq, contracting the edge will not affect that path, and will leave the other paths untouched. Otherwise, no path uses pq, and so it can be deleted.). Note that one of s, t could be the same vertex as one of x, y; for simplicity we will assume that $\{s, t\} \cap \{x, y\} = \emptyset$, but this does not change our proof in any detail. We show that our assumption on the existence of s, t and x, y with the above properties leads to a contradiction. Let $\kappa'_G(s, t) = k_1$ and $\kappa'_G(x, y) = k_2$. We use the following facts several times.

1. Any cutset of size less than k_1 that separates s and t in G_1 cannot include p or q. (If it did, it would also separate s and t in G.)

2. $\kappa'_{G_1}(x,y) = k_2$ since $\kappa'_{G_2}(x,y) = k_2 - 1$.

We define a vertex tri-partition of a graph G as follows: (A, B, C) is a vertex tri-partition of G if A, B, and C partition $V(G)$, B contains only white vertices, and there are no edges between A and C. (That is, removing the white vertices in B disconnects A and C.)

Since $\kappa'_{G_1}(s,t) = k_1 - 1$, there is a vertex-tri-partition (S, M, T) such that $|M| = k_1 - 1$ and $s \in S$ and $t \in T$. From Fact 1 above, M cannot contain p or q. For the same reason, it is also easy to see that p and q cannot be both in S (or both in T); otherwise M would be a cutset of size $k_1 - 1$ in G. Therefore, assume without loss of generality that $p \in S, q \in T$.

Similarly, since $\kappa'_{G_2}(x,y) = k_2 - 1$, there is a vertex-tri-partition (X, N', Y) in G_2 with $|N'| = k_2 - 1$ and $x \in X$ and $y \in Y$. We claim that N' contains the contracted vertex pq for otherwise N' would be a cutset of size $k_2 - 1$ in G. Therefore, it follows that (X, N, Y) where $N = N' \cup \{p, q\} - \{pq\}$ is a vertex-tri-partition in G that separates x from y. Note that $|N| = k_2$ and N includes *both* p and q. For the latter reason we note that (X, N, Y) is a vertex-tri-partition also in G_1.

Subsequently, we work with the two vertex tri-partitions (S, M, T) and (X, N, Y) in G_1 (we stress that we work in G_1 and not in G or G_2). Recall that $s, p \in S$, and $t, q \in T$, and that M has size $k_1 - 1$; also, N separates x from y, and $p, q \in N$. Fig. 1 (a) below shows these vertex tri-partitions. Since M and N contain only white vertices, all terminals are in S or T, and in X or Y. We say that $S \cap X$ is *diagonally opposite* from $T \cap Y$, and $S \cap Y$ is diagonally opposite from $T \cap X$. Let A, B, C, D denote $S \cap N, X \cap M, T \cap N$ and $Y \cap M$ respectively, with I denoting $N \cap M$; note that A, B, C, D, I partition $M \cup N$.

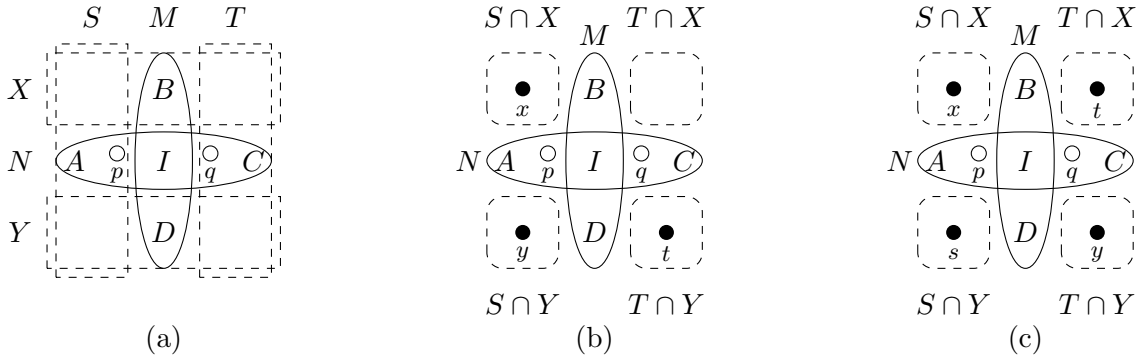


Figure 4.1: Part (a) illustrates the vertex tri-partitions (S, M, T) and (X, N, Y).
In parts (b) and (c), we consider possible locations of the terminals s, t, x, y.

We assume without loss of generality that $x \in S$. If we also have $y \in S$, then $x \in S \cap X$ and $y \in S \cap Y$; therefore, one of x, y is diagonally opposite from t, suppose this is x. Fig. 1 (b) illustrates this case. Observe that $A \cup I \cup B$ separates x from y; since x and y are k_2-connected and $|N = A \cup I \cup C| = k_2$, it follows that $|B| \geq |C|$. Similarly, $C \cup I \cup D$ separates t from s, and since C contains q, Fact 1 implies that $|C \cup I \cup D| \geq k_1 > |B \cup I \cup D = M| = k_1 - 1$. Therefore, $|C| > |B|$, and we have a contradiction.

Hence, it must be that $y \notin S$; so $y \in T \cap Y$. The argument above shows that x and t cannot be diagonally opposite, so t must be in $T \cap X$. Similarly, s and y cannot be diagonally opposite, so $s \in S \cap Y$. Fig. 1 (c) shows the required positions of the vertices. Now, N separates s from t and contains p, q; therefore, from fact 1, $|N| \geq k_1 > |M|$. But M separates x from y, and fact 2 implies that x, y are k_2-connected in G_1; therefore, $|M| \geq k_2 = |N|$, and we have a contradiction. □

4.3 Packing Steiner Trees and Forests in General Graphs

Consider a graph $G(V, E)$, with its vertex set V partitioned into $T_1, T_2, \ldots T_h, W$. We refer to each T_i as a group of *terminals*, and W as the set of Steiner or white vertices; we use $T = \bigcup_i T_i$ to denote the set of all terminals. A Steiner forest for this graph is a forest that is a subgraph of G, such that each T_i is entirely contained in a single tree of this forest. (Note that T_i and T_j can be in the same tree.) For any group T_i of terminals, we define $\kappa'(T_i)$, the element-connectivity of T_i, as the largest k such that for every $u, v \in T_i$, the element-connectivity of u and v in the graph G is at least k.

We say two Steiner forests for G are element-disjoint if they share no edges or Steiner vertices. (Every Steiner forest must contain all the terminals.) The Steiner forest packing problem is to find as many element-disjoint Steiner forests for G as possible. By inserting a Steiner vertex between any pair of adjacent terminals, we can assume that there are no edges between terminals, and then the problem of finding element-disjoint Steiner forests is simply that of finding Steiner forests that do not share any Steiner vertices. A special case is when $h = 1$ in which case we seek a maximum number of element-disjoint Steiner trees.

Proposition 4.6. *If $k = \min_i \kappa'_G(T_i)$, there are at most k element-disjoint Steiner forests in G.*

Proof. Let S be a set of k white vertices that separates vertices u and v in T_i. Any tree that contains both u and v must contain a vertex of S. Hence, we can pack at most k trees that contain all of T_i. □

Cheriyan and Salavatipour [63] proved that if there is a single group T of terminals, with $\kappa'(T) = k$, then there always exist $\Omega(k/\log|T|)$ Steiner trees. Their algorithm proceeds by using Theorem 4.3, the global element-connectivity reduction of [105], to delete and contract edges between Steiner vertices, while preserving $\kappa'(T) = k$. Then, once we obtain a bipartite graph G' with terminals on one side and Steiner vertices on the other side, randomly color the Steiner vertices using $k/6\log|T|$ colors; they show that with high probability, each color class connects the terminal set T, giving $k/6\log|T|$ trees. The bipartite case can be cast as a special case of packing bases of a polymatroid and a variant of the random coloring idea is applicable in this more general setting [71]; a derandomization is also provided in [71], thus yielding a deterministic polynomial time algorithm to find $\Omega(k/\log|T|)$ element-disjoint Steiner trees.

In this section, we give algorithms for packing element-disjoint Steiner forests, where we are given h groups of terminals $T_1, T_2, \ldots T_h$. The approach of [63] encounters two difficulties. First, we cannot reduce to a bipartite instance, using only the global-connectivity version of the Reduction Lemma. In fact, our strengthening of the Reduction Lemma to preserve local connectivity was motivated by this; using it allows us once again assume that we have a bipartite graph $G'(T \cup W, E)$. Second, we cannot apply the random coloring algorithm on the bipartite graph G' directly; we give an example at the end of this section to show that this approach does not work. One reason for this is that, unlike the Steiner tree case, it is no longer a problem of packing bases of a submodular function. To overcome this second difficulty we use a decomposition technique followed by the random coloring algorithm to prove that there always exist $\Omega(k/(\log|T|\log h))$ element-disjoint forests. We believe that the bound can be improved to $\Omega(k/\log|T|)$.

In order to pack element-disjoint Steiner forests we borrow the basic idea from [59] in the *edge-connectivity* setting for Eulerian graphs; this idea was later used by Lau [124] in the much more difficult non-Eulerian case. The idea at a high level is as follows: If all the terminals are k-connected then we can treat the terminals as forming one group and reduce the problem to that of packing Steiner trees. Otherwise, we can find a cut $(S, V \setminus S)$ that separates some groups from others. If

the cut is chosen appropriately we may be able to treat one side, say S, as containing a single group of terminals and pack Steiner *trees* in them *without* using the edges crossing the cut. Then we can shrink S and find Steiner forests in the reduced graph; unshrinking of S is possible since we have many trees on S. In [59, 124] this scheme works to give $\Omega(k)$ edge-disjoint Steiner forests. However, the approach relies strongly on properties of edge-connectivity as well as the properties of the packing algorithm for Steiner trees. These do not generalize easily for element-connectivity. Nevertheless, we show that the basic idea can be applied in a slightly weaker way (resulting in the loss of an $O(\log h)$ factor over the Steiner tree packing factor). We remark that the reduction to a bipartite instance using the Reduction Lemma plays a critical role. A key definition is the notion of a good separator given below.

Definition 4.7. *Given an graph $G(V, E)$ with terminal sets $T_1, T_2, \ldots T_h$, such that for all i, $\kappa'(T_i) \geq k$, we say that a set S of white vertices is a* good separator *if (i) $|S| \leq k/2$ and (ii) there is a component of $G - S$ in which all terminals are $k/2 \log h$-element-connected.*

Note that the empty set is a good separator if all terminals are $k/2 \log h$-element-connected.

Lemma 4.8. *For any instance of the Steiner forest Packing problem, there is a polynomial-time algorithm that finds a good separator.*

Proof. Let $G(V, E)$ be an instance of the Steiner forest packing problem, with terminal sets $T_1, T_2, \ldots T_h$ such that each T_i is k-element-connected. If T is $\frac{k}{2 \log h}$-element connected, the empty set S is a good separator.

Otherwise, there is some set of white vertices of size less than $\frac{k}{2 \log h}$ that separates some of the terminals from others. Let S_1 be a minimal such set, and consider the two or more components of $G - S_1$. Note that each T_i is entirely contained in a single component, since T_i is at least k-element-connected, and $|S_1| < k$. Among the components of $G - S_1$ that contain terminals, consider a component G_1 with the fewest sets of terminals; G_1 must have at most $h/2$ sets from $T_1, \ldots T_h$. If the set of all terminals in G_1 is $\frac{k}{2 \log h}$ connected, we stop, otherwise, find in G_1 a set of white vertices S_2 with size less than $\frac{k}{2 \log h}$ that separates terminals of G_1. Again, find a component G_2 of $G_1 - S_2$ with fewest sets of terminals, and repeat this procedure until we obtain some subgraph G_ℓ in which all the terminals are $\frac{k}{2 \log h}$-connected. We can always find such a subgraph, since the

number of sets of terminals is decreasing by a factor of 2 or more at each stage, so we find at most $\log h$ separating sets S_j. Now, we observe that the set $S = \bigcup_{j=1}^{\ell} S_j$ is a good separator. It separates the terminals in G_ℓ from the rest of T, and its size is at most $\log h \times \frac{k}{2\log h} = k/2$; it follows that each set of terminals T_i is entirely within G_ℓ, or entirely outside it. By construction, all terminals in G_ℓ are $\frac{k}{2\log h}$ connected. To see that this algorithm runs in polynomial time, it suffices to observe that if the sets of white vertices $S_1, S_2, \ldots$ exist, they can be found in polynomial time via min-cut algorithms. Once no such set S_ℓ exists in $G_{\ell-1}$, we have found the desired separator. □

We can now prove our main result of this section, that we can always find a packing of $\Omega(\frac{k}{\log|T|\log h})$ Steiner forests.

Theorem 4.9. *Given a graph $G(V,E)$, with terminal sets $T_1, T_2, \ldots T_h$, such that for all i, $\kappa'(T_i) \geq k$, there is a polynomial-time algorithm to pack $\Omega(k/\log|T|\log h)$ element-disjoint Steiner forests in G.*

Proof. The proof is by induction on h. The base case of $h = 1$, follows from [63, 71]; G contains at least $\frac{k}{6\log|T|}$ element-disjoint Steiner *trees*, and we are done.

We may assume G is bipartite by using the Reduction Lemma. Find a good separator S, and a component G_ℓ of $G - S$ in which all terminals are $\frac{k}{2\log h}$-connected. Now, since the terminals in G_ℓ are $\frac{k}{2\log h}$-connected, use the algorithm of [63] to find $\frac{k}{12\log h\log|T|}$ element-disjoint Steiner trees containing all the terminals in G_ℓ; none of these trees uses vertices of S. Number these trees from 1 to $\frac{k}{12\log h\log|T|}$; let $\mathcal{T}_j$ denote the jth tree.

The set S separates G_ℓ from the terminals in $G - G_\ell$. If S is not a minimal such set, discard vertices until it is. If we delete G_ℓ from G, and add a clique between the white vertices in S to form a new graph G', it is clear that the element-connectivity between any pair of terminals in G' is at least the element-connectivity they had in G. The graph G' has $h' \leq h-1$ groups of terminals; by induction, we can find $\frac{k}{12\log|T|\log h} < \frac{k}{12\log|T|\log h'}$ element-disjoint Steiner forests for the terminals in G'. As before, number the forests from 1 to $\frac{k}{12\log h\log|T|}$; we use $\mathcal{F}_j$ to refer to the jth forest. These Steiner forests may use the newly added edges between the vertices of S; these edges do not exist in G. However, we claim that the Steiner forest $\mathcal{F}_j$ of G', together with the Steiner tree $\mathcal{T}_j$ in G_ℓ gives a Steiner forest of G. The only way this might not be true is if $\mathcal{F}_j$ uses some edge added

between vertices $u, v \in S$. However, every vertex in S is adjacent to a terminal in G_ℓ, and all the terminals of G_ℓ are in every one of the Steiner trees we generated. Therefore, there is a path from u to v in $\mathcal{T}_j$. Hence, deleting the edge between u and v from $\mathcal{F}_j$ still leaves each component of $\mathcal{F}_j \cup \mathcal{T}_j$ connected.

Therefore, for each $1 \leq j \leq \frac{k}{12 \log h \log |T|}$, the vertices in $\mathcal{F}_j \cup \mathcal{T}_j$ induce a Steiner forest for G. To see that this algorithm runs in polynomial-time, note that we can find a good separator S in polynomial time, and we then recurse on the graph G'. As G' has $h' \leq h - 1$ groups of terminals, there are at most $h \leq n$ levels of recursion. □

A Counterexample to the Random Coloring Algorithm for Packing Steiner Forests.

We first define a graph H_k, which we use subsequently. H_k has two black vertices x and y, and k white vertices, each incident to both x and y. (That is, there are k disjoint paths of white vertices from x to y.) Given a graph G, we define the operation of inserting H_k along an edge $pq \in E(G)$ as follows: Add the vertices and edges of H_k to G, delete the edge pq, and add edges from p to x and q to y. (If we collapsed H_k to a single vertex, we would have subdivided the edge pq.) Figure 2 below shows H_4 and the effect of inserting H_4 along an edge.

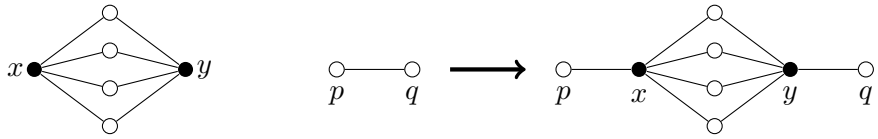


Figure 4.2: On the left, the graph H_4. On the right, inserting it along a single edge pq.

We now describe the construction of our counterexample. We begin with 2 black vertices s and t, and k vertex-disjoint paths between them, each of length $k + 1$; there are no edges besides the ones just described. Each of the k^2 vertices besides s and t is white. It is obvious that s and t are k-element-connected in this graph. Now, to form our final graph G_k, insert a copy of H_k along each of the $k(k-1)$ edges between a pair of white vertices. Fig. 4.3 below shows the construction of G_3.

The following claims are immediate:

- The vertices s and t are k-element-connected in G_k.

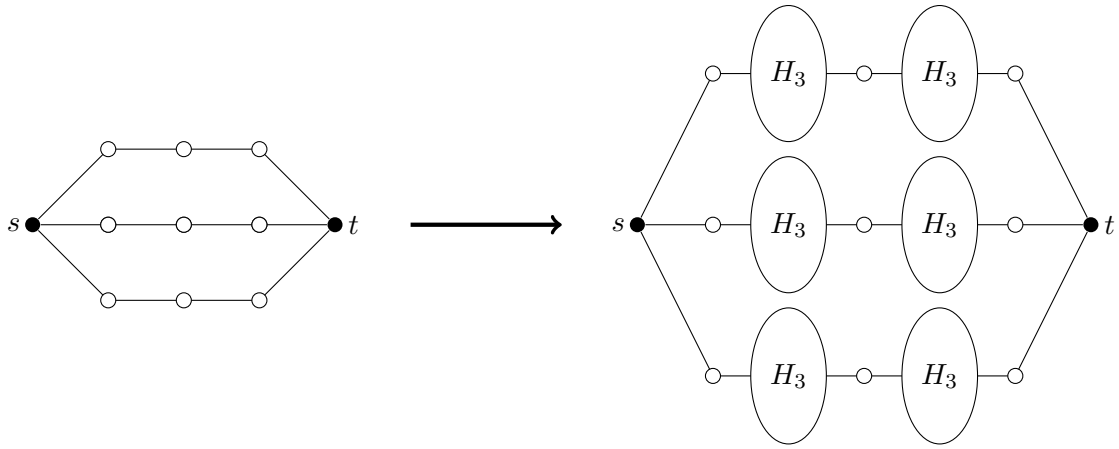


Figure 4.3: The construction of G_3.

- For every copy of H_k, the vertices x and y are k-white connected in G_k.
- The graph G_k is bipartite, with the white vertices and the black vertices forming the two parts.

We use G_k as an instance of the Steiner-forest packing problem; s and t form one group of terminals, and for each copy of H_k, the vertices x and y of that copy form a group. From our claims above, each group is k-element-connected.

If we use the algorithm of Cheriyan and Salavatipour, there are no edges between white vertices to be deleted or contracted, so we move directly to the coloring phase. If colors are assigned to the white vertices randomly, it is easy to see that no color class is likely to connect up s and t. The probability that a white vertex is given color i is $\frac{c \log |T|}{k}$, for some constant c. The vertices s and t can be connected iff the same color is assigned to all the white vertices on one of the k paths from s to t in the graph formed from G_k by contracting each H_k to a single vertex. The probability that *every* vertex on such a path will receive the same color is $\left(\frac{c \log |T|}{k}\right)^k$; using the union bound over the k paths gives us the desired result.

4.4 Packing Steiner Trees and Forests in Planar Graphs

We now prove much improved results for restricted classes of graphs, in particular planar graphs. If G is planar, we show the existence of $\lceil k/5 \rceil - 1$ element-disjoint Steiner forests.[4] The (simple) technique extends to graphs of fixed genus to prove the existence of $\Omega(k)$ Steiner forests where the constant depends mildly on the genus. We believe that there exist $\Omega(k)$ Steiner forests in any H-minor-free graph where H is fixed; it is shown in [1] that there exist $\Omega(k)$ Steiner *trees* in H-minor-free graphs. Our technique for planar graphs does not extend directly, but generalizing this technique allows us to make partial progress; by using our general graph result and some related ideas, in Section 4.5 we prove that in graphs of any fixed treewidth, there exist $\Omega(k)$ element-disjoint Steiner trees if the terminal set is k-element-connected.

The intuition and algorithm for planar graphs are easier to describe for the Steiner tree packing problem and we do this first; we later discuss the algorithm for packing Steiner forests in Section 4.4.2. We achieve the improved bound by observing that planarity restricts the use of many white vertices as "branch points" (that is, vertices of degree ≥ 3) in forests. Intuitively, even in the case of packing trees, if there are terminals $t_1, t_2, t_3, \ldots$ that must be in every tree, and white vertices $w_1, w_2, w_3 \ldots$ that all have degree 3, it is difficult to avoid a $K_{3,3}$ minor.[5] Note, however, that degree 2 white vertices behave like edges and do not form an obstruction. We capture this intuition more precisely by showing that there must be a pair of terminals t_1, t_2 that are connected by $\Omega(k)$ degree-2 white vertices; we can contract these "parallel edges", and recurse.

We describe below an algorithm for packing Steiner trees. Through the rest of the section, we assume $k > 10$; otherwise, $\lceil k/5 \rceil - 1 \leq 1$, and we can always find 1 Steiner tree in a connected graph.

Given an instance of the Steiner tree packing problem in planar graphs, we construct a *reduced instance* as follows: Use the Reduction Lemma to delete and contract edges between white vertices to obtain a planar graph with vertex set $T \cup W$, such that W is a stable set. Now, for each vertex $w \in W$ of degree 2, connect the two terminals that are its endpoints directly with an edge, and delete w. (All edges have unit capacity.) We now have a planar *multigraph*, though the only parallel

[4] Note that in the special case of packing Steiner trees, the paper of Aazami *et al.* [1] shows that there are $\lfloor k/2 \rfloor - 1$ element-disjoint Steiner trees.

[5] Strictly speaking, this is not true, though the intuition is helpful.

edges are between terminals, as these were the only edges added while deleting degree-2 vertices in W. Note that this reduction preserves the element-connectivity of each pair of terminals; further, any set of element-disjoint trees in this reduced instance corresponds to a set of element-disjoint trees in the original instance.

Lemma 4.10. *In a reduced instance of the planar Steiner tree packing problem, if T is k-element-connected, there are two terminals t_1, t_2 with at least $\lceil k/5 \rceil - 1$ parallel edges between them.*

The proof of this lemma is rather intricate, and we defer a complete proof to Section 4.4.1. First, though, we show that Lemma 4.10 allows us to pack $\lceil k/5 \rceil - 1$ disjoint trees.

Theorem 4.11. *Given an instance of the Steiner tree packing problem on a planar graph G with terminal set T, if $\kappa'(T) \geq k$, there is a polynomial-time algorithm to find at least $\lceil k/5 \rceil - 1$ element-disjoint Steiner trees in G. Moreover, in each tree, the white (non-terminal) vertices all have degree 2.*

Proof. We prove this theorem by induction on $|T|$; if $|T| = 2$, there are k disjoint *paths* in G from one terminal to the other, so we are done (including the guarantee of degree 2 for white vertices).

Otherwise, apply the Reduction Lemma to construct a reduced instance G', preserving the element-connectivity of T. Now, from Lemma 4.10, there exist a pair of terminals t_1, t_2 that have $\lceil k/5 \rceil - 1$ parallel edges between them (Note that the parallel edges between t_1 and t_2 may have non-terminals on them in the original graph but they have degree 2.). Contract t_1, t_2 into a single terminal t, and consider the new instance of the Steiner tree packing problem with terminal set $T' = T \cup \{t\} - \{t_1, t_2\}$. It is easy to see that the element-connectivity of the terminal set is still at least k; by induction, we can find $\lceil k/5 \rceil - 1$ Steiner trees containing all the terminals of T', with the property that all non-terminals have degree 2. Taking these trees together with $\lceil k/5 \rceil - 1$ edges between t_1 and t_2 gives $\lceil k/5 \rceil - 1$ trees in G' that span the original terminal set T. □

It remains only to prove Lemma 4.10, which we now do. As this involves some technical arguments about the structure of planar graphs, we recommend Section 4.4.1 be skipped on a first reading.

4.4.1 The Proof of Lemma 4.10

The following structural result is key to the proof of Lemma 4.10.

Lemma 4.12. *Let $G(T \cup W, E)$ be a planar graph with minimum degree 3, in which W is a stable set. There exists a vertex $t \in T$ of degree at most 10, with at most 5 neighbors in T.*

Proof. Our proof uses the *discharging* technique. Assume, for the sake of contradiction, that every vertex $t \in T$ has degree at least 11, or has at least 6 neighbors in T. By multiplying Euler's formula by 4, we observe that for a planar graph $G(V, E)$ with face set F, $(2|E|-4|V|)+(2|E|-4|F|) = -8$. We rewrite this as $\sum_{v \in V}(d(v)-4)+\sum_{f \in F}(l(f)-4) = -8$, where $d(v)$ and $l(f)$ denote the degree of vertex v and length of face f respectively.

Now, in our given graph G, assign $d(v) - 4$ units of *charge* to each vertex $v \in T \cup W$, and assign $l(f) - 4$ units of charge to each face f: Note that the net charge on the graph is negative. (It is equal to -8.) We describe rules for redistributing the charge through the graph such that after redistribution, if every vertex $t \in T$ has degree at least 11 or has at least 6 neighbors in T, the charge at each vertex and face will be non-negative. But no charge is added or removed (it is merely rearranged), and so we obtain a contradiction.

We use the following rules for distributing charge:

1. Every terminal $t \in T$ distributes $1/3$ unit of charge to each of its neighbors in W.
2. Every terminal $t \in T$ distributes $1/2$ unit of charge to each triangular face f it is incident to, unless the face contains 3 terminals. In this case, it distributes $1/3$ unit of charge to the face.

We now observe that every vertex of W and every face has non-negative charge. Each vertex $u \in W$ has degree at least 3 (the graph has minimum degree 3), so its initial charge was at least -1. It did not give up any charge, and rule 1 implies that it received $1/3$ from each of its (at least 3) neighbors, all of which are in T. Therefore, u has non-negative charge after redistribution. If a face f has length 4 or more, it already had non-negative charge, and it did not give up any. If f is a triangle, it starts with charge -1. It is incident to at least 2 terminals, since W is a stable set; we argue that it gains 1 unit of charge, to end with charge 0. From rule 2, if f is incident to

2 terminals, it gains 1/2 unit from each of them, and if it is adjacent to 3 terminals, it gains 1/3 unit from each of them.

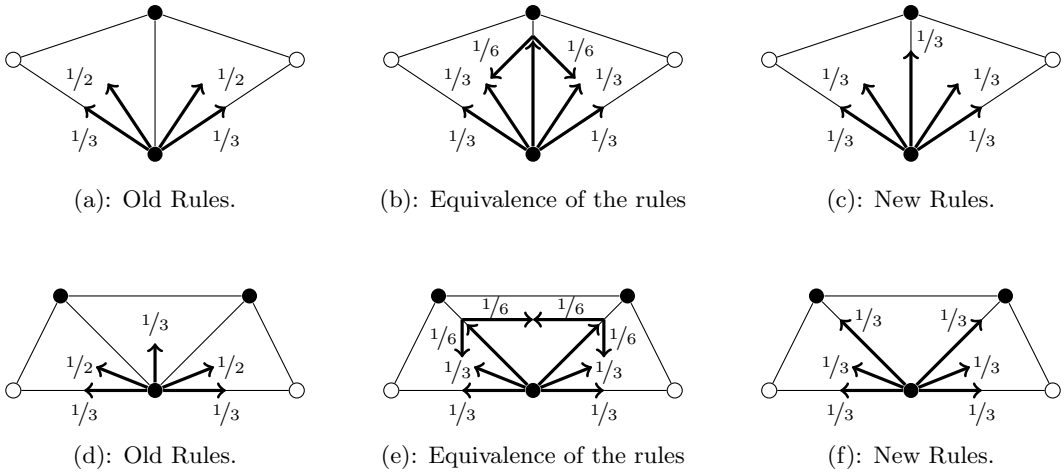
It remains only to argue that each terminal $t \in T$ has non-negative charge after redistribution. For ease of analysis, we describe a slightly modified version of the discharging in which each terminal loses at least as much charge as under the original rules, and show that each terminal has non-negative charge under the new discharging rules, listed below:

1. Every terminal t gives 1/3 unit of charge to *every* neighbor.
2. Every terminal $t \in T$ gives 1/3 unit of charge to each adjacent triangle.
3. Every terminal t gets back 1/3 unit of charge from each face f such that both t's neighbors on f are black.

We first prove that every terminal t loses at least as much charge as under the original rules; see also Fig. 4.4. The terminal t is now giving 1/3 unit of charge to all its black neighbors, besides giving this charge to its white neighbors. It is giving less charge (1/3 instead of 1/2) to some triangular neighbors, but every triangle is incident to a black vertex t' besides t; this neighbor of t received an extra 1/3 unit of charge from t, and it can give $1/6 = 1/2 - 1/3$ to each face incident to the edge $t - t'$. That is, the extra charge of 1/3 given by t to t' is enough to compensate for the fact that t may give 1/6 units less charge to the two faces incident to $t - t'$. Finally, note that if both t's neighbors on some face f are black, the original rules require t to give only 1/3 unit to f, which it also does under the new rules. However, it has given 1/3 unit of charge to these two black neighbors, and they do not need to use this to compensate for t giving too little charge to f; therefore, they may each return 1/6 unit of charge to t.

We now argue that every terminal has non-negative charge under the new rules. Let $t \in T$ have degree d; we consider three cases:

1. If $d \geq 12$, t gives away 1/3 to each of its d neighbors and d incident faces, so the total charge it gives away is $2d/3$. (It may also receive some charge, but we ignore this.) Therefore, the net charge on t is $(d - 4) - 2d/3 = (d/3) - 4$; as $d \geq 12$, this cannot be negative.



(a): Old Rules. (b): Equivalence of the rules (c): New Rules.

(d): Old Rules. (e): Equivalence of the rules (f): New Rules.

Figure 4.4: Terminals lose at least as much charge under the new rules.
Part (a) shows the charge given away by a terminal under the original rules, while part (c) shows the charge given away under the new rules; the triangles now receive less charge. Part (b) shows that the extra 1/3 unit of charge given to the black neighbor under the new rules can be split equally among the two triangles, which has the same effect as giving 1/2 unit to the triangles. Similarly, part (d) shows the charge given away by a terminal under the original rules, while part (f) shows the charge under the new rule 3: The central triangular face receives 1/3 unit of charge, but also returns 1/3 charge to the terminal as both its neighbors on this face are black. Part (e) shows that the extra 1/3 unit of charge given to each black neighbor under the new rules can be split among the triangles, so the effect is the same as giving 1/3 unit of charge to the central face, and 1/2 to each of the other faces.

2. If $d = 11$, we count the number of triangles incident to t. If there are 10 or fewer, t gives away 1/3 unit of charge to each of its 11 neighbors, and at most 10/3 to its adjacent triangles, so the net charge on t is at least $(11 - 4) - 11/3 - 10/3 = 0$. If t is incident to 11 triangles, it must be adjacent to at least 6 black vertices, as each triangle incident to t must be adjacent to a black neighbor of t, and no more than 2 triangles incident to t can share a neighbor of t. Since t has degree 11 and at least 6 black neighbors, some pair of black neighbors of t are on a common face, and t must receive 1/3 unit of charge from this face. It follows that the charge on t is at least $(11 - 4) - 11/3 - 11/3 + 1/3 = 0$.

3. If $d \leq 10$, t has at least 6 black neighbors by hypothesis. It has at most $d-6$ white neighbors, so there are at least $6 - (d - 6) = 12 - d$ faces f such that both t's neighbors on f are black. (Delete the white neighbors; there are at least 6 faces incident to t on which both its neighbors are black. When each white vertex is added back, it can only decrease the number of such faces by 1.) The terminal t gives away 1/3 unit of charge to each of its d neighbors and at most d incident triangles, and receives 1/3 unit of charge from each face on which both its

neighbors are black. Therefore, the net charge on t is at least $(d-4) - 2d/3 + (12-d)/3 = 0$.

□

Proof of Lemma 4.10. Let G be the planar multigraph of the reduced instance. Since T is k-element-connected in G, every terminal has degree at least k in G. Construct a planar graph G' from G by keeping only a single copy of each edge; from Lemma 4.12 above, some terminal t has degree at most 10, and at most 5 black neighbors. Let w denote the number of white neighbors of t, and b the number of black neighbors. Since each white vertex is incident to only a single copy of each edge in G, there must be at least $\lceil (k-w)/b \rceil$ copies in G of some edge between t and a black neighbor. But $b \leq 5$ and $b + w \leq 10$. Therefore, it is easy to verify since $k \geq 10$, the smallest possible value of $\lceil (k-w)/b \rceil$ is $\lceil (k-5)/5 \rceil = \lceil k/5 \rceil - 1$; this completes the proof. □

4.4.2 Packing Steiner Forests in Planar Graphs

For the Planar Steiner forest Packing problem, we use an algorithm very similar to that for packing Steiner trees above. Now, as input, we are given sets $T_1, \ldots T_h$ of terminals that are each internally k-connected, but some T_i and T_j may be poorly connected. The algorithm described above for packing Steiner trees encounters a technical difficulty when we try to extend it to Steiner forests. Lemma 4.10 can be used at the start to merge some two terminals. Precisely as before, as long as each T_i contains at least 2 terminals, Lemma 4.10 is true, so we can contract some pair of terminals t_1, t_2 that have $\lceil k/5 \rceil - 1$ parallel edges between them. Note that if t_1, t_2 are in the same T_i, after contraction, we have an instance in which T_i contains fewer terminals, and we can apply induction. If t_1, t_2 are in different sets T_i, T_j, then after contracting, all terminals in T_i and T_j are pairwise k-connected, so we can merge these two groups into a single set.

However, as the algorithm proceeds it may get stuck in the following situation: it merges all terminals from some group T_i into a single terminal. Now this terminal does not require any more connectivity to other terminals although other groups are not yet merged together. In this case we term this terminal as dead. In proving the crucial Lemma 4.10, we argued that in the multigraph G of the reduced instance, every terminal has degree at least k (since it is k-element-connected to other terminals), and in the graph G' in which we keep only a single copy of each edge, some

terminal has degree at most 10; therefore, there are $\lceil k/10 \rceil$ copies of some edge. However, in the Steiner forest problem, some T_i may contain only a single dead terminal t (after several contraction steps). The terminal t may be poorly connected to the remaining terminals; therefore, it may have degree less than k in the multigraph G. If t is the unique low-degree terminal in G', we may not be able to find a pair of terminals with a large number of edges between them. Thus, in the presence of dead terminals, Lemma 4.10 no longer applies; we illustrate this with a concrete example at the end of Section 4.4.

We solve this problem by eliminating a set T_i when it has only a single dead terminal t. One cannot simply delete this terminal or replace it by a single white vertex, as several paths connecting other terminals may pass through t. Instead, we replace the dead terminal t with a "well-linked" collection of white vertices so that distinct paths through t can now use disjoint white vertices from this collection. It might be most natural to replace t by a clique of white vertices, but this would not preserve planarity; instead, we replace a dead terminal t with a grid of white vertices, which ensures that the resulting graph is still planar. We then apply the Reduction Lemma to remove edges between the newly added white vertices and proceed with the merging process. We formalize this intuition in the following lemma:

Lemma 4.13. *Let $G(V, E)$ with a given $T \subseteq V$ be a planar graph, and $t \in T$ be an arbitrary terminal of degree d. Let G' be the graph constructed from G by deleting t, and inserting a $d \times d$ grid of white vertices, with the edges incident to t in G made incident to distinct vertices on one side of the new grid in G'. Then:*

1. *G' is planar.*
2. *For every pair u, v of terminals in G', $\kappa'_{G'}(u, v) = \kappa'_G(u, v)$.*
3. *Any set of element-disjoint subgraphs of G' corresponds to a set of element-disjoint subgraphs of G.*

Proof. See Figure 4.5 showing this operation; it is easy to observe that given a planar embedding of G, one can construct a planar embedding of G'. It is also clear that a set of element-disjoint subgraphs in G' correspond to such a set in G; every subgraph that uses a vertex of the grid can contain the terminal t.

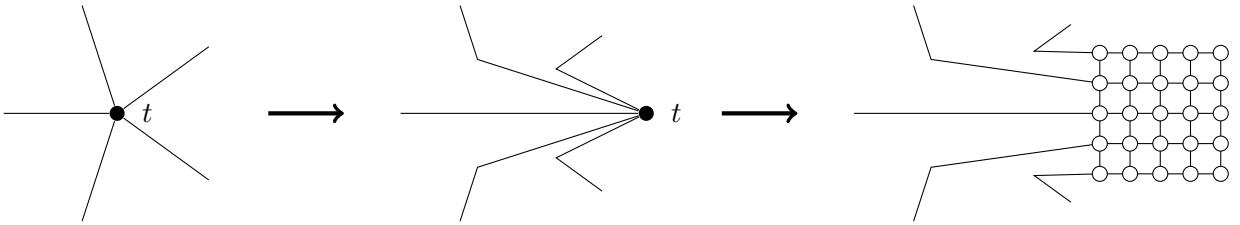


Figure 4.5: Replacing a terminal by a grid of white vertices preserves planarity and element-connectivity.

It remains only to argue that the element-connectivity of every other pair of terminals is preserved. Let u, v be an arbitrary pair of terminals; we show that their element-connectivity in G' is at least their connectivity $\kappa'(u, v)$ in G. Fix a set of $\kappa'(u, v)$ paths in G from u to v; let $\mathcal{P}$ be the paths that use the terminal t, and let $\ell = |\mathcal{P}|$. We locally modify these $\ell \leq d$ paths in $\mathcal{P}$ by routing them through the grid, so we obtain $\kappa'(u, v)$ element-disjoint paths in G'.

Let $\mathcal{P}_u$ denote the set of prefixes from u to t of the ℓ paths in $\mathcal{P}$, and let $\mathcal{P}_v$ denote the suffixes from t to v of these paths. Let H denote the $d \times d$ grid that replaces t in G'; we use $\mathcal{P}'_u$ and $\mathcal{P}'_v$ to denote the corresponding paths in G' from u to vertices of H, and from vertices in H to v respectively. Let $\mathcal{I}$ and $\mathcal{O}$ denote the vertices of H incident to paths in $\mathcal{P}'_u$ and $\mathcal{P}'_v$. It is not difficult to see that there are a set of disjoint paths in the grid H connecting the ℓ distinct vertices in $\mathcal{I}$ to those in $\mathcal{O}$; using the paths of $\mathcal{P}'_u$, together with the paths through H and the paths of $\mathcal{P}'_v$ gives us a set of disjoint paths in G' from u to v. □

A Counterexample to Lemma 4.10 for Planar Steiner Forest: Recall that in Section 4.4.2, we pointed out that in the presence of dead terminals (after all terminals in some T_i have been contracted to a single vertex), Lemma 4.10 may no longer apply. As a concrete example, consider the graph G_k defined at the end of Section 4.3. (See also Fig. 4.3, and note that G_k is planar.) We have one terminal set $T_1 = \{s, t\}$, and other sets T_i containing the two terminals of each copy of H_k. After several contraction steps, each copy of H_k may have been contracted together to form a single terminal; each such terminal is only 2-connected to the rest of the graph. In the reduced instance, there is only a single copy of each edge, and Lemma 4.10 does not hold.

Extensions: Our result for planar graphs can be generalized to graphs of fixed genus; Ivanco [107]

generalized a similar result of Borodin [33] on planar graphs to show that a graph G of genus g has an edge such that the sum of the degrees of its endpoints is at most $2g + 13$ if $0 \leq g \leq 3$ and $4g + 7$ otherwise. As the non-terminals have degree at least 3 and form a stable set, this implies that there is a terminal of degree at most $2g + 10$ (if $g \leq 3$) or $4g + 4$ (if $g > 3$). Using this result instead of Lemma 4.12, one can prove a result similar to Lemma 4.10, arguing that there are two terminals with $\lceil k/c \rceil$ parallel edges between them, where $c \leq 4g + 8$; we have not attempted to optimize this constant c. Thus, we obtain an algorithm for packing $\lceil k/c \rceil$ Steiner forests.

Aazami *et al.* [1] also give algorithms for packing Steiner trees in graphs of fixed genus, and graphs excluding a fixed minor. We thus make the following natural conjecture:

Conjecture 4.14. *Let $G = (V, E)$ be a H-minor-free graph, with terminal sets $T_1, T_2, \ldots T_h$, such that for all i, $\kappa'(T_i) \geq k$. There exist $\Omega(k/c)$ element-disjoint Steiner forests in G, where c depends only on the size of H.*

We note that Lemma 4.10 fails to hold for H-minor-free graphs, and in fact fails even for bounded treewidth graphs. Thus, our approach cannot be directly generalized. However, instead of attempting to contract together just two terminals connected by many parallel edges, we may be able contract together a constant number of terminals that are "internally" highly connected. Using Theorem 4.9 and other ideas, we prove in Section 4.5 that this approach suffices to pack many trees in graphs with small treewidth. We believe that these ideas together with the structural characterization of H-minor-free graphs by Robertson and Seymour [144] should lead to a positive resolution of Conjecture 4.14.

Finally, we note that our algorithms can be applied to the problem of packing *edge*-disjoint Steiner forests in planar graphs. See [1] for details; we provide a sketch here. Given a planar graph G and disjoint terminal sets $T_1, T_2, \ldots T_h$ that are each internally k-edge-connected, we begin by replacing each non-terminal of degree greater than 4 with a grid, precisely as in Lemma 4.13. This preserves planarity and edge-connectivity; thus, we have a planar graph G' in which each T_i is k-edge-connected, and all non-terminals have degree at most 4. But it now follows that each T_i is at least $k/2$-*element*-connected in G'; if this were not true, suppose that X is a set of non-terminals such that $|X| < k/2$ and deleting X separates terminals of some T_i. Now, as each vertex of X has

degree at most 4, there are fewer than $2k$ edges incident to X; this implies that some component of $G' - X$ is incident to fewer than k edges, and contains some (but not all) terminals of T_i. But this contradicts the fact that T_i is k-edge-connected in G'. Hence each T_i must be at least $k/2$-*element*-connected. Now, we can find $\lceil k/10 \rceil - 1$ element-disjoint Steiner forests in G'; these correspond to a set of $\lceil k/10 \rceil - 1$ edge-disjoint forests in G.

4.5 Packing Trees in Graphs of Bounded Treewidth

Let $G(V, E)$ be a graph of treewidth $\leq r - 1$, with terminal set $T \subseteq V$ such that $\kappa'(T) \geq k$. In this section, we give an algorithm to find, for any fixed r, $\Omega(k)$ element-disjoint Steiner trees in G. Our approach is similar to that for packing Steiner trees in planar graphs, where we argued in Lemma 4.10 that there exist two terminals t_1, t_2 with $\Omega(k)$ parallel edges between them, so we could contract them together and recurse on a smaller instance. In graphs of bounded treewidth, this is no longer the case; see the end of this section for an example in which no pair of terminals is connected by many parallel edges. However, we argue that there exists a small set of terminals $T' \subset T$ that is highly "internally connected", so we can find $\Omega(k)$ disjoint trees connecting all terminals in T', without affecting the connectivity of terminals in $T - T'$. We can then contract together T' and the white vertices used in these trees to form a single new terminal t, and again recurse on a smaller instance. The following lemma captures this intuition:

Lemma 4.15. *If $G(V, E)$ is a bipartite graph of treewidth at most $r - 1$, with terminal set $T \subset V$ such that $T \geq 2^r$, $\kappa'(T) \geq k$, there exists a set $S \subseteq V - T$ such that there is a component G' of $G - S$ containing $k/12r^2 \log(3r)$ element-disjoint Steiner trees for the (at least 2) terminals in G'. Moreover, these trees in G' can be found in polynomial time.*

Given this lemma, we prove below that for any fixed r, we can pack $\Omega(k)$ element-disjoint trees in graphs of treewidth at most $r - 1$. The proof combines ideas of Theorem 4.11 and Theorem 4.9.

Theorem 4.16. *Let $G = (V, E)$ be a graph of treewidth at most $r - 1$. For any terminal set $T \subseteq V$ with $\kappa'_G(T) \geq k$, there exist $\Omega(k/12r^2 \log(3r))$ element-disjoint Steiner trees on T.*

Proof. As for Theorem 4.11, we prove this theorem by induction. Let G be a graph of treewidth at most $r - 1$, with terminal set T. If $|T| \leq 2^r$, we have $k/6 \log |T| \geq k/6r$ element-disjoint trees

from the tree-packing algorithm of Cheriyan and Salavatipour [63] in *arbitrary* graphs.

Otherwise, we use the Reduction Lemma to ensure that G is bipartite. Let S be a set of white vertices guaranteed to exist from Lemma 4.15. If S is not a minimal such set, discard vertices until it is. Now, find $k/12r^2 \log(3r)$ element-disjoint trees containing all terminals in some component G' of $G - S$; note that each vertex of S is incident to some terminal in G', and hence to every tree. (This follows from the minimality of S and the fact that G is bipartite.) Modify G by contracting all of G' to a single terminal t, and make it incident to every vertex of S. It is easy to see that all terminals in the new graph are k-element-connected; therefore, we now have an instance of the Steiner tree packing problem on a graph with fewer terminals. The new graph has treewidth at most $r - 1$, so by induction, we have $k/12r^2 \log(3r)$ element-disjoint trees for the terminals in this new graph; taking these trees together with the $k/12r^2 \log(3r)$ trees of G' gives $k/12r^2 \log(3r)$ trees of the original graph G. □

We devote the rest of this section to proving the crucial Lemma 4.15. Subsequently, we may assume without loss of generality (after using the Reduction Lemma), that the graph G is bipartite; we may further assume that $k \geq 12r^2 \log(3r)$ and $|T| \geq 2^r$. First, observe that G has a small cutset that separates a few terminals from the rest.

Proposition 4.17. *G has a cutset C of size at most r such that the union of some components of $G - C$ contains between r and $2r$ terminals.*

Proof. Fix a (rooted) tree-decomposition $\mathcal{T}$ of G. Every non-leaf node of $\mathcal{T}$ corresponds to a cutset, and each node of $\mathcal{T}$ contains at most r vertices of G. Let v be a deepest node in $\mathcal{T}$ such that the the subtree rooted at each child of v has no more than $2r$ terminals. The nodes of G contained in v clearly form a cutset C of size at most r. If any subtree of $\mathcal{T}$ rooted at a child of v contains at least r terminals not contained in C, we are done. Otherwise, greedily select children of v until the total number of terminals in the associated subtrees not contained in C is between r and $2r$. □

We find the set S and component of $G - S$ in which we contract together a small number of terminals by focusing on the cutset C and components of $G - C$ that are guaranteed to exist from the previous proposition. We introduce some notation before proceeding with the proof:

1. Let C be a cutset of size at most r, and let V' be the vertices of the union of some components of $G - C$ containing between r and $2r$ terminals in total.

2. Since terminals in V' are k-connected to the terminals in the rest of the graph, and $|C| \leq r \ll k$, C contains at least one black vertex. Let C' be the set of black vertices in C.

3. Let $G' = G[V' \cup C']$ be the graph induced by V' and C'.

We omit a proof of the following straightforward proposition; the second part of the statement follows from the fact that each terminal in V' is k-connected to terminals outside G', and these paths to terminals outside G' must go through the cutset C of size at most r.

Proposition 4.18. *The graph G' contains between r and $3r$ terminals (as C' may contain up to r terminals), and each terminal in V' is at least k/r-connected to some terminal in C'.*

Let T' be the set of terminals in G'. If $\kappa'_{G'}(T') \geq k/2r^2$, we can easily find a set of white vertices satisfying Lemma 4.15: Let S be the set of vertices of G that are adjacent (in G) to vertices of G'. It is obvious that S separates G' from the rest of G, and all terminals in T' are highly connected; from the tree packing result of [63], we can find the desired disjoint trees in G'. Finally, note that all vertices of S are white, as the only neighbors of G' are either white vertices of the cutset C or the neighbors of the black vertices in C, all of which are white as G is bipartite.

However, it may not be the case that all terminals of T' are highly connected in G'. In this event, we use the following simple algorithm (very similar to that in the proof of Lemma 4.8) to find a highly-connected subset of T': Begin by finding a set S_1 of at most $k/2r^2$ white vertices in G' that separates terminals of T'. Among the components of $G' - S_1$, pick a component G_1 with at least one terminal of V'. If all terminals of G_1 are $k/2r^2$ connected, stop; otherwise, find in G_1 a set S_2 of at most $k/2r^2$ white vertices that separates terminals of G_1, pick a component G_2 of $G_1 - S_2$ that contains at least one terminal of V', and proceed in this manner until finding a component G_ℓ in which all terminals are $k/2r^2$ connected.

Claim 4.19. *We perform at most r iterations of this procedure before we stop, having found some subgraph G_ℓ in which all the (at least 2) terminals are $k/2r^2$ connected.*

Proof. At least one terminal of C' must be lost every time we find such a set S_i; if this is true, the claim follows. To see that this is true, observe that when we find a cutset S_{i+1} in G_i, there is a component that we do *not* pick that contains a terminal t. If this terminal t is in C', we are done; otherwise, it must be in V'. But from Proposition 4.18 all terminals in V' are k/r connected to some terminal in C', and so some terminal of C' must be in the same component as t. When we stop with the subgraph G_ℓ, it contains at least one terminal $t' \in V'$, and at least one terminal of C' to which t' is highly connected; therefore, G_ℓ contains at least 2 terminals. □

All terminals in the subgraph G_ℓ are $k/2r^2$-connected, and there are at most $3r$ of them, so we can find $k/12r^2 \log(3r)$ disjoint trees *in* G_ℓ that connect them, using the tree-packing result of [63]. Let S be the set of vertices of G that are adjacent (in G) to vertices of G_ℓ; obviously, S separates G_ℓ from the rest of G, and to satisfy Lemma 4.15, it merely remains to verify that S only contains white vertices. Every terminal in $G' - G_\ell$ was separated from G_ℓ by white vertices in some S_i, and terminals in $G - G'$ can only be incident to white vertices of the cutset C, which are not in G', let alone G_ℓ. This completes the proof of Lemma 4.15.

A Counterexample to the Existence of 2 Terminals Connected by $\Omega(k)$ "Parallel Edges"

Recall that in the case of planar graphs (or graphs of bounded genus), we argued that there must be two terminals t_1, t_2 with $\Omega(k)$ "parallel edges" between them. (That is, there are $\Omega(k)$ degree-2 white vertices adjacent to t_1 and t_2.) This is not necessarily the case even in graphs of treewidth 3: The graph $K_{3,k}$, the complete bipartite graph with 3 vertices on one side and k on the other, has treewidth 3. If the three vertices on one side are the terminal set T and the k vertices of the other side are non-terminals, it is easy to see that $\kappa'(T) = k$, but every white vertex has degree 3.

In this example, there are only 3 terminals, so the tree-packing algorithm of Cheriyan and Salavatipour [63] would allow us to find $\Omega(k/\log|T|) = \Omega(k)$ trees connecting them. Adding more terminals incident to all the white vertices would raise the treewidth, so this example does not immediately give us a low-treewidth graph with a large terminal set such that there are few parallel edges between any pair of terminals. However, we can easily extend the example by defining a graph G_h as follows: Let $T_1, T_2, \ldots T_h$ be sets of 2 terminals each, let $W_1, W_2, \ldots W_{m-1}$ each be

sets of k white vertices, and let all the vertices in each W_i be adjacent to both terminals in T_i and both terminals in T_{i+1}. (See Fig. 4.6 below.) The graph G_h has $2h$ terminals, $T = \bigcup_i T_i$ is k-element-connected, and it is easy to verify that G_h has treewidth 4. However, every white vertex has degree 4, so there are no "parallel edges" between terminals. (One can modify this example to construct a counterexample graph G_h with treewidth 3 by removing one terminal from each alternate T_i.)

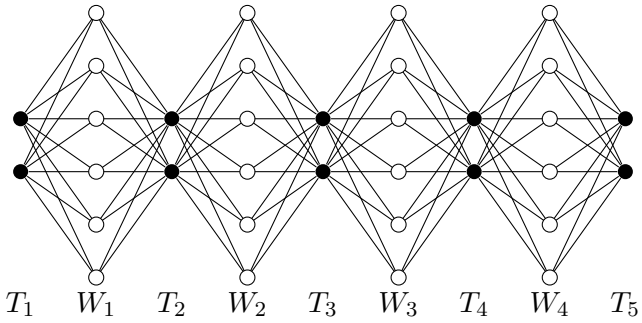


Figure 4.6: A graph of treewidth 4 with many terminals, but no "parallel edges".

4.6 Concluding Remarks

In this chapter, we generalized the reduction step of Hind and Oellermann [105] to handle local element connectivity. In this chapter, we demonstrated applications of our Reduction Lemma to packing disjoint Steiner trees and forests, and in the next, we give an application to network design. We believe that the ability to obtain bipartite graphs while preserving the element-connectivity of all pairs of terminals is very useful, and that the Reduction Lemma will find many applications in the future.

There are a few natural questions on packing element-disjoint Steiner forests that remain to be answered. First, we believe that our bound on the number of element-disjoint Steiner forests in a general graph can be improved from $\Omega(k/(\log |T| \log h))$ to $\Omega(k/\log |T|)$; an algorithm to achieve this would be of interest.

Second, it should be possible to prove Conjecture 4.14, on packing disjoint Steiner forests in graphs excluding a fixed minor. Chekuri and Ene [44] extended the techniques of this chapter to show that one can pack $\Omega(k)$ element-disjoint Steiner forests in graphs of fixed treewidth (in

Section 4.5, we only gave algorithms for packing Steiner trees), providing further evidence for the conjecture.

Finally, in a natural generalization of the Steiner forest packing problem, each non-terminal/white vertex has a *capacity*, and the goal is to pack element-disjoint forests subject to these capacity constraints. In general graphs, it is easy to reduce this problem to the uncapacitated/unit-capacity version (for example, by replacing a white vertex of capacity c by a clique of size c), but this is not necessarily the case for restricted classes of graphs. In particular, it would be of interest to show that it is possible to pack $\Omega(k)$ forests for this capacitated planar Steiner forest packing problem. An obvious first step is to prove this for packing element-disjoint Steiner trees in planar graphs. It is likely that this is possible, as one can *fractionally* pack $\Omega(k)$ element-disjoint Steiner trees in capacitated planar graphs; this follows from the recent work of Demaine *et al.* [72], showing that there is an $O(1)$-approximation for node-weighted Steiner Tree in planar graphs.

Chapter 5

Single-Sink Network Design with Vertex-Connectivity Requirements

5.1 Introduction [1]

In the Survivable Network Design Problem, the input is an undirected graph $G(V, E)$ with edge costs, and an integer connectivity requirement $R_{uv} > 0$ for each pair of vertices u, v. The goal is to find a minimum-cost subgraph H such that, for each pair u, v, there are R_{uv} disjoint paths between u and v. If these paths are required to be edge-disjoint, the problem is referred to as EC-SNDP, while if the paths must be vertex-disjoint, we refer to the problem as VC-SNDP.

SNDP already captures as special cases a variety of fundamental connectivity problems in combinatorial optimization such as:

- Minimum Spanning Tree, which is the special case when $R_{uv} = 1$ for all u, v.
- Steiner Tree, the special case when there is a set $T \subseteq V$ and $R_{uv=1}$ iff $u, v \in T$.
- Steiner Forest, the special case when $R_{uv} \in \{0, 1\}$ for all u, v.
- λ-Edge-Connected Spanning Subgraph, which is the special case of EC-SNDP when $R_{uv} = \lambda$ for all u, v.

Each of these problems has been extensively studied on its own, along with many other special cases of SNDP, and all but the first are NP-hard and APX-hard to approximate. Besides its theoretical interest, SNDP has obvious applications to the design of robust networks. A feasible solution to an EC-SNDP instance guarantees that each pair of vertices u and v will be able to communicate even if $R_{uv} - 1$ edges fail; similarly, a feasible solution to a VC-SNDP instance guarantees that u and v will be connected even if $R_{uv} - 1$ other vertices or edges fail.

[1] This chapter is based on joint work with Chandra Chekuri; portions have previously appeared in [52, 53]. The original version of [53] is available at `www.springerlink.com`.

The EC-SNDP problem is well understood: Jain's [109] seminal work on iterated rounding showed a 2-approximation for EC-SNDP, improving previous results [91, 154]. This was extended by Fleischer *et al.* and Cheriyan *et al.* [79, 64] to Element-Connectivity-SNDP, which is the variant in which there is a set of terminals $T \subseteq V$, $R_{uv} > 0$ iff $u, v \in T$, and the goal is to find a min-cost subgraph in which there are R_{uv} element-disjoint paths. (See Chapter 4 for a definition of element-connectivity and related discussion.) These techniques also extend to VC-SNDP when each $R_{uv} \in \{0, 1, 2\}$ via the setpair relaxation [82]. An important question that was open for many years[2] was to understand the approximability of VC-SNDP when the connectivity requirements exceed 2.

In this chapter we consider several *single-sink* network design problems with *vertex connectivity* requirements. Let $G = (V, E)$ be a given undirected graph on n nodes with a specified sink/root vertex r and a subset of terminals $T \subseteq V$, with $|T| = h$. Each terminal t has a demand $d_t > 0$ that needs to be routed to the root along each of k vertex-disjoint paths. In the following discussion, we assume for simplicity that $d_t = 1$ for each terminal t. The goal in all the problems is to find a routing (a selection of paths) for the terminals so as to minimize the cost of the routing. We obtain problems of increasing generality and complexity based on the cost function on the edges. In the basic SS-k-Connectivity problem, each edge e has a non-negative cost c_e, and the objective is to find a minimum-cost subgraph H of G that contains the desired disjoint paths for each terminal. Thus, this is the special case of VC-SNDP in which there is a set of terminals $T \subseteq V$ and a root t; $R_{tr} = k$ for each $t \in T$, and $R_{uv} = 0$ for all other pairs u, v. (Note that when $k = 1$, this is the well-known Steiner Tree problem, which has a 1.388 approximation [36].)

We then consider generalizations of SS-k-Connectivity to single-sink network design problems where the cost of an edge depends on the amount of demand/traffic that is routed along it. In the SS-k-Rent-or-Buy problem there is a parameter M with the following interpretation: An edge e can either be bought for a cost of $c_e \cdot M$, in which case any number of terminals can use it, or e can be rented at the cost of c_e per terminal. In other words, the cost of an edge e is $c_e \cdot \min\{M, |T_e|\}$ where T_e is the set of terminals whose paths use e. In the Uniform-SS-k-Buy-at-Bulk problem, the cost of an edge e is $c_e \cdot f(|T_e|)$ for some given sub-additive function

[2]There have been many recent developments, subsequent to our work discussed in this chapter; see the description of related work for details.

$f : \mathbb{R}^+ \to \mathbb{R}^+$. In the NON-UNIFORM-SS-$k$-BUY-AT-BULK problem the cost of an edge e is $f_e(|T_e|)$ for some edge-dependent sub-additive function $f_e : \mathbb{R}^+ \to \mathbb{R}^+$.

All of the three problems SS-k-CONNECTIVITY, SS-k-RENT-OR-BUY and SS-k-BUY-AT-BULK described above are NP-hard and also APX-hard to approximate even for $k = 1$. In this chapter we focus on polynomial-time approximation algorithms for these network design problems when $k > 1$.

The more general problems SS-k-RENT-OR-BUY and SS-k-BUY-AT-BULK are motivated by general BUY-AT-BULK type network design problems which arise naturally in the design of telecommunication networks [146, 7, 43]. Economies of scale imply that in practice, bandwidth on a link can be purchased/provisioned in integer units of different *cable-types*; that is, there are some b cable-types with capacities $u_1 < u_2 < \ldots < u_b$ and costs $w_1 < w_2 < \ldots < w_b$ such that $w_1/u_1 > \ldots > w_b/u_b$. This naturally leads to sub-additive edge-cost cost functions. For an overview of real-world fault-tolerant models in optical network design similar to SS-k-BUY-AT-BULK with $k = 2$, see [43, 141, 148].

5.1.1 Related Work

We have already mentioned several papers on the edge-connectivity and element-connectivity versions of SNDP [91, 154, 109, 79, 65]. Few algorithmic results were known for VC-SNDP with $R_{\max} = \max_{uv} R_{uv} > 2$ until recently. One special case that had received significant attention is the k-CONNECTED-SUBGRAPH problem, where the goal is to find a min-cost k-connected subgraph spanning the entire input graph G. Cheriyan, Vempala and Vetta [64] gave an $O(\log k)$ approximation when $k < \sqrt{n/6}$ and an $\sqrt{n/\varepsilon}$ approximation for $k < (1-\varepsilon)n$. Kortsarz and Nutov [120] improved this for large k; their algorithm had a ratio of $O(\ln k \cdot \min\{\sqrt{k}, \frac{n}{n-k} \ln k\})$. Fakcharoenphol and Laekhanukit [75] further improved this to an $O(\log^2 k)$-approximation for all k. These results use an algorithm of Frank and Tardos [84] for finding k-outconnected subgraphs in directed graphs.

Kortsarz, Krauthgamer and Lee [119] showed that for the general VC-SNDP, even when $R_{uv} \in \{0, k\}$ for each pair of vertices (u, v), there is no $2^{\log^{1-\varepsilon} n}$-approximation for any $\varepsilon > 0$ unless $\mathcal{NP} \subseteq DTIME(n^{\texttt{polylog}(n)})$. However, their hardness result requires k to be n^δ for some constant

$\delta > 0$; in this same setting they show that SS-k-Connectivity is hard to approximate to within an $\Omega(\log n)$ factor. A natural question to ask is whether SS-k-Connectivity, and more generally VC-SNDP, admits a good approximation when the maximum requirement $R_{\max}$ is small; this question is relevant from both practical and theoretical perspectives. In fact, no counterexample is known to the possibility of iterated rounding yielding a ratio of $R_{\max}$ for VC-SNDP; see [79] for a discussion of this subject. For SS-k-Connectivity in particular, no non-trivial (that is, $o(|T|)$) approximation was known until recently even when $k = 3$! Chakraborty, Chuzhoy and Khanna [39] developed some fundamental new insights and showed an $O(k^{O(k^2)} \log^4 n)$-approximation for SS-k-Connectivity via the setpair relaxation; we discuss connections between our work and that of [39] in Section 5.1.2. Chakraborty *et al.* [39] also improved the hardness results of [119]; they proved that VC-SNDP with $R_{uv} \in \{0, k\}$ does not admit a k^δ approximation for all $k > k_0$ for some constants δ, k_0. Further, they showed that the set pair relaxation has an integrality gap of $\tilde{\Omega}(k^{1/3})$.

Subsequent to [39], we obtained our initial results for SS-k-Connectivity, SS-k-Rent-or-Buy and SS-k-Buy-at-Bulk; using different techniques, we gave a simple *reverse*-greedy algorithm for SS-k-Connectivity that achieves an approximation ratio of $O(3^k k! \cdot k^2 \log |T|)$, an improvement over the $O(k^{O(k^2)} \log^4 n)$ ratio from [39]. (The problems SS-k-Rent-or-Buy and SS-k-Buy-at-Bulk were not considered in [39].) Independently, Chuzhoy and Khanna [68] gave an alternate analysis of a randomized variant of a similar greedy algorithm, showing that it achieved an approximation ratio of $O(k \log |T|)$ for SS-k-Connectivity. Their analysis used an element-connectivity based approach; shortly thereafter, we realized that the Reduction Lemma from Chapter 4 could be used to considerably simplify the proof of the main technical result in [68]. We discuss these approaches in more detail below; this chapter contains both our original analysis (in Section 5.2.2), and the improved element-connectivity based proof (in Section 5.2.1).

There have been several very recent developments on SS-k-Connectivity and, more generally, VC-SNDP. In addition to the $O(k \log |T|)$ approximation for SS-k-Connectivity, Chuzhoy and Khanna [68] gave an $O(k^7 \log^2 |T|)$-approximation for the more general variant of SS-k-Connectivity with costs on vertices, instead of edges. Nutov [137] studied the same problems, obtaining ratios of $O(k^2 \log |T|)$ and $O(k^2 \log^2 |T|)$ for SS-k-Connectivity and its vertex-cost

variant; his approach used uncrossing arguments for the problem of connectivity augmentation (i.e., increasing the connectivity between each $t \in T$ and r from $k-1$ to k). Using more ideas related to edge-covers of crossing families of sets, Nutov also improved the approximation ratio for k-Connected-Subgraph to $O(\log k \cdot \log \frac{n}{n-k})$ [138]. In a significant breakthrough, Chuzhoy and Khanna [69] obtained the first non-trivial approximation algorithms for the *general* VC-SNDP; their techniques were based on existing element-connectivity algorithms, and yielded an $O(k^3 \log n)$-approximation. Shortly thereafter, Nutov [139] gave algorithms for finding minimum-cost covers of so-called "uncrossable bifamilies", using these to obtain improved results for several problems, including an $O(k^2)$-approximation for SS-k-Connectivity and an $O(k^2 \log |T|)$-approximation for its vertex-cost variant[3], and an $O(k^4 \log^2 |T|)$-approximation for general VC-SNDP with vertex costs.

In addition to the basic connectivity problems such as VC-SNDP, there has been much work on Buy-at-Bulk and related problems. Until recently, almost all results were for the case of $k = 1$, though both the single sink and more general multi-commodity problems were considered. Our starting point for Buy-at-Bulk is the Rent-or-Buy cost function which can be modeled with two cable-types, one with unit capacity and the other with essentially infinite capacity. Gupta *et al.* [99]gave a constant-factor approximation for both the single-sink and multi-commodity variants of Rent-or-Buy. This simple cost function, in addition to its inherent interest, has played an important role in the development of algorithms for several problems [99]; in particular, results for Buy-at-Bulk have built on insights developed for Rent-or-Buy. Constant-factor approximations for Uniform-SS-k-Buy-at-Bulk when $k = 1$ were given by [99, 95, 149]; Awerbuch and Azar [19] gave an $O(\log n)$-approximation for the more general multi-commodity variant, using embeddings into tree metrics.

For Non-Uniform-SS-k-Buy-at-Bulk with $k = 1$, a randomized $O(\log n)$ approximation was given by Meyerson, Munagal and Plotkin [133]; this was later derandomized by Chekuri, Khanna and Naor [49]. Charikar and Karagiazova [40] gave the first approximation algorithm for the multi-commodity variant, obtaining an approximation ratio of $2^{O(\sqrt{\log h \log\log h})}$.[4] Chekuri *et al.* [48]

[3] In fact, the algorithms of [139] obtain these approximation ratios even for the more general problems in which terminals have differing connectivity requirements in $\{1, \ldots, k\}$.

[4] This result assumes that each demand is for one unit of traffic.

gave a poly-logarithmic approximation for the multi-commodity version, and later improved and extended this to obtain an $O(\log^4 h)$-approximation for the variant with vertex costs [47], where h is the number of pairs with non-zero demand. Recently, Kortsarz and Nutov [122] obtained an $O(\log^3 n)$-approximation for NON-UNIFORM-BUY-AT-BULK.[5]

There have been fewer results for BUY-AT-BULK and related problems when $k > 1$. Antonakapoulos *et al.* [9], motivated by problems in fault-tolerant optical network design, introduced the PROTECTED-BUY-AT-BULK network design problem. In this problem, there are h terminal pairs $(s_1, t_1), \ldots, (s_h, t_h)$, and each pair (s_i, t_i) needs to route its demand along two vertex-disjoint paths. The cost of the paths/routing is given by $\sum_e c_e f(x_e)$ where x_e is the total traffic on edge e induced by the paths; here f is a concave/sub-additive function induced by the cable-types in question. In [9] this problem was reduced to the corresponding single-sink problem at the expense of a poly-logarithmic increase in the approximation ratio. An $O(1)$ approximation for the single-sink problem was derived in [9] for the special case of PROTECTED-BUY-AT-BULK when there is only a single cable type. An open question raised in [9] is whether one can find a good approximation for the single-sink problem even for the case of two cable-types; we answer this question affirmatively in this chapter, giving a poly-logarithmic approximation for any fixed number of cable types. Subsequent to the work described here, Gupta, Krishnaswamy and Ravi [97, 98] considered RENT-OR-BUY and BUY-AT-BULK when terminal pairs must route their flow along *edge-disjoint* paths; they gave algorithms for the multi-commodity versions of RENT-OR-BUY with $k > 1$, and for UNIFORM-BUY-AT-BULK when $k = 2$.

For further references on the large literature on related network design problems, we refer the reader to [121] for a recent survey, to [73, 152, 39, 98] for various pointers to approximation algorithms on connectivity problems, and to [146, 99, 5, 48, 9, 98] for pointers to algorithms on BUY-AT-BULK network design and related problems.

5.1.2 Overview of Results and Algorithmic Techniques:

We analyze simple combinatorial algorithms for the three single-sink vertex-connectivity network design problems described earlier; our algorithms are natural extensions of known combinatorial

[5] This ratio is obtained when the total amount of traffic to be supported by the network is polynomially bounded.

algorithms for the $k = 1$ case. We prove bounds on the approximation ratio of the algorithms using two techniques: one based on element-connectivity, and the second based on the duals of natural LP relaxations. (The LP relaxations are used only for the analyses of our combinatorial algorithms.) This leads to the following results:

- An $O(k \log |T|)$ approximation for SS-k-Connectivity.
- An $O(k \log |T|)$ approximation for SS-k-Rent-or-Buy.
- An $O((\log |T|)^{O(b)})$ approximation for the SS-k-Buy-at-Bulk with b cable-types when $k = 2$.
- A $2^{O(\sqrt{\log h})}$ approximation for Non-Uniform-SS-k-Buy-at-Bulk for each fixed k.

As mentioned above, our initial analyses used the LP-based approach, obtaining weaker bounds of $O(k^{O(k)} \log |T|)$; for SS-k-Connectivity, this improved the ratio of $O(k^{O(k^2)} \log^4 n)$ from [39]. The algorithm of [39] was based on solving an LP relaxation; the authors used an optimal fractional solution to argue about the costs of connecting a terminal t to other terminals via disjoint paths. We give a simple combinatorial algorithm and analyze the *dual* of a natural linear programming relaxation. For SS-k-Connectivity, a (online) greedy algorithm is to order the terminals arbitrarily and add terminals one by one while maintaining a feasible solution for the current set of terminals. When $k = 1$, (that is, for Steiner Tree), this greedy algorithm gives an $O(\log |T|)$ approximation. However, it can be shown easily that even for $k = 2$, this same algorithm (and in fact any deterministic online algorithm), can return solutions of value $\Omega(|T|)\mathrm{OPT}$. Interestingly, our algorithm for SS-k-Connectivity applies the greedy strategy in *reverse* and has a good approximation ratio!

The LP dual-based analysis we present is inspired by the dual-packing arguments that have been used earlier for the node-weighted Steiner Tree problem [96] and the single-sink Buy-at-Bulk problems [49, 47]. These prior arguments were for $k = 1$, where distance-based arguments via balls grown around terminals can be used. For $k \geq 2$ these arguments do not apply. Nevertheless, we show the effectiveness of the dual-packing approach by using non-uniform balls. These non-uniform balls are derived in a natural fashion by solving an auxiliary min-cost flow problem for each terminal and interpreting the dual of the min-cost flow LP. We believe that this interpretation is of technical

interest. We also use this dual based analysis to analyze algorithms for the Rent-or-Buy and Buy-at-Bulk problems, although we require more sophisticated machinery.

Chuzhoy and Khanna [68] obtained results for SS-k-Connectivity independently and concurrently; their algorithm is essentially the same as ours, but their analysis relies on an important structural decomposition of a feasible integral solution to the problem. They proved that this algorithm is an $O(k \log |T|)$-approximation for SS-k-Connectivity, significantly improving the dependence on k we obtain via the dual-based approach. The heart of their structural result is that any optimal solution to SS-k-Connectivity can be (approximately) decomposed into a collection of k *element-disjoint* paths from each terminal to other terminals. Using the Reduction Lemma from Chapter 4, we give an extremely simple proof of this decomposition result; thus, we show that our algorithm has an approximation ratio of $O(k \log |T|)$ in Section 5.2.1. Note that the hardness results of [119, 39] imply that the approximation ratio of any algorithm has to depend on k in some form, but it may be possible to obtain an $O(k)$-approximation.

Thus, we obtain two bounds on the approximation ratio of our SS-k-Connectivity algorithm; one based on element-connectivity, and the other based on the dual of a natural LP relaxation. Though the approximation ratio it yields is weaker, we believe that our dual-based analysis is of independent technical value. The exponential dependence on k is an artifact of a combinatorial lemma we prove on intersecting path systems. If a natural conjecture regarding our dual packing of the non-uniform balls is true, this would imply that the dual-based analysis could also yield an approximation ratio with a polynomial dependence on k. Further, this dual-based analysis is crucial to our algorithms for SS-k-Buy-at-Bulk, which is not considered in [68].

Before we discuss the more general SS-k-Rent-or-Buy and SS-k-Buy-at-Bulk problems, we note that [39] observed that the SS-k-Connectivity approximation ratio applies also to the Subset-k-Connectivity problem; here the objective is to find a min-cost subgraph such that T is k-connected. It is also easy to see that the approximation ratio for the single-sink version only worsens by a factor of k if the terminals have different connectivity requirements in $\{1, 2, \ldots, k\}$.

For the SS-k-Rent-or-Buy problem, ours is the first non-trivial result for any $k \geq 2$. Our algorithm is a straightforward generalization of the simple random-sampling algorithm of Gupta *et al.* [99] for $k = 1$. Again we give two analyses: the first, using the element-connectivity approach

and the strict cost-sharing framework of [99] gives an $O(k \log |T|)$ approximation ratio. The second extends the dual-based analysis used for SS-k-Connectivity, and again obtains a ratio with exponential dependence on k.

The only non-trivial algorithm previously known for the SS-k-Buy-at-Bulk problem with $k > 1$ was due to [9]; they gave an $O(1)$ approximation for $k = 2$ in the *single-cable* setting, and certain other results that can be derived from it. Our algorithm for Uniform-SS-k-Buy-at-Bulk uses ideas from [9] together with a natural clustering strategy previously used for $k = 1$. This is where the dual-based analyses of SS-k-Connectivity and SS-k-Rent-or-Buy are useful: The algorithm for SS-k-Rent-or-Buy randomly samples some terminals, and buys (infinite-capacity) edges connecting them to the route. It then rents edges to k-connect each unsampled terminal to sampled terminals that are near it. For the Buy-at-Bulk clustering application we need an extra *balance* condition which ensures that the number of unsampled terminals connected to any sampled terminal is no more than βM, where $\beta \geq 1$ is not too large. The dual-based analysis allows us to guarantee precisely this condition, and hence we obtain an $O(\log |T|)^{O(b)}$-approximation when $k = 2$. We believe that our algorithm and analysis for SS-k-Buy-at-Bulk can be extended to $k > 2$; this is discussed further in Section 5.4. Using additional ideas from [9], our algorithms for SS-k-Rent-or-Buy and SS-k-Buy-at-Bulk when $k = 2$ can be extended to the multi-commodity (as opposed to the single-sink) setting.

For Non-Uniform-SS-k-Buy-at-Bulk, we analyze a simple randomized greedy inflation algorithm (suggested by Charikar and Kargiazova [40] for $k = 1$), and show that it achieves a non-trivial approximation for each fixed k; see Section 5.4.1 for intuition and details.

Chapter Outline

In Section 5.2, we consider SS-k-Connectivity, and give an $O(k \log |T|)$-approximation. We also provide our more complex original LP-based analysis of the weaker approximation ratio $O(f(k)k^2 \log |T|)$ in Section 5.2.2 ; this analysis is of independent interest, and unlike the simpler proof for SS-k-Connectivity, it extends to the more general SS-k-Buy-at-Bulk. The reader who does not wish to focus on SS-k-Buy-at-Bulk may safely skip Section 5.2.2 on first reading.

We consider SS-k-Rent-or-Buy in Section 5.3. Using techniques of Gupta *et al.* [99], we show that our simpler analysis for SS-k-Connectivity extends to SS-k-Rent-or-Buy, yielding an $O(k \log |T|)$-approximation ratio for this problem as well. However, we also extend the LP-based analysis to this problem in Sections 5.3.1 to 5.3.3 in order to ensure a certain balance condition; these sections may also be skipped.

In Section 5.4, we consider the uniform and non-uniform problems separately. For Uniform-SS-k-Buy-at-Bulk, we combine ideas from the LP-based analysis for SS-k-Rent-or-Buy and [9] to give an approximation algorithm for $k = 2$. We study the harder Non-Uniform-SS-k-Buy-at-Bulk in Section 5.4.1, and describe an algorithm achieving the weaker approximation ratio of $O(k \cdot 2^{O(\sqrt{\log n})})$.

5.2 Connectivity

In this section we analyze a simple reverse-greedy algorithm for SS-k-Connectivity. Formally, the input to the problem is an edge-weighted graph $G = (V, E)$, an integer k, a specified root vertex r, and a set of terminals $T \subseteq V$. (Throughout this chapter, we use h to denote $|T|$.) The goal is to find a min-cost edge-induced subgraph H of G such that H contains k vertex-disjoint paths from each terminal t to r.

The key concept is that of *augmentation*. Let $T' \subseteq T$ be a subset of terminals and let H' be a subgraph of G that is feasible for T'. For a terminal $t \in T \setminus T'$, a set of k paths $p_1, \ldots, p_k$ is said to be an augmentation for t with respect to T' if (i) p_i is a path from t to some vertex in $T' \cup \{r\}$ (ii) the paths are internally vertex disjoint and (iii) a terminal $t' \in T'$ is the endpoint of at most one of the k paths. Note that the root is allowed to be the endpoint of more than one path. The following proposition is easy to show via a simple min-cut argument.

Proposition 5.1. *If $p_1, p_2, \ldots, p_k$ is an augmentation for t with respect to T' and H' is a feasible solution to T' then $H \cup (\bigcup_i p_i)$ is a feasible solution for $T' \cup \{t\}$.*

Given T' and t, the augmentation cost of t with respect to T' is the cost of a min-cost set of paths that augment t wrt to T'. (Thus, one can find the augmentation cost of t in polynomial time using, for instance, algorithms for min-cost flow.) If T' is not mentioned, we implicitly assume that

$T' = T \setminus \{t\}$. With this terminology and Proposition 5.1, it is easy to see that the algorithm below finds a feasible solution.

REVERSE-GREEDY:
Let $t \in T$ be a terminal of minimum augmentation cost.
Recursively solve the instance of SS-k-CONNECTIVITY on G, with terminal set $T' = T - \{t\}$.
Augment t wrt T', paying (at most) its augmentation cost.

The rest of the section is devoted to showing that REVERSE-GREEDY achieves a good approximation. The key step in the analysis of the algorithm is to bound the augmentation cost of terminals, as shown in the following lemma:

Lemma 5.2. *Given an instance of* SS-k-CONNECTIVITY, *let* OPT *denote the cost of an optimal solution. For each terminal t, let $AugCost(t)$ denote the augmentation cost of t. Then,* $\sum_t AugCost(t) \leq 8k \cdot \mathrm{OPT}$.

In our initial proof of Lemma 5.2, we obtained a weaker bound on the augmentation cost of terminals; see Section 5.2.2. Chuzhoy and Khanna [68] then proved Lemma 5.2, with a weaker constant ($18k+3$ instead of $8k$); as mentioned previously, their proof was technically involved. We give a simple proof of this lemma, but first show that it suffices to obtain the desired approximation ratio.

Given Lemma 5.2, it is easy to see that the minimum augmentation cost of a terminal (and hence the cost paid by REVERSE-GREEDY in the last step) is at most $8k\mathrm{OPT}/h$. In fact, since the *average* augmentation cost of a terminal is also bounded by this value, the lemma also allows one to prove that a greedy algorithm with a random ordering of terminals suffices to obtain an $O(k \log |T|)$-approximation.

RANDOM-GREEDY:
Permute the terminals randomly.
Let t_j denote the jth terminal in the permutation and let $T_j = \{t_1, \ldots, t_j\}$.
Subgraph $H \leftarrow \emptyset$
For $i = 1$ to $|T|$.
  Add to H a min-cost augmentation of t_i with respect to T_{i-1}.
Output the subgraph H.

Note that this is a greedy algorithm except for the initial randomization. This randomization

is crucial; as mentioned previously, even for $k = 2$ there exist permutations that yield a solution of cost $\Omega(|T| \cdot \mathrm{OPT})$. Thus the order of terminals is of considerable importance in the performance of the greedy algorithm. This is in contrast to the case of $k = 1$, namely Steiner Tree, for which the greedy online algorithm does have a performance ratio of $O(\log |T|)$.

Lemma 5.2 and a simple inductive proof give the following theorem.

Theorem 5.3. *Algorithm* Reverse-Greedy *is an $O(k \log h)$-approximation algorithms for* SS-k-Connectivity. *Algorithm* Random-Greedy *obtains an approximation ratio of $O(k \log h)$ in expectation.*

Proof. A straightforward induction on h shows that the cost of the solution returned by Reverse-Greedy is at most $8kH_h\mathrm{OPT}$, where H_h denotes the hth harmonic number. In the base case of $h = 1$, we obtain an optimal solution. For $h > 1$, let t be the terminal of minimum augmentation cost. The induction hypothesis implies that the solution returned by the recursive call on $T' = T \setminus \{t\}$ has cost at most $8kH_{h-1}\mathrm{OPT}$, as an optimal soluion on T' clearly has cost at most OPT. Together with the augmentation cost of t, we pay at most $8kH_h\mathrm{OPT}$.

To see that the same bound on the expected cost of the solution returned by Random-Greedy, we simply use the fact that the expected augmentation cost of a random terminal is at most $8k\mathrm{OPT}/h$. □

We prove the crucial Lemma 5.2 in Section 5.2.1; the proof is combinatorial, and uses ideas from Chapter 4 together with some ideas from [68]. We also give a proof of a weaker bound in Section 5.2.2. This latter proof may be of independent interest, and extends to more general problems that we consider later.

5.2.1 An Element-Connectivity Based Proof of Lemma 5.2

The main ingredient in the proof of Lemma 5.2, as shown by [68], is the following weaker statement involving paths that are *element-disjoint*, as opposed to vertex-disjoint.

Lemma 5.4 (Element-Connectivity, [68])**.** *Given an instance of* SS-k-Connectivity *let* OPT *denote the cost of an optimal solution. For each terminal t, let $ElemCost(t)$ denote the minimum*

cost of a set of k internally vertex-disjoint paths from any terminal t to $T \cup \{r\} - t$. Then, $\sum_{t \in T} ElemCost(t) \leq 2\mathrm{OPT}$.

It is shown in [68] that one can prove Lemma 5.2 by repeatedly invoking Lemma 5.4 to obtain a large collection of paths from each $t \in T$ to other terminals, and applying a flow-scaling argument. The heart of the proof of the crucial Lemma 5.4, is a structural theorem of [68] on *spiders*: A spider is a tree containing at most a single vertex of degree greater than 2. If such a vertex exists, it is referred to as the *head* of the spider, and each leaf is referred to as a *foot*. Thus, a spider may be viewed as a collection of disjoint paths (called *legs*) from its feet to its head. If the spider has no vertex of degree 3 or more, any vertex of the spider may be considered its head. Vertices that are not the head or feet are called intermediate vertices of the spider. Our Reduction Lemma from Chapter 4 allows us to give an extremely easy inductive proof of the Spider Decomposition Theorem below,[6] greatly simplifying the proof of [68].

Theorem 5.5 ([68])**.** *Let $G(V, E)$ be a graph with a set $B \subseteq V$ of black vertices such that every pair of black vertices is k-element connected. There is a subgraph H of G whose edges can be partitioned into spiders such that:*

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

Before giving the formal short proof we remark that if the graph is bipartite then the collection of spiders is trivial to see: they are simply the edges between the black vertices and the stars rooted at each white vertex! Thus the Reduction Lemma effectively allows us to reduce the problem to a trivial case.

Proof. We prove this theorem by induction on the number of edges between white vertices in G. As the base case, we have a graph G with no edges between white vertices; therefore, G is bipartite. (Recall that there are no edges between black vertices.) Each pair of black vertices is k-element

[6] In the decomposition theorem of [68], the spiders satisfy a certain additional technical condition; the proof of Lemma 5.2 in [68] relies on this condition. We give a modified proof of Lemma 5.2 that does not require the condition.

connected, and hence every black vertex has at least k white neighbors. Let every $b \in B$ mark k of its (white) neighbors arbitrarily. Every white vertex w that is marked at least twice becomes the head of a spider, the feet of which are the black vertices that marked w. For each white vertex w marked only once, let b be its neighbor that marked it, and b' be another neighbor. We let $b-w-b'$ be a spider with foot b and head b'. It is easy to see that the spiders are disjoint, and that they satisfy all the other desired conditions.

For the inductive step, consider a graph G with an edge pq between white vertices. If all black vertices are k-element connected in $G_1 = G - pq$, then we can apply induction, and find the desired subgraph of G_1 and hence of G. Otherwise, by Theorem 4.1.1, we can find the desired set of spiders in $G_2 = G/pq$. If the new vertex $v = pq$ is not in any spider, this set of spiders exists in G, and we are done. Otherwise, let S be the spider containing v. If v is not the head of S, let x, y be its neighbors in S. Either x and y are both adjacent to p, or both adjacent to q, or (without loss of generality) x is adjacent to p and y to q. Therefore, we can replace the path $x - v - y$ in S with one of $x-p-y$, $x-q-y$, or $x-p-q-y$. If v is the head of S, we know that it has at least 2 feet. If at least 2 legs of S are incident to each of p and q, we can create two new spiders S_p and S_q, with heads p and q respectively; S_p contains the legs of S incident to p, and S_q the legs incident to q. If all the legs of S are incident to p, we let p be the head of the spider in G; the case in which all legs are incident to q is symmetric. If neither of these cases holds, it follows that (without loss of generality) exactly one leg ℓ of S is incident to p, with the remaining legs being incident to q. We let q be the head of the new spider, and add p to the leg ℓ. □

The authors of [68] showed that, once we have the Spider Decomposition Theorem, it is very easy to prove Lemma 5.4.

Proof of Lemma 5.4.([68]) In an optimal solution H to an instance of SS-k-Connectivity, every terminal is k-vertex-connected to the root. Let the terminals be black vertices, and non-terminals be white; it follows that all the terminals are k-element connected to the root in H, and hence to each other. Therefore, we can find a subgraph of H of total cost at most OPT which can be partitioned into spiders as in Theorem 5.5. For each spider S and every terminal t that is a foot of S, we find a path entirely contained within S from t to another terminal. Each edge of S is in at most two such paths; since the spiders are disjoint and each terminal is a foot of k spiders, we

obtain the desired result.

If the head of S is a terminal, the path for each foot is simply the leg of S from that foot to the head. Each edge of S is in a single path. If the head of S is a white vertex, it has at least two feet. Fix an arbitrary ordering of the feet of S; the path for foot i follows leg i from the foot to the head, and then leg $i+1$ from the head to foot $i+1$. (The path for the last foot follows the last leg, and then leg 1 from the head to the foot.) It is easy to see that each edge of S is in exactly two paths; this completes the proof. □

Finally, we give a proof of Lemma 5.2 that relies only on the statement of Lemma 5.4. Our proof is a technical modification of the one in [68] and as previously remarked, does not need to rely on the additional condition on the spiders that [68] guarantees. Our proof also gives a slightly stronger bound on $\sum_t AugCost(t)$ than that of [68].

Proof of Lemma 5.2. We give an algorithm to find an augmentation for each terminal that proceeds in $4k^2$ iterations: In each iteration, for every terminal t, it finds a set of k internally vertex-disjoint paths from t to other terminals or the root. Let $\mathcal{P}_i(t)$ denote the set of paths found for terminal t in iteration i. These paths have the following properties:

1. For each terminal t, every other terminal is an end-point of fewer than $4k^2 + 2k$ paths in $\bigcup_i \mathcal{P}_i(t)$.

2. In each iteration i, $\sum_t Cost(\mathcal{P}_i(t)) \leq 4k\mathrm{OPT}$.

Given these two properties, we can prove the theorem as follows: Separately for each terminal t, send 1 unit of flow along each of the paths in $\bigcup_i \mathcal{P}_i(t)$; we thus have a flow of $4k^2 \cdot k$ units from t to other terminals. Scale this flow down by $4k^2 \cdot (k + \frac{1}{2})/k$, to obtain a flow of $\frac{k^2}{k+1/2} > k - 1/2$ from t to other terminals. After the scaling step, the net flow through any vertex (terminal or non-terminal) is at most 1, since the maximum flow through a vertex before scaling was $4k^2 + 2k$. Let $FlowCost(t)$ denote the cost of this scaled flow for terminal t; if we now scale the flow *up* by a factor of 2, we obtain a flow of value greater than $2k-1$ from t to other terminals, in which the flow through any vertex besides t is at most 2. Therefore, by the integrality of min-cost flow, we can find an integral flow of $2k-1$ units from t to other terminals, of total cost at most $2FlowCost(t)$. Let E_t be the set of edges used in this integral flow; it follows that $cost(E_t) \leq 2FlowCost(t)$. It is

also easy to see that E_t contains k disjoint paths from t to k distinct terminals, by observing that a hypothetical cutset of size $k-1$ contradicts the existence of the flow of value $2k-1$ in which the flow through a vertex is at most 2.

Therefore, we have found k disjoint paths from t to k other terminals, of total cost $2FlowCost(t)$. To bound the cost over all terminals, we note that from the second property above, we have $\sum_t FlowCost(t) \leq 4k^2 \cdot 4k\mathrm{OPT}/\left(4k^2\frac{k+1/2}{k}\right)$, which is less than $4k\mathrm{OPT}$. It follows that the total cost of the set of paths is at most $2\sum_t FlowCost(t) < 8k\mathrm{OPT}$.

It remains only to show that we can find a set of paths for each terminal in every iteration that satisfies the two desired properties. The proof below uses induction on the number of iterations i to prove property 1: After i iterations, for each terminal t, every other terminal is an end-point of fewer than $i+2k$ paths in $\bigcup_i \mathcal{P}_i(t)$.

In iteration i, for each terminal t, let $Blocked(t)$ denote the set of terminals in $T-t$ that have been the endpoints of at least $(i-1)+k$ paths in $\bigcup_{j=1}^{i-1}\mathcal{P}_j(t)$. (Note that the root r is never in any $Blocked(t)$.) Since the total number of paths that have been found so far is $(i-1)k$, $|Blocked(t)| < k$. Construct a directed graph D on the set of terminals, with edges from each terminal t to the terminals in $Blocked(t)$. Since the out-degree of each vertex in D is at most $k-1$, there is a vertex of in-degree at most $k-1$; therefore, the digraph D is $2k-2$ degenerate and so can be colored using $2k-1$ colors. Let $C_1, C_2, \ldots C_{2k-1}$ denote the color classes in a proper coloring of D; if $t_1, t_2 \in C_j$, then in iteration i, $t_1 \notin Blocked(t_2)$ and $t_2 \notin Blocked(t_1)$. For each color class C_j in turn, consider the terminals of C_j as black, and the non-terminals and terminals of other classes as white. There is a graph of cost OPT in which every terminal of C_j is k-vertex-connected to the root, so C_j is k-element-connected to the root in this graph even if terminals not in C_j are regarded as white vertices. From Lemma 5.4, for every C_j, we can find a set of internally disjoint paths from each $t \in C_j$ to $C_j \cup \{r\} - \{t\}$ of total cost at most 2OPT. If these paths contain other terminals in $T - C_j$ as intermediate vertices, trim them at the first terminal they intersect. It follows that $\sum_j \sum_{t\in C_j} Cost(\mathcal{P}_i(t)) < 4k\mathrm{OPT}$, establishing property 2 above.

To conclude, we show that for each terminal t, after iteration i, every other terminal is an end-point of fewer than $i+2k$ paths in $\bigcup_{j=1}^{i}\mathcal{P}_j(t)$. Let C be the color class containing t; if $t' \in Blocked(t)$, at most one new path in $\mathcal{P}_i(t)$ ends in t', as the paths for t are disjoint except at

terminals in C, and $t' \notin C$. By induction, before this iteration t' was the endpoint of fewer than $(i-1)+2k$ paths for t, and so after this iteration, it cannot be the endpoint of $i+2k$ paths for t. If $t' \notin \mathit{Blocked}(t)$, it was the endpoint of at most $(i-1)+k-1$ paths for t before this iteration; even if all the k paths for t in this iteration ended at t', it is the endpoint of at most $i+2k-2$ paths for t after the iteration. This gives us the desired property 1, completing the proof. □

Lemma 5.2 and Lemma 5.4 have applications to more general problems including the node-weighted version of SS-k-Connectivity [68], which we do not discuss in this thesis, and rent-or-buy and buy-at-bulk network design [52]. (See Sections 5.3 and 5.4.)

5.2.2 An LP-Based Bound on Augmentation Costs

In this section, we give a weaker bound on the augmentation cost of the terminals than that of Lemma 5.2; we prove the following lemma:

Lemma 5.6. *Given an instance of* SS-k-Connectivity *on h terminals, let* OPT *denote the cost of an optimal solution, and let $\mathit{AugCost}(t)$ denote the augmentation cost of terminal t. Then,* $\min_t \mathit{AugCost}(t) \leq 2f(k)k^2 \cdot \frac{\mathrm{OPT}}{h}$ *where* $f(k) \leq 3^k k!$. *It also follows that* $\sum_t \mathit{AugCost}_t \leq 2f(k)k^2 \log h \cdot \mathrm{OPT}$.

Our proof proceeds by constructing a natural linear program for the problem and using a dual-based argument. We will describe a feasible dual solution of cost at least $\frac{h \min_t \mathit{AugCost}(t)}{2f(k)k^2}$. As this is a lower bound on OPT, we obtain the lemma. The primal and its dual linear programs for SS-k-Connectivity are shown below. We remark that our linear program is based on a path-formulation unlike the standard cut-based (setpair) formulation for VC-SNDP [82, 79]. However, the optimal solution values of the two relaxations are the same. The path-formulation is more appropriate for our analysis; this is inspired by a similar approach in [49, 47].

In the primal linear program below, and throughout the chapter, we let $\mathcal{P}_t^k$ denote the collection of all sets of k vertex-disjoint paths from t to the root r. We use the notation $\vec{P}$ to abbreviate $\{p_1, p_2, \ldots p_k\}$, an unordered set of k disjoint paths in $\mathcal{P}_t^k$. Finally, we say that an edge $e \in \vec{P}$ if there is some $p_j \in \vec{P}$ such that $e \in p_j$. In the LP, the variable x_e indicates whether or not the edge e is in the solution. For each $\vec{P} \in \mathcal{P}_t^k$, the variable $f_{\vec{P}}$ is 1 if terminal t selects the k paths of $\vec{P}$ to connect to the root, and 0 otherwise.

$$\textbf{Primal-Conn} \quad \min \sum_{e \in E} c_e x_e$$

$$\sum_{\vec{P} \in \mathcal{P}_t^k} f_{\vec{P}} \geq 1 \qquad (\forall t \in T)$$

$$\sum_{\vec{P} \in \mathcal{P}_t^k | e \in \vec{P}} f_{\vec{P}} \leq x_e \qquad (\forall t \in T, e \in E)$$

$$x_e, f_{\vec{P}} \in [0,1]$$

$$\textbf{Dual-Conn} \quad \max \sum_{t \in T} \alpha_t$$

$$\sum_t \beta_e^t \leq c_e \qquad (\forall e \in E)$$

$$\alpha_t \leq \sum_{e \in \vec{P}} \beta_e^t \qquad \left(\forall \vec{P} \in \mathcal{P}_t^k\right)$$

$$\alpha_t, \beta_t^e \geq 0$$

The value $f_{\vec{P}}$ can be thought of as the amount of "flow" sent from t to the root along the set of paths in $\vec{P}$. The first constraint requires that for each terminal, a total flow of at least 1 unit must be sent along various sets of k disjoint paths.

Overview of the Dual-Packing Analysis

We prove Lemma 5.6 based on a dual-packing argument. In order to do this we first interpret the variables and constraints in **Dual-Conn**. There is a dual variable α_t for each $t \in T$. We interpret α_t as the total cost that t is willing to pay to connect to the root. In addition there is a variable β_e^t which is the amount that t is willing to pay on edge e. The dual constraint $\sum_t \beta_e^t \leq c_e$ requires that the total payments on an edge from all terminals is at most c_e. In addition, for each terminal t, the total payment α_t should not exceed the min-cost k-disjoint paths to the root with costs given by the β_e^t payments of t on the edges.

Let $\alpha = \min_t AugCost(t)$. To prove Lemma 5.6 it is sufficient to exhibit a feasible setting for the dual variables in which $\alpha_t \geq \alpha/(f(k)k^2)$ for each terminal t. How do we do this? To understand the overall plan and intuition, we first consider the Steiner Tree problem (the case of $k = 1$). In this case, $\alpha = \min_t AugCost(t)$ is the shortest distance between any two terminals. For each t consider the ball of radius $\alpha/2$ centered around t; these balls are *disjoint*. Hence, setting $\alpha_t = \alpha/2$ and $\beta_e^t = c_e$ for each e in t's ball (and $\beta_e^t = 0$ for other edges) yields a feasible dual solution. This interpretation is well-known and underlies the $O(\log |T|)$ bound on the competitiveness of the greedy algorithm for online Steiner Tree problem.

Extending the above intuition to $k > 1$ is substantially more complicated. We again to wish to

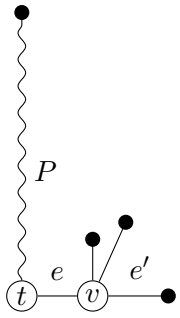


Figure 5.1: An example which shows that there may not be a ball of radius $\Omega(\alpha)$ that is centered at terminal t and is disjoint from other terminals.

define balls of radius $\Omega(\alpha)$ that are disjoint. As we remarked earlier, for $k = 1$ one can work with distances in the graph and the ball of radius $\alpha/2$ is well defined. For $k > 1$, there may be multiple terminals at close distance d from a terminal t, but nevertheless $AugCost(t)$ could be much larger than d. The reason for this is that t needs to reach k terminals via *vertex disjoint* paths and there may be a vertex v whose removal disconnects t from *all* the nearby terminals. Consider the example in Figure 5.1 above, where filled circles denote other terminals: The terminal t is willing to pay for e and edges on P but not e'. There does not appear to be a natural notion of a ball; however, we show that one can define some auxiliary costs on the edges (that vary based on t) which can then be used to define a ball for t. The complexity of the analysis comes from the fact that the balls for different terminals t are defined by different auxiliary edge costs. Now we show how the auxiliary costs can be defined.

We can obtain the augmentation cost of a terminal t via a min-cost flow computation in an associated *directed* graph $G_t(V_t, E_t)$ constructed from G in the following standard way: make 2 copies v^+ and v^- of each vertex $v \neq t$, with a single edge/arc between them, and for each undirected edge uv in G, edges from u^+ to v^- and v^+ to u^-. Further, we add a new vertex r_t as sink, and for each terminal $\hat{t}$ other than t, add a 0-cost edge from $\hat{t}^+$ to r_t. Recall that an augmentation for t is a set of k disjoint paths from t that end at distinct terminals in $T \setminus \{t\}$, or the root. While constructing G_t, then, the root is also considered a terminal, and we make k copies of it to account for the fact that multiple paths in the augmentation can end at the root; each such copy is also connected to the sink r_t. We now ask for a minimum cost set of k disjoint paths

from t to r_t[7]; these correspond to a minimum-cost augmentation for t. It is useful to use a linear programming formulation for the min-cost flow computation. The linear program for computing the augmentation cost of t, and its dual are shown below. We refer to these as **Primal-Aug**(t) and **Dual-Aug**(t) respectively.

Primal-Aug(t) $\quad \min \sum_{e \in E_t} c_e f_e$

$$\begin{aligned}
\sum_{e \in \delta^-(r_t)} f_e &\geq k \\
\sum_{e \in \delta^-(v)} f_e &= \textstyle\sum_{e=\delta^+(v)} f_e \quad (\forall v \neq t, r_t) \\
f_e &\leq 1 \quad (\forall e \in E_t) \\
f_e &\geq 0 \quad (\forall e \in E_t)
\end{aligned}$$

Dual-Aug(t) $\quad \max k \cdot \Pi - \sum_e z_e^t$

$$\begin{aligned}
\Pi - \pi_t(u) &\leq c_e + z_e^t \quad (\forall e = (u, r_t)) \\
\pi_t(v) - \pi_t(u) &\leq c_e + z_e^t \quad (\forall e = (u, v), u \neq t, v \neq r_t) \\
\pi_t(v) &\leq c_e + z_e^t \quad (\forall e = (t, v) \in E_t) \\
z_e^t &\geq 0 \quad (\forall e \in E_t)
\end{aligned}$$

Note that the cost of an optimal solution to **Primal-Aug**(t) is equal to $AugCost(t)$. The interesting aspect is the interpretation of the dual variables. The variables z_e^t are auxiliary costs on the edges. One can then interpret the dual **Dual-Aug**(t) as setting z_e^t values such that the distance from t, with modified cost of each edge e set to $c_e + z_e^t$, is equal to Π for every other terminal t'. Thus the modified costs create a ball around t in which all terminals are at equal distance!

Thus, the overall game plan of the proof is the following. For each t solve **Primal-Aug**(t) and find an appropriate solution to **Dual-Aug**(t) (this requires some care). Use these dual variables to define a notion of a non-uniform ball around t in the original graph G. This leads to a feasible setting of variables in **Dual-Conn** (with the balls being approximately disjoint). Although the scheme at a high level is fairly natural, the technical details are non-trivial and somewhat long. In particular, one requires an important combinatorial lemma on intersecting path systems that was formulated in [39] — here we give an improved proof of a slight variant that we need. The use of this lemma leads to the exponential dependence on k in Lemma 5.6. A certain natural conjecture regarding the non-uniform balls, if true, would lead to a polynomial dependence on k.

[7]Note that we do not make two copies of t, as we will never use an incoming edge to t in a min-cost set of paths. All edges are directed out of the unique copy of t.

Details of the Proof

We first give a combinatorial way to obtain optimal solutions to **Primal-Aug**(t) and **Dual-Aug**(t). This allows us to interpret the variable settings in a useful way. For each $i \leq k$, we let $AugCost_i(t)$ denote the minimum cost of a set of i disjoint paths from t to r_t; $AugCost(t)$, the augmentation cost of t, is precisely $AugCost_k(t)$. A minimum cost set of k paths can be found in several ways; we focus on the (combinatorial) *successive-shortest-paths* algorithm [4, 118]. In this algorithm, we start with the graph G_t, find a shortest path from t to r_t, and then construct the residual graph. Iterating k times, we obtain a min-cost set of k paths, by repeatedly finding shortest t-to-r_t paths in the appropriate residual graphs. Let $G_t(i)$ be the residual graph obtained after i iterations of the successive shortest paths algorithm, and let Q_i be the set of i disjoint paths found after i iterations. If ℓ_i is the length of the shortest path found in $G_t(i-1)$, then by the properties of the algorithm, $\ell_i = AugCost_i(t) - AugCost_{i-1}(t)$. This gives us an integral optimal solution to **Primal-Aug**(t). Also, note that an augmenting path found in iteration i may not be one of the final set of k paths found by the algorithm since the paths are found in residual graphs.

We construct an optimal feasible solution to **Dual-Aug**(t) as follows. For each vertex v, $\pi_t(v)$, the potential of v, is set to the shortest-path distance from t to v in the residual graph $G_t(k-1)$ (the final residual graph). The potential of r_t, set to Π, is given by the shortest-path distance of r_t from t in the graph $G_t(k-1)$. Note that $\Pi = AugCost_k(t) - AugCost_{k-1}(t)$. With potentials set according to this rule, there may be edges $e = (u, v)$ such that $\pi_t(v) - \pi_t(u) > c_e$; on these edges, we set $z_e^t = \pi_t(v) - \pi_t(u) - c_e$, satisfying the constraints (with equality). It is not hard to see that this is an optimal solution for **Dual-Aug**(t); see [118] for a discussion of the successive-shortest-path algorithm and the associated potentials, using slightly different notation. Here, we merely observe that these settings satisfy the complementary slackness conditions. In particular, we characterize the edges with $z_e^t > 0$ in the following claim, and then prove a few more properties of the dual variables set by this process.

Claim 5.7. *For all $e \in E_t$, $z_e^t > 0$ only if $e \in Q_{k-1} \cap Q_k$, where Q_i denotes the set of i disjoint paths found by the successive shortest paths algorithm after i iterations.*

Proof. If $z_e^t > 0$ for $e = (u, v)$, the shortest-path distance from t to v in $G_t(k-1)$ must be greater than the sum of c_e and the shortest-path distance from t to u. This implies that $e = (u, v)$ cannot

exist in the residual graph $G_t(k-1)$, and so it must be an edge in Q_{k-1}. Further, it must be in Q_k, the final set of min-cost k paths; if not, it would need to be in Q_{k-1} but not Q_k. This means that the shortest path from t to r_t would have to use the edge (v, u), with cost $-c_e$ (as (u, v) does not exist in the residual graph). This implies that the shortest path to u in $G_t(k-1)$ goes through v, and hence $\pi_t(v) = \pi_t(u) + c_e$, contradicting the fact that $z_e^t > 0$. □

Proposition 5.8. Π*, the shortest-path distance from* t *to* r_t *in* $G_t(k-1)$*, is at least* $\frac{1}{k}$*AugCost*(t)*.*

Proof. Let p be a longest path in a set of k disjoint $t - r_t$ paths in G_t of total cost $AugCost(t)$; the length of p is at least $\frac{AugCost(t)}{k}$. The length of the remaining $k-1$ paths is, then, at most $(1-1/k)AugCost(t)$; this implies that $AugCost_{k-1}(t)$, the cost of a minimum-cost set of $k-1$ paths, is also at most this value. Therefore, the shortest-path distance from t to r_t in $G_t(k-1)$, which is precisely $AugCost_k(t) - AugCost_{k-1}(t)$, is at least $\frac{AugCost(t)}{k}$. □

For each terminal v, let $\Pi(t)$ denote the value of Π in the linear program **Dual-Aug**(t). From Proposition 5.8, we have $\frac{AugCost(t)}{k} \leq \Pi(t) \leq AugCost(t)$. (Note that for $k = 1$, $\Pi(t)$ is precisely the distance between t and its closest terminal.) For each edge e, we think of $c_e + z_e^t$ as an "adjusted" length/cost ℓ_e^t for e. Let $d_t(v)$ be the the shortest-path distance from t to v in G_t, under the distance function ℓ_e^t. Recall that we set $\pi_t(v)$, the potential of v, to be the shortest-path distance from t to v in the residual graph $G_t(k-1)$; the constraints of **Dual-Aug**(t) enforce that $d_t(v) \geq \pi_t(v)$.

Claim 5.9. *Under the distance function* $\ell_e^t = c_e + z_e^t$*, any path from* t *to* r_t *in* G_t *has length at least* $\Pi(t)$*.*

Proof. Note that $\Pi(t)$ is the the potential of r_t in our solution to **Dual-Aug**(t). The shortest-path distance $d_t(r_t)$ from t to r_t is at least this potential. □

Claim 5.10. $\sum_e z_e^t \leq (k-1)\Pi$.

Proof. If this were not true, our solution to **Dual-Aug**(t) would have value less than Π, and hence less than $AugCost(t)$. But this is an optimal solution, with value equal to the minimum-cost set of k paths from t to r_t, giving a contradiction. □

Using the settings from the **Dual-Aug**(t) programs, we now describe a feasible setting for the dual program **Dual-Conn**. Note that **Dual-Conn** refers to the original (undirected) graph G,

while for each **Dual-Aug**(t), we worked with a directed graph G_t which contained 2 copies of each vertex and an additional sink r_t. For ease of notation, we subsequently describe, for each terminal t, how β_e^t should be set in the appropriate graph G_t. In G, for every undirected edge $e = uv$ we set β_e^t to be $\max\{\beta^t_{\overrightarrow{e}}, \beta^t_{\overleftarrow{e}}\}$, where $\overrightarrow{e}, \overleftarrow{e}$ correspond to the directed edges (u, v) and (v, u) respectively.

Let $\Pi_{min} = \min_t \Pi(t)$. For each terminal t, set $\alpha_t = \alpha = \Pi_{min}/2k$; we must now give a setting for the β_e^t variables, for which we use the dual program **Dual-Aug**(t). Consider the (directed) edge $e = (u, v)$: If $d_t(u) \geq \alpha$, set $\beta_e^t = 0$; otherwise, set $\beta_e^t = \min\{c_e, \alpha - d_t(u)\}$. We interpret this as setting $\beta_e^t = c_e$ on edges e within the ball of radius α using *the edge costs* ℓ_e^t. If e is outside the ball, $\beta_e^t = 0$, and on the border, we set β_e^t depending on the portion of e within the ball. This is the ball referred to earlier (in the overview of the proof) with auxiliary costs; it is a ball of radius α using the distance metric ℓ_e^t, as opposed to the "ordinary"costs c_e.

The following two lemmas prove the (approximate) feasibility of the variable settings for **Dual-Conn**. Lemma 5.11 is the more difficult of the two, and we temporarily defer its proof.

Lemma 5.11. *Let $f(k) = 3^k \cdot k!$. For each edge e, $\sum_t \beta_e^t \leq f(k)c_e$.*

Lemma 5.12. *For all $\vec{P} \in \mathcal{P}_t^k$, $\alpha_t \leq \sum_{e\in\vec{P}} \beta_e^t$.*

Proof. Consider any set of k paths $\vec{P} \in \mathcal{P}_t^k$. We focus on the corresponding set of k directed paths $\vec{P'}$ in the directed graph G_t, and showing that $\sum_{e\in\vec{P'}} \beta_e^t \geq \alpha_t$. Since for each edge $e = uv$ in G, we set $\beta_e^t = \max\{\beta^t_{(u,v)}, \beta^t_{(v,u)}\}$, this shows that $\sum_{e\in\vec{P}} \beta_e^t \geq \alpha_t$.

Since all the paths of $\vec{P'}$ end in a terminal (the root), they must each leave the ball of ℓ_e^t radius α_t centered at t. Let E' be the set of edges in the ball of radius α_t; note that an edge $e = (u, v)$ is entirely in the ball if $d_t(u) + \ell_e^t \leq \alpha_t$, and e may be *partially* in the ball if $d_t(u) < \alpha_t < d_t(u) + \ell_e^t$. Without loss of generality, we assume that no edges are partially in the ball; such an edge can be subdivided into two pieces, one inside the ball, and the other outside.

For each path $p \in \vec{P'}$, the length of p inside the ball is at least α_t; that is, $\sum_{e\in p\cap E'} \ell_e^t \geq \alpha_t$. Since the k paths in $\vec{P'}$ are disjoint, $\sum_{e\in\vec{P'}\cap E'} \ell_e^t \geq k\alpha_t$. As $\ell_e^t = c_e + z_e^t$, and inside this ball, $\beta_e^t = c_e$, we have $\sum_{e\in\vec{P'}\cap E'} \beta_e^t + \sum_{e\in\vec{P'}\cap E'} z_e^t \geq k\alpha_t$.

We claim that $\sum_{e\in E'} z_e^t \leq (k-1)\alpha_t$; this implies that $\sum_{e\in\vec{P'}\cap E'} \beta_e^t \geq \alpha_t$, completing the proof.

It remains only to prove the claim that $\sum_{e \in E'} z_e^t \leq (k-1)\alpha_t$; consider any edge e in the ball of radius α_t with $z_e^t > 0$. From Claim 5.7, such an edge must be in Q_{k-1}, the first set of $k-1$ minimum-cost paths from t to r_t. Further, for *any* edge $(u,v) \in Q_{k-1}$, since (v,u) is a negative-cost edge in $G_t(k-1)$, $\pi_t(v) > \pi_t(u)$. Consider any path $q \in Q_{k-1}$; we show that $\sum_{e \in q \cap E'} z_e^t \leq \alpha_t$, which will prove the desired claim.

If x is the last vertex of q within the ball of radius α_t, let q' be the sub-path of q from t to x. For an edge $e = (u,v) \in q'$, $z_e^t \leq \pi_t(v) - \pi_t(u)$. Therefore, $\sum_{e \in q'} z_e^t \leq \pi_t(x) - \pi_t(t) = \pi_t(x)$. But $\pi_t(x) \leq d_t(x) \leq \alpha_t$, and hence $\sum_{e \in q'} z_e^t \leq \alpha_t$. □

Given Lemmas 5.11 and 5.12, we can now prove Lemma 5.6, bounding the minimum augmentation cost of a terminal. This completes the proof of Theorem 5.3.

Proof of Lemma 5.6. Using the settings for α_t and β_e^t as described above, we have a solution for **Dual-Conn**, in which one set of constraints is violated by a factor of $f(k)$. With these settings, $\sum_t \alpha_t = h\frac{\Pi_{min}}{2k}$. Further, $\Pi_{min} \geq \min_t AugCost(t)/k$. Therefore, $\min_t AugCost(t) \leq \frac{2k^2}{h} \sum_t \alpha_t$. Shrinking all values of α_t and β_e^t by a factor of $f(k)$, we obtain a feasible solution for **Dual-Conn**, which must have cost at most OPT. Therefore, $\min_t AugCost(t) \leq 2f(k)k^2 \cdot \frac{\text{OPT}}{h}$. □

In order to complete our proof of Lemma 5.6, we now need only to prove Lemma 5.11; first, we give some intuition underlying our proof. Suppose that (u,v) is an edge such that $\sum_t \beta_e^t > f(k)c_e$. Since each $\beta_e^t \leq c_e$, there must exist more than $f(k)$ terminals t such that $\beta_e^t > 0$; let S be the set of such terminals. Informally, for each terminal in S, u must be in its ball of radius α, and so each terminal must be fairly close to u. The essential idea, then, is that if all the terminals in S are so close to u, and hence to each other, one of them would have been able to find disjoint paths to k other terminals without going as far as Π_{min}. In order to analyze this formally we need a definition and a combinatorial lemma on intersecting paths in a graph.

Definition 5.13. *Given a graph $G = (V,E)$ a triple (X,P,v) is an intersecting path system for a set $X \subset V$ if P is a collection of (simple) paths such that $|P| = |X|$ and for each $t \in X$ there is a path $p_t \in P$ that connects t to $v \in V$. The vertex v is referred to as the root of the system and for $t, u \in X$, $prefix_t(u)$ denotes the subpath of p_u from u to its first intersection with p_t.*

In [39] a useful combinatorial lemma on intersecting path systems was crucially used in the analysis. Below we give a similar lemma with slightly improved bounds on the parameters as well as a different proof.

Lemma 5.14. *Let (X, P, v) be an intersecting path system in graph $G(V, E)$ and let $f(k)$ be the function $3^k \cdot k!$. If $|X| \geq f(k)$, there exists $k+1$ distinct vertices $t, u_1, u_2, \ldots u_k$ in X such that for $1 \leq i < j \leq k$, $\mathit{prefix}_t(u_i)$ and $\mathit{prefix}_t(u_j)$ are vertex-disjoint, except perhaps at the point of intersection with p_t.*

As observed by the authors of [39], a perfect binary tree with 2^{k-1} leaves as vertices of X shows that $f(k)$ has to be larger than 2^{k-1} for the above lemma to be true. We believe that $f(k) = 2^{k-1} + 1$ suffices; in particular, one can verify that this holds for $k = 2, 3$.

Proof of Lemma 5.14. Given an intersecting path system, let p_i be the path from $x_i \in X$ to v; we abuse notation and use $\mathit{prefix}_i(j)$ instead of $\mathit{prefix}_{x_i}(x_j)$ to denote the subpath of p_j from x_j to its first intersection with p_i.

Suppose p_i is a first path that p_j intersects. (Note that p_j may intersect other paths simultaneously with p_i.) Then, $\mathit{prefix}_i(j)$ is entirely disjoint from any other path in the system, except perhaps at the point of intersection with p_i. If this prefix intersected some path $p_{i'}$, that would contradict the choice of p_i as the first path intersected by p_j.

Construct a directed graph $H(X, E_H)$, with edge set E_H described as follows: For each $x_j \in X$, if p_i is a first path intersected by p_j, insert an edge from x_j to x_i. (In the event of ties, they may be broken arbitrarily.) Note that $|E_H| = |X|$, and by applying Proposition 5.15 below to each connected component, H has a proper 3-coloring. Let X_1 be a color class such that the total in-degree of X_1 is at least $|X|/3$. Let $X' \subseteq X_1$ be the set of vertices in X_1 with in-degree at least 1. We consider two cases:

If $|X'| \leq \frac{|X|}{3k}$, there is a vertex $t \in X'$ with in-degree at least k. This vertex t, together with its in-neighbors, satisfy the lemma.

If $|X'| > \frac{|X|}{3k}$, consider the intersecting path system obtained from (X, P, v) by considering only the paths beginning at terminals in X'. Since $X' > f(k-1)$, by induction, we obtain a terminal t, and $k-1$ other terminals (without loss of generality, by renumbering) $x_1, x_2, \ldots x_{k-1}$ such that for each $1 \leq i < j \leq k-1$, $\mathit{prefix}_t(x_i)$ and $\mathit{prefix}_t(x_j)$ are disjoint. These terminals, together with

an in-neighbor x_k of t (as X' is an independent set, $x_k \notin X'$), satisfy the lemma, as *prefix*$_t(x_k)$ is disjoint from every other path in (X, P, v). □

Proposition 5.15. *If $H(V, E)$ is a connected graph such that $|V| = |E|$, H has a proper 3-coloring.*

Proof of Lemma 5.11. If $\sum_t \beta_e^t > f(k)c_e$, we have a set S of more than $f(k)$ terminals that all have $\beta_e^t > 0$ on edge $e = (u, v)$. Each terminal $t \in S$ has a path p_t of length at most α (under the ℓ_e^t distances); these paths form an intersecting path system. From Lemma 5.14, we can find a terminal t, and k distinct terminals $x_1, x_2, \ldots, x_k$ such that for each x_i, x_j, *prefix*$_t(x_i)$ and *prefix*$_t(x_j)$ are vertex-disjoint, except perhaps at the point of intersection with p_t.

Note that since each path from a terminal to u has length at most α, we can find short paths from t to each of $x_1, \ldots, x_k$ by following p_t until y_i, the point of intersection with *prefix*$_t(x_i)$, and then following *prefix*$_t(x_i)$ "backwards" until x_i. Consider any such path from t to x_i; recall that in G_t, $d_t(r_t)$ was at least $\Pi(t)$, where r_t was the added "sink" vertex. One might conclude that since the path from t to y_i has length less than α, and the same is true for the path from y_i to x_i, we have a path from t to a terminal of length less than $2\alpha = \Pi_{min}/k$, contradicting Claim 5.9. This is incorrect, because for an edge e *different terminals may have distinct values for* ℓ_e^t. In particular, $d_{x_i}(y_i) \leq \alpha$; that is, the distance from x_i to y_i is short using edge costs $l_e^{x_i}$. This does *not* imply that the length of the path from y_i to x_i is necessarily low using edge costs ℓ_e^t.

However, a similar argument using the multiple disjoint path prefixes suffices to complete our proof. Consider the path from t to y_i; $d_t(y_i) \leq \alpha$. That is, this path is short using edge costs $\ell_e^t = c_e + z_e^t$. The path from y_i to r_t using *prefix*$_t(x_i)$ has c_e length less than α (since we know that its length is less than α using edge costs $l_e^{x_i} = c_e + z_e^{x_i}$). But since $d_t(r_t) \geq \Pi(t)$, the length of this path from y_i to r_t using edge costs z_e^t must be greater than $\Pi(t) - 2\alpha$. But since $\alpha \leq \Pi_{min}/2k$, the z_e^t length of this path must be greater than $\Pi(t)(1 - \frac{1}{k})$. This is true for each path *prefix*$_t(x_i)$, $1 \leq i \leq k$, and *these paths are entirely disjoint.* Therefore, $\sum_e z_e^t$ is greater than $k\left(\Pi(t)(1 - \frac{1}{k})\right) = (k-1)\Pi(t)$. But this contradicts Claim 5.10, completing the proof. □

5.3 Rent-or-Buy

In this section we describe and analyze a simple algorithm for the SS-k-Rent-or-Buy problem. Recall that the input to this problem is the same as that for SS-k-Connectivity, with an additional parameter M. As before, the goal is to connect terminals to the root using k disjoint paths, but now we wish to construct networks with sufficient capacity to simultaneously support traffic from each terminal to the root; in contrast, the requirement in SS-k-Connectivity was merely that each terminal be highly connected to the root. More formally, for each terminal $t \in T$, we must find k vertex-disjoint paths to the root r. The objective is to minimize the total cost of the chosen paths, where the cost of an edge e is $c_e \cdot \min\{M, |T_e|\}$ and T_e is the set of terminals whose paths contain e. In other words, an edge can either be bought at a price of Mc_e in which case any number of terminals can use it or an edge can be rented at a cost of c_e per terminal. Our algorithm given below is essentially the same as the random marking algorithm that has been shown to give an $O(1)$ approximation for the case of $k = 1$ [99].

Rent-Or-Buy-Sample:

1. Sample each terminal independently with probability $1/M$ to form a set $T' \subseteq T$.

2.1 Let H be a solution to SS-k-Connectivity on G with terminal set T'.

$\langle\langle$*Each terminal in T' is k-connected in H to the root.*$\rangle\rangle$

2.2 *Buy* the edges of H, paying Mc_e for each edge $e \in H$.

3. For each non-sampled terminal t:

Greedily *rent* disjoint paths to connect t to k distinct sampled terminals or the root.

It is easy to see that the algorithm is correct. Note that a non-sampled terminal can always find feasible paths since the root can be the endpoint of all k paths. We can easily generalize the algorithm and analysis to the case where each terminal t has a demand d_t to be routed to the root.

It is straightforward to bound the expected cost of the edges bought in Step 2.2.

Lemma 5.16. *Let* OPT *be the cost of an optimal solution to the given instance of* SS-k-Rent-or-Buy. *If we use algorithm* Reverse-Greedy *to find the graph H in Step 2.1 of* Rent-Or-Buy-Sample*, the expected cost of edges bought in Step 2.2 is at most $O(k \log |T|) \cdot \text{OPT}$.*

Proof. For each set of terminals T' sampled in Step 1, we show the existence of a solution H' to the SS-k-Connectivity instance defined on T' such that the expected cost of buying edges in H' is precisely OPT. Since we find the graph H by using the $O(k \log |T|)$-approximation for SS-k-Connectivity from Section 5.2, we obtain the lemma.

Before we describe the soution H' on the sampled terminals T', consider a fixed optimal solution with cost OPT to the original SS-k-Rent-or-Buy instance, and let B be the set of bought edges in this solution. Construct a graph G' from G by reducing the cost of edges in B to 0. It is easy to see that in the given optimal solution to SS-k-Rent-or-Buy, every terminal routes flow to the root along k disjoint paths of minimum cost in G'. Let $\vec{P}_t$ denote the k disjoint paths used by terminal t, and let $rent(t)$ be the cost in G' of the paths in $\vec{P}_t$. (This is how much the terminal t pays to rent edges.) Clearly, $\text{OPT} = Mcost(B) + \sum_{t \in T} rent(t)$.

We now describe H': Simply use B, together with $\bigcup_{t \in T'} \vec{P}_t$. (Note that though we do not know the optimal B or any $\vec{P}_t$, we are only proving the existence of H'.) The cost of H' is $Mcost(B) + M \sum_{t \in T'} rent(t)$; the expected cost is $Mcost(B) + M \sum_{t \in T} \Pr[t \in T'] \cdot rent(t)$. But since each vertex is sampled with probability $1/M$, we conclude that the expected cost of H' is $Mcost(B) + \sum_{t \in T} rent(t) = \text{OPT}$. □

To analyze the cost of the edges rented in Step 3 of Rent-Or-Buy-Sample, one can use the elegant *strict cost-shares* framework of Gupta *et al.* [99] for sampling algorithms for rent-or-buy and related problems. If one uses the algorithm Reverse-Greedy to find the subgraph H in Step 2.1, we obtain an $O(k \log h)$-approximation, as Reverse-Greedy is an $O(k \log h)$-approximation to SS-k-Connectivity that is also $O(k \log h)$-strict. (See Definition 5.19 and Lemma 5.22 below for details.) That is, we obtain the following lemma:

Lemma 5.17. *Let* OPT *be the cost of an optimal solution to the given instance of* SS-k-Rent-or-Buy*. If we use algorithm* Reverse-Greedy *to find the graph H in Step 2.1 of* Rent-Or-Buy-Sample*, the expected cost of edges rented in Step 3 is at most $O(k \log |T|) \cdot \text{OPT}$.*

Before proving this lemma, we note that one can bound the approximation ratio of Rent-or-Buy-Sample as an immediate corollary of Lemmas 5.16 and 5.17.

Theorem 5.18. *Algorithm* Rent-Or-Buy-Sample *is a randomized $O(k \log |T|)$-approximation for* SS-k-Rent-or-Buy.

It remains only to prove Lemma 5.17. The following definition and lemmas were given by Gupta *et al.* [99] for analysis of Rent-or-Buy and related problems when $k = 1$, but they extend completely to our setting.

Definition 5.19 ([99])**.** *Let $\mathcal{A}$ be a deterministic algorithm for* SS-k-Connectivity. *Let $\mathcal{I}$ be any instance of* SS-k-Connectivity *on a graph $G(V, E)$, with edge cost function $c\colon E \to \mathbb{R}^+$, and with terminal set $T \subseteq V$. Let $\mathrm{OPT}(\mathcal{I})$ denote the cost of an optimal solution to $\mathcal{I}$, and for any terminal t, let $\mathcal{I}_{-t}$ denote the instance of* SS-k-Connectivity *on G with the same cost function c, but with terminal set $T - \{t\}$. Let $H(\mathcal{I}_{-t})$ denote the solution returned by $\mathcal{A}$ on the instance $\mathcal{I}_{-t}$, and let $Aug(\mathcal{I}, t)$ denote the minimum cost of a set of k vertex-disjoint paths from t to k distinct terminals or the root when the cost of edges in $H(\mathcal{I}_{-t})$ has been reduced to 0.*

Let χ be a function that, for any such instance $\mathcal{I}$ of SS-k-Connectivity, *assigns a non-negative real value $\chi(\mathcal{I}, t)$ to each terminal $t \in T$. The function χ is a* β-strict cost-sharing *method for $\mathcal{A}$ if, for each instance $\mathcal{I}$:*

- $\sum_{t \in T} \chi(\mathcal{I}, t) \leq \mathrm{OPT}$.
- *For all $t \in T$, $Aug(t) \leq \beta \cdot \chi(\mathcal{I}, t)$.*

Definition 5.20 ([99])**.** *An algorithm $\mathcal{A}$ for* SS-k-Connectivity *is β-strict if there exists a β-strict cost sharing function χ for $\mathcal{A}$.*

Lemma 5.21. *The algorithm* Reverse-Greedy *is $O(k \log |T|)$-strict for* SS-k-Connectivity.

Proof. Let $t_1, t_2, \ldots t_{|T|}$ be the order in which the terminals are connected to the root by Reverse-Greedy, and let $T_i = \{t_1, \ldots t_{i-1}\}$. Recall from Section 5.2 that the augmentation cost of a terminal t with respect to terminal set $T' \subseteq T - t$ is the minimum-cost set of k vertex-disjoint paths from t to k distinct terminals in T' or the root. Let $Aug_{RG}(t_i)$ be the augmentation cost of t_i with respect to T_i. It is easy to see that the function $\chi(\mathcal{I}, t_i) = Aug_{RG}(t_i)/(8k \log |T|)$ satisfies the second condition of Definition 5.19 with $\beta = 8k \log |T|$. From Theorem 5.3, it follows that the first condition is also satisfied. □

Lemma 5.22 ([99]). *If, in Step 2.1 of algorithm* Rent-Or-Buy-Sample *for* SS-k-Rent-or-Buy*, one uses a β-strict α-approximation algorithm for* SS-k-Connectivity *to find the graph H, the cost of edges rented in Step 3 is at most $O(\alpha+\beta)\mathrm{OPT}$.*

Lemma 5.17 is now an immediate consequence of the two lemmas above. This completes our analysis of Rent-Or-Buy-Sample.

In the rest of this section, we give another direct and somewhat complex analysis of Rent-or-Buy-Sample that proves a slightly weaker bound than the above for reasons that we discuss now. One of our motivations to understand SS-k-Rent-or-Buy is for its use in obtaining algorithms for the SS-k-Buy-at-Bulk problem. For $k = 1$, previous algorithms for SS-k-Buy-at-Bulk [95, 99] could use an algorithm for SS-k-Rent-or-Buy essentially as a black box. However, for $k \geq 2$ there are important technical differences and challenges that we outline in Section 5.4. We cannot, therefore, use an algorithm for SS-k-Rent-or-Buy as a black box. In a nutshell, the extra property that we need is the following. In the sampling algorithm Rent-Or-Buy-Sample, there is no bound on the number of unsampled terminals that may route flow to any specific sampled terminal. In the Buy-at-Bulk application we need an extra *balance* condition which ensures that unsampled terminals route to sampled terminals in such a way that no sampled terminal receives more than βM demand where $\beta \geq 1$ is not too large. We prove the following technical lemma that shows that β can be chosen to be $O(f(k)k\log^2 h)$.

Lemma 5.23. *Consider an instance of* Rent-or-Buy *and let* OPT *be the value of an optimal fractional solution to the given instance. Then for each terminal t we can find paths $P_1^t, P_2^t, \ldots, P_o^t$ with the following properties: (i) $o \geq (k-1/2)M$, (ii) the paths originate at t and end at distinct terminals or the root, and (iii) no edge e is contained in more than M paths for any terminal t. Moreover the total rental cost of the paths is $O(f(k)e^{O(k^2)} \cdot k^5 \log h) \cdot M \cdot \mathrm{OPT}$ and no terminal is the end point of more than $O(f(k)k\log^2 h \cdot M)$ paths.*

The proof of the above lemma is non-trivial. We are able to prove it by first analyzing the sampling based algorithm directly via the natural LP relaxation for SS-k-Rent-or-Buy. Although the underlying ideas are inspired by the ones for SS-k-Connectivity, the proof itself is long and technical; the reader may skip the rest of this section (that is, Sections 5.3.1 to 5.3.3) on first

reading; the statement of Lemma 5.23 above is all that is needed for our SS-k-Buy-at-Bulk algorithms and analysis.

For technical reasons we analyze a slight variant of the algorithm where in the first step we sample each terminal with probability $4k \log k/M$ instead of $1/M$. A trivial modification to the proof of Lemma 5.16 bounds the expected cost of edges bought in Step 2.2 by $O(k^2 \log k \log |T|)\mathrm{OPT}$. It remains to bound the cost of step 3, in which we rent edges from every non-sampled terminal using k disjoint paths to sampled terminals. In the next subsection, we prove the following lemma which is at the heart of the analysis.

Lemma 5.24. *The expected cost of the augmentation in step 3 of the algorithm is* $O(f(k)k^6 \log h) \cdot \mathrm{OPT}$*, where* OPT *is the cost of an optimal fractional solution .*

Lemmas 5.16 and 5.24 together imply the theorem below.

Theorem 5.25. *There is a* $O(f(k)k^6 \log h)$*-approximation for the* SS-k-Rent-or-Buy *problem.*

In the rest of the section we assume, for technical reasons, that M is an integer, and that $h/M = \Omega(\log h)$; otherwise, one can greedily rent paths for each terminal t separately to give an $O(\log h)$ approximation.

5.3.1 The Augmentation Cost

As we did for k-Connectivity, we bound the expected augmentation cost in terms of the value of a linear programming relaxation **ROB-Primal** for the SS-k-Rent-or-Buy problem. (Note that the augmentation cost now depends on the set of sampled terminals.) We use similar notation here, with $\mathcal{P}_t^k$ denoting the collection of all sets of k disjoint paths from a terminal t to the root r, and $\vec{P}$ denoting $\{p_1, p_2, \ldots p_k\}$, a set of k disjoint paths in $\mathcal{P}_t^k$. In the LP, the variable x_e indicates whether or not the edge e is *bought* in the solution, and the variable y_e^t indicates whether edge e is *rented* by terminal t. For each $\vec{P} \in \mathcal{P}_t^k$, the variable $f_{\vec{P}}$ denotes how much "flow" is sent by t along the paths in $\vec{P}$. We also give the dual of **ROB-Primal** that we call **ROB-Dual**.

ROB-Primal

$$\begin{aligned}
\min \sum_{e\in E} M\cdot c_e x_e \;+\; \sum_{e\in E}\sum_{t\in T} c_e y_e^t & & \\
\sum_{\vec{P}\in\mathcal{P}_t^k} f_{\vec{P}} &\geq 1 & (\forall t\in T)\\
\sum_{\vec{P}\in\mathcal{P}_t^k | e\in\vec{P}} f_{\vec{P}} &\leq x_e + y_e^t & (\forall t\in T, e\in E)\\
x_e, f_{\vec{P}}, y_e^t &\geq 0 &
\end{aligned}$$

ROB-Dual

$$\begin{aligned}
\max \sum_{t\in T}\alpha_t & & \\
\beta_e^t &\leq c_e & (\forall t\in T, e\in E)\\
\sum_{t\in T}\beta_e^t &\leq M\cdot c_e & (\forall e\in E)\\
\alpha_t &\leq \sum_{e\in\vec{P}}\beta_e^t & \left(\forall \vec{P}\in\mathcal{P}_t^k\right)\\
\alpha_t, \beta_e^t &\geq 0 &
\end{aligned}$$

As before, we construct a feasible solution to the dual program, with α_t for terminal t closely related to augmentation costs. Again, we pay $\beta_e^t = c_e$ on edges within a "ball" of radius α_t, using edge costs modified by z_e^t variables. Note that, in contrast to the dual for k-Connectivity, here we are allowed to have M terminals paying for an edge. We again use a dual-packing argument. However, the proof for the SS-k-Rent-or-Buy problem requires more intricate analysis in order to deal with the added complexity that arises from bought and rented edges. The analysis is also somewhat different because the algorithm has one sampling stage followed by *simultaneous* augmentation for all non-sampled terminals. This means that for each terminal t, α_t will have to be roughly the augmentation cost of t, unlike the SS-k-Connectivity case, where we set α_t to be the minimum augmentation cost of any terminal. Further, the augmentation cost for a non-sampled terminal depends on the set of sampled terminals, so we work with *expected* augmentation costs. For each terminal t, we construct the directed graph $G_t(V_t, E_t)$ precisely as before, making two copies of each vertex other than t, and a new sink r_t. However, we now ask for a set of kM paths in G_t from t to r_t through kM distinct terminals, with no more than M of these paths sharing an edge. The intuition here is that once we have paths to kM terminals, sampling terminals with probability proportional to $1/M$ will give us about k sampled terminals, and we will find disjoint paths to them with reasonable probability — this requires that no more than M of the kM paths use any edge. More formally, we use the natural linear program **ROB-Primal-Aug**(t), shown below.

$$\min \sum_{e \in E_t} c_e f_e$$

$$f_e \leq 1 \qquad (\forall e = (y, r_t) \in E_t)$$

$$f_e \leq M \qquad (\forall e \neq (y, r_t) \in E_t)$$

$$\sum_{e=\delta^-(v)} f_e = \sum_{e=\delta^+(v)} f_e \qquad (\forall v \neq t, r_t)$$

$$\sum_{e=\delta^-(r_t)} f_e \geq kM$$

$$f_e \geq 0 \qquad (\forall e \in E_t)$$

The linear program **ROB-Primal-Aug**(t) has an integral optimal solution since all capacities are integer valued (recall that M is an integer). Let $\mathrm{OPT}_{Aug}(t)$ denote the cost of an optimum solution.

Lemma 5.26. *With probability at least* $1-1/k$*, the augmentation cost of* t *is at most* $2k\mathrm{OPT}_{Aug}(t)/M$*.*

With Lemma 5.26 in place, we proceed to show a setting of feasible dual variables in **ROB-Dual** to prove the following lemma.

Lemma 5.27. *There exists a feasible setting of variables to* ***ROB-Dual*** *such that* $\sum_t \alpha_t \geq \Omega(\frac{1}{f(k)k^5 \log h}) \sum_t \mathrm{OPT}_{Aug}(t)/M$*.*

Lemmas 5.26 and 5.27 do not quite suffice to prove Lemma 5.24. This is because Lemma 5.26 does not guarantee a bound on the expected augmentation cost of t. To prove Lemma 5.24 we need somewhat more general versions of the above lemmas. We state and prove these in the next subsection.

5.3.2 The Proof of Lemma 5.24

We prove Lemma 5.24, using more general versions of Lemmas 5.26 and 5.27.

First, we wish to bound the probability that the augmentation cost of t is more than $\frac{2k\mathrm{OPT}_{Aug}(t)}{M}$. Given an optimal solution to **ROB-Primal-Aug**(t) on the graph $G_t(V_t, E_t)$, let E_t^1 be the set of

edges with $f_e > 0$, the support of this solution. Consider the directed graph $G_t^1(V_t, E_t^1)$, with flow given by the f_e values; there is a flow of kM from t to r_t. While there exists a t-to-r_t path with non-zero flow in G_t^1 of length at least $2\text{OPT}_{Aug}(t)/M$, decrement flow along this path by 1 unit, and delete any edges with 0 flow. This decreases the cost of the flow by at least $2\text{OPT}_{Aug}(t)/M$, so at most $M/2$ such paths can be found. Let $G_t^2(V_t, E_t^2)$ be the graph with edges with remaining non-zero flow. There is a flow of at least $(k - 1/2)M$ from t to r_t in $G_t^2(V_t, E_t^2)$, and there is no directed t to r_t path of length at least $2\text{OPT}_{Aug}(t)/M$. We will argue that in the graph G_t^2, the probability that we *cannot* find k disjoint paths to sampled terminals is low; since all paths in G_t^2 to terminals have length at most $2\text{OPT}_{Aug}(t)/M$, the total length of these paths will be no more than $2k\text{OPT}_{Aug}(t)/M$.

Lemma 5.28. *Let T' be the set of at least $(k - 1/2)M$ terminals through which t sends flow to r_t in G_t^2. Let $U = \{u_1, u_t, \ldots, u_i\} \subset T'$ be any set of $i < k$ terminals from T' such that there are i disjoint paths from t to U, one to each u_j. A terminal $x \in T' - U$ is said to be* bad *for U if G_t^2 does not contain $i + 1$ disjoint paths from t to $U \cup \{x\}$, one to each of these terminals. For any such set U, the number of terminals that are bad for U is at most iM.*

Proof. If we cannot find such a set of $i + 1$ paths to x and U, there must be a cut-set of i edges whose deletion separates t from x and U. Let C be a cut-set of i edges *nearest* t that separates t from U: More precisely, let H_t be the graph containing edges of E_t^2, but with all capacities reduced to 1. In the residual graph after finding i paths from t to U in H_t, C is the cut-set induced by the set of vertices reachable from t, and those unreachable from t. If x is reachable from t in $H_t - C$, we have $i + 1$ disjoint paths from t to x and U. If it is not reachable, any path from t to x in H_t must go through one of the i edges in C. But each edge of C carries at most M units of flow in G_t^2, and therefore there are at most iM such terminals x that are bad for U. Equivalently, at least $(k - 1/2 - i)M \geq M/2$ terminals are good for U. □

Now, we can prove the following more general version of Lemma 5.26:

Lemma 5.29. *Suppose each terminal is sampled with probability $\rho \cdot 4k \log k/M$ for an integer $\rho \geq 1$. The probability that the augmentation cost of t is more than $2k\text{OPT}_{Aug}(t)/M$ is at most $1/k^{2\rho-1}$.*

Proof. We give an algorithm to construct a set of k disjoint paths using edges of G_t^2, and bound the failure probability of this algorithm. For the purpose of this analysis, suppose that terminals were sampled in k *phases*; in each phase, a terminal is selected independently with probability $4\rho \log k/M$, and a terminal is considered available if it is sampled in any phase. It is easy to see that the probability a terminal is sampled under this model is less than the probability it is sampled if we perform "single-stage" sampling with probability $4\rho k \log k/M$.

Our algorithm attempts, in each phase, to increase the number of disjoint paths it can find to sampled terminals; if it cannot do this, we say that it has failed. At the end of phase i, assuming the algorithm has not failed so far, it has i disjoint paths to i sampled terminals. We bound the probability that it cannot find another disjoint path to a sampled terminal during phase $i+1$ by $1/(k^{2\rho})$:

Let U be the set of i terminals found during the first i phases, such that we have i disjoint paths from t to each of the terminals of U. From Lemma 5.28, at least $(k - 1/2 - i)M > M/2$ terminals are not bad for U; if *any* of these terminals is sampled during phase $i+1$, the algorithm will succeed in this phase. Since terminals are sampled with probability $4\rho \log k/M$ in each phase, the probability that the algorithm will fail in this phase is at most $\left(1 - \frac{4\rho \log k}{M}\right)^{M/2}$, which is at most $1/k^{2\rho}$.

Therefore, the probability that the algorithm fails overall, using the union bound, is at most $1/k^{2\rho-1}$. □

To obtain the feasible setting for **ROB-Dual**, we use the dual to the augmentation LP **ROB-Primal-Aug**(t), as we did for k-Connectivity:

$$\begin{aligned}
\max \quad kM \cdot \Pi - \left(M \sum_{e \neq (u,r_t)} z_e^t\right) - \left(\sum_{e=(u,r_t)} z_e^t\right) & \\
\Pi - \pi_t(u) &\leq c_e + z_e^t \quad (\forall e = (u, r_t) \in E_t) \\
\pi_t(v) - \pi_t(u) &\leq c_e + z_e^t \quad (\forall e = (u, v) \in E_t, u \neq t, v \neq r_t) \\
\pi_t(v) &\leq c_e + z_e^t \quad (\forall e = (t, v) \in E_t) \\
z_e^t &\geq 0 \quad (\forall e \in E_t)
\end{aligned}$$

As we did for k-CONNECTIVITY, we describe an optimal solution to **ROB-Dual-Aug**(t) based on the residual graph $G_t(kM-1)$ after a minimum-cost $kM-1$ units of flow have been sent from t to the sink r_t. The potential $\pi_t(v)$ for a vertex v is the distance from t to v in $G_t(kM-1)$, and z_e^t values are set to satisfy all the constraints. Again, the complementary slackness conditions allow one to verify that this is an optimal solution. As before, we think of $c_e + z_e^t$ as a "modified" length ℓ_e^t for edge e, with $d_t(v)$ being the shortest-path distance from t to v in G_t under this modified length; as for k-CONNECTIVITY, note that the **ROB-Dual-Aug**(t) constraints imply that $d_t(v) \geq \pi_t(v)$.

Note that $\Pi(t) = \Pi$ is the potential of r_t in the residual graph. In our construction of a feasible solution to **ROB-Dual**, we use the following technical claims:

Proposition 5.30. *For each terminal t, $\Pi(t) \geq \mathrm{OPT}_{Aug}(t)/kM$. Under the distance function ℓ_e^t, any path from t to r_t in G_t has length at least $\Pi(t)$. Finally, $\sum_{e \neq (u,r_t)} z_e^t \leq k\Pi(t)$.*

Proof. Similar to the proofs of Proposition 5.8, and Claims 5.9 and 5.10. □

We now use the dual variables from **ROB-Dual-Aug**(t) to set the corresponding variables for **ROB-Dual**. Let $\Pi_{max} = \max_t \Pi(t)$. If $\Pi(t) < \Pi_{max}/h^2$, set $\alpha_t = 0$, and $\beta_e^t = 0$ for all e. Otherwise, set $\alpha_t = 2^{\lfloor \log(\Pi(t)/2k) \rfloor}$, and $\beta_e^t = c_e$ within the ℓ_e^t ball of radius α_t. Note that there are now only $2 \log h$ distinct values of α_t, and $\frac{\Pi(t)}{4k} < \alpha_t \leq \frac{\Pi(t)}{2k}$. We show that these settings are "nearly" feasible for **ROB-Dual**.

Lemma 5.31. *For each edge e, $\sum_t \beta_e^t \leq O(k \log h \cdot f(k+2) \cdot Mc_e)$*

Proof. Recall that there are only $2 \log h$ distinct non-zero values of α_t. Let T_i denote the set of terminals with the ith value of α_t; we show that for all i, $\sum_{t \in T_i} \beta_e^t \leq 2kf(k+2)Mc_e$, proving the lemma.

Suppose, by way of contradiction, there were some edge $e = (u, v)$ and some set of terminals T_i such that this were not true. Draw a directed graph on T_i, with edges from each terminal t to the (at most) kM other terminals in T_i that it routes flow to in G_t^2. (Note that some of the kM terminals that t routes flow to may not be in T_i.) Since $|T_i| > 2kf(k+2)M$, and each vertex in this graph has out-degree at most kM, the graph has an independent set I of size $f(k+2)$.

The paths from each terminal in I to u form an intersecting path system; from Lemma 5.14, there exists a terminal t and $u_1, u_2, \ldots u_{k+2}$ in I, such that $\mathit{prefix}_t(u_i)$ and $\mathit{prefix}_t(u_j)$ are disjoint

for all $i \neq j$. Since I was an independent set, t does not route flow through any u_i in G_t, and hence $z_e^t = 0$ on the edges in G_t from each u_i to r_t. Exactly as we argued for Lemma 5.11, $\sum_{e \in prefix_t(u_i)} z_e^t$ must be at least $\Pi(t)(1 - 1/k)$, and these $k+2$ prefixes are all disjoint. Therefore, $\sum_{e \in E_t, e \neq (u,r_t)} z_e^t > k\Pi(t)$. But this contradicts Proposition 5.30, completing the proof. □

Lemma 5.32. *For all $\vec{P} \in \mathcal{P}_t^k$, $\alpha_t \leq \sum_{e \in \vec{P}} \beta_e^t$.*

Proof. If $\alpha_t = 0$, the lemma is clearly true. Suppose the lemma is not true; consider $\vec{P} \in \mathcal{P}_t^k$, a set of k paths that witness this. Since all the paths end in the root (also a terminal), they must each leave the ball of ℓ_e^t radius α centered at t. Let E' be the set of edges of $\vec{P}$ in this ball[8]; by definition, $\sum_{e \in E'} \ell_e^t = k\alpha$. We show that $\sum_{e \in E'} z_e^t \leq (k-1)\alpha$. As $\ell_e^t = c_e + z_e^t$, and for each edge in the ball of radius α, $\beta_e^t = c_e$ it follows directly that $\sum_{e \in E'} \beta_e^t \geq \alpha$, giving a contradiction.

For each edge $e = (u, v) \in E'$, with $z_e^t > 0$, charge it to the interval $[\pi_t(u), \pi_t(v))$, the length of which is at least z_e^t. For every vertex v in the ball of radius α, $\pi_t(v) \leq d_t(v) \leq \alpha$. By Lemma 5.33 immediately below, no point in the interval $[0, \alpha]$ gets charged more than $k-1$ times, which implies that $\sum_{e \in E'} z_e^t \leq (k-1)\alpha$. □

Lemma 5.33. *For any potential γ, there are at most $k-1$ edges $e = (u, v)$ such that $z_e^t > 0$ and $\pi(u) \leq \gamma < \pi(v)$:*

Proof. First, note that for any edge $e = (u, v)$ with $z_e^t > 0$, it must be the case that $\pi(u) < \pi(v)$, and if $z_e^t > 0$ on edge e, e does not exist in $G_t(kM - 1)$, so it must be carrying M units of the minimum-cost $kM - 1$ total flow from t to r_t. Consider the set of vertices U within distance γ of t in $G_t(kM - 1)$; for each edge $e = (u, v)$ whose endpoints bracket potential γ, it must be the case that $u \in U, v \notin U$. Note that there is no edge $e' = (v', u')$ with $v' \notin U, u' \in U$ that is carrying any of the $kM - 1$ units of flow; if there were such an edge carrying flow, the edge (u', v') would have negative cost in $G_t(kM - 1)$, and hence $\pi(v') < \gamma$, contradicting the fact that $v' \notin U$. Therefore, the total flow on edges from U to $V_t - U$ is at most $kM - 1$. But any edge $e = (u, v)$ with $z_e^t > 0$ such that $\pi(u) \leq \gamma < \pi(v)$ must be carrying M units of this flow, and so there can be at most $k - 1$ such edges. □

[8]For ease of notation, we ignore edges partially contained in the ball of radius α; this does not affect the proof in any detail.

Proof of Lemma 5.27. *There is a feasible setting of variables to* ***ROB-Dual*** *such that:*

$$\sum_t \alpha_t \geq \Omega(\frac{1}{f(k)k^5 \log h}) \sum_t \mathrm{OPT}_{Aug}(t)/M$$

For each terminal t, $\Pi(t) \geq \mathrm{OPT}_{Aug}(t)/kM$, and so $\sum_t \mathrm{OPT}_{Aug}(t)/kM \leq \sum_t \Pi(t)$. For each terminal t with $\alpha_t > 0$, $\Pi(t) \leq 4k\alpha_t$, and for each terminal with $\alpha_t = 0$, $\Pi(t) \leq 1/h^2\Pi_{max}$. Therefore, $\sum_t \mathrm{OPT}_{Aug}(t)/kM \leq 4k(1 + 1/h) \sum_t \alpha_t$.

However, using the values of α_t and β_e^t for **ROB-Dual** as described above, we do not have a feasible solution; instead, from Lemma 5.31, one constraint is violated by a factor of at most $O(k \log h \cdot f(k+2)) = O(k^3 f(k) \log h)$. Therefore, we shrink all values of α_t and β_e^t by this factor, and obtain a feasible solution such that $\sum_t \alpha_t \geq \Omega(\frac{1}{f(k)k^5 \log h}) \sum_t \mathrm{OPT}_{Aug}(t)/M$. □

Lemma 5.29 strengthened Lemma 5.26, but this still does not suffice to prove Lemma 5.24, which requires a bound on the *expected* augmentation cost. We now present a proof of this bound.

Given an instance $\mathcal{I}$ of Rent-or-Buy, let $\mathcal{I}_i$ denote the instance in which the parameter M is replaced by $i \cdot M$. (With this notation, $\mathcal{I}_1$ denotes the original instance.) We use OPT_i to denote the cost of an optimal solution to $\mathcal{I}_i$; it is easy to see that $\mathrm{OPT}_i \leq i \cdot \mathrm{OPT}$, where OPT — which is the same as OPT_1 — denotes the cost of an optimal solution to the original instance. (Given an optimal solution to $\mathcal{I}_1$, use the same solution on $\mathcal{I}_i$ (that is the same set of paths for the terminals), paying i times as much to "buy" any edge bought in the original solution.)

For each instance $\mathcal{I}_i$ and terminal t, we can construct a corresponding Augmentation Linear Program **ROB-Primal-Aug**$(t)^i$, finding paths to $k \cdot iM$ terminals, with no more than iM paths sharing a vertex; let $\mathrm{OPT}^i_{Aug}(t)$ denote the cost of an optimal solution to **ROB-Primal-Aug**$(t)^i$. The following claim is a consequence of Lemma 5.27:

Claim 5.34. $\sum_t \mathrm{OPT}^i_{Aug}(t)/iM \leq O(f(k)k^5 \log h) \cdot \mathrm{OPT}_i$.

Let ρ_i denote the probability that the augmentation cost of t is more than $2k\mathrm{OPT}^i_{Aug}(t)/iM$ when terminals are sampled with probability $4k \log k/M$.

Lemma 5.35. *The probability ρ_i is at most $1/k^{2i-1}$.*

Proof. From Lemma 5.29, it follows directly that if terminals are sampled with probability $i \cdot \frac{4k \log k}{iM}$, the probability that the augmentation cost of t is more than $\frac{2k\text{OPT}^i_{Aug}(t)}{iM}$ is at most $1/k^{2i-1}$. □

Intuitively, this lemma says that the probability the augmentation cost of a terminal is much larger than $\text{OPT}^i_{Aug}(t)/iM$ is decreasing geometrically with i; while Claim 5.34 shows that (on average) $\text{OPT}^i_{Aug}(t)/iM$ is only increasing linearly with i. This allows us to easily bound the expected augmentation costs of terminals.

An upper bound on the expected augmentation cost of t, denoted by $\mathbb{E}[AugCost(t)]$, is given by the following sum:

$$2k\text{OPT}^1_{Aug}(t)/M + \sum_{i>1} \rho_{i-1} \cdot 2k\text{OPT}^i_{Aug}/iM$$

We can now prove Lemma 5.24: *The expected cost of the augmentation in Step 3 of the algorithm* Rent-or-Buy-Sample *is* $O(f(k)k^6 \log h) \cdot \text{OPT}$.

Proof of Lemma 5.24.

$$\begin{aligned}
\sum_t \mathbb{E}[AugCost(t)] &\leq 2k \left(\sum_t \frac{\text{OPT}^1_{Aug}(t)}{M} + \sum_{i>1} \left(\rho_{i-1} \sum_t \frac{\text{OPT}^i_{Aug}(t)}{iM} \right) \right) \\
&\leq 2k \left(\sum_t O(f(k)k^5 \log h) \cdot \text{OPT}_1 + \sum_{i>1} \rho_{i-1} O(f(k)k^5 \log h) \cdot \text{OPT}_i \right) \\
&= O(f(k)k^6 \log h) \left(\text{OPT} + \sum_{i>1} \frac{1}{k^{2i-3}} i \cdot \text{OPT} \right) \\
&= O(f(k)k^6 \log h) \cdot \text{OPT}. \qquad (\text{since } k \geq 2)
\end{aligned}$$

□

5.3.3 The Proof of Lemma 5.23

Given an instance of Rent-or-Buy, an optimal solution to the linear program **ROB-Primal-Aug**(t) gives, for each terminal t, a set of kM paths $P^t_1, P^2_2, \ldots, P^t_{kM}$ from t to kM distinct terminals[9], such that no more than M paths use any edge, and the total length of these paths is $\text{OPT}_{Aug}(t)$. For terminal t, let $Sinks(t)$ denote the set of endpoints of these kM paths; we say that t routes flow to each terminal in $Sinks(t)$. From Lemma 5.27, $\sum_t \text{OPT}_{Aug}(t) \leq$

[9]The root may appear as the end vertex of more than 1 path.

$f(k)k^5 \log h \cdot M \cdot \mathrm{OPT}$, where OPT is the cost of an optimal solution to the Rent-or-Buy instance.

These paths satisfy Lemma 5.23, except for the balance condition, that no terminal v be the endpoint of more than $O(f(k)k \log^2 h \cdot M)$ paths. We say that a terminal is *overloaded* if it is the endpoint of more than this number of paths. That is, v is overloaded if the number of terminals t such that $v \in Sink(t)$ is more than $O(f(k)k \log^2 h \cdot M)$. In this section, we argue that one can *reroute* some of the paths to ensure this balance, without increasing their length significantly. We say that a terminal t can *safely* reroute flow from v to u if $v \in Sink(t), u \notin Sink(t)$, and after deleting the $t - v$ path, we can add a path from t to u such that no more than M paths of t use any edge.

Lemma 5.36. *Let $X \cup \{v\}$ be a set of terminals, such that for each $x \in X$, $v \in Sink(x)$. The paths from each terminal $x \in X$ to v form an intersecting path system (X, P, v); let $p(x)$ denote the path from x to v. If $|X| \geq 2kf(k)M$, there exist terminals $t, u_1, u_2, \ldots u_k$ in X such that for each $1 \leq i \leq k$, $u_k \notin Sink(t)$, and for each $i \neq j$, $prefix_t(u_i)$ and $prefix_t(u_j)$ are disjoint, except perhaps at the point of intersection with $p(t)$. Then, t can safely reroute flow from v to some u_i.*

Proof. All the claims in the lemma except the last follow directly from the proof of Lemma 5.32. It remains to show that t can safely reroute flow from v to some u_i. We refer to $\{prefix_t(u_i) : 1 \leq i \leq k\}$ as the set of candidate paths for t, and use y_i to denote the vertex where $p(t)$ intersects $prefix_t(u_i)$.

Let $G_t(V_t, E_t)$ denote the directed graph used in **ROB-Primal-Aug**(t), with E_t^1 being the set of edges with non-zero flow. In the directed graph $G_t^1(V_T, E_t^1)$, there is a flow of kM from t to r_t, including a flow of 1 unit from t to r_t through the path from t to v. The current capacity of each edge in E_t^1 is M; consider a modified max-flow instance $G_t^2(V_t, E_t^2)$ in which:

1. The capacity of every edge e not on one of the candidate paths is reduced to f_e, the current flow through that edge.

2. The edge from every terminal in $T \setminus Sinks(t)$ to r_t is deleted, the edge from v to r_t is deleted, and 0-cost directed edges of capacity 1 are added from each u_i to r_t.

We argue that there are kM paths in G_t^2 from t to r_t via distinct terminals. The terminal v will no longer be in $Sinks(t)$; in its place, some u_i will be added. No more than M paths will share

any edge, and the total cost of these paths will increase only by the total length of the candidate paths.

Note that of the kM paths originally used by t, all but the path to r_t through v still exist in G_t^2. Consider the residual graph G_t^r after sending flow through the remaining $kM - 1$ paths (excluding the path through v). The path $p(t)$ from t to v still exists in the residual graph. Therefore, each y_i is still reachable from t in G_t^r. Further, each u_i is not originally in $Sinks(t)$, the edge from u_i to r_t still exists in G_t^r. Finally, we show that $\exists 1 \leq i, j \leq k$ such that u_i is reachable from y_j; this implies that there is a path from t to r_t in the graph G_t^r, and hence we find a set of kM paths as desired, showing that t can safely reroute flow from t to u_i.

Suppose this were not true; let S be the set of vertices reachable from t in G_t^r. As argued above, the set S must include each y_j, but not any u_i. Consider the cut induced by $[S, V_t - S]$, as there are k disjoint paths from each y_i to u_i, each of capacity M, the capacity of this cut is at least kM. But the total flow out of S is at most $kM - 1$, and hence some vertex in $V_t - S$ is reachable from t, giving a contradiction. □

Corollary 5.37. *If a terminal t routes flow to kM other terminals, using paths of total length L, and we perform a reroute operation for t such that each candidate path for t has length at most ℓ, the total length of the kM paths obtained after this step is at most $L + kl$.*

In order to obtain a set of paths that satisfies Lemma 5.23, we repeatedly find a large set of terminals X that share a common sink v, and find one terminal $t \in X$ that can safely reroute to another terminal $u \in X$. We refer to this operation as a *reroute for t*, or as a *reroute at v*.

We now give an algorithm to reroute flow from terminals, at the end of which, no terminal receives more than $O(f(k)k \log^2 h \cdot M)$ flow. This is the missing balance condition of Lemma 5.23. Note that re-routing may increase the length of the kM paths found by each terminal (though from Corollary 5.37, the increase will not be very much); for ease of exposition, we ignore this increase in lengths for the moment. After proving that we can achieve the desired balance, we present a very slight modification to the algorithm that accounts for the increase in lengths. The following lemma describes this recursive algorithm.

Lemma 5.38. *There is an algorithm to safely reroute flow such that no terminal is an endpoint of more than $O(f(k)k \log h \cdot M)$ paths.*

Proof. Construct a directed graph H on the terminals by adding an edge from each terminal t to the kM terminals in $Sinks(t)$. The out-degree of each terminal is kM, and hence the *average* in-degree of a terminal is kM; observe that an overloaded terminal has in-degree at least $O(f(k)k \log h \cdot M)$, which is significantly above the average. Let T_1 be the set of terminals with in-degree greater than k^2M and $T_2 = T \setminus T_1$; note that $|T_1| < \frac{|T|}{k} = h/k$. All overloaded terminals are in T_1, and further, each terminal in T_2 is far from being overloaded. Now, assuming that the *only* terminals are in T_1, recursively reroute flow safely so that no terminal in T_1 is overloaded. That is, we reroute flow for terminals in T_1, such that no terminal in T_1 is a sink for more than $O(f(k)k \log(h/k)M)$ other terminals in T_1.

After re-routing flow at terminals in T_1, each of those terminals is no longer a sink for many other terminals in T_1; however, such terminals may still be overloaded due to terminals in T_2. Note that terminals in T_2 are currently far from overloaded. Finally, in our induction step, we show how terminals in T_2 can reroute most of their flow from terminals in T_1 to other terminals in T_2. At the end of this step, each terminal in T_1 will receive at most $O(f(k)k \log k \cdot M)$ flow from T_2; together with at most $O(f(k)k \log(h/k) \cdot M)$ flow it received from T_1, it receives a total flow of at most $O(f(k)k \log h \cdot M)$.

We process each overloaded terminal in T_1 in turn. Consider v, such a terminal; we process v in *phases* until it is no longer overloaded. At the beginning of a phase, since v is overloaded, it must be receiving more than $O(f(k)k \log k \cdot M)$ flow from terminals in T_2; let S be the set of such terminals routing flow to v. Pick an arbitrary set $S' \subseteq S$ of $2kf(k)M$ such terminals in T_2; from Lemma 5.36, there exist terminals $t, u \in S'$ such that t can safely reroute to u, and set $S = S \setminus \{t, u\}$. Repeat this process until $|S| < 2kf(k)M$. At this point, almost half the terminals originally routing flow to v have been safely rerouted, and no terminal has received more than one unit of flow from another of these terminals. This completes a single phase. By the end of $O(\log h)$ phases, v can no longer be overloaded (as the number of terminals routing flow to v is falling by a constant factor in successive phases), and no terminal in T_2 receives more than one unit of flow in each phase.

Clearly, after each overloaded terminal in T_1 is processed, none of them is receiving too much flow. As for terminals in T_2, each such terminal t receives at most $O(\log h)$ additional flow for each

of the kM terminals in $Sinks(t)$, and since $t \in T \setminus T_1$, it originally received at most k^2M flow. Therefore, the total flow received by t is at most $k^2M + O(\log h)kM = O(f(k)k \log h \cdot M)$. □

Lemma 5.38 gives a set of paths from each terminal t to kM distinct terminals, such that no more than M paths for t share any vertex, and no terminal is the endpoint of more than $O(f(k)k^2 \log h \cdot M)$ paths. This still does not suffice for Lemma 5.23, since even though the original set of kM paths for each terminal t has length $\mathrm{OPT}_{Aug}(t)$, the balancing algorithm of Lemma 5.38 may increase these path lengths significantly. In the next lemma, we give a revised version of the algorithm that does not significantly increase path lengths.

Lemma 5.39. *Given a set of terminals T, and, for each $t \in T$, paths of total length L_t from t to kM distinct terminals in T, there is an algorithm to find a set of paths from t to $(k - 1/2)M$ distinct terminals, of total length no more than $O(e^{O(k^2)}L_t)$, and such that no terminal v is the endpoint of more than $O(f(k)k \log h \log n \cdot M)$ paths.*

Proof. We assume, for this lemma, that the length of each edge in the graph is an integer in the range $[1, n^3]$; it is easy to achieve this by pre-processing the graph, at the cost of an additional factor of $1 + o(1)$ in the approximation ratio.[10]

As before, we let T_1 denote the set of terminals with degree greater than k^2M, and recursively solve the subproblem with terminal set T_1. Note that we never again reroute flow for terminals in T_1, and so the total path length for $v \in T_1$ does not increase in the inductive step.

Now, it remains only to process terminals in T_1 that receive more than $O(f(k)k \log n \log k \cdot M)$ flow from T_2. Again, we will try to repeatedly reroute flow from a terminal $t \in T_2$ to another terminal $u \in T_2$, but now we have to be careful not to increase the total path length for t significantly. Therefore, we distinguish between *short* and *long* paths for t. Each of of the kM paths from t to other terminals is said to be short if its length is at most $4k$ times the average, and long otherwise. More precisely, if L currently denotes the total path length for t, a path from t to some other terminal is short if its length is at most $4L/M$, and long otherwise. Note that at most $M/4$ of the paths for t can be long. As we reroute flow for t, its total path length will change, and so a path that is long at one time may be short at another. The total path length for t can change only when we

[10] As we discuss later, this can be avoided.

reroute for t; let L_t^i denote this total length after we have performed i reroutes for t. $L_t^0 = L_t$ is the original total length for t. Every time we reroute for $t \in T_2$, we decrease the number of paths that end in T_1, and increase the number that end in T_2. Therefore, the total number of reroutes for t is at most kM. We ensure that when we perform the ith reroute, $L_t^i \leq (1 + 8k/M)L_t^{i-1}$. Therefore, when the algorithm concludes, the total path length for t is at most $L_t^0(1 + 8k/M)^{kM} \approx e^{8k^2}L_t$. Thus, we achieve the desired bound on path lengths.

We can now describe the algorithm more precisely: As before, we process terminals in turn, and in phases. Let v be a terminal receiving more than $O(f(k)k \log n \log k \cdot M)$ flow from T_2, and such that more than half of these paths ending at v are short for their respective sources. That is, if we use S to denote the set of terminals $t \in T_2$ such that $v \in Sinks(t)$, and the path from t to v is short for t, the set S contains more than half the terminals in T_2 currently routing flow to v. Partition S into groups $S_1, S_2, \ldots S_{5 \log n}$, such that $t \in S_i$ if the path from t to v has length at least 2^i and less than 2^{i+1}. While some group S_i has more than $2kf(k)M$ terminals, apply the balancing algorithm from Lemma 5.38 to process v for group S_i; this ensures that no terminal $t \in S_i$ receives more than $O(\log h)$ additional flow. Observe that each reroute for a terminal t only changes the path length for t, and after t is rerouted, v is no longer in $Sinks(t)$, so no terminal moves from S_i to S_j in the course of processing v. Further, when t is rerouted, if its path to v has length ℓ, all the candidate paths for t have length at most 2ℓ, and by Corollary 5.37, the total increase in its path lengths is at most $2k\ell$. If this is the ith reroute for t, $\ell \leq 4L_t^{i-1}/M$, and hence $L_t^i \leq (1 + 8k/M)L_t^{i-1}$.

After running this procedure for each S_i, the total flow into v from short paths is at most $2kf(k) \cdot 5 \log nM$. Initially, more than half the flow into v was from short paths, so we have decreased the flow into v by at least a third. Note that after we have completed processing v, it may still be overloaded due to flow from long paths. If it is no longer overloaded, we may implicitly remove it from T_1.

After processing v, we move on to another overloaded vertex $w \in T_1$ such that at least than half the flow it receives from T_2 is along short paths. If there is *no* such vertex w, more than half the paths from T_2 to T_1 must be long. In particular, it follows that there is some vertex $t \in T_2$ such that more than half the paths from t to T_1 are long. Delete all paths from t to T_1; since there are at most $M/4$ long paths, we have deleted at most $M/2$ paths in total, and so t has at least

$(k - 1/2)M$ paths remaining. The terminal t no longer contributes to overloading any terminal in T_1, so we can remove it from consideration. After all such terminals $t \in T_2$ have been deleted, some overloaded vertex in T_1 must receive more than half its flow from T_2, and hence we can process it.

At the end of this algorithm, each vertex in T_1 is no longer overloaded, and the total path length for each terminal t is at most $e^{O(k^2)}L_t$. We only need to ensure that no vertex in T_2 is overloaded; this is straightforward. A terminal in $t \in T_2$ receives at most $O(\log h)$ flow every time a terminal from T_1 in $Sinks(t)$ is processed. Every time $v \in T_1$ is processed, it loses at least a third of the flow it received, and hence v is processed $O(\log h)$ times. Finally, there are at most kM terminals of T_1 in $Sinks(t)$, and hence the total additional flow received by $t \in T_2$ is at most $O(\log^2 h \cdot kM)$. □

As $\sum_t L_t \leq O(k^5 f(k) \cdot \log h)M \cdot \mathrm{OPT}$ from Lemma 5.27, and after rerouting, we increase L_t only by a factor of $e^{O(k^2)}$, Lemma 5.39 shows that we can obtain a set of at least $(k-1/2)M$ paths from t to distinct terminals, no more than M of which share a vertex, and such that each terminal is an endpoint of at most $O(f(k)k \log h \log n \cdot M)$ paths. These almost satisfy the requirements of Lemma 5.23; the factor of $\log n$ should be replaced with $\log h$. We briefly sketch how to achieve this, but omit the details, which are very similar to those presented above: When processing a terminal $v \in T_1$, group terminals into $2 \log h$ groups instead of $O(\log n)$ groups based on the lengths of their paths to t. Terminals with paths of lengths significantly less than (less than $1/h^2$ times as long as) the longest paths can be grouped with each other; as their path lengths are so small, they do not significantly contribute to the total.

5.4 Buy-at-Bulk Network Design

In this section we consider the SS-k-Buy-at-Bulk problem. In Section 5.4.1, we give an algorithm for non-uniform SS-k-Buy-at-Bulk that works for any k. First, though, we consider the special case of uniform SS-k-Buy-at-Bulk when $k = 2$, and obtain a better approximation ratio when the number of cable types is not too large.

We begin by briefly summarizing the algorithm and sketching its analysis, before providing details of the proofs. Each terminal $t \in T$ wishes to route one unit of demand to the root along k vertex disjoint paths. More generally, terminals may have different demands, but we focus on

the unit-demand case for ease of exposition. There are b cable-types; the ith cable has capacity u_i and cost w_i per unit length. Let $\hat{f} : \mathbb{R}^+ \to \mathbb{R}^+$ be a sub-additive function[11] where $\hat{f}(x)$ is the minimum-cost set of cables whose total capacity is at least x. The goal is to find a routing for the terminals so that $\sum_e c_e \cdot \hat{f}(x_e)$ is minimized where x_e is the total flow on edge e. One can assume that the cables exhibit economy of scale; that is, $w_i/u_i > w_{i+1}/u_{i+1}$ for each i. Therefore, there is some parameter g_{i+1}, with $u_i < g_{i+1} < u_{i+1}$, such that if the flow on an edge is at least g_{i+1}, it is more cost-effective to use a single cable of type $i+1$ than g_{i+1}/u_i cables of type i. Consistent with this notation, we set $g_1 = 1$; since all our cables have capacity at least u_1, if an edge has non-zero flow, it must use a cable of type at least 1.

Our overall algorithm follows the same high-level approach as that of the previous single-sink algorithms for the $k = 1$ problem [95, 99]. The basic idea is as follows: Given an instance in which the demand at each terminal is of value at least g_i, it is clear that cable types 1 to $i-1$ can be effectively ignored.[12] The goal is now to aggregate or cluster the demand from the terminals to some cluster centers such that the aggregated demand at the cluster centers is at least g_{i+1}. Suppose we can argue the following two properties of the aggregation process: (i) the cost of sending the demand from the current terminals to the cluster centers is comparable to that of OPT and (ii) there exists a solution on the cluster centers of cost not much more than OPT. Then we have effectively reduced the problem to one with fewer cables, since the demand at the cluster centers is at least g_{i+1}. We can thus recurse on this problem. For $k = 1$ this outline can be effectively used to obtain an $O(1)$ approximation *independent* of the number of cable types.

There are several obstacles to using this approach for $k > 1$: The most significant of these is that it is difficult to argue that there is a solution on the new cluster centers of cost not much more than OPT. In the case of $k = 1$, this is fairly easy, as the new cluster centers can pretend to randomly send the demand back to the original terminals; for higher k, since centers need to send demand along k disjoint paths, this is no longer straightforward. In particular, the approach involves terminals routing their demand via other terminals selected as cluster centers. Suppose $k = 2$ and t routes its demand to the cluster centers t' and t''; then t' and t'' route t's demand via

[11] Any sub-additive $\hat{f}$ can conversely be approximated by a collection of cable-types.

[12] Cables of type $j < i$ might be useful, but they can be removed at the expense of a small increase in the approximation ratio.

two separate disjoint paths each; thus the clustering and rerouting leads to an exponential increase in the amount of demand being routed. Further, when $k = 1$, one can exploit in several ways the fact that there is a near optimal solution which is a tree.

To deal with these issues, we perform a 2-stage aggregation process that is more complex than previous methods: First, given centers with demand g_i, we cluster demand to produce a new set of centers with demand u_i, using a result of [9]. Second, given centers with demand u_i, we use some ideas from Section 5.3 for Rent-or-Buy to produce a new set of centers with demand g_{i+1}. The algorithm of [9] that we use in the first stage applies only for $k = 2$; our ideas for the second stage can be extended to arbitrary k. We describe the two-stage aggregation process to go from a set of centers with demand g_i to a new set of centers with demand g_{i+1} below; we can then recurse.

Lemma 5.40 ([9])**.** *Given an instance of* SS-2-Buy-at-Bulk *with center set T in which all demands are at least g_i, let* OPT_i *denote the cost of an optimal solution to the instance, and let H denote an feasible solution to the* SS-2-Connectivity *instance on the terminal set T where the cost of edge e is $w_i c_e$. Then, there is a polynomial time algorithm to find a set of terminals $T' \subseteq T$ such that*

1. *Every $t \in T$ can route g_i units of flow to 2 centers in T' via disjoint paths in H.*
2. *The total flow on any edge in H is $O(1)u_i$.*
3. *The demand at each $t' \in T'$ is at least u_i and at most $7u_i$.*[13]
4. *There is a solution to the new buy-at-bulk instance on T' of expected cost at most $O(1)\mathrm{OPT}_i$.*

We briefly describe how this lemma is used: Given an instance of SS-2-Buy-at-Bulk with in which all demands are at least g_i, we can effectively assume that an optimal solution only uses cables of type i to b; let OPT_i denote the cost of an optimal solution to this instance. As the new SS-2-Buy-at-Bulk instance on T' has good expected cost, we can apply recursion. Thus, it remains to argue that the cost of routing flow from $T \setminus T'$ to T' is $O(1)\mathrm{OPT}_i$, but this is straightforward: Consider a solution of cost OPT_i to the original instance; the set of edges with

[13]The algorithm as described in [9] enforces a weaker version of condition 3; the demand at each $t' \in T'$ is at least u_i, and at many centers, the demand is at most $7u_i$. The centers of so-called star-like jumbo clusters may have higher demand, but the algorithm can be easily extended so that such high demand centers have their demand split into smaller units.

installed cables k-connects T to the root, and the cost on each edge e is at least $w_i c_e$, as we only use cables of type i or higher. Thus, the cost of an optimal solution H to the SS-2-Connectivity instance on T is at most OPT_i; if we find a 2-approximate solution using the algorithm of [79] (one could also use our algorithm for arbitrary k from Section 5.2, obtaining a slightly weaker approximation ratio), the cost of H is at most $2\mathrm{OPT}_i$. But in rerouting demand from $T \setminus T'$ to T', there is only $O(1)u_i$ flow on any edge e in H, so we only need to buy a constant number of copies of cable i, paying $O(1)w_i c_e$. This completes the first aggregation stage.

We now have an instance of SS-2-Buy-at-Bulk with center set T in which each center has demand $\approx u_i$, and with an optimal solution of cost at most $\mathrm{OPT}'_i = O(1)\mathrm{OPT}_i$. Consider a modified instance in which all demands are set equal to u_i, the cable capacity u_{i+1} is set to infinity and the cable-types $i+2$ to ℓ are eliminated. Clearly, the cost of an optimal solution to this modified instance is no more than OPT'_i; simply replace each cable of higher capacity with a single cable of type $i+1$. However, we now have an instance of Rent-or-Buy with $M = g_{i+1}/u_i$. We can thus perform our second stage of aggregation; the key idea here is to use Lemma 5.23 from Section 5.3 which guarantees a desired balance condition. In the standard rent-or-buy problem and analysis, we do not care how many unsampled terminals route to a sampled terminal since a sampled terminal would essentially have infinite capacity to the root. In SS-k-Buy-at-Bulk we aggregate demand to the next cable type; to argue that there is a solution on the sampled terminals of cost not much more than OPT'_i, we ensure that the total worst-case demand that can be routed to a sampled terminal is not much more than the capacity of the next cable-type. We give an example at the end of this section which illustrates that the balancing is really necessary; a naive strategy of sampling terminals with probability proportional to u_i/g_{i+1} and routing each unsampled terminal to the k nearest sampled terminals, and recursing, can lead to bad performance.

Lemma 5.41. *Given an instance of* SS-k-Buy-at-Bulk *with center set T in which all demands are at least u_i, let OPT'_i denote the cost of an optimal solution to the instance. There is a randomized polynomial-time algorithm to find a set $T' \subseteq T$ and route flow from each terminal $t \in T \setminus T'$ to k centers in T' along disjoint paths such that:*

1. *The total demand at any center in T' is at most $O(f(k)k\log^2 h \cdot g_{i+1})$.*
2. *The total cost of routing flow from $T \setminus T'$ to T' is at most $O(f(k)k^6 e^{O(k^2)}\log h)\mathrm{OPT}$ with*

high probability.

3. *The expected cost of the optimal solution to the new* SS-k-Buy-at-Bulk *instance on* T' *is at most* $O(f(k)k^2 \log k \log^3 h)\mathrm{OPT}'_i$.

Proof. We developed Lemma 5.23 so that we could implement the second stage of aggregation; the balance condition there allows us to enforce condition 1 of this lemma. Given the instance of SS-k-Buy-at-Bulk on T in which all demands are at least u_i, set $M = u_i/g_{i+1}$ and sample terminals with probability $O(k \log k \log h/M)$ to form the set T'. From Lemmas 5.28 and 5.29 for Rent-or-Buy, each non-sampled terminal will be able to route its demand of u_i along k disjoint paths to sampled terminals using a subset of its $(k - 1/2)M$ paths from Lemma 5.23 with high probability . Further, the total cost of the k paths for each terminal is $O(f(k)k^6 e^{O(k^2)} \log h)\mathrm{OPT}$.

It remains only to argue that there is a good solution on the sampled terminals T' forming the new centers. First, we observe that Lemma 5.23 ensures that the demand at each sampled vertex is $O(f(k)k \log^2 h)M \cdot u_i = O(f(k)k \log^2 h \cdot g_{i+1})$, and terminals are sampled with probability $O(k \log k \log h/M)$. Therefore, the *expected* demand at any terminal is $O(f(k)k^2 \log k \log^3 h)u_i$. From the subadditivity of the function $\hat{f}$ (that is, the economies of scale exhibited by the cable types), the expected cost of a solution on the sampled terminals is $O(f(k)k^2 \log k \log^3 h) \cdot \mathrm{OPT}'_i$. □

After Lemma 5.41, we now have an instance in which all demands are at least g_{i+1}; we have thus successfully eliminated cable type i. Theorem 5.42 completes the proof of our main result for SS-k-Buy-at-Bulk.

Theorem 5.42. *There is an* $(O(\log h))^{3b}$*-approximation for* SS-2-Buy-at-Bulk *with* b *cable-types.*

Proof. We prove a ratio of $(O(\log^3 h))^b$ for $k = 2$; note that terms depending on k (such as $f(2) = 3$ are absorbed into the $O(\cdot)$ notation.

Let OPT denote the cost of an optimal solution to the given instance $\mathcal{I}$; initially, the demand at each terminal is 1. Now, cluster demand into centers with demand between u_1 and $7u_1$ using the first aggregation step to produce a new instance $\mathcal{I}_1$. From Lemma 5.40 and the subsequent discussion, the cost of rerouting demand to these new centers is $O(1)\mathrm{OPT}$, and there is a solution

to $\mathcal{I}_1$ of cost $\mathrm{OPT}_1 = O(1)\mathrm{OPT}$. Consider the modified instance $\mathcal{I}_1'$ in which the demand at all the new centers is reduced to exactly u_1; clearly, the new instance $\mathcal{I}_1'$ has a solution of cost at most OPT_1. We will solve $\mathcal{I}_1'$ and scale up our solution by a factor of 7 to obtain a feasible solution to $\mathcal{I}_1$.

Given the instance $\mathcal{I}_1'$, we use Lemma 5.41 to construct a new instance $\mathcal{I}_2$ with sampled terminals, such that all demands are at most $O(\log^2 h)g_2$. Again, we construct a reduced version $\mathcal{I}_2'$ in which all demands at sampled terminals are exactly g_2; we scale this solution up by a factor of $O(\log^2 h)$ to obtain a feasible solution for $\mathcal{I}_2$. The expected cost of $\mathcal{I}_2'$ is $\mathrm{OPT}_2 = O(\log h)\mathrm{OPT}_1$, since terminals are sampled with probability $O(\log h)u_1/g_2$, and, if sampled, have demand g_2. Now, as all demands are at least g_2, we can remove cable 1, and by induction, solve the instance $\mathcal{I}_2'$ and obtain a solution of cost at most $(O(\log^3 h))^{b-1} \cdot \mathrm{OPT}_2$. Paying at most $O(\log^2 h)$ times this amount, we obtain a solution to $\mathcal{I}_2$ of cost at most $(O(\log^3 h))^{b-1} \cdot O(\log^2 h)\mathrm{OPT}_2$. Now, each terminal of $\mathcal{I}_1'$ that was not sampled must route flow to 2 sampled terminals using disjoint paths; from Lemma 5.23, the total cost of these paths is $O(\log h)\mathrm{OPT}_1$. Therefore, we have a solution to $\mathcal{I}_1'$ of cost $(O(\log^3 h))^{b-1} \cdot O(\log^3 h)\mathrm{OPT}_1 + O(\log h)\mathrm{OPT}_1$. A feasible solution for $\mathcal{I}_1$ can be obtained by scaling this up by a factor of 7, and the rerouting cost to convert the instance $\mathcal{I}$ into $\mathcal{I}_1$ was at most $O(1)\mathrm{OPT}$. Therefore, we have found a solution to the original instance I of cost at most $(O(\log^3 h))^{b-1} \cdot O(\log^3 h)\mathrm{OPT}_1 + O(\log h)\mathrm{OPT}_1 + O(1)\mathrm{OPT} = (O(\log^3 h))^b\mathrm{OPT}$. □

Lemma 5.23 is crucial to the second stage of aggregation in our algorithm for SS-k-Buy-at-Bulk. The following example shows that balancing is really necessary.

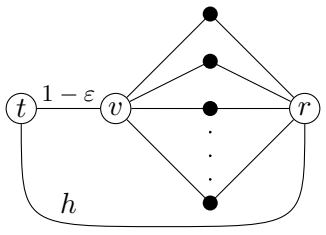


Figure 5.2: An instance of SS-2-Buy-at-Bulk which shows that it is necessary to balance aggregated flow.

In the figure above, the vertex r denotes the root, and t, together with h other terminals (the filled circles) must each be 2-connected to the root. Let each terminal have demand 1, and suppose

we have a cable of capacity 1 and cost 1, and a cable of capacity $2\sqrt{h}$ with cost $\sqrt{h}$. All edge costs are 1, except for the edge from t to v, with cost $1-\varepsilon$, and the edge from t to r, with cost h. There is a simple solution of cost $O(h)$, in which we install 2 cables of capacity 1 on each edge. On the other hand, if we sample terminals with probability $1/\sqrt{h}$ and t is sampled, the 2 shortest paths from *every* non-sampled terminal will be to t and r. Therefore, if t is sampled, its expected demand is roughly $h - \sqrt{h} = h \cdot (1 - o(1))$. To route its demand to the root along 2 disjoint paths, t must use the edge of cost h, and even using the higher-capacity cable, must pay cost $O(h^2)$. Further, t is sampled with probability $1/\sqrt{h}$, so the expected cost of the solution on sampled terminals is $\Omega(h^{3/2})$, which is a factor of $\sqrt{h}$ more than optimal.

In this situation, the balance condition of Lemma 5.23 is precisely what is needed. Instead of accumulating $\Omega(h)$ demand at t, we observe that $\sqrt{h}$ other terminals have been sampled. Therefore, non-sampled terminals should send demand to these other sampled terminals in a balanced manner, even at the expense of slightly increasing the cost of routing their demand. Once this is done, each sampled terminal has not much more than $\sqrt{h}$ demand, and hence we can argue that there is a good solution on the sampled terminals.

5.4.1 Non-Uniform Buy-at-Bulk

We now consider the non-uniform version of SS-k-Buy-at-Bulk. In this version, for each edge e of the graph G there is a given sub-additive cost function f_e and routing x units of demand on e results in a cost of $f_e(x)$. The uniform version is a special case where $f_e = c_e \cdot f$ for a single sub-additive function f. The non-uniform buy-at-bulk problem is considerably harder than its uniform variant and we refer the reader to [133, 49, 40, 48] for prior work and related pointers. We have already mentioned that prior to this work, for $k \geq 2$ the SS-k-Buy-at-Bulk problem did not admit a non-trivial approximation even for the (uniform) 2-cable problem. For the non-uniform single-sink problem there are essentially two approximation algorithms known for $k = 1$, one from [133] and the other from [40]. The algorithm of Charikar and Kargiazova [40] admits a natural generalization for $k \geq 2$. We are able to analyze this algorithm using our result for SS-k-Connectivity to show that it has an approximation ratio of $2^{O(\sqrt{\log h})}$. Note that this bound is essentially the same as the one shown in [40] for the multi-commodity problem since the recurrence

that we obtain is analyzed in a similar fashion. We remark that the [40] proves a bound of $O(\log^2 h)$ for the single-sink problem. However, for $k \geq 2$ the analysis of the recurrence changes dramatically from that for $k = 1$. Although the bound we show is not impressive, the randomized inflated greedy algorithm of [40] is extremely simple and elegant. It is easy to implement and amenable to heuristic improvement and has shown to be effective in some empirical evaluation [10]. The bound also suggests that a poly-logarithmic approximation may be possible.

We now describe the algorithm of [40] adapted to SS-k-Buy-at-Bulk. We assume that each terminal has unit demand to begin with.

Random-Inflated-Greedy:

1. Pick a random permutation π of the terminals in T.

Let $t_1, \ldots, t_h$ be the order of terminals according to π.

2. For $i = 1$ to h in that order:

    Greedily route h/i units of demand from t_i to the root r along k disjoint paths using the cheapest cost paths in the network built by the previous $i - 1$ terminals.

Note that the algorithm routes h/i units of demand for t_i although only one unit of demand is required to be routed. We refer the reader to [40] for the background and intuition behind the design of the above algorithm. Each terminal is routed greedily but the cost of routing on an edge depends on the routing of the previous terminals. More precisely, if x_e^{i-1} is the amount of demand routed on an edge e by the first $i - 1$ terminals then the cost of routing an additional h/i units for terminal i on e is given by $c_e^i = f_e(x_e^{i-1} + h/i) - f_e(x_e^{i-1})$. One can use a min-cost flow computation with costs c_e^i to find the cheapest k disjoint paths from t_i to r. It is easy to see that the algorithm is correct; in the case of $k = 1$, it is known to have an approximation ratio of $O(\log^2 h)$ for $k = 1$ [40]. However, for $k \geq 2$ we are able to establish the following theorem.

Theorem 5.43. *For any fixed k,* Random-Inflated-Greedy *is a $2^{O(\sqrt{\log h})}$-approximation for the non-uniform version of* SS-k-Buy-at-Bulk *with unit-demands. For arbitrary demands there is a $\log D \cdot 2^{O(\sqrt{\log h})}$ approximation algorithm where D is the ratio of the maximum to minimum demands.*

It may seem surprising that the analysis of the algorithm changes dramatically from $k = 1$ to $k = 2$; in fact, this is for some of the same reasons that we described for the uniform version of SS-k-Buy-at-Bulk. The analysis of Theorem 5.43 also involves terminals routing their demand via other terminals. Suppose $k = 2$ and t routes its demand to t' and t''; then t' and t'' route t's demand via two separate disjoint paths each; thus the rerouting leads to an exponential increase in the amount of demand being routed.

We now give a proof of the above theorem which follows the analysis from [40], pointing out the place where we rely on our analysis for SS-k-Connectivity. Let OPT denote the cost of an optimum solution to the problem.

Throughout the analysis below, the expectations of various quantities are with respect to the randomness in picking π. Let C_i denote the *expected* cost of the algorithm in routing the ith terminal in the permutation, and let $C = \sum_{i=1}^{h} C_i$ be an upper bound on the expected total cost of the paths found by the algorithm.

Lemma 5.44. *Let E' be a set of edges such that $G[E']$ contains k disjoint paths from each of the first i terminals to the root, and minimizing the quantity $X_i = \sum_{e \in E'} f_e(h/i)$. That is, X_i denotes the optimal cost of routing h/i flow on each edge of a set $E' \subseteq E$ such that $G[E']$ contains k disjoint paths from each of the first i terminals to the root. The expected cost of X_i is at most* OPT.

Proof. We describe a feasible set of edges E' on which to route h/i flow such that E' contains k disjoint paths from each of the first i terminals to the root, and in expectation, $\sum_{e \in E'} f_e(h/i) \leq$ OPT. Clearly, this implies that for an optimal choice of edges, the expected cost is at most OPT. Fix a given optimal solution O of cost OPT for the entire SS-k-Buy-at-Bulk instance. The feasible solution E' we construct is just the union of the sets of k paths that O uses to connect each of the first i terminals to the root. We simply show that for each edge e, the expected cost we pay for it is at most the cost paid by O.

For any edge e, if O sends at least h/i flow along e, then regardless of the choice of the first i terminals, we pay no more for e than O does. Now suppose O sends $j < h/i$ units of flow for some j terminals along e. The edge e will be selected for E' iff one of these j terminals is among the first i; the probability of this is at most $ji/h < 1$. Therefore, the expected cost we pay for e is at most $\frac{ji}{h} f_e(h/i) \leq f(\frac{ji}{h} \cdot \frac{h}{i}) = f_e(j)$. (The first inequality follows from the sub-additivity of each f_e.)

Therefore, again our payment for e is at most as much as that of O. Hence, $\mathbb{E}[cost(E')] \leq \text{OPT}$, and so $\mathbb{E}[X_i] \leq \text{OPT}$. □

Corollary 5.45. *For $1 \leq i \leq h$, $C_i \leq \text{OPT}$.*

Lemma 5.46. *For any i', i such that $i' < i$, we have $C_i \leq \frac{8k \cdot \text{OPT}}{i'} + \frac{k}{i'} \sum_{1 \leq j \leq i'} \frac{jC_j}{i}$.*

The above two bounds on C_i lead to a proof of Theorem 5.43. Before providing a formal proof, which analyzes the above recurrence with $i' = i/2^{O(\sqrt{\log i})}$, we give a proof of Lemma 5.46.

Proof. First, consider the case that $i' = i - 1$. To upper bound C_i it is sufficient to route the demand of t_i to k distinct terminals in $\{t_1, \ldots, t_{i-1}\}$ via disjoint paths and then piggyback on their paths to the root. (We ignore, for simplicity of description, the case that some of the k paths may terminate directly at the root.) Let A_i be the expected cost of routing h/i demand from t_i to k distinct terminals, and B_i the expected cost of piggybacking onto the paths of those terminals. We bound these separately and use the fact that $C_i \leq A_i + B_i$.

Let R be the set of the first i terminals. Let X_i denote the optimum cost of routing h/i flow on each edge in a set E' such that there are k disjoint paths in E' from each terminal in R to the root. Consider the graph $H = G[E']$ where the cost of each edge $e \in E'$ is $f_e(h/i)$. For $j \leq i$, let $AugCost(t_j)$ be the cost of connecting terminal t_j to k other distinct terminals in R (or the root) via disjoint paths in the graph H. From our main lemma in Section 5.2, Lemma 5.2, $\sum_{1 \leq j \leq i} AugCost(t_j) \leq 8k \cdot X_i$. Since the ith terminal is equally likely to be any of the first i terminals, the expected cost of $AugCost(t_i)$ is at most $8k \cdot X_i/i$. From Lemma 5.44, the expected cost of X_i is at most OPT. Therefore $A_i \leq 8k \cdot \text{OPT}/i$.

Now we bound B_i. Let t_j with $j \leq i - 1$ be one of the k terminals that t_i routes to and piggybacks on. Recall that t_j has already routed h/j demand on k disjoint paths to the root and paid C_j. What is the incremental cost of t_i using the paths of t_j to route its h/i demand? Due to the sub-additivity of the cost functions f_e, we claim that the expected cost of this piggybacking is at most $(h/i)/(h/j) \cdot C_j \leq j/i \cdot C_j$. We omit the proof of this simple claim (see [40]). The k terminals that t_i routes to are equally likely to be in any of the first $i - 1$ positions in the permutation. Therefore, $B_i \leq \frac{k}{i-1} \sum_{1 \leq j \leq i-1} \frac{jC_j}{i}$ and this yields the recurrence for C_i when $i' = i - 1$.

Now consider some $i' < i$. Since π is a random permutation, the first i' terminals together with the ith terminal are equally likely to be any set of $i'+1$ terminals; therefore the above arguments can be easily adapated to give the desired recurrence on C_i. □

In [40], for $k = 1$, the recurrence $C_i \leq 2\mathrm{OPT}/i + \frac{1}{i-1}\sum_{j=1}^{i-1}\frac{jC_j}{i}$ is derived and analyzed. For the first term which corresponds to A_i, one can use a simple argument since an optimum solution can be assumed to be a tree-like. For $k \geq 2$ we need to use the much more complex argument given by Lemma 5.6. However, the main change is in the second term in the recurrence where the multiplicative factor of k makes the recurrence behave very differently for $k \geq 2$. In order to bound the ratio, we use a trick inspired by the ideas in [40] for the multicommodity case. In terms of the algorithm, this corresponds to making t_i piggyback only via terminals $t_1, \ldots, t_{i'}$ for some i' that is sufficiently small compared to i. This helps since the terminals $t_1, \ldots, t_{i'}$ route much larger demand than t_i and hence the piggybacking cost of t_i is a smaller factor. This counterbalances the effect of the rerouted demand using more and more paths in the case of $k \geq 2$.

Proof of Theorem 5.43. In Lemma 5.46, we derived a recurrence for C_i, the expected cost of the algorithm in the ith step. Let γ denote the quantity $8k \cdot \mathrm{OPT}$. Recall that $C_i \leq \mathrm{OPT}$ for all $1 \leq i \leq h$, and

$$C_i \leq \frac{1}{i'}\gamma + \frac{k}{i'}\sum_{j=1}^{i'}\frac{j}{i}C_j$$

for $i' < i$ which implies that

$$iC_i \leq \frac{i}{i'}\gamma + \frac{k}{i'}\sum_{j=1}^{i'} jC_j.$$

Let $D_i = iC_i$. We show by induction on i that $D_i \leq 2^{\sqrt{4\log i \log(k+1)}} \cdot \gamma$. Since $C_i \leq \mathrm{OPT}$ for all i, it follows that $D_i \leq 2^{\sqrt{4\log i \log(k+1)}} \cdot \gamma$ for $i \leq 2^{\sqrt{4\log i \log(k+1)}} \cdot (8k)$. Hence, it is sufficient to prove the induction hypothesis for i larger than $2^{\sqrt{4\log i \log(k+1)}} \cdot (8k)$. Choosing $i' = i/2^{\sqrt{\log i \log(k+1)}}$, we obtain:

$$
\begin{aligned}
D_i &\le \frac{i}{i'}\gamma + \frac{k}{i'}\sum_{j=1}^{i'} D_j \;\le\; \frac{i}{i'}\gamma + \frac{k}{i'}(i' D_{i'}) \;\le\; \frac{i}{i'}\gamma + k D_{i'} \\
&\le 2^{\sqrt{\log i \log(k+1)}} \cdot \gamma + k \cdot 2^{\sqrt{4 \log i' \log(k+1)}} \cdot \gamma \\
&\le 2^{\sqrt{2 \log i' \log(k+1)}} \cdot \gamma + k \cdot 2^{\sqrt{4 \log i' \log(k+1)}} \cdot \gamma \qquad (\text{since } i' \ge \sqrt{i}) \\
&= (k+1) \cdot 2^{\sqrt{4 \log i' \log(k+1)}} \cdot \gamma \\
&\le (k+1)\gamma \cdot 2^{\sqrt{4 \log(k+1)\left(\log i - \sqrt{\log i \log(k+1)}\right)}} \\
&= \gamma \cdot 2^{\log(k+1) + \sqrt{4 \log(k+1) \log i \left(1 - \frac{\sqrt{\log i \log(k+1)}}{\log i}\right)}} \\
&= \gamma \cdot 2^{\sqrt{4 \log i \log(k+1)}\left(\frac{\log(k+1)}{\sqrt{4 \log i \log(k+1)}} + \left(1 - \frac{\sqrt{\log i \log(k+1)}}{\log i}\right)^{1/2}\right)} \\
&\le \gamma \cdot 2^{\sqrt{4 \log i \log(k+1)}\left(\frac{\log(k+1)}{\sqrt{4 \log i \log(k+1)}} + 1 - \frac{\sqrt{\log i \log(k+1)}}{2 \log i}\right)} \\
&\le \gamma \cdot 2^{\sqrt{4 \log i \log(k+1)}\left(1 + \frac{\log(k+1)}{\sqrt{4 \log i \log(k+1)}} - \frac{\sqrt{\log(k+1)}}{2\sqrt{\log i}}\right)} \\
&= \gamma \cdot 2^{\sqrt{4 \log i \log(k+1)}}.
\end{aligned}
$$

Therefore the total cost C is upper bounded as:

$$
\begin{aligned}
C &\le \sum_{i=1}^{h} C_i \;\le\; \sum_{i=1}^{h} D_i/i \;\le\; \sum_{i=1}^{h} 2^{\sqrt{4 \log i \log(k+1)}} \cdot \gamma/i \\
&\le 2^{\sqrt{4 \log h \log(k+1)}} \cdot \gamma \sum_{i=1}^{h} 1/i \\
&= O(2^{\sqrt{4 \log h \log(k+1)}} \cdot \gamma \cdot \log h) \\
&= O(2^{\sqrt{4 \log h \log(k+1)}} 8k \log h) \cdot \mathrm{OPT}.
\end{aligned}
$$

□

5.5 Concluding Remarks

In this chapter we demonstrated that simple and natural extensions of algorithms developed for single-sink connectivity problems when $k = 1$ extend to the case of fixed $k > 1$. In particular, we gave $O(k \log |T|)$-approximations for SS-k-Connectivity and SS-k-Rent-or-Buy. We also studied Uniform-SS-k-Buy-at-Bulk, giving an $O(\log |T|)^{O(b)}$ approximation when $k = 2$, and

Non-Uniform-SS-k-Buy-at-Bulk, giving an $O(2^{O(\sqrt{\log h})})$-approximation for any fixed k. In addition to our dual-based proofs, we also gave simpler analyses for SS-k-Connectivity and SS-k-Rent-or-Buy based on the structural decomposition of [68].

Recently, many new insights have been obtained into the structure of VC-SNDP and related problems from the work of Chuzhoy and Khanna [68, 69] and Nutov [137, 138, 139]; these have opened up many directions for further exploration. One question of interest is whether there exist algorithms for these problems with approximation ratios that are purely functions of k (and not of n, or $|T|$). Nutov recently showed an $O(k^2)$-approximation for SS-k-Connectivity; it may be possible to extend this result to more general problems. Another potential approach is to use Jain's iterated rounding technique [109]; a first step would be to prove the following conjecture: In any (fractionally optimal) extreme point solution to a natural LP relaxation, there is a variable with value $\Omega(1/k)$. If the conjecture were true, one could round such a variable up to 1, and "recurse" on the remaining problem.[14] No counterexample to this conjecture is known; in the worst instances that have been constructed [2], there are variables with value $\Omega(1/\sqrt{k})$.

For SS-k-Buy-at-Bulk, we believe that our results can be extended to $k > 2$ to achieve an approximation of $(k^k \log h)^{O(b)}$. We further conjecture that a $k^{O(b)}\text{polylog}(n)$-approximation exists. For the non-uniform buy-at-bulk problem, the analysis in Section 5.4.1 hints at the existence of a poly-logarithmic approximation ratio.

[14] There are additional technical details we do not discuss here.

Chapter 6

Capacitated Network Design

6.1 Introduction [1]

Recall that in the basic Survivable Network Design Problem, as described in Chapter 5, the input is a graph $G(V, E)$ with a cost on each edge, and an integer requirement R_{uv} for each unordered pair of nodes (u, v). The goal is to find a minimum-cost subgraph in which there are R_{uv} disjoint paths from u to v, or equivalently a minimum-cost subgraph such that the minimum cut separating u from v has size at least R_{uv}. One may require that either the minimum edge-cut or minimum vertex-cut separating u and v be large; the corresponding problems are referred to as EC-SNDP and VC-SNDP respectively. Jain, in an influential paper [109], obtained a 2-approximation for EC-SNDP via the standard cut-based LP relaxation using the iterated rounding technique. In contrast, the VC-SNDP problem is harder; see Chapter 5 for a full discussion.

In this chapter we consider the *capacitated* Survivable Network Design Problem, referred to as Capacitated-SNDP, which is the following generalization of EC-SNDP: The input consists of an undirected n-vertex multi-graph $G(V, E)$ and an integer requirement R_{uv} for each unordered pair of nodes (u, v). Each edge e of G has both a cost c_e and an integer capacity u_e. The goal is to find a minimum-cost subgraph H of G such that for each pair of nodes u, v, the capacity of the minimum-cut between u and v in H is at least R_{uv}.

Although the 2-approximation for EC-SNDP mentioned above has been known since 1998, the approximability of Capacitated-SNDP has essentially been wide open even in very restricted special cases. Similar to SNDP, Capacitated-SNDP is motivated by both practial and theoretical considerations. These problems find applications in the design of resilient networks such as in telecommunication infrastructure. In such networks it is often quite common to have equipment

[1] This chapter is based on joint work with Deeparnab Chakrabarty, Chandra Chekuri, and Sanjeev Khanna.

with different discrete capacities; this leads naturally to design problems such as CAPACITATED-SNDP. We note that a different and somewhat related problem is also referred to by the same name, especially in the operations research literature. In this version the subgraph H has to *simultaneously* support a flow of R_{uv} between each pair of nodes (u, v); this is more closely related to multicommodity flows and BUY-AT-BULK network design. Our version is more related to connectivity problems such as SNDP.

As far as we are aware, the version of CAPACITATED-SNDP that we study was introduced (in the approximation algorithms literature) by Goemans *et al.* [91] in conjunction with their work on SNDP. They made several observations on CAPACITATED-SNDP: (i) CAPACITATED-SNDP reduces to SNDP if all capacities are the same, (ii) there is an $O(\min\{m, R_{\max}\})$ approximation where m is the number of edges in G and $R_{\max} = \max_{uv} R_{uv}$ is the maximum requirement, and (iii) if *multiple* copies of an edge are allowed then there is an $O(\log R_{\max})$-approximation.[2] We note that in the capacitated case $R_{\max}$ can be exponentially large in n, the number of nodes of the graph. Carr *et al.* [37] observed that the natural cut-based LP relaxation has an unbounded integrality gap even for the graph consisting of only two nodes s, t connected by parallel edges with different capacities. Motivated by this observation and the goal of obtaining improved approximation ratio for CAPACITATED-SNDP, [37] strengthened the basic cut-based LP by using *Knapsack-Cover* inequalities. (Several subsequent papers in approximation algorithms have fruitfully used these inequalities.) Using these inequalities, [37] obtained a $\beta(G) + 1$ approximation for CAPACITATED-SNDP where $\beta(G)$ is the maximum cardinality of a *bond* in the underlying simple graph: a bond is a minimal set of edges that separates some pair of vertices with positive demand. Although $\beta(G)$ could be $\Theta(n^2)$ in general, for certain topologies — for instance, if the underlying graph is a line or a cycle — this gives a constant factor approximation.

The results described above naturally lead to several questions. What is the approximability of CAPACITATED-SNDP? Should we expect a poly-logarithmic approximation or even a constant factor approximation? If not, what are interesting and useful special cases to consider? Do the Knapsack-Cover inequalities guarantee LP relaxations of small integrality gap in the worst case? What is the approximability of CAPACITATED-SNDP if one allows multiple copies? Does this

[2] See the discussion in Section 6.1.1 below for a definition of "multiple copies".

relaxed version of the problem allow a constant factor approximation?

In this chapter we make some progress towards answering the above questions by obtaining positive as well as negative results. Our approach is to consider some simple yet illuminating special cases, which yield considerable insight into these questions.

6.1.1 Overview of Results

We first discuss results for CAPACITATED-SNDP where multiple copies are not allowed. We initiate our study by considering the *global connectivity* version of CAPACITATED-SNDP, where we want a min-cost subgraph with global min-cut at least R; in other words, there is a "uniform" requirement $R_{ij} = R$ for *all* pairs (i, j). We refer to this as the CAP-R-CONNECTED SUBGRAPH problem; the special case when all capacities are unit corresponds to the classical minimum cost λ-EDGE-CONNECTED SPANNING SUBGRAPH problem, which is known to be APX-hard [78]. We show the following positive result for arbitrary capacities.

Theorem 6.1. *There is a randomized $O(\log n)$-approximation algorithm for the* CAP-R-CONNECTED SUBGRAPH *problem.*

The same techniques give an algorithm for a slightly more general problem, CAPACITATED-SNDP when requirements are "nearly uniform".

Theorem 6.2. *There is a randomized algorithm with running time $n^{O(\gamma)}$ that obtains an $O(\gamma \log n)$ approximation for* CAPACITATED-SNDP *when $R_{uv} \in [R, \gamma R]$ for all u, v.*

To prove Theorems 6.1 and 6.2, we begin with a natural LP relaxation for the problem. Almost all positive results previously obtained for the unit capacity case are based on this relaxation. As remarked already, this LP has an unbounded integrality gap even for a graph with two nodes (and hence for CAP-R-CONNECTED SUBGRAPH). We strengthen the relaxation by adding the valid Knapsack-Cover inequalities. Although we do not know of a polynomial time algorithm to separate over these inequalities, following [37], we find a violated inequality *only if* the current fractional solution does not satisfy certain useful properties. Our main technical tool both for finding a violated inequality and subsequently rounding the fractional solution is Karger's theorem on the number of small cuts in undirected graphs [113].

We believe the approach outlined above may be useful in other network design applications. As a concrete illustration, we use it to solve an interesting and natural generalization of CAP-R-CONNECTED SUBGRAPH, namely, the k-WAY–$\mathcal{R}$-CONNECTED SUBGRAPH problem. In addition to costs and capacities on the edges, the input consists of $(k-1)$ integer requirements $R_1, \ldots, R_{k-1}$, such that $R_1 \leq R_2 \leq \ldots \leq R_{k-1}$. The goal is to find a minimum-cost subgraph H of G such that for each $1 \leq i \leq k-1$, the capacity of any $(i+1)$-way cut of G is at least R_i.[3] It is easy to see that CAP-R-CONNECTED SUBGRAPH is precisely the k-WAY–$\mathcal{R}$-CONNECTED SUBGRAPH, with $k = 2$. Note that the k-WAY–$\mathcal{R}$-CONNECTED SUBGRAPH problem is not a special case of the general CAPACITATED-SNDP as the cut requirements for the former problem are not expressible as pairwise connectivity constraints. Interestingly, our techniques for CAP-R-CONNECTED SUBGRAPH can be naturally extended to handle the multiway cut requirements, yielding the following generalization of Theorem 6.1.

Theorem 6.3. *There is a randomized $O(k \log n)$-approximation algorithm with running time $n^{O(k)}$ for the k-WAY–$\mathcal{R}$-CONNECTED SUBGRAPH problem.*

We remark that even for the unit-capacity case of this problem, it is not clear how to obtain a better ratio than that guaranteed by the above theorem. We discuss this further in Section 6.2.4.

Once the pairwise connectivity requirements are allowed to vary arbitrarily, the CAPACITATED-SNDP problem seems to become distinctly harder. Surprisingly, the difficulty of the general case starts to manifest even for the simplest representative problem in this setting, where there is only one pair (s,t) with $R_{st} > 0$; we refer to this as the *single pair* problem. The only known positive result for this seemingly restricted case is a polynomial-factor approximation that follows from the results in [91, 37] for general CAPACITATED-SNDP. We give several negative results to suggest that this special case may capture the essential difficulty of CAPACITATED-SNDP. In particular, we start by observing that the LP with knapsack cover inequalities has an $\Omega(n)$ integrality gap even for the single-pair problem.[4] Next we show that the single pair problem is $\Omega(\log \log n)$-hard to approximate.

[3] An i-way cut $\mathcal{C}$ of a graph $G(V,E)$ is a partition of its vertices into i non-empty sets $V_1, \ldots, V_i$; we use $\delta(\mathcal{C})$ to denote the set of edges with endpoints in different sets of the partition $\mathcal{C}$. The *capacity* of an i-way cut $\mathcal{C}$ is the total capacity of edges in $\delta(\mathcal{C})$.

[4] In [37] it is mentioned that there is a series-parallel graph instance of CAPACITATED-SNDP such that the LP with knapsack-cover inequalities has an integrality gap of at least $\lfloor \beta(G)/2 \rfloor + 1$. However, no example is given; it is not clear if the gap applied to a single pair instance or if $\beta(G)$ could be as large as n in the construction.

Theorem 6.4. *The single pair* Capacitated-SNDP *problem cannot be approximated to a factor better than* $\Omega(\log\log n)$ *unless* $NP \subseteq DTIME(n^{\log\log\log n})$.

The above theorem is a corollary of the results in Chuzhoy *et al.*'s work on the hardness of related network design problems [67]. We state it as a theorem to highlight the status of the problem, and give a complete proof in Section 6.3.2. We further discuss this connection at the end of this section. We prove a much stronger negative result for the single pair problem in *directed* graphs. Since in the unit-capacity case, polynomial-time minimum-cost flow algorithms solve the single-pair problem exactly even in directed graphs, the hardness result below shows a stark contrast between the unit-capacity and the non-unit capacity cases.

Theorem 6.5. *In* directed *graphs, the single pair* Capacitated-SNDP *cannot be approximated to a factor better than* $2^{\log^{(1-\delta)} n}$ *for any* $0 < \delta < 1$, *unless* $NP \subseteq DTIME(n^{\texttt{polylog}(n)})$. *Moreover, this hardness holds for instances in which there are only two distinct edge capacities.*

Allowing Multiple Copies

Given the negative results above for even the special case of the single-pair Capacitated-SNDP, it is natural to consider the relaxed version of the problem where multiple copies of an edge can be chosen. Specifically, for any integer $\imath \geq 0$, i copies of e can be bought at a cost of $i \cdot c(e)$ to obtain a capacity $i \cdot u(e)$. In some applications, such as in telecommunication networks, this is a reasonable model. As we discussed, this model was considered by Goemans *et al.* [91] who gave an $O(\log R_{\max})$ approximation for Capacitated-SNDP. This follows from a simple $O(1)$ approximation for the case when all requirements are in $\{0, R\}$. The advantage of allowing multiple copies is that one can group request pairs into classes and separately solve the problem for each class while losing only the number of classes in the approximation ratio. For instance, one easily obtains a 2-approximation for the single pair problem even in directed graphs, in contrast to the difficulty of the problem when multiple copies are not allowed. Note that this also implies an easy $2k$ approximation where k is the number of pairs (u, v) with $R_{uv} > 0$. We address the approximability of Capacitated-SNDP with multiple copies of edges allowed. When $R_{\max}$ is large, we improve the $\min\{2k, O(\log R_{\max})\}$-approximation discussed above via the following.

Theorem 6.6. *In undirected graphs, there is an $O(\log k)$-approximation algorithm for* Capacitated-SNDP *with multiple copies, where k is the number of pairs (u, v) with $R_{uv} > 0$.*

Both our algorithm and analysis are inspired by the $O(\log k)$-competitive online algorithm for the Steiner Forest problem by Berman and Coulston [30], and the subsequent adaptation of these ideas for the Priority Steiner Forest problem by Charikar *et al.* [41]. However, we believe the analysis of our algorithm is more transparent (although it gets weaker constants) than the original analysis of [30].

We complement our algorithmic result by showing that the multiple copy version is $\Omega(\log \log n)$-hard to approximate. This hardness holds even for the *single-source* Capacitated-SNDP where we are given a source node $s \in V$, and a set of terminals $T \subseteq V$, such that $R_{ij} > 0$ iff $i = s$ and $j \in T$. Observe that single-source Capacitated-SNDP is a simultaneous generalization of the classical Steiner Tree problem ($R_{ij} \in \{0, 1\}$) as well as both Cap-R-Connected Subgraph and single-pair Capacitated-SNDP.

Theorem 6.7. *In undirected graphs, single source* Capacitated-SNDP *with multiple copies cannot be approximated to a factor better than $\Omega(\log \log n)$ unless $NP \subseteq DTIME(n^{\log \log \log n})$.*

The above theorem, like Theorem 6.4, also follows easily from the results of [67]; we provide a proof in Section 6.3.2. We note that the hardness reduction above creates instances with super-polynomially large capacities. For such instances, our $O(\log k)$-approximation strongly improves on the previously known approximation guarantees.

6.1.2 Related Work

Network design has a large literature in a variety of areas including computer science and operations research. Practical and theoretical considerations have resulted in numerous models and results. We briefly mention some work that allows the reader to compare the model we consider here to related models. As we mentioned earlier, our version of Capacitated-SNDP is a direct generalization of SNDP and hence is concerned with (capacitated) connectivity between request node pairs. We refer the reader to Chapter 5, the survey [121], and some recent and previous papers [91, 109, 79, 68, 69, 139] for pointers to literature on network design for connectivity. A different model arises

if one wishes to find a min-cost subgraph that supports multicommodity flow for the request pairs; in this model each node pair (u, v) needs to routes a flow of R_{uv} in the chosen graph and these flows simultaneously share the capacity of the graph. We observe that if multiple copies of an edge are allowed then this problem is essentially equivalent to the non-uniform Buy-at-Bulk network design problem that has received substantial recent attention [133, 40, 48, 5]; see also Chapter 5. We refer the reader to [48] for an overview of related work on Buy-at-Bulk network design. If multiple copies are not allowed, the approximability of this flow version is not well-understood; for example if the flow for each pair is only allowed to be routed on a single path then even checking feasibility of a given subgraph is NP-Hard since the problem captures the well-known Edge-Disjoint Paths and Unsplittable Flow problems [117, 50, 6, 21]. Andrews and Zhang [8] have recently considered special cases of this problem with uniform capacities while allowing some congestion on the chosen edges.

The single pair Capacitated-SNDP is a special case of the Fixed Charge Network Flow (FCNF) problem. In this problem, each edge of the graph has a capacity u_e, a fixed cost c_e and a per-unit-flow cost ℓ_e, and each vertex of the graph has a demand for the net flow through it. Given flow x_e on an edge e, the cost incurred on the edge is $(c_e + \ell(e) \cdot x_e)$. The goal is to find the cheapest flow satisfying the demands. Note that if $\ell(e)$ is 0, and the only vertices with non-zero demands are s and t, we get the single pair Capacitated-SNDP. The FCNF problem has been intensively studied in the OR-literature (see for instance, [131, 106, 140]; we point the reader to [136] for a survey), however, to our knowledge, no non-trivial approximation algorithm[5] is known for the problem. For undirected graphs, Chuzhoy *et al.* [67] show that the single commodity FCNF problem is hard to approximate to a factor better than $O(\log \log n)$; as mentioned above, their hardness also holds for the special case of single pair Capacitated-SNDP. (See Section 6.3.2.) No better hardness of approximation is known for the Capacitated-SNDP problem with multiple demand pairs.

The Knapsack-Cover inequalities with which we strengthen the linear programming relaxations for Capacitated-SNDP were first used to design approximation algorithms by Carr *et al.* [37].

[5] Carr *et al.* [37] observe that the case of FCNF with exactly two non-zero demands can be essentially reduced to Capacitated-SNDP, thus they get a $(\beta(G) + 1)$ factor approximation for this case, where $\beta(G) \leq \binom{n}{2}$ is the size of the largest bond separating s from t.

However, these inequalities were previously studied extensively in the OR literature [23, 102, 155].

The k-Way–$\mathcal{R}$-Connected Subgraph problem that we consider does not appear to have been considered previously even in the unit-capacity case.

Chapter Outline

We begin with an $O(\log n)$-approximation for the Cap-R-Connected Subgraph problem in Section 6.2; we also prove that one can obtain a similar approximation ratio if the requirements are "nearly uniform". We show that these techniques can be extended to the k-Way–$\mathcal{R}$-Connected Subgraph problem, proving Theorem 6.3 in Section 6.2.4.

We then present several hardness results even for special cases in Section 6.3: First, in Section 6.3.1 we show that the natural LP has integrality gap $\Omega(n)$, even when strengthened with the Knapsack-Cover inequalities which are useful when requirements are uniform. Then, in Section 6.3.2, we show that Capacitated-SNDP is $\Omega(\log \log n)$-hard to approximate in undirected graphs, even when multiple copies of edges are allowed. (As discussed above, this follows from the work of [67].) In Section 6.3.3, we show that in directed graphs, even the single-pair special case of Capacitated-SNDP is $2^{\log^{(1-\delta)} n}$ hard to approximate.

Finally, in Section 6.4, we give an $O(\log k)$-approximation algorithm for Capacitated-SNDP in undirected graphs when multiple copies are allowed, where $k \leq \binom{n}{2}$ denotes the number of vertex pairs (u, v) such that $R_{uv} > 0$.

6.2 The Cap-R-Connected Subgraph problem

In this section, we prove Theorem 6.1, giving an $O(\log n)$-approximation for Cap-R-Connected Subgraph. We start by writing a natural linear programming relaxation for the problem; the integrality gap of this LP can be arbitrarily large. To deal with this, we introduce additional valid inequalities, called the *Knapsack-Cover* inequalities, that must be satisfied by any integral solution. We show how to round this strengthened LP, obtaining an $O(\log n)$-approximation.

Having described a good algorithm for Cap-R-Connected Subgraph (when all pairwise requirements R_{uv} are equal) we then prove Theorem 6.2 for Capacitated-SNDP with *nearly* uniform requirements, giving an $O(\gamma \log n)$ approximation when all requirements are in $[R, \gamma R]$ for

some fixed R. We also extend Theorem 6.1 to prove Theorem 6.3, giving an $O(k \log n)$ approximation for k-Way–$\mathcal{R}$-Connected Subgraph.

6.2.1 The Standard LP Relaxation and Knapsack-Cover Inequalities

We assume without any loss of generality that the capacity of any edge is at most R. For each subset $S \subseteq 2^V$, we use $\delta(S)$ to denote the set of edges with exactly one endpoint in S. For a set of edges A, we use $u(A)$ to denote $\sum_{e \in A} u_e$. We say that a set of edges A satisfies (the cut induced by) S if $u(A \cap \delta(S)) \geq R$. Note that we wish to find the cheapest set of edges which satisfies every subset $\emptyset \neq S \subset V$. The following is the LP relaxation of the standard integer program capturing the problem.

$$\min \sum_{e \in E} c_e x_e \qquad \text{(Std LP)}$$

$$\sum_{e \in \delta(S)} u_e x_e \geq R \qquad (\forall S \subseteq V)$$

$$0 \leq x_e \leq 1 \qquad (\forall e \in E)$$

The following example shows that (Std LP) can have integrality gap as bad as R.

Example 1: Consider a graph G on three vertices p, q, r. Edge pq has cost 0 and capacity R; edge qr has cost 0 and capacity $R - 1$; and edge pr has cost C and capacity R. To achieve a global min-cut of size at least R, any integral solution must include edge pr, and hence must have cost C. In contrast, in (Std LP) one can set $x_{pr} = 1/R$, and obtain a total cost of C/R.

In the previous example, any integral solution in which the mincut separating r from $\{p, q\}$ has size at least R must include edge pr, even if qr is selected. The following valid inequalities are introduced precisely to enforce this condition. More generally, let S be a set of vertices, and A be an arbitrary set of edges. Define $R(S, A) = \max\{0, R - u(A \cap \delta(S))\}$ to be the *residual* requirement of S that must be satisfied by edges in $\delta(S) \setminus A$. That is, any feasible solution has $\sum_{e \in \delta(S) \setminus A} u_e x_e \geq R(S, A)$. However, any integral solution also satisfies the following stronger requirement

$$\sum_{e \in \delta(S) \setminus A} \min\{R(S, A), u_e\} x_e \geq R(S, A)$$

and thus these inequalities can be added to the LP to strengthen it. These additional inequalities are referred to as *Knapsack-Cover* inequalities, or simply KC inequalities, and were used by [37] in design of approximation algorithms for Capacitated-SNDP.

Below, we write a LP relaxation, (KC LP), strengthened with the Knapsack- Cover inequalities. Note that the original constraints correspond to KC inequalities with $A = \emptyset$; we simply write them explicitly for clarity.

$$\min \sum_{e \in E} c_e x_e \qquad \text{(KC LP)}$$

$$\sum_{e \in \delta(S)} u_e x_e \geq R \qquad (\forall S \subseteq V) \qquad \text{(Original Constraints)}$$

$$\sum_{e \in \delta(S) \setminus A} \min(u_e, R(S,A)) x_e \geq R(S,A) \qquad (\forall A \subseteq E, \forall S \subseteq V) \qquad \text{(KC inequalities)}$$

$$0 \leq x_e \leq 1 \qquad (\forall e \in E)$$

The Linear Program (KC LP), like the original (Std LP), has exponential size. However, unlike (Std LP), we do not know of the existence of an efficient separation oracle for (KC LP). Nevertheless, as we show below, we do not need to solve (KC LP); it suffices to get to what we call a *good* fractional solution.

Definition 6.8. *Given a fractional solution x, we say an edge e is* nearly integral *if $x_e \geq \frac{1}{40 \log n}$, and we say e is* highly fractional *otherwise.*

Definition 6.9. *For any $\alpha \geq 1$, a cut in a graph G with capacities on edges, is an* α-mincut *if its capacity is within a factor α of the minimum cut of G.*

Theorem 6.10. [Theorems 4.7.6 and 4.7.7 of [113]] *The number of α-mincuts in an n-vertex graph is at most $n^{2\alpha}$. Moreover, the set of all α-mincuts can be found in $O(n^{2\alpha} \log^2 n)$ time with high probability.*

Given a fractional solution x to the edges, we let A_x denote the set of nearly integral edges, that is, $A_x := \{e \in E : x_e \geq \frac{1}{40 \log n}\}$. Define $\hat{u}(e) = u_e x_e$ to be the fractional capacity on the edges. Let $\mathcal{S} := \{S \subseteq V : \hat{u}(\delta(S)) \leq 2R\}$. The solution x is said to be *good* if it satisfies the following three conditions:

(a) The global mincut in G with capacity $\hat{u}$ is at least R; equivalently x satisfies the original constraints.

(b) The KC inequalities are satisfied for the set A_x and the sets in $\mathcal{S}$. Note that if (a) is satisfied, then by Theorem 6.10, $|\mathcal{S}| \leq n^4$.

(c) $\sum_{e \in E} c_e x_e$ is at most the value of the optimum solution to (KC LP).

Note that a good solution need not be *feasible* for (KC LP) as it is required to satisfy only a subset of KC inequalities. We use the ellipsoid method to get such a solution; such a method was also used in [37].

Lemma 6.11. *There is a randomized algorithm that computes a good fractional solution with high probability.*

Proof. We start by guessing the optimum value M of (KC LP) and add the constraint $\sum_{e \in E} c_e x_e \leq M$ to the constraints of (KC LP). If the guessed value is too small, a good solution may not exist; however, a simple binary search suffices to identify the smallest feasible value of M. With this constraint in place, we will use the ellipsoid method to compute a solution that satisfies (a), (b), and (c) with high probability. Since we do not know of a polynomial-time separation oracle for KC inequalities, we will simulate a separation oracle that verifies condition (b), a subset of KC inequalities, in polynomial time. Specifically, we give a randomized polynomial time algorithm such that given a solution x that violates condition (b), the algorithm detects the violation with high probability and outputs a violated KC inequality. We now describe the entire process.

Given a solution x we first check if condition (a) is satisfied. This can be done in polynomial time by $O(n)$ max-flow computations. If (a) is not satisfied, we have found a violated constraint. Once we have a solution that satisfies (a), we know that $|\mathcal{S}| \leq n^4$. By Theorem 6.10, the set $\mathcal{S}$ can be computed in polynomial-time with high probability. Thus we can check condition (b) in polynomial-time, and with high probability find a violating constraint for (b) if one exists. Once we have a solution that satisfies both (a) and (b), we check if $\sum_{e \in E} c_e x_e \leq M$. If not, we have once again found a violated constraint for input to the ellipsoid algorithm. Thus in polynomially many rounds, where each round runs in polynomial time, the ellipsoid algorithm combined with the simulated separation oracle, either returns a solution x that satisfies (a), (b), and $\sum_{e \in E} c_e x_e \leq M$,

with high probability, or proves that the system is infeasible. Using binary search, we find the smallest M for which a solution x is returned satisfying conditions (a), (b) and $\sum_{e\in E} c_e x_e \leq M$. Since M is less than the optimum value of (KC LP), we get that the returned x is a good fractional solution with high probability. □

6.2.2 The Rounding and Analysis

Given a good fractional solution x, we now round it to get a $O(\log n)$ approximation to Cap-R-Connected Subgraph. A useful tool for our analysis is the following Chernoff bound (see, for instance, [134]):

Lemma 6.12. *Let $X_1, X_2, \ldots X_k$ be a collection of independent random variables in $[0,1]$, let $X = \sum_{i=1}^{k} X_i$, and let $\mu = \mathbb{E}[X]$. The probability that $X \leq (1-\delta)\mu$ is at most $e^{-\mu\delta^2/2}$.*

We start by selecting A_x, the set of all nearly integral edges. Henceforth, we lose the subscript and denote the set as simply A. Let $F = E \setminus A$ denote the set of all highly fractional edges; for each edge $e \in F$, select it with probability $(40 \log n \cdot x_e)$. Let $F^* \subseteq F$ denote the set of selected highly fractional edges. The algorithm returns the set of edges $E_A := A \cup F^*$.

It is easy to see that the expected cost of this solution E_A is $O(\log n)\sum_{e\in E} c_e x_e$, and hence by condition (c) above, within $O(\log n)$ times that of the optimal integral solution. Thus, to prove Theorem 6.1, it suffices to prove that with high probability, E_A satisfies every cut in the graph G; we devote the rest of the section to this proof. We do this by separately considering cuts of different capacities, where the capacities are with respect to $\hat{u}$ (recall that $\hat{u}_e = u_e x_e$). Let $\mathcal{L}$ be the set of cuts of capacity at least $2R$, that is, $\mathcal{L} := \{S \subseteq V : \hat{u}(\delta(S)) > 2R\}$.

Lemma 6.13. $\mathbf{Pr}[\ \forall S \in \mathcal{L} :\ u(E_A \cap \delta(S)) \geq R]\ \geq\ 1 - \frac{1}{2n^{10}}$.

Proof. We partition $\mathcal{L}$ into sets $\mathcal{L}_2, \mathcal{L}_3, \ldots$ where $\mathcal{L}_j := \{S \subseteq V : jR < \hat{u}(\delta(S)) \leq (j+1)R\}$. Note that Theorem 6.10 implies $|\mathcal{L}_j| \leq n^{2(j+1)}$ by condition (a) above. Fix j, and consider an arbitrary cut $S \in \mathcal{L}_j$. If $u(A \cap \delta(S)) \geq R$, then S is clearly satisfied by E_A. Otherwise, since the total $\hat{u}$-capacity of S is at least jR, we have $\hat{u}(F \cap \delta(S)) \geq \hat{u}(\delta(S)) - u(A \cap \delta(S)) \geq (j-1)R$. Thus

$$\sum_{e \in F \cap \delta(S)} \frac{u_e}{R} x_e \geq (j-1)$$

Recall that an edge $e \in F$ is selected in F^* with probability $(40 \log n \cdot x_e)$. Thus, for the cut S, the expected value of $\sum_{e \in F^* \cap \delta(S)} \frac{u_e}{R} \geq 40(j-1)\log n$. Since $u_e/R \leq 1$, we can apply Lemma 6.12 to get that the probability that S is not satisfied is at most $e^{-16 \log n(j-1)} = 1/n^{16(j-1)}$. Applying the union bound, the probability that there exists a cut in $\mathcal{L}_j$ not satisfied by E_A is at most $n^{2(j+1)}/n^{16(j-1)} = n^{18-14j}$. Thus probability that some cut in $\mathcal{L}$ is not satisfied is bounded by $\sum_{j \geq 2} n^{18-14j} \leq 2n^{-10}$ if $n \geq 2$. Hence with probability at least $1 - 1/2n^{10}$, $A \cup F^*$ satisfies all cuts in $\mathcal{L}$. □

One might naturally attempt the same approach for the cuts in $\mathcal{S}$ (recall that $\mathcal{S} = \{S \subseteq V : \hat{u}(\delta(S)) \leq 2R\}$) modified as follows. Consider any cut S, which is partly satisfied by the nearly integral edges A. The fractional edges contribute to the residual requirement of S, and since x_e is scaled up for fractional edges by a factor of $40 \log n$, one might expect that F^* satisfies the residual requirement, with the $\log n$ factor providing a high-probability guarantee. This intuition is correct, but the KC inequalities are crucial. Consider Example 1; edge pr is unlikely to be selected, even after scaling. In the statement of Lemma 6.12, it is important that each random variable takes values in $[0, 1]$; thus, to use this lemma, we need the expected capacity from fractional edges to be large compared to the maximum capacity of an individual edge. But the KC inequalities, in which edge capacities are "reduced", enforce precisely this condition. Thus we get the following lemma using an analysis similar to the one above.

Lemma 6.14. $\mathbf{Pr}[\ \forall S \in \mathcal{S} :\ u(\delta(E_A \cup \delta(S))) \geq R]\ \geq\ 1 - \frac{1}{n^{12}}$.

Proof. By Theorem 6.10, the number of cuts in $\mathcal{S}$ is at most n^4; it thus suffices to show that for any $S \in \mathcal{S}$, the probability it is not satisified by E_A is at most n^{-16}. Assume S is not satisfied by A, otherwise we are done.

Since x is good, by condition (b) above, we have $\sum_{e \in \delta(S) \cap F} \min\{u_e, R(S, A)\} x_e \geq R(S, A)$. Thus:

$$\sum_{e \in \delta(S) \cap F} \frac{\min\{u_e, R(S, A)\}}{R(S, A)} x_e \geq 1$$

Once again since the coefficient of x_e is at most 1, as in the proof of Lemma 6.13, we get that the probability S is not satisfied by F^* is at most $e^{-16 \log n} \leq n^{-16}$, and we are done. □

Theorem 6.1 follows from the two previous lemmas.

6.2.3 Capacitated-SNDP with Nearly Uniform Requirements

The algorithm described above can be extended to the case where requirements are *nearly* uniform, that is, if $R_{uv} \in [R, \gamma R]$ for all pairs $(u, v) \in V \times V$. We obtain an $O(\gamma \log n)$-approximation, while increasing the running time by a factor of $O(n^{4\gamma})$. We work with a similar LP relaxation; for each set $S \subseteq 2^V$, we use $R(S) = \max_{u \in S, v \notin S}\{R_{uv}\}$ to denote the requirement of S. Now, the original constraints are of the form

$$\sum_{e \in \delta(S)} u_e x_e \geq R(S)$$

for each set S, and we define the residual requirement for a set as $R(S, A) = \min\{0, R(S) - u(A \cap \delta(S))\}$. The KC inequalities use this new definition of $R(S, A)$.

Given a fractional solution x to the KC LP, we modify the definitions of highly fractional and nearly integral edges: An edge e is said to be nearly integral if $x_e \geq \frac{1}{40\gamma \log n}$, and highly fractional otherwise. Again, for a fractional solution x, we let A_x denote the set of nearly integral edges; the set $\mathcal{S}$ of small cuts is now $\{S \subseteq V : \hat{u}(\delta(S)) \leq 2\gamma R\}$. From the cut-counting theorem, $|\mathcal{S}| \leq n^{4\gamma}$. We use $\mathcal{L}$ to denote the set of *large* cuts, the sets $\{S \subseteq V : \hat{u}(\delta(S)) > 2\gamma R\}$.

As before, a fractional solution x is *good* if the original constraints are satisfied, and the KC Inequalities are satisfied for the set of edges A_x and the sets in $\mathcal{S}$. These constraints can be checked in time $O(n^{4\gamma+2} \log^2 n)$, so following the proof of Lemma 6.11, for constant γ, we can find a good fractional solution in polynomial time.

The rounding and analysis proceed precisely as before: For each highly fractional edge e, we select it for the final solution with probability $40\gamma \log n \cdot x_e$. The expected cost of this solution is at most $O(\gamma \log n)$ times that of the optimal integral solution, and analogously to the proofs of Lemmas 6.13 and 6.14, one can show that the solution satisfies all cuts with high probability. This completes the proof of Theorem 6.2.

6.2.4 The k-Way–$\mathcal{R}$-Connected Subgraph Problem

The k-Way–$\mathcal{R}$-Connected Subgraph problem that we define is a natural generalization of the well-studied min-cost λ-Edge-Connected Spanning Subgraph problem. The latter problem is motivated by applications to fault-tolerant network design where any $\lambda - 1$ edge failures should

not disconnect the graph. However, there may be situations in which global λ-connectivity may be too expensive or infeasible. For example, the underlying graph G may have a single cut-edge but we still wish a subgraph that is as close to 2-edge-connected as possible. We could model the requirement by k-Way–$\mathcal{R}$-Connected Subgraph (in the unit-capacity case) by setting $R_1 = 1$ and $R_2 = 3$; that is, at least 3 edges have to be removed to partition the graph into 3 disconnected pieces.

Our proof of Theorem 6.3 is similar to that of Theorem 6.1, but we use the following lemma on counting k-way cuts in place of Theorem 6.10.

Lemma 6.15 (Lemma 11.2.1 of [113])**.** *In an n-vertex graph, the number of k-way cuts with capacity at most α times that of a minimum k-way cut is at most $n^{2\alpha(k-1)}$.*

To prove Theorem 6.3, we work with the generalization of (KC LP) given below. For any i-way cut $\mathcal{C}$ and for any set of edges A, we use $R(\mathcal{C}, A)$ to be $\max\{0, R_i - u(A \cap \delta(C)\}$.[6]

$$\min \sum_{e \in E} c_e x_e \qquad \text{(k-way KC LP)}$$

$$\sum_{e \in \delta(\mathcal{C})} u(e) x_e \geq R_i \qquad (\forall i, \forall i\text{-way cuts } \mathcal{C}) \qquad \text{(Original Constraints)}$$

$$\sum_{e \in \delta(\mathcal{C}) \setminus A} \min\{u(e), R(\mathcal{C}, A)\} x_e \geq R(\mathcal{C}, A) \qquad (\forall A \subseteq E, \forall i, \forall i\text{-way cuts } \mathcal{C}) \qquad \text{(KC inequalities)}$$

$$0 \leq x_e \geq 1 \qquad (\forall e \in E)$$

As before, given a fractional solution x to this LP, we define A_x (the set of nearly integral edges) to be $\{e \in E \colon x_e \geq \frac{1}{40k \log n}\}$. Define $\hat{u}(e) = u(e)x_e$ to be the fractional capacity on the edges. Let $\mathcal{S}_i := \{\mathcal{C} : \mathcal{C}$ is an $(i+1)$-way cut and $\hat{u}(\delta(C)) \leq 2R_i\}$. The solution x is said to be *good* if it satisfies the following three conditions:

(a) If the capacity of e is $\hat{u}(e)$, the capacity of any $(i+1)$-way cut in G is at least R_i; equivalently x satisfies the original constraints.

[6]For ease of notation, we assume that for any edge e, $u(e) \leq R_1$. This is not without loss of generality, but the proof can be trivially generalized: In the constraint for each $(i+1)$-way cut $\mathcal{C}$ such that $e \in \delta(\mathcal{C})$, simply use the minimum of $u(e)$ and R_i.

(b) The KC inequalities are satisfied for the set A_x and the sets in $\mathcal{S}_i$, for each $1 \leq i \leq k-1$. Note that if (a) is satisfied, then by Lemma 6.15, $|\mathcal{S}_i| \leq n^{4i}$.

(c) $\sum_{e \in E} c(e)x_e$ is at most the value of the optimum solution to the linear program (k-way KC LP).

Following the proof of Lemma 6.11, it is completely straightforward to verify that there is a randomized algorithm that computes a good fractional solution with high probability in $n^{O(k)}$ time.

Once we have a good fractional solution, our algorithm is to select A_x, the set of nearly integral edges, and to select each highly fractional edge $e \in E \setminus A_x$ with probability $40k \log n \cdot x_e$. If F^* denotes the highly fractional edges that were selected, we return the solution $A_x \cup F^*$. As before, it is trivial to see that the expected cost of this solution is $O(k \log n)$ times that of the optimal integral solution. We show below that for any $i \leq k-1$, we satisfy all $(i+1)$-way cuts with high probability; taking the union bound over the $k-1$ choices of i yields the theorem.

As in Lemmas 6.13 and 6.14, we separately consider the "large" and "small" $(i+1)$-way cuts. First, consider any small cut $\mathcal{C}$ in $\mathcal{S}_i$. From the Chernoff bound (Lemma 6.12) and the KC inequality for $\mathcal{C}$ and A_x, it follows that the probability we fail to satisfy $\mathcal{C}$ is at most $1/n^{19k}$. From the cut-counting Lemma 6.15, there are at most $n^{4i} < n^{4k}$ such small cuts, so we satisfy all the small $(i+1)$-way cuts with probability at least $1 - \frac{1}{n^{15k}}$.

For the large $(i+1)$-way cuts $\mathcal{L}$, we separately consider cuts of differing capacities. For each $j \geq 2$, let $\mathcal{L}(j)$ denote the $(i+1)$-way cuts $\mathcal{C}$ such that $jR_i \leq \hat{u}(\mathcal{C}) \leq (j+1)R_i$. Consider any cut $\mathcal{C} \in \mathcal{L}_j$; if $u(A_x \cap \delta(C)) \geq R_i$, then the cut $\mathcal{C}$ is clearly satisfied. Otherwise, $\hat{u}(\delta(\mathcal{C}) \setminus A_x) \geq (j-1)R_i$. But since we selected each edge e in $\delta(\mathcal{C}) \setminus A_x$ for F^* with probability $40k \log n \cdot x_e$, the Chernoff bound implies that we do not satisfy $\mathcal{C}$ with probability at most $\frac{1}{n^{19k(j-1)}}$. The cut-counting Lemma 6.15 implies there are most $n^{2i(j+1)} < n^{2k(j+1)}$ such cuts, so we fail to satisfy any cut in $\mathcal{L}(j)$ with probability at most n^{21-17j}. Taking the union bound over all j, the failure probability is at most $2n^{-13}$.

6.3 Hardness of Approximation for Capacitated-SNDP

In this section we show that the integrality gap with KC inequalities is $\Omega(n)$ even for single-pair Capacitated-SNDP in undirected graphs. Moreover, when the underlying graph is directed, we prove that the single-pair problem is hard to approximate to within a factor of $2^{\log^{(1-\delta)} n}$ for any $\delta > 0$.

6.3.1 Integrality Gap with KC Inequalities

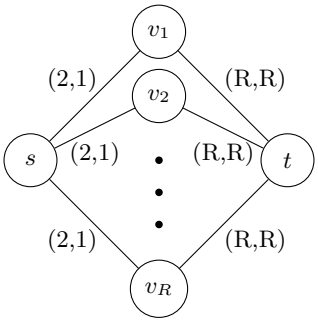


Figure 6.1: An example with integrality gap $\Omega(n)$ for the strengthened LP. Label (u, c) on an edge denotes a capacity of u and cost of c for that edge.

We show that for any positive integer R, there exists a single-pair Capacitated-SNDP instance G with $(R + 2)$ vertices such that the integrality gap of the natural LP relaxation strengthened with KC inequalities is $\Omega(R)$. The instance G consists of a source vertex s, a sink vertex t, and R other vertices $v_1, v_2, \ldots, v_R$. There is an edge of capacity 2 and cost 1 (call these *small* edges) between s and each v_i, and an edge of capacity R and cost R between each v_i and t (*large* edges). We have $R_{st} = R$. Clearly, an optimal integral solution must select at least $R/2$ of the large edges (in addition to small edges), and hence has cost greater than $R^2/2$. The instance is depicted in Figure 6.1: Label (u, c) on an edge denotes capacity u and cost c.

We now describe a feasible LP solution: set $x_e = 1$ on each small edge e, and $x_{e'} = 2/R$ on each large edge e'. The cost of this solution is R from the small edges, and $2R$ from the large edges, for a total of $3R$. This is a factor of $R/6$ smaller than the optimal integral solution, proving the desired integrality gap.

It remains only to verify that this is indeed a feasible solution to (KC LP). Consider the

constraint corresponding to sets S, A. As edges in $A \setminus \delta(S)$ play no role, we may assume $A \subseteq \delta(S)$. If A includes a large edge, or at least $R/2$ small edges, the residual requirement $R(S, A)$ that must be satisfied by the remaining edges of $\delta(S)$ is 0, and so the constraint is trivially satisfied. Let A consist of $a < R/2$ small edges; the residual requirement is thus $R - 2a$. Let $\delta(S)$ contain i large edges and thus $R - i$ small edges. Now, the contribution to the left side of the constraint from small edges in $\delta(S) \setminus A$ is $2(R-i-a) = (R-2a)+(R-2i)$. Therefore, the residual requirement is satisfied by small edges alone unless $i > R/2$. But the contribution of large edges is $i \cdot \frac{2}{R} \cdot (R-2a)$ which is greater than $R-2a$ whenever $i > R/2$. Thus, we satisfy each of the added KC inequalities.

6.3.2 Hardness of Approximation for Capacitated-SNDP in Undirected Graphs

In this section, we prove Theorem 6.7 via a reduction from the Priority Steiner Tree problem. In Priority Steiner Tree, the input is an undirected graph $G(V, E)$ with a cost c_e and a priority $P(e) \in \{1, 2, \ldots, k\}$ for each edge e. (We assume k is the highest and 1 the lowest priority.) We are also given a root r and a set of terminals $T \subseteq V - \{r\}$; each terminal $t \in T$ has a desired priority $P(t)$. The goal is to find a minimum-cost Steiner Tree in which the unique path from each terminal t to the root consists only of edges of priority $P(t)$ or higher.[7]

Chuzhoy *et al.* [67] showed that one cannot approximate the Priority Steiner Tree problem within a factor better than $\Omega(\log \log n)$ unless $NP \subseteq DTIME(n^{\log \log \log n})$, even when all edge costs are 0 or 1. Here, we show an approximation-preserving reduction from this problem to Capacitated-SNDP with multiple copies; this also applies to the basic Capacitated-SNDP, as the copies of edges do not play a significant role in the reduction.

Given an instance $\mathcal{I}_{pst}$ of Priority Steiner Tree on graph $G(V, E)$ with edge costs in $\{0, 1\}$, we construct an instance $\mathcal{I}_{cap}$ of Capacitated-SNDP defined on the graph G as the underlying graph. Fix R to be any integer greater than $2m^3$ where m is the number of edges in the graph G. We now assign a capacity of $u_e = R^i$ to each edge e with priority $P(e) = i$ in $\mathcal{I}_{pst}$. Each edge e of cost 0 in $\mathcal{I}_{pst}$ has cost $c_e = 1$ in $\mathcal{I}_{cap}$, and each edge e of cost 1 in $\mathcal{I}_{pst}$ has cost $c_e = m^2$ in $\mathcal{I}_{cap}$. Finally, for each terminal t, set $R_{tr} = R^i$ if $P(t) = i$; for every other pair of vertices (p, q), $R_{pq} = 0$.

[7] It is easy to see that a minimum-cost subgraph containing such a path for each terminal is a tree; given any cycle, one can remove the edge of lowest priority.

Let C denotes the cost of an optimal solution to $\mathcal{I}_{pst}$; note that $C \leq m$; we now argue that $\mathcal{I}_{pst}$ has an optimal solution of cost C iff $\mathcal{I}_{cap}$ has an optimal solution of of cost between Cm^2 and $Cm^2 + m < (C+1)m^2$. Given a solution E^* to $\mathcal{I}_{pst}$ of cost C, simply select the same edges for $\mathcal{I}_{cap}$; the cost in $\mathcal{I}_{cap}$ is at most $Cm^2 + m$ since in $\mathcal{I}_{cap}$, we pay 1 for each edge in E^* that has cost 0 in $\mathcal{I}_{pst}$. This is clearly a feasible solution to $\mathcal{I}_{cap}$ as each terminal t has a path to r in E^* containing only edges with priority at least $P(t)$, which is equivalent to having capacity at least R_{tr}. Conversely, given a solution E' to $\mathcal{I}_{cap}$ with cost in $[Cm^2, (C+1)m^2)$, select a single copy of each edge in E' as a solution to $\mathcal{I}_{pst}$; clearly the total cost is at most C. To see that this is a feasible solution, suppose that E' did not contain a path from some terminal t to the root r using edges of priority $P(t)$ or more. Then there must be a cut separating t from r in which all edges of E' have capacity at most $R^{P(t)-1}$. But since E' supports a flow of $R^{P(t)}$ from t to r, it must use at least R edges (counting with multiplicity); this implies that the cost of E' is at least $R \geq (C+1)m^2$, a contradiction.

We remark that a similar reduction also proves that the single-pair Capacitated-SNDP problem without multiple copies is $\Omega(\log \log n)$ hard to approximate: One can effectively encode an instance of single-source Fixed-Charge Network Flow (FCNF, [67]), very similar to single-source Capacitated-SNDP with multiple copies, as an instance of single-pair Capacitated-SNDP *without* multiple copies: Create a new sink t^*, and connect t^* to each original terminal t with a single edge of cost 0 and capacity R_{tr}. The only way to send flow $\sum_{t \in T} R_{tr}$ flow from t^* to the source s is for each terminal t to send R_{tr} to s. Thus, Theorem 6.4 is a simple consequence of the $\Omega(\log \log n)$ hardness for single-source FCNF [67].

6.3.3 Hardness of Approximation in Directed Graphs

We now prove Theorem 6.5 via a reduction from the Label Cover problem [13].

Definition 6.16 (*The* Label Cover *Problem).* *The input consists of a bipartite graph $G(A \cup B, E)$ such that the degree of every vertex in A is d_A and degree of every vertex in B is d_B, a set of labels L_A and a set of labels L_B, and a relation $\pi_{(a,b)} \subseteq L_A \times L_B$ for each edge $(a,b) \in E$. Given a labeling $\phi : A \cup B \to L_A \cup L_B$, an edge $e = (a,b) \in E$ is said to be* consistent *iff $(\phi(a), \phi(b)) \in \pi_{(a,b)}$. The goal is to find a labeling that maximizes the fraction of consistent edges.*

The following hardness result for Label Cover is a well-known consequence of the PCP theorem [16] and Raz's Parallel Repetition theorem [143].

Theorem 6.17 ([16, 143])**.** *For any $\varepsilon > 0$, there does not exist a poly-time algorithm to decide if a given instance of* Label Cover *has a labeling where all edges are consistent* (Yes-Instance)*, or if no labeling can make at least $\frac{1}{\gamma}$ fraction of edges to be consistent for $\gamma = 2^{\log^{1-\varepsilon} n}$* (No-Instance)*, unless $NP \subseteq DTIME(n^{\texttt{polylog}(n)})$.*

We now give a reduction from Label Cover to the single-pair Capacitated-SNDP in directed graphs. In our reduction, the only non-zero capacity values will be 1, d_A, and d_B. We note that Theorem 6.17 holds even when we restrict to instances with $d_A = d_B$. Thus our hardness result will hold on single-pair Capacitated-SNDP instances where there are only two distinct non-zero capacity values.

Given an instance I of Label Cover with m edges, we create in polynomial-time a directed instance I' of single-pair Capacitated-SNDP such that if I is a Yes-Instance then I' has a solution of cost at most $2m$, and otherwise, every solution to I' has cost $\Omega(m\gamma^{\frac{1}{4}})$. This establishes Theorem 6.5 when we choose $\varepsilon = \delta/2$.

The underlying graph $G'(V', E')$ for the single-pair Capacitated-SNDP instance is constructed as follows. The set V' contains a vertex v for every $v \in A \cup B$. We slightly abuse notation and refer to these sets of vertices in V' as A and B as well. Furthermore, for every vertex $a \in A$, and for every label $\ell \in L_A$, the set V' contains a vertex $a(\ell)$. Similarly, for every vertex $b \in B$, and for every label $\ell \in L_B$, the set V' contains a vertex $b(\ell)$. Finally, V' contains a source vertex s and a sink vertex t. The set E' contains the following directed edges:

- For each vertex a in A, there is an edge from s to the vertex a of cost 0 and capacity d_A. For each vertex $b \in B$, there is an edge from b to t of cost 0 and capacity d_B.
- For each vertex $a \in A$, and for all labels ℓ in L_A, there is an edge from a to $a(\ell)$ of cost d_A and capacity d_A. For each vertex $b \in B$, and for all labels ℓ in L_B, there is an edge from $b(\ell)$ to b of cost d_B and capacity d_B. These two types of edges are the only edges with non-zero cost.

- For every edge $(a, b) \in E$, and for every pair of labels $(\ell_a, \ell_b) \in \pi_{(a,b)}$, there is an edge from $a(\ell_a)$ to $b(\ell_b)$ of cost 0 and capacity 1.

This completes the description of the network G'. The requirement R_{st} between s and t is m, the number of edges in the LABEL COVER instance. It is easy to verify that the size of the graph G' is at most quadratic in the size of the LABEL COVER instance, and that G' can be constructed in polynomial-time.

Lemma 6.18. *If the* LABEL COVER *instance is a* YES-INSTANCE*, then G' contains a subgraph of cost $2m$ which can realize a flow of value m from s to t.*

Proof. Let ϕ be any labeling that consistently labels all edges in $G(A \cup B, E)$. Let $E_1 \subseteq E'$ be the set of all edges of cost 0 in E', and let $E_2 \subseteq E'$ be the set of edges $\{(a, a(\phi(a))) \mid a \in A\} \cup \{(b(\phi(b)), b) : b \in B\}$. We claim that $E_1 \cup E_2$ is a feasible solution for the single-pair CAPACITATED-SNDP instance. Note that the total cost of edges in $E_1 \cup E_2$ is $|A|d_A + |B|d_B = 2m$. We now exhibit a flow of value m from s to t in $G''(V', E_1 \cup E_2)$. A flow of value d_A is sent along the path $s \to a \to a(\phi(a))$ for all $a \in A$. From $a(\phi(a))$, a unit of flow is sent to the d_A vertices of the form $\{b(\phi(b)) \mid b \in B \text{ and } (a, b) \in E\}$; this is feasible because ϕ consistently labels all edges in E. Thus each vertex of the form $b(\phi(b))$ where $b \in B$ receives d_B units of flow, since the degree of b is d_B in G. A flow of value d_B is sent to t along the path $b(\phi(b)) \to b \to t$. Thus s sends out a flow of value $|A|d_A = m$, or equivalently, t receives a flow of value $|B|d_B = m$. □

Lemma 6.19. *If the* LABEL COVER *instance is a* NO-INSTANCE*, then any subgraph of G' that realizes a flow of m units from s to t has cost $\Omega(m\gamma^{\frac{1}{4}})$.*

Proof. Let $\rho = \gamma^{1/4}/2$, and $M = 32/15$. Assume by way of contradiction, that there exists a subgraph $G''(V', E'')$ of G' of cost strictly less than $\frac{\rho m}{M}$ that realizes m units of flow from s to t. We say a vertex $a \in A$ is *light* if the number of edges of the form $\{(a, a(\ell)) \mid \ell \in L_A\}$ in G'' is less than ρ. Similarly, we say a vertex $b \in B$ is *light* if the number of edges of the from $\{(b(\ell), b) \mid \ell \in L_B\}$ in G'' is less than ρ. All other vertices in $A \cup B$ are referred to as *heavy* vertices. Note that at most $1/M$ fraction of vertices in A could be heavy, for otherwise the total cost of the edges in E'' would exceed $\frac{|A|}{M} \cdot \rho \cdot d_A = \frac{\rho m}{M}$. Similarly, at most $1/M$ fraction of vertices in B could be heavy.

Now fix any integral s-t flow f of value m in G''; an integral flow exists since all capacities are integers. We start by deleting from G'' all heavy vertices. Since at most $1/M$ fraction of either A or B are deleted, the total residual flow in this network is at least $(1 - \frac{2}{M})m = \frac{m}{16}$ (recall that $M = 32/15$) since at most d_A units of flow can transit through a vertex in A, and at most d_B units of flow can transit through a vertex in B.

Let F be a decomposition of the residual flow into unit flow paths. Note that $|F| = m/16$. By construction of G', every flow path $f \in F$ is of the form $s \to a \to \ell_a \to \ell_b \to b \to t$ where the pair $(\ell_a, \ell_b) \in \pi_{(a,b)}$. We say that an edge $(a, b) \in E$ is a *good* edge if there is a flow path f of the above form, and we say f is a certificate for edge (a, b) being good. Note that every flow path f is a certificate of exactly one edge (a, b). We claim that there are at least $\frac{m}{16\rho^2}$ good edges in G. It suffices to show that for any edge $(a, b) \in E$, at most ρ^2 flow paths in F can certify that (a, b) as a good edge. Since a and b are both light vertices, we know that $|\{(a, \ell_a) \mid \ell_a \in L_A\} \cap E''| \leq \rho$ and $|\{(\ell_b, b) \mid \ell_b \in L_B\} \cap E''| \leq \rho$. Now using the fact that each edge (ℓ_a, ℓ_b) has unit capacity, it follows that at most ρ^2 paths in F can certify (a, b) as a good edge. Hence the number of good edges in E is at least $\frac{m}{16\rho^2}$.

We now show the existence of a labeling ϕ that makes at least $\frac{1}{\gamma}$ fraction of the edges to be consistent, contradicting the fact that we were given a No-Instance of Label Cover. For a vertex $a \in A$, let $\Gamma(a) := \{\ell_a \in L_A \mid (a, \ell_a) \in E''\}$. Similarly, we define $\Gamma(b)$ for each vertex $b \in B$. Consider the following random label assignment: each vertex $a \in A$ is assigned uniformly at random a label from $\Gamma(a)$, and each vertex in B is assigned uniformly at random a label in $\Gamma(b)$. For any good edge (a, b), the probability that the random labeling makes it consistent is at least $\frac{1}{\rho^2}$ since $|\Gamma(a)|$ and $|\Gamma(b)|$ are both less than ρ (as a and b are light), and there exists an $\ell_a \in \Gamma_A$ and $\ell_b \in \Gamma_B$ such that $(\ell_a, \ell_b) \in \pi_{(a,b)}$. Thus, in expectation, at least $\frac{1}{\rho^2}$ fraction of good edges are made consistent by the random assignment. Hence there exists a labeling ϕ that $\frac{m}{16\rho^4} = \frac{m}{\gamma}$ edges in G consistent. □

Since the graph G' can be constructed from G in polynomial time, it follows that a poly-time $(\gamma^{1/4}/5)$-approximation algorithm for single-pair Capacitated-SNDP would give a poly-time algorithm to decide whether a given instance of Label Cover is a Yes-Instance or a No-Instance.

6.4 Capacitated-SNDP with Multiple Copies of Edges

We now consider the variant of Capacitated-SNDP when multiple copies of any edge e can be chosen; that is, for any edge e and integer $i \geq 0$, i copies of e can be bought at a cost $i \cdot c_e$ to obtain a capacity of $i \cdot u_e$ between e's endpoints. Recall that we proved $\Omega(\log \log n)$ hardness of approximation for this variant in Theorem 6.7. Still, allowing multiple copies of an edge to be chosen appears to make the problem easier, and Goemans *et al.* [91] give a $O(\log R_{\max})$-approximation algorithm for the problem; for completeness, we describe an algorithm achieving this ratio in Section 6.4.2. In general, however, $R_{\max}$ may be exponentially large, and hence this does not guarantee a sub-polynomial approximation ratio. In Section 6.4.1, we give a $O(\log k)$-approximation algorithm, where k is the number of vertex pairs (u, v) with $R_{uv} > 0$.

6.4.1 An $O(\log k)$-Approximation

Our algorithm for Capacitated-SNDP when multiple copies of an edge can be chosen is based on that of Berman and Coulston [30] for online Steiner Forest; however, we believe our proof is simpler and more illuminating than that of [30], though the constant we obtain is weaker. For notational convenience, we rename the pairs $(s_1, t_1), \cdots, (s_k, t_k)$, and denote the requirement R_{s_i,t_i} as R_i; the vertices s_i, t_i are referred to as *terminals*. We also assume that the pairs are so ordered that $R_1 \geq R_2 \geq \cdots \geq R_k$.

We first give an intuitive overview of the algorithm. The algorithm considers the pairs in decreasing order of requirements, and maintains a *forest solution* connecting the pairs that have been already been processed; that is, if we retain a single copy of each edge in the partial solution constructed so far, we obtain a forest F. For any edge e on the path in F between s_j and t_j, the total capacity of copies of e will be at least R_j. When considering s_i, t_i, we connect them as cheaply as possible, assuming that edges previously selected for F have 0 cost. (Note that this can be done since we are processing the pairs in decreasing order of requirements and for each edge already present in F, the capacity of its copies is at least R_i.) The key step of the algorithm is that *in addition* to connecting s_i and t_i, we also connect the pair to certain other components of F that are "nearby". The cost of these additional connections can be bounded by the cost of the direct connection costs between the pairs. These additional connections are useful in allowing subsequent

pairs of terminals to be connected cheaply. In particular, they allow us to prove a $O(\log k)$ upper bound on the approximation factor.

We now describe the algorithm in more detail. The algorithm maintains a forest F of edges that have already been bought; F satisfies the invariant that, after iteration $i-1$, for each $j \leq i-1$, F contains a unique path between s_j and t_j. In iteration i, we consider the pair s_i, t_i. We define the cost function $c_i(e)$ as $c_i(e) := 0$ for edges e already in F, and $c_i(e) := c(e) + \frac{R_i}{u(e)}c(e)$, for edges $e \notin F$. Note that for an edge $e \notin F$, the cost $c_i(e)$ is sufficient to buy enough copies of e to achieve a total capacity of R_i. Thus it suffices to connect s_i and t_i and pay cost $c_i(e)$ for each edge; in the Cap-SNDP solution we would pay at most this cost and get a feasible solution. However, recall that our algorithm also connects s_i and t_i to other "close" components; to describe this process, we introduce some notation:

We now describe the algorithm in more detail. The algorithm maintains a forest F of edges that have already been bought; F satisfies the invariant that, after iteration $i-1$, for each $j \leq i-1$, F contains a unique path between s_j and t_j. In iteration i, we consider the pair s_i, t_i. We define the cost function $c_i(e)$ as $c_i(e) := 0$ for edges e already in F, and $c_i(e) := c(e) + \frac{R_i}{u(e)}c(e)$, for edges $e \notin F$. Note that for an edge $e \notin F$, the cost $c_i(e)$ is sufficient to buy enough copies of e to achieve a total capacity of R_i. Thus it suffices to connect s_i and t_i and pay cost $c_i(e)$ for each edge; in the Cap-SNDP solution we would pay at most this cost and get a feasible solution. However, recall that our algorithm also connects s_i and t_i to other "close" components; to describe this process, we introduce some notation:

For any vertices p and q, we use $d_i(p, q)$ to denote the distance between p and q according to the metric given by edge costs $c_i(e)$. We let $\ell_i := d_i(s_i, t_i)$ be the cost required to connect s_i and t_i, given the current solution F. We also define the *class* of a pair (s_j, t_j), and of a component:

- For each $j \leq i$, we say that pair (s_j, t_j) is in *class* h if $2^h \leq \ell_j < 2^{h+1}$.
  Equivalently, $\texttt{class}(j) = \lfloor \log \ell_j \rfloor$.

- For each connected component X of F, $\texttt{class}(X) = \max_{(s_j,t_j)\in X} \texttt{class}(j)$.

Now, the algorithm connects s_i (respectively t_i) to component X if $d_i(s_i, X)$ (resp. $d_i(t_i, X)$) $\leq 2^{\min\{\texttt{class}(i),\texttt{class}(X)\}}$. That is, if X is close to the pair (s_i, t_i) compared to the classes they are in, we connect X to the pair. As we show in the analysis, this extra connection cost can be charged

to some pair (s_j, t_j) in the component X. The complete algorithm description is given below.

```
CAPACITATED-SNDP-MC:
F ← ∅                                         《F is the forest solution returned》
For i ← 1 to k
   For each edge e ∈ F, c_i(e) ← 0
   For each edge e ∉ F, c_i(e) ← c_e + (R_i/u_e)c_e
   ℓ_i ← d_i(s_i, t_i)
   Add to F a shortest path (of length ℓ_i) from s_i to t_i under distances c_i(e)
   class(i) ← ⌊log ℓ_i⌋
   For each connected component X of F
         If d_i(s_i, X) ≤ 2^min{class(i),class(X)}
               Add to F a shortest path connecting s_i and X
   For each connected component X of F
         If d_i(t_i, X) ≤ 2^min{class(i),class(X)}
               Add to F a shortest path connecting t_i and X
   Buy ⌈R_i/u_e⌉ copies of each edge e added during this iteration.
```

Note that though the forest F may change several times during a single iteration i of the outer loop, the costs $c_i(e)$ are fixed at the beginning of each iteration. Also, the components of F may change during the final loops; thus, these loops run over the components that have not been merged with the component containing s_i and t_i.

We show that this algorithm Capacitated-SNDP-MC gives an $O(\log k)$ approximation, which proves Theorem 6.6. The structure of our proof is as follows: Recall that ℓ_i was the direct connection cost between s_i and t_i; in addition to paying ℓ_i to connect these vertices, the algorithm also buys additional edges connecting s_i and t_i to existing components. We first show (in Lemma 6.21) that the total cost of extra edges bought can be charged to the direct connection costs; thus, it suffices to show that $\sum_i \ell_i \leq O(\log k)\mathrm{OPT}$, where OPT is the cost of an optimal solution. To prove this (Lemma 6.22), we bucket the pairs (s_i, t_i) into $O(\log k)$ groups based on $\texttt{class}(i)$, and show that in each bucket h, $\sum_{i:\texttt{class}(i)=h} \ell_i \leq O(\mathrm{OPT})$.

Proposition 6.20. Capacitated-SNDP-MC *returns a feasible solution.*

Proof. We prove by induction on i that after iterations 1 through i, F contains a path between s_j and t_j for each $j \leq i$. Further, for each edge e added in iteration i, the total capacity of the copies of e is at least R_i.

Consider any iteration i; as we only add edges to F, the hypothesis is still satisfied for each

pair s_j, t_j with $j < i$. Since we add to F a shortest path between s_i and t_i, F clearly contains the desired path. Consider any edge e on this path: If it was added in iteration $j < i$, the total capacity of its copies is at least $R_j \geq R_i$; if it was added during iteration i, the total capacity of its copies is $\lceil \frac{R_i}{u_e} \rceil u_e \geq R_i$. Thus, the capacity of (the copies of) any edge along the path from s_i to t_i is at least R_i, giving a feasible solution. □

Lemma 6.21. *The total cost of all edges bought by* CAPACITATED-SNDP-MC *is at most* $9 \sum_{i=1}^{k} \ell_i$.

Proof. Let F_i denote the set of edges added to F during iteration i. First, note the total cost paid for copies of edge $e \in F_i$ is $\lceil \frac{R_i}{u(e)} \rceil c(e) < c(e) + \frac{R_i}{u_e} c(e) = c_i(e)$. Thus, it suffices to show:

$$\sum_{i=1}^{k} \sum_{e \in F_i} c_i(e) \leq 9 \sum_{i=1}^{k} \ell_i$$

We prove that the total cost of the *additional* edges bought is at most $8 \sum_{i=1}^{k} \ell_i$; this clearly implies the desired inequality. It is *not* true that for each i, the total cost of additional edges bought during iteration i is at most $8\ell i$. Nonetheless, a careful charging scheme proves the needed bound on total cost. In iteration i, suppose we connect the pair (s_i, t_i) to the components $X_1, \ldots, X_r$. We charge the cost of connecting (s_i, t_i) and component X_j to the connection cost ℓ_j of a pair (s_j, t_j) in X_j. This is possible since we know the additional connection cost is at most $2^{\texttt{class}(X_j)}$. Care is required to ensure no pair is overcharged. To do so, we introduce some notation.

At any point during the execution of the algorithm, for any current component X of F, we let $\texttt{Leader}(X)$ be a pair $(s_i, t_i) \in X$ such that $\texttt{class}(i) = \texttt{class}(X)$. For integers $h \leq \texttt{class}(X)$, $h\text{-}\texttt{Leader}(X)$ will denote a pair (s_j, t_j) in X; we explain how this pair is chosen later. (Initially, $h\text{-}\texttt{Leader}(X)$ is undefined for each component X.)

Now, we have to account for additional edges bought during iteration i; these are edges on a shortest path connecting s_i (or t_i) to some other component X; we assume w.l.o.g. that the path is from s_i to X. Consider any such path P connecting s_i to a component X; we have $\sum_{e \in P} c_i(e) = d_i(s_i, X) \leq 2^{\min\{\texttt{class}(i), \texttt{class}(X)\}}$. Let $h = \lfloor \log d_i(s_i, X) \rfloor$: Charge all edges on this path to $h\text{-}\texttt{Leader}(X)$ if it is defined; otherwise, charge all edges on the path to $\texttt{Leader}(X)$. In either case, the pair (s_i, t_i) becomes the $h\text{-}\texttt{Leader}$ of the new component just formed. Note that a pair (s_i, t_i) could simultaneously be the $h_1\text{-}\texttt{Leader}$, $h_2\text{-}\texttt{Leader}$, etc. for a component X if (s_i, t_i)

connected to many components during iteration i. However, it can never be the h-`Leader` of a component for $h > \texttt{class}(i)$, and once it has been charged as h-`Leader`, it is never charged again as h-`Leader`. Also observe that if a pair is in a component X whose h-`Leader` is defined, subsequently, it always stays in a component in which the h-`Leader` is defined.

For any i, we claim that the total charge to pair (s_i, t_i) is at most $8\ell_i$, which completes the proof. Consider any such pair: any charges to the pair occur when it is either `Leader` or h-`Leader` of its current component. First, consider charges to (s_i, t_i) as `Leader` of a component. Such a charge can only occur when connecting some s_j (or t_j) to X. Furthermore, if $h = \lfloor \log d_j(s_j, X) \rfloor \leq \texttt{class}(X) = \texttt{class}(i)$, the h-`Leader`(X) must be *currently undefined*, for otherwise the h-`Leader`(X) would have been charged. Subsequently, the h-`Leader` of the component containing (s_i, t_i) is always defined, and so (s_i, t_i) will never again be charged as a `Leader`(X) by a path of length in $[2^h, 2^{h+1})$. Therefore, the total charge to (s_i, t_i) as `Leader` of a component is at most $\sum_{h=1}^{\texttt{class}(i)} 2^{h+1} < 2^{\texttt{class}(i)+2} \leq 4\ell_i$.

Finally, consider charges to (s_i, t_i) as h-`Leader` of a component. As observed above, $h \leq \texttt{class}(i)$. Also for a fixed h, a pair is charged at most once as h-`Leader`. Since the total cost charged to (s_i, t_i) as h-`Leader` is at most 2^{h+1}; summing over all $h \leq \texttt{class}(i)$, the total charge is less than $2^{\texttt{class}(i)+2} = 4\ell_i$.

Thus, the total charge to (s_i, t_i) is at most $4\ell_i + 4\ell_i = 8\ell_i$, completing the proof. □

Lemma 6.22. *If* OPT *denotes the cost of an optimal solution to the instance of* Capacitated-SNDP *with multiple copies, then* $\sum_{i=1}^{k} \ell_i \leq 64(\lceil \log k \rceil + 1)\text{OPT}$.

Proof. Let C_h denote $\sum_{i:\texttt{class}(i)=h} \ell_i$. Clearly, $\sum_{i=1}^{k} \ell_i = \sum_h C_h$. The lemma follows from the two sub-claims below:

Sub-Claim 1: $\sum_h C_h \leq (2(\lceil \log k \rceil + 1)) \cdot \max_h C_h$

Sub-Claim 2: For each h, $C_h \leq 32\text{OPT}$.

Proof of Sub-Claim 1. Let $h' = \max_i \texttt{class}(i)$. We have $C_{h'} \geq 2^{h'}$, and for any terminal i such that $\texttt{class}(i) \leq h' - (\lceil \log k \rceil + 1)$, we have $\ell_i \leq \frac{2^{h'+1}}{2k}$. Thus, the total contribution from such classes is at most $\frac{2^{h'}}{k} \cdot k = 2^{h'}$, and hence:

$$\sum_{h=h'-\lceil \log k \rceil}^{h'} C_h \geq \frac{\sum_h C_h}{2}, \text{ which implies}$$

$$\max_{h'-\lceil \log k \rceil \leq h \leq h'} C_h \geq \frac{\sum_h C_h}{2(\lceil \log k \rceil + 1)}. \qquad \square$$

$\square$

It remains to show Sub-Claim 2, that for each h, $C_h \leq 32\text{OPT}$. Fix h. Let $\mathcal{S}_h$ denote the set of pairs s_i, t_i such that $\texttt{class}(i) = h$. Our proof will go via the natural primal and dual relaxations for the Cap-SNDP problem. In particular, we will exhibit a solution to the dual relaxation of cost $C_h/32$. To do so we will require the following claim. Define $\texttt{ball}(s_i, r)$, a *ball* of radius r around s_i as containing the set of vertices v such that $d_i(s_i, v) \leq r$ and the set of edges $e = uv$ such that $d_i(s_i, \{u, v\}) + c_i(e) \leq r$. An edge e is *partially* within the ball if $d_i(s_i, \{u, v\}) < r < d_i(s_i, \{u, v\}) + c_i(e)$. Subsequently, we assume for ease of exposition that no edges are partially contained within the balls we consider; this can be achieved by subdividing the edges as necessary. Similarly, we define $\texttt{ball}(t_i, r)$, the ball of radius r around t_i. Two balls are said to be *disjoint* if they contain no common vertices.

Claim 6.23. *There exists a subset of pairs, $\mathcal{S}'_h \subseteq \mathcal{S}_h$, $|\mathcal{S}'_h| \geq |\mathcal{S}_h|/2$, and a collection of $|\mathcal{S}'_h|$ disjoint balls of radius $2^h/4$ centred around either s_i or t_i, for every pair $(s_i, t_i) \in \mathcal{S}'_h$.*

We prove this claim later; we now use it to complete the proof of Sub-Claim 2. First we describe the LP. Let the variable x_e denote whether or not edge e is in the Capacitated-SNDP solution. Let $\mathcal{P}_i$ be the set of paths from s_i to t_i. For each $P \in \mathcal{P}_i$, variable f_P denotes how much flow t sends to the root along path P. We use $u_i(e)$ to refer to $\min\{R_i, u_e\}$, the effective capacity of edge e for pair (s_i, t_i).

Primal $\min \sum_{e \in E} c_e x_e$

$$\sum_{P \in \mathcal{P}_i} f_P \geq R_i \qquad (\forall i \in [k])$$

$$\sum_{P \in \mathcal{P}_i | e \in P} f_P \leq u_i(e) x_e \qquad (\forall i \in [k], e \in E)$$

$$x_e \geq 0 \qquad (\forall e \in E)$$

$$f_P \geq 0 \qquad (\forall i \in [k], \forall P \in \mathcal{P}_i)$$

Dual $\max \sum_{t \in T} R_i \alpha_i$

$$\sum_i u_i(e) \beta_{i,e} \leq c_e \qquad (\forall e \in E)$$

$$\alpha_i \leq \sum_{e \in P} \beta_{i,e} \qquad (\forall i \in [k], P \in \mathcal{P}_i)$$

$$\alpha_i, \beta_{i,e} \geq 0 \qquad (\forall e \in E, \forall i \in [k])$$

We now describe a feasible dual solution of value at least $C_h/32$ using Claim 6.23. For $(s_i, t_i) \in \mathcal{S}'_h$, if there is a ball B around s_i (or equivalently t_i), we define $\beta_{i,e} = c(e)/u_i(e)$ for each edge in the ball. Since the balls are disjoint, the first inequality of the dual is clearly satisfied. Set $\alpha_i = 2^h/8R_i$. For any path $P \in \mathcal{P}_i$, we have

$$\sum_{e \in P} \beta_{i,e} = \frac{1}{R_i} \sum_{e \in P \cap B} \frac{R_i c(e)}{u_i(e)} \geq \frac{1}{2R_i} \sum_{e \in P \cap B} \frac{R_i c(e)}{u(e)} + c(e) \geq \frac{1}{2R_i} \sum_{e \in P \cap B} c_i(e) \geq \frac{1}{2R_i} \frac{2^h}{4} = \alpha_i$$

where the first inequality used $u_i(e) \leq R_i$, the second follows from the definition of $c_i(e)$, and the last inequality follows from the definition of $\texttt{ball}(s_i, 2^h/4)$. Thus, $\alpha_i = 2^h/8R_i$ is feasible along with these $\beta_{i,e}$'s. This gives a total dual value of

$$\frac{2^h}{8} \cdot |\mathcal{S}'_h| \geq \frac{2^h}{16} \cdot |\mathcal{S}_h| \geq \frac{1}{32} \sum_{i \in \mathcal{S}_h} \ell_i = \frac{C_h}{32}$$

where the last inequality follows from the fact that $\texttt{class}(i) = h$. This proves the lemma modulo Claim 6.23, which we now prove.

Proof of Claim 6.23. We process the pairs in $\mathcal{S}_h$ in the order they are processed by the original algorithm and grow the balls. We abuse notation and suppose these pairs are $(s_1, t_1), \ldots, (s_p, t_p)$. We maintain a collection of disjoint balls of radius $r = 2^h/4$, initially empty.

At stage i, we try to grow a ball of radius r around either s_i or t_i. If this is not possible, the ball around s_i intersects that around some previous terminal in $\mathcal{S}'_h$, say s_j; similarly, the ball around t_i intersects that of a previous terminal, say t_ℓ. Let v be a vertex in $\texttt{ball}(s_i, r)$ and $\texttt{ball}(s_j, r)$. We have $d_i(s_i, s_j) \leq d_i(s_i, v) + d_i(v, s_j) \leq d_i(s_i, v) + d_j(v, s_j) < 2^h/2$. (The second inequality follows because for any $j < i$ and any edge e, $c_i(e) \leq c_j(e)$.) Similarly, we have $d_i(t_i, t_\ell) < 2^h/2$.

Now, we observe that s_j and t_ℓ could not have been in the same component of F at the beginning of iteration i of Cap-SNDP-MC; otherwise $d_i(s_i, t_i) \leq d_i(s_i, s_j) + d_i(t_i, t_\ell) < 2^h$, contradicting that $\texttt{class}(i) = h$. But since $d_i(s_i, s_j) \leq 2^h/2$ and $\texttt{class}(i) = \texttt{class}(j) = h$, we connect s_i to the component of s_j during iteration i; likewise, we connect t_i to the component of t_ℓ during this iteration. Hence, at the end of the iteration, s_i, t_i, s_j, t_ℓ are all in the same component. As a result, the number of components of F containing pairs of $\mathcal{S}_h$ *decreases* by at least one during the iteration.

It is now easy to complete the proof: During any iteration of F corresponding to a pair $(s_i, t_i) \in \mathcal{S}_h$, the number of components of F containing pairs of $\mathcal{S}_h$ can go up by at most one. Say that an iteration *succeeds* if we can grow a ball of radius r around either s_i or t_i, and *fails* otherwise. During any iteration that fails, the number of components decreases by at least one; as the number of components is always non-negative, the number of iterations which fail is no more than the number which succeed. That is, $|\mathcal{S}'_h| \geq |\mathcal{S}_h - \mathcal{S}'_h|$. □

This completes our proof of Lemma 6.22.

Theorem 6.6 is now a straightforward consequence of Lemmas 6.21 and 6.22:

Proof of Theorem 6.6. The total cost of edges bought by the algorithm is at most $\sum_{i=1}^{k} \sum_{e \in F_i} c_i(e) \leq 9 \sum_{i=1}^{k} \ell_i$, by Lemma 6.21. But $\sum_{i=1}^{k} \ell_i \leq 64(\lceil \log k \rceil + 1)\text{OPT}$, by Lemma 6.22, and hence the total cost paid by CAP-SNDP-MC is at most $O(\log k)\text{OPT}$. □

6.4.2 An $O(\log R_{\max})$-Approximation

We briefly describe the $O(\log R_{\max})$-approximation of [154] for CAPACITATED-SNDP,[8] obtained as a consequence of the following theorem for the special case when all non-zero demands are identical:

Theorem 6.24 ([154])**.** *In undirected graphs, there is a* 4*-approximation algorithm for* CAPACITATED-SNDP *with multiple copies if* $R_{uv} \in \{0, R\}$ *for each pair of vertices* (u, v)*.*

We defer the proof of this theorem, first showing how it yields the desired result:

Theorem 6.25 ([154])**.** *In undirected graphs, there is an* $O(\log R_{\max})$*-approximation algorithm for* CAPACITATED-SNDP *with multiple copies, where* $R_{\max} = \max_{uv} R_{uv}$*.*

Proof. Given an arbitrary instance $\mathcal{I}_1$ of CAPACITATED-SNDP with multiple copies, let OPT denote the cost of an optimal solution. One can create a new instance $\mathcal{I}_2$ by raising each requirement R_{uv} to $2^{\lceil \log R_{uv} \rceil}$ (intuitively, to the next power of 2) while increasing the cost to at most 2OPT; simply take two copies of each edge in an optimal solution to $\mathcal{I}_1$. This solution is clearly feasible for $\mathcal{I}_2$.

Now, for each $j \in \{0, 1, \ldots \lceil \log R_{\max} \rceil\}$ in turn, create an instance $\mathcal{I}_2(j)$ in which we retain only the requirements equal to 2^j. (That is, if $R_{uv} = 2^j$ in instance $\mathcal{I}_2$, $R_{uv} = 2^j$ in instance $\mathcal{I}_2(j)$

[8] Note that the presentation of the analysis in [154] is slightly different.

and $R_{uv} = 0$ in each of the other $\lceil \log R_{\max} \rceil - 1$ newly created instances.) Each instance $\mathcal{I}_2(j)$ has an optimal solution of cost at most 2OPT; we solve each separately using the algorithm of Theorem 6.24 and take the union of the $\lceil \log R_{\max} \rceil$ solutions, each of cost at most 8OPT. Together, these are feasible for $\mathcal{I}_2$, and hence $\mathcal{I}_1$; their total cost is at most $8\lceil \log R_{\max} \rceil \mathrm{OPT}$. □

Thus, it remains only to prove Theorem 6.24, for the special case when all demands are 0 or R. For any set $S \subseteq 2^V$, let $f(S) = 1$ if S separates some pair (u, v) with $R_{uv} = R$, and let $f(S) = 0$ otherwise.[9] We use the following natural LP relaxation for the problem:

$$\min \sum_{e \in E} c(e) x_e \tag{LP1}$$

$$\sum_{e \in \delta(S)} u(e) x_e \geq R f(S) \qquad \left(\forall S \subseteq 2^V\right)$$

$$x_e \geq 0 \qquad (\forall e \in E)$$

We show that this LP has integrality gap 4 via a reduction to the Steiner Forest problem. Define an auxiliary cost function $c'(e) = c(e) + \frac{R}{u(e)} c(e)$ for each edge e. Let (LP2) denote the following linear program:

$$\min \sum_{e \in E} c'(e) z_e \tag{LP2}$$

$$\sum_{e \in \delta(S)} z_e \geq f(S) \qquad \left(\forall S \subseteq 2^V\right)$$

$$1 \geq z_e \geq 0 \qquad (\forall e \in E)$$

Claim 6.26. *An optimal solution to (LP2) has cost at most twice that of an optimal solution to (LP1).*

Proof. Given an optimal solution x^* to (LP1), we set $z_e = \frac{u_e}{R} x^*_e$. This is clearly feasible for (LP2). The contribution of edge e to the objective function of (LP2) is $c'(e) z_e = c(e) z_e + \frac{R}{u(e)} c(e) z_e \leq c(e) z_e + c(e) x^*_e \leq 2 c(e) x^*_e$. (The last inequality follows because $z_e \leq x^*_e$.) □

Claim 6.27 ([154])**.** *The integrality gap of (LP2) is at most* 2.

[9] The function f is *proper*; see, for instance, [92].

Proof. (LP2) is simply the standard LP relaxation for the Steiner Forest instance given by the pairs (u, v) such that $R_{uv} = R$, and with edge costs $c'(e)$. This LP is well known to have integrality gap 2 (see the primal-dual 2-approximation in [154], for example). □

Claim 6.28. *Given an integral solution to* $(LP2)$*, one can construct an integral solution to* $(LP1)$ *of the same cost.*

Proof. Given an integral solution z to (LP2), take $\lceil \frac{R}{u(e)} \rceil$ copies of edge e if $z_e = 1$. This is clearly feasible for (LP1); it has cost at most $\frac{R}{u(e)} c(e) + c(e) = c'(e) z_e$. □

Thus, in polynomial time, we obtain a 4-approximate integral solution to (LP1); this completes the proof of Theorem 6.24.

6.5 Concluding Remarks

In this chapter we addressed the approximability of Capacitated-SNDP, giving new algorithms and hardness results for several special cases of the problem. We gave an $O(\log n)$ approximation for Cap-R-Connected Subgraph, which is a capacitated generalization of the well-studied min-cost λ-Edge-Connected Spanning Subgraph problem. A natural question for further study is whether we can improve this to obtain an $O(1)$ approximation, as is possible for λ-Edge-Connected Spanning Subgraph. If not, can one prove super-constant hardness of approximation? In fact, it may be possible to show an $O(1)$ integrality gap for (KC LP); we do not know of any instances where the gap is super-constant. However, showing a constant integrality gap for (KC LP) may require new rounding techniques, as our methods rely on scaling up the LP variables by a factor of $\Omega(\log n)$. Algorithms not based on an LP relaxation would also be of significant interest, even if the approximation ratio obtained is weaker; such methods might apply to more general problems where the integrality gap for (KC LP) is $\Omega(n)$.

We also highlight the difficulty of Capacitated-SNDP by focusing on the single pair problem, and showing both super-constant hardness and an $\Omega(n)$ integrality gap example, even when the LP is strengthened with Knapsack-Cover inequalities. We believe that understanding the single pair problem is the key to understanding the general case. In particular, we do not have a non-trivial algorithm even for instances in which the edge capacities are either 1 or $R_{\max} = \max_{uv} R_{uv}$. (Note

that when $R_{\max}$ is polynomially bounded, one can reduce any instance of Capacitated-SNDP to an instance in which edges either have capacity 1 and cost 0, or capacity $R_{\max}$ and arbitrary cost.) We also showed that in directed graphs, even the single-pair problem is $2^{\log^{1-\delta} n}$-hard to approximate; this is a striking illustration of the fact that Capacitated-SNDP is significantly harder than the uncapacitated version. We believe that the problem is indeed very hard to approximate: In recent work, we showed that the problem is $\mathcal{APX}$-Hard even when $R_{\max}$ is polynomially bounded (note that in our $\Omega(\log \log n)$ hardness result, we used super-polynomial requirements); we believe that it should be possible to extend these hardness results.

Finally, we noted that allowing multiple copies of edges makes the problem easier, and gave an $O(\log k)$-approximation. In practice it may desirable to not allow too many copies of an edge to be used. It is therefore of interest to examine the approximability of Capacitated-SNDP if we allow only a small number of copies of an edge. Does the problem admit a non-trivial approximation if we allow $O(1)$ copies, or perhaps even $O(\log n)$ copies? This investigation may further serve to differentiate the easy and difficult cases of Capacitated-SNDP.

Chapter 7

Conclusions

In this thesis, we presented algorithms for several fundamental problems related to vehicle routing and network design. In Chapter 2, we gave improved algorithms for Orienteering and several variants in both undirected and directed graphs. In Chapters 3 through 6, we considered several problems in the field of fault-tolerant network design, where the goal was typically to find low-cost networks that satisfied a certain connectivity requirement. In particular, we designed networks resilient to *vertex* failures, an area which was poorly understood until recently. In Chapter 3 we gave algorithms to find low-cost 2-vertex-connected subgraphs of any desired size, and in Chapter 5 we gave simple combinatorial algorithms for several *single-sink* network design problems with vertex-connectivity requirements. Our algorithms are simple and efficient, and we believe they will be of use in solving such problems, which have numerous applications in the design and management of robust telecommunications networks. For example, our randomized inflated-greedy algorithm for Non-Uniform-SS-k-Buy-at-Bulk (based on the work of [40]) is easy to implement and heuristically improve; it was also effective in empirical evaluation conducted by [10].

In Chapter 4, the Reduction Lemma we proved allowed us to drastically simplify graphs while preserving the element-connectivity of all pairs of terminals. We gave applications to packing Steiner trees and forests, and to SS-k-Connectivity. We believe that this lemma will find several additional applications in the future, particularly in light of the recent work [68] showing further connections between vertex-connectivity and element-connectivity.

Finally, in Chapter 6, we considered *capacitated* versions of the Survivable Network Design Problem. We gave hardness results and the first non-trivial approximation algorithms for several special cases of the problem. However, we still do not fully understand the approximability of Capacitated-SNDP; there is much scope for further work. In particular, as we have few algorithmic results for general Capacitated-SNDP, it would be interesting to obtain improved

algorithms when the input structure is restricted; see the discussion below.

There are several natural directions for further research on problems discussed in this thesis; many questions of interest have been listed in the concluding remarks at the end of each chapter. We discuss one broad area for future investigation here:

In many applications of graph problems, the instances that occur in practice have a highly restricted structure. For example, graphs that arise in network design applications are often planar, or nearly planar. Similarly, road networks that form the underlying graphs in vehicle routing applications are typically nearly planar. Many well-known $\mathcal{NP}$-Hard problems become significantly easier on planar graphs; Baker [22] gave Polynomial-Time Approximation Schemes (PTASes) for problems such as Vertex Cover and Independent Set. Subsequently, Arora *et al.* [14] gave a PTAS for TSP in planar graphs; Klein extended this to the Subset-TSP problem [115]. More recently, PTASes were obtained for the planar versions of Steiner Tree [35] and Steiner Forest [28]. We recently extended these ideas to give a PTAS for the Prize-Collecting Steiner Tree problem [45]; a similar result was independently achieved by Bateni *et al.* [29]. All these problems are $\mathcal{APX}$-Hard on general graphs, but the structure imposed by planarity makes their planar versions more tractable. It is natural to ask whether one can obtain similarly improved ratios for the problems considered in this thesis. In particular, is there a PTAS for Orienteering or k-Stroll in undirected planar graphs? (Note that this is unlikely for Orient-TW, as there is no $o(\log n)$-approximation known even when the input graph is a line.) The techniques used to obtain PTASes for TSP, Steiner Tree, Steiner Forest and Prize-Collecting Steiner Tree do not appear to extend to the k-MST, k-Stroll, or Orienteering problems; new approaches may be required.

Several of the other problems we consider are only known to admit poly-logarithmic approximations, and may not have PTASes even in planar graphs. Still, it may be possible to prove constant-factor approximations in planar settings. As merely one example of a problem where this is possible, note that we gave $O(1)$-approximations for packing element-disjoint Steiner trees and forests in planar graphs, though there is an $\Omega(\log n)$ lower bound on their approximability in general graphs. In *directed* planar graphs, Gharan and Saberi [89] recently showed an $O(1)$-approximation for TSP; can this be extended to directed Orienteering or k-Stroll? Can we

obtain constant-factor approximations for k-2VC or Budget-2VC? Is this possible for VC-SNDP, at least when $R_{max} = \max_{uv} R_{uv}$ is small? Some evidence for this was given by Borradaile and Klein [34], who gave a PTAS for a variant of EC-SNDP in which $R_{\max} = 2$. Similarly, can one obtain improved approximations for Rent-or-Buy or Buy-at-Bulk?

Other classes of problems in which the input has restricted structure are also of interest. For example, the input graphs could be sparse or have bounded treewidth, or edge costs could arise from a metric in which vertices are embedded. Such problems often arise in applications, and algorithms that exploit such restricted structure are likely to be of both significant theoretical and practical interest.

References

[1] A. Aazami, J. Cheriyan, and K. Jampani. Approximation Algorithms and Hardness Results for Packing Element-Disjoint Steiner Trees in Planar Graphs. In *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 1–14. Springer, 2009.

[2] A. Aazami, J. Cheriyan, and B. Laekhanukit. A Bad Example for the Iterative Rounding Method for Mincost k-Connected Spanning Subgraphs. Manuscript, 2010.

[3] A. Agrawal, P. N. 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. Preliminary version in *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing (STOC)*, 134–144, 1991.

[4] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, NJ, 1993.

[5] M. Andrews. Hardness of Buy-at-Bulk Network Design. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 115–124, 2004.

[6] M. Andrews, J. Chuzhoy, S. Khanna, and L. Zhang. Hardness of the Undirected Edge-Disjoint Paths Problem with Congestion. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 226–241, 2005.

[7] M. Andrews and L. Zhang. The Access Network Design Problem. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 40–49, 1998.

[8] M. Andrews and L. Zhang, 2009. Personal Communication.

[9] S. Antonakopoulos, C. Chekuri, B. Shepherd, and L. Zhang. Buy-at-Bulk Network Design with Protection. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 634–644, 2007.

[10] S. Antonakopoulos and L. Zhang. Heuristics for Fiber Installation in Optical Network Optimization. In *Proceedings of the 50th Annual IEEE Global Telecommunications Conference (GLOBECOM), 2007*, pages 2342–2347, 2007.

[11] E.M. Arkin, J.S.B. Mitchell, and G. Narasimhan. Resource-Constrained Geometric Network Optimization. In *Proceedings of the 14th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 307–316, 1998.

[12] S. Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems*. PhD thesis, University of California at Berkeley, 1994.

[13] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The Hardness of Approximate Optima in Lattices, Codes, and Systems of Linear Equations. *Journal of Computer and System Sciences*, 54(2):317–331, 1997. Preliminary version in *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 724–733, 1993.

[14] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn. A Polynomial-Time Approximation Scheme for Weighted Planar Graph TSP. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 33–41, 1998.

[15] S. Arora and G. Karakostas. A $(2 + \varepsilon)$-Approximation Algorithm for the k-MST Problem. *Mathematical Programming A*, 107(3):491–504, 2006. Preliminary version in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 754–759, 2000.

[16] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and the Hardness of Approximation Problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*,14–23,1992.

[17] A. Asadpour, M.X. Goemans, A. Madry, S.O. Gharan, and A. Saberi. An $O(\log n/\log\log n)$-Approximation Algorithm for the Asymmetric Traveling Salesman Problem. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[18] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties*. Springer Verlag, Berlin Heidelberg, 1999.

[19] B. Awerbuch and Y. Azar. Buy-at-bulk Network Design. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 542–547, 1997.

[20] 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. Preliminary version in *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC)*, 277–283, 1995.

[21] Y. Azar and O. Regev. Combinatorial Algorithms for the Unsplittable Flow Problem. *Algorithmica*, 44(1):49–66, 2006. Preliminary version in *Proceedings of the 8th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 15–29, 2001.

[22] Brenda S. Baker. Approximation Algorithms for NP-Complete Problems on Planar Graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994. Preliminary version in *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 265–273,1983.

[23] E. Balas. Facets of the Knapsack Polytope. *Mathematical Programming*, 8(1):146–164, 1975.

[24] E. Balas. The Prize Collecting Traveling Salesman Problem. *Networks*, 19(6):621–636, 1989.

[25] N. Bansal, A. Blum., S. Chawla, and A. Meyerson. Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 166–174. ACM New York, NY, USA, 2004.

[26] R. Bar-Yehuda, G. Even, and S. Shahar. On Approximating a Geometric Prize-Collecting Traveling Salesman Problem with Time Windows. *Journal of Algorithms*, 55(1):76–92, 2005. Preliminary version in *Proceedings of the 11th Annual European Symposium on Algorithms (ESA)*, 55–66, 2003.

[27] MH Bateni and J. Chuzhoy. Approximation Algorithms for the Directed k-Stroll and k-Tour Problems. In *Proceedings of the 13th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, page To appear, 2010.

[28] MH Bateni, MT Hajiaghayi, and D. Marx. Approximation Schemes for Steiner Forest on Planar Graphs and Graphs of Bounded Treewidth. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 211–220, 2010.

[29] MH Bateni, MT Hajiaghayi, and D. Marx. Prize-Collecting Network Design on Planar Graphs. Manuscript, 2010.

[30] P. Berman and C. Coulston. On-line Algorithms for Steiner Tree Problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 344–353. ACM, 1997.

[31] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation Algorithms for Orienteering and Discounted-Reward TSP. *SIAM Journal on Computing*, 37(2):653–670, 2007. Preliminary version in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 46–55, 2003.

[32] A. Blum, R. Ravi, and S. Vempala. A Constant-Factor Approximation Algorithm for the k-MST Problem. *Journal of Computer and System Sciences*, 58(1):101–108, 1999. Preliminary version in *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, 442–448, 1996.

[33] O.V. Borodin. Coupled Colorings of Graphs on a Plane. *Metody Diskret. Analiz* [In Russian], 45:21–27, 1987.

[34] G. Borradaile and P. Klein. The Two-Edge Connectivity Survivable Network Problem in Planar Graphs. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 485–501, 2008.

[35] G. Borradaile, P. Klein, and C. Mathieu. An $O(n \log n)$-Approximation Scheme for Steiner Tree in Planar Graphs. *ACM Transactions on Algorithms*, 5(3):1–31, 2009.

[36] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. An Improved LP-Based Approximation for Steiner Tree. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 583–592, 2010.

[37] R.D. Carr, L.K. Fleischer, V.J. Leung, and C.A. Phillips. Strengthening Integrality Gaps for Capacitated Network Design and Covering Problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 106–115, 2000.

[38] R.D. Carr and S. Vempala. Randomized Metarounding. *Random Structures and Algorithms*, 20(3):343–352, 2002. Preliminary version in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, 58–62, 2000.

[39] T. Chakraborty, J. Chuzhoy, and S. Khanna. Network Design for Vertex Connectivity. In *Proceedings of the 40th annual ACM Symposium on Theory of Computing (STOC)*, pages 167–176, 2008.

[40] M. Charikar and A. Karagiozova. On Non-Uniform Multicommodity Buy-at-Bulk Network Design. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 176–182, 2005.

[41] M. Charikar, J.S. Naor, and B. Schieber. Resource Optimization in QoS Multicast Routing of Real-Time Multimedia. *IEEE/ACM Transactions on Networking*, 12(2):340–348, 2004. Preliminary version in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 1518–1527, 2000.

[42] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, Trees, and Minimum Latency Tours. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 36–45. IEEE Computer Society, 2003.

[43] C. Chekuri, P. Claisse, R.J. Essiambre, S. Fortune, D.C. Kilper, W. Lee, N.K. Nithi, I. Saniee, B. Shepherd, C.A. White, et al. Design Tools for Transparent Optical Networks. *Bell Labs Technical Journal*, 11(2):129–143, 2006.

[44] C. Chekuri and A. Ene, 2009. Personal Communication.

[45] C. Chekuri, A. Ene, and N. Korula. Prize-Collecting Steiner Tree and Forest in Planar Graphs. Manuscript, 2010.

[46] C. Chekuri, G. Even, A. Gupta, and D. Segev. Set Connectivity Problems in Undirected Graphs and the Directed Steiner Network Problem. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 532–541, 2008.

[47] C. Chekuri, MT Hajiaghayi, G. Kortsarz, and M.R. Salavatipour. Approximation Algorithms for Node-Weighted Buy-at-Bulk Network Design. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1265–1274, 2007.

[48] C. Chekuri, MT Hajiaghayi, G. Kortsarz, and M.R. Salavatipour. Approximation Algorithms for Non-Uniform Buy-at-Bulk Network Design. *SIAM Journal on Computing*, 39(5):1772–1798, 2010. Preliminary version in *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 677–686, 2006.

[49] C. Chekuri, S. Khanna, and J.S. Naor. A Deterministic Algorithm for the Cost-Distance Problem. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 232–233, 2001.

[50] 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*, 2:137–146, 2006.

[51] C. Chekuri and N. Korula. Approximation Algorithms for Orienteering with Time Windows. arXiv.org preprint. `arXiv:0711.4825v1`.

[52] C. Chekuri and N. Korula. Single-Sink Network Design with Vertex Connectivity Requirements. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 131–142, 2008.

[53] C. Chekuri and N. Korula. A Graph Reduction Step Preserving Element-Connectivity and Applications. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, pages 254–265, 2009.

[54] C. Chekuri and N. Korula. Pruning 2-Connected Graphs. *Algorithmica*, to appear. Preliminary version in *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 119–130, 2008.

[55] C. Chekuri, N. Korula, and M. Pál. Improved Algorithms for Orienteering and Related Problems. *ACM Transactions on Algorithms*, to appear. Preliminary version in *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 661–670, 2008.

[56] C. Chekuri and A. Kumar. Maximum Coverage Problem with Group Budget Constraints and Applications. In *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 72–83. Springer, 2004.

[57] C. Chekuri and M. Pál. A recursive greedy algorithm for walks in directed graphs. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 245–253. IEEE Computer Society, 2005.

[58] C. Chekuri and M. Pál. An $O(\log n)$ Approximation for the Asymmetric Traveling Salesman Path Problem. *Theory of Computing*, 3:197–209, 2007. Preliminary version in *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 95–103, 2005.

[59] C. Chekuri and F.B. Shepherd. Approximate Integer Decompositions for Undirected Network Design Problems. *SIAM Journal on Discrete Mathematics*, 23(1):163–177, 2008.

[60] K. Chen and S. Har-Peled. The Orienteering Problem in the Plane Revisited. *SIAM Journal on Computing*, 38(1):385–397, 2008. Preliminary version in *Proceedings of the 22nd Annual ACM Symposium on Computational Geometry (SoCG)*, 247–254, 2006.

[61] J. Cheriyan, 2008. Personal Communication.

[62] J. Cheriyan and M.R. Salavatipour. Hardness and Approximation Results for Packing Steiner Trees. *Algorithmica*, 45(1):21–43, 2006. Preliminary version in *Proceedings of the 12th Annual European Symposium on Algorithms (ESA)*, 180–191, 2004.

[63] J. Cheriyan and M.R. Salavatipour. Packing Element-Disjoint Steiner Trees. *ACM Transactions on Algorithms*, 3(4), 2007. Preliminary version in *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 52-61, 2005.

[64] J. Cheriyan, S. Vempala, and A. Vetta. An Approximation Algorithm for the Minimum-Cost k-Vertex Connected Subgraph. *SIAM Journal on Computing*, 32(4):1050–1055, 2003. Preliminary version in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, 306–312, 2002.

[65] J. Cheriyan, S. Vempala, and A. Vetta. Network Design via Iterative Rounding of Setpair Relaxations. *Combinatorica*, 26(3):255–275, 2006.

[66] F.A. Chudak, T. Roughgarden, and D.P. Williamson. Approximate k-MSTs and k-Steiner Trees via the Primal-Dual Method and Lagrangean Relaxation. *Mathematical Programming*, 100(2):411–421, 2004. Preliminary version in *Proceedings of the 8th International Conference on Integer Programming and Combinatorial Optimization (IPCO)* , 60–70, 2001.

[67] J. Chuzhoy, A. Gupta, J.S. Naor, and A. Sinha. On the Approximability of Some Network Design Problems. *ACM Transactions on Algorithms*, 4(2):1–17, 2008. Preliminary version in *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 943–951, 2005.

[68] J. Chuzhoy and S. Khanna. Algorithms for single-source vertex connectivity. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 105–114, 2008.

[69] J. Chuzhoy and S. Khanna. An $O(k^3 \log n)$-Approximation Algorithm for Vertex-Connectivity Survivable Network Design. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 437–441, 2009.

[70] A. Cobham. The Intrinsic Computational Difficulty of Functions. In *Proceedings of the 1964 Congress for Logic, Methodology and Philosophy of Science*, pages 24–30. North-Holland Publishing Company, 1964.

[71] G. Călinescu, C. Chekuri, and J. Vondrák. Disjoint Bases in a Polymatroid. *Random Structures and Algorithms*, 35(4):418–430, 2009.

[72] E. Demaine, M.T. Hajiaghayi, and P. Klein. Node-Weighted Steiner Tree and Group Steiner Tree in Planar Graphs. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 328–340, 2009.

[73] D.S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1996.

[74] J. Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.

[75] J. Fakcharoenphol and B. Laekhanukit. An $O(\log^2 k)$-Approximation Algorithm for the k-Vertex Connected Spanning Subgraph Problem. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 153–158. ACM, 2008.

[76] U. Feige, M.M. Halldórsson, G. Kortsarz, and A. Srinivasan. Approximating the Domatic Number. *SIAM Journal on Computing*, 32(1):172–195, 2002. Preliminary version in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, 134–143, 2000.

[77] U. Feige, D. Peleg, and G. Kortsarz. The Dense k-Subgraph Problem. *Algorithmica*, 29(3):410–421, 2001. Preliminary version in *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 692–701, 1993.

[78] C.G. Fernandes. A Better Approximation Ratio for the Minimum k-Edge-Connected Spanning Subgraph Problem. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 629–638, 1997.

[79] L. Fleischer, K. Jain, and D.P. Williamson. Iterative Rounding 2-Approximation Algorithms for Minimum-Cost Vertex Connectivity Problems. *Journal of Computer and System Sciences*, 72(5):838–867, 2006. Preliminary versions in *Proceedings of the 8th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 115–129, 2001 and *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 339–347, 2001.

[80] A. Frank. On Connectivity Properties of Eulerian Digraphs. *Annals of Discrete Mathematics*, 41:179–194, 1989.

[81] A. Frank. Augmenting Graphs to Meet Edge-Connectivity Requirements. *SIAM Journal on Discrete Mathematics*, 5(1):25–53, 1992. Preliminary version in *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 708–718, 1990.

[82] A. Frank and T. Jordán. Minimal Edge-Covers of Pairs of Sets. *Journal of Combinatorial Theory, Series B*, 65(1):73–110, 1995.

[83] A. Frank, T. Király, and M. Kriesell. On Decomposing a Hypergraph into k Connected Sub-Hypergraphs. *Discrete Applied Mathematics*, 131(2):373–383, 2003.

[84] A. Frank and É. Tardos. An Application of Submodular Flows. *Linear Algebra and its Applications*, 114:329–348, 1989.

[85] G.N. Frederickson and B. Wittman. Approximation Algorithms for the Traveling Repairman and Speeding Deliveryman Problems with Unit-Time Windows. In *Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 119–133. Springer, 2007.

[86] A.M. Frieze, G. Galbiati, and F. Maffioli. On the Worst-Case Performance of Some Algorithms for the Asymmetric Traveling Salesman Problem. *Networks*, 12(1):23–39, 1982.

[87] N. Garg. A 3-Approximation for the Minimum Tree Spanning k Vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 302–309, 1996.

[88] N. Garg. Saving an Epsilon: a 2-Approximation for the k-MST Problem in Graphs. In *Proceedings of the 37th Annual ACM Symposium on Theory of computing (STOC)*, pages 396–402, 2005.

[89] S.O. Gharan and A. Saberi. The Asymmetric Traveling Salesman Problem on Graphs with Bounded Genus. arXiv.org preprint `arXiv:0909.2849`, 2009.

[90] M. X. Goemans and D. P. Williamson. A General Approximation Technique for Constrained Forest Problems. *SIAM Journal on Computing*, 24:296–317, 1995. Preliminary version in *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 307–316, 1992.

[91] M.X. Goemans, A.V. Goldberg, S. Plotkin, D.B. Shmoys, E. Tardos, and D.P. Williamson. Improved Approximation Algorithms for Network Design Problems. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 223–232, 1994.

[92] M.X. Goemans and D.P. Williamson. The Primal-Dual Method for Approximation Algorithms and its Application to Network Design Problems. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1996.

[93] B.L. Golden, L. Levy, and R. Vohra. The Orienteering Problem. *Naval Research Logistics*, 34(3):307–318, 1987.

[94] M. Grötschel, A. Martin, and R. Weismantel. The Steiner Tree Packing Problem in VLSI Design. *Mathematical Programming*, 78(2):265–281, 1997.

[95] S. Guha, A. Meyerson, and K. Munagala. A Constant Factor Approximation for the Single Sink Edge Installation Problem. *SIAM Journal on Computing*, 38(6):2426–2442, 2009. Preliminary version in *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, 383–388, 2001.

[96] S. Guha, A. Moss, J.S. Naor, and B. Schieber. Efficient Recovery from Power Outage. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 574–582, 1999.

[97] A. Gupta, R. Krishnaswamy, and R. Ravi. Online and Stochastic Survivable Network Design. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 685–694, 2009.

[98] A. Gupta, R. Krishnaswamy, and R. Ravi. Tree Embeddings for Two-Edge-Connected Network Design. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1521–1539, 2010.

[99] A. Gupta, A. Kumar, M. Pál, and T. Roughgarden. Approximation via Cost Sharing: Simpler and Better Approximation Algorithms for Network Design. *Journal of the ACM*, 54(3):11, 2007. Preliminary versions in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, 365–372, 2003, and *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 606–617, 2003.

[100] G. Gutin and A.P. Punnen, editors. *The Traveling Salesman Problem and its Variations*. Springer, Berlin, 2002.

[101] MT Hajiaghayi and K. Jain. The Prize-Collecting Generalized Steiner Tree Problem via a New Approach of Primal-Dual Schema. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 631–640, 2006.

[102] P.L. Hammer, E.L. Johnson, and U.N. Peled. Facets of Regular 0–1 Polytopes. *Mathematical Programming*, 8(1):179–206, 1975.

[103] J. Håstad. Some Optimal Inapproximability Results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001. Preliminary version in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, 1–10, 1997.

[104] M. Held and R.M. Karp. The Traveling-Salesman and Minimum Cost Spanning Trees. *Operations Research*, 18:1138–1162, 1970.

[105] H.R. Hind and O. Oellermann. Menger-Type Results for Three or More Vertices. *Congressus Numerantium*, pages 179–204, 1996.

[106] D.S. Hochbaum and A. Segev. Analysis of a Flow Problem with Fixed Charges. *Networks*, 19(3):291–312, 1989.

[107] J. Ivanco. The Weight of a Graph. *Annals of Discrete Mathematics*, pages 113–116, 1992.

[108] B. Jackson. Some Remarks on Arc-Connectivity, Vertex Splitting, and Orientation in Graphs and Digraphs. *Journal of Graph Theory*, 12(3):429–436, 2006.

[109] K. Jain. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem. *Combinatorica*, 21(1):39–60, 2001. Preliminary version in *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 448–457, 1998.

[110] K. Jain, M. Mahdian, and M. Salavatipour. Packing Steiner Trees. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 266–274, 2003.

[111] K. Jain, I. Măndoiu, V.V. Vazirani, and D.P. Williamson. A Primal-Dual Schema Based Approximation Algorithm for the Element Connectivity Problem. In *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 484–489, 1999.

[112] D.S. Johnson, M. Minkoff, and S. Phillips. The Prize Collecting Steiner Tree Problem: Theory and Practice. In *Proceedings of the 11th annual ACM-SIAM Symposium on Discrete Algorithms (SODA(*, pages 760–769, 2000.

[113] D. Karger. *Random Sampling in Graph Optimization Problems.* PhD thesis, Stanford University, 1994.

[114] T. Király and L.C. Lau. Approximate Min-Max Theorems for Steiner Rooted-Orientations of Graphs and Hypergraphs. *Journal of Combinatorial Theory, Series B*, 98(6):1233–1252, 2008. Preliminary version in *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 283–292, 2006.

[115] P. N. Klein. A Subset Spanner for Planar Graphs: With Application to Subset TSP. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 749–756. ACM, 2006.

[116] J. Kleinberg and D.P. Williamson, 1998. Unpublished Note.

[117] J.M. Kleinberg. *Approximation Algorithms for Disjoint Paths Problems.* PhD thesis, Massachusetts Institute of Technology, 1996.

[118] B.H. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer Verlag, Berlin, 3rd edition, 2008.

[119] G. Kortsarz, R. Krauthgamer, and J.R. Lee. Hardness of Approximation for Vertex-Connectivity Network Design Problems. *SIAM Journal on Computing*, 33(3):704–720, 2004. Preliminary version in *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 185–199, 2002.

[120] G. Kortsarz and Z. Nutov. Approximating k-node Connected Subgraphs via Critical Graphs. *SIAM Journal on Computing*, 35(1):247–257, 2005. Preliminary version in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, 138–145, 2004.

[121] G. Kortsarz and Z. Nutov. Approximating Minimum Cost Connectivity Problems. In T.F. Gonzalez, editor, *Handbook of Approximation algorithms and Metaheuristics*. CRC Press, 2007.

[122] G. Kortsarz and Z. Nutov. Approximating Some Network Design Problems with Node Costs. In *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 231–243. Springer, 2009.

[123] M. Kriesell. Edge-Disjoint Trees Containing Some Given Vertices in a Graph. *Journal of Combinatorial Theory, Series B*, 88(1):53–65, 2003.

[124] L.C. Lau. Packing Steiner Forests. In *Proceedings of the 11th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 362–376, 2005.

[125] L.C. Lau. An Approximate Max-Steiner-Tree-Packing Min-Steiner-Cut Theorem. *Combinatorica*, 27(1):71–90, 2007. Preliminary version in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 61–70, 2004.

[126] L.C. Lau, J.S. Naor, M.R. Salavatipour, and M. Singh. Survivable Network Design with Degree or Order Constraints. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 651–660, 2007.

[127] L.C. Lau, J.S. Naor, M.R. Salavatipour, and M. Singh. Survivable Network Design with Degree or Order Constraints. *SIAM Journal on Computing*, 39(3):1062–1087, 2009.

[128] E.L. Lawler, A.H.G. Rinnooy Kan, J.K. Lenstra, and D.B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons Inc, 1985.

[129] L. Lovász. On Some Connectivity Properties of Eulerian Graphs. *Acta Mathematica Hungarica*, 28(1):129–138, 1976.

[130] W. Mader. A Reduction Method for Edge-Connectivity in Graphs. *Annals of Discrete Mathematics*, 3:145–164, 1978.

[131] T.L. Magnanti, P. Mirchandani, and R. Vachani. Modeling and Solving the Two-Facility Capacitated Network Loading Problem. *Operations Research*, 43(1):142–157, 1995.

[132] Karl Menger. Zur Allgemeinen Kurventheorie. *Fundamenta Mathematicae* [In German], 10:96–115, 1927.

[133] A. Meyerson, K. Munagala, and S. Plotkin. Cost-Distance: Two Metric Network Design. *SIAM Journal on Computing*, 38(4):1648–1659, 2008. Preliminary version in *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 383–388, 2000.

[134] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[135] V. Nagarajan and R. Ravi. Poly-logarithmic Approximation Algorithms for Directed Vehicle Routing Problems. In *Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 257–270, 2007.

[136] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*, chapter Section II.6.4. Wiley-Interscience Series In Discrete Mathematics And Optimization. John Wiley and Sons, 1988.

[137] Z. Nutov. A Note on Rooted Survivable Networks. *Information Processing Letters*, 109(19):1114–1119, 2009.

[138] Z. Nutov. An Almost $O(\log k)$-Approximation for k-Connected Subgraphs. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 912–921, 2009.

[139] Z. Nutov. Approximating Minimum Cost Connectivity Problems via Uncrossable Bifamilies and Spider-Cover Decompositions. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 417–426, 2009.

[140] F. Ortega and L.A. Wolsey. A Branch-and-Cut Algorithm for the Single-Commodity, Uncapacitated, Fixed-Charge Network Flow Problem. *Networks*, 41(3):143–158, 2003.

[141] R. Ramaswami and K.N. Sivarajan. *Optical Networks: A Practical Perspective*. Morgan Kaufmann, 2nd edition, 2002.

[142] 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. Preliminary version in *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 546–555, 1994.

[143] R. Raz. A Parallel Repetition Theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998. Preliminary version in *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC)*, 447–456, 1995.

[144] N. Robertson and PD Seymour. Graph Minors:: XVII. Taming a Vortex. *Journal of Combinatorial Theory, Series B*, 77(1):162–210, 1999.

[145] M.A. Safari and M. Salavatipour. A Constant Factor Approximation for Minimum λ-Edge-Connected k-Subgraph with Metric Costs. In *Proceedings of the 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 233–246, 2008.

[146] FS Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Approximating the Single-Sink Link-Installation Problem in Network Design. *SIAM Journal on Optimization*, 11(3):595–610, 2001.

[147] P.D. Seymour. Nowhere-Zero 6-flows. *Journal of Combinatorial Theory, Series B*, 30(2):130–135, 1981.

[148] T.E. Stern and K. Bala. *Multiwavelength Optical Networks: A Layered Approach*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[149] K. Talwar. The Single-Sink Buy-at-Bulk LP Has Constant Integrality Gap. In *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 475–486, 2002.

[150] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial Mathematics, Philadelphia PA, 2001.

[151] J.N. Tsitsiklis. Special Cases of Traveling Salesman and Repairman Problems with Time Windows. *Networks*, 22(3):263–282, 1992.

[152] V.V. Vazirani. *Approximation Algorithms*. Springer Verlag, Berlin, 2001.

[153] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 2001.

[154] D.P. Williamson, M.X. Goemans, M. Mihail, and V.V. Vazirani. A Primal-Dual Approximation Algorithm for Generalized Steiner Network Problems. *Combinatorica*, 15(3):435–454, 1995. Preliminary version in *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, 708–717, 1993.

[155] L.A. Wolsey. Faces for a Linear Inequality in 0–1 Variables. *Mathematical Programming*, 8(1):165–178, 1975.

[156] Hehui Wu and Douglas B. West. Packing of Steiner Trees and S-connectors in Graphs. Unpublished manuscript, 2010.