Approximation Algorithms for Network Interdiction and Fortification Problems

Adam Whitley
Clemson University, whitle4@gmail.com

Follow this and additional works at: https://tigerprints.clemson.edu/all_dissertations
Part of the Computer Sciences Commons

Recommended Citation
https://tigerprints.clemson.edu/all_dissertations/646

This Dissertation is brought to you for free and open access by the Dissertations at TigerPrints. It has been accepted for inclusion in All Dissertations by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.
APPROXIMATION ALGORITHMS FOR NETWORK INTERDICATION AND FORTIFICATION PROBLEMS

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

by
Adam Aleron Whitley
December 2010

Accepted by:
Brian C. Dean, Ph.D., Committee Chair
Wayne Goddard, Ph.D.
David Jacobs, Ph.D.
Pradip Srimani, Ph.D.
Abstract

This dissertation discusses algorithms and results on several NP-hard graph problems which can all be classified as network interdiction and network fortification problems.

The first problem studied, the multiway cut problem, is a generalization of the well-studied s-t min-cut problem, in which we must remove a minimum-cost subset of edges from a graph so that $r > 2$ designated terminals become disconnected from each other. We investigate an approximation algorithm for general $r$ with a relatively simple analysis guaranteeing an approximation ratio $\leq 1.4647 - \varepsilon_r$, where $\varepsilon_r$ is a small constant related to the number of terminals $r$. This improves on the $1.5 - \frac{1}{r}$ guarantee of Călinescu et al. [9] and is somewhat simpler than the $1.3438 - \varepsilon_r$ result of Karger et al. [51]. We also perform three types of computational experiments to obtain empirical results and offer observations for the $r = 4$ case, based on small discretized instances.

Next, we introduce a generalization of the multiway cut problem, called the $k$-hurdle multiway cut problem, in which every terminal-terminal path must be cut not merely once, but $k > 1$ times. We present a half-integrality proof implying a 2-approximation to the problem, a $(1, k - 1)$-pseudo approximation result, and also a true approximation algorithm with performance guarantee $2(1 - \frac{1}{k})$. This guarantee is unlikely to be improved upon, as we demonstrate an approximation-preserving reduction from the well-known vertex cover problem.

A related problem we also study is the $k$-hurdle multicut problem, where we have a list of source-sink pairs $(i, j)$, and each source must be separated from its sink $k_{ij}$ times. We present a $(\log(n), [(1 - \varepsilon)k_{\text{max}}])$-pseudo approximation algorithm, with approximation guarantee $O(\log(n))$ that guarantees all hurdles will be satisfied if all $k_{ij} = O(1)$. We also obtain a 2-approximation for trees, which holds in both the standard $k$-hurdle multicut problem and a vertex variant, and we consider a hybrid problem of $k$-hurdle multiway cut and $k$-hurdle multicut, in which different hurdle counts $k_{ij}$ exist between each pair of terminals $(i, j)$, and all form an ultrametric. We demonstrate
the half-integrality of this problem, implying a 2-approximation.

Finally, we consider a problem called the network knapsack problem, which is a special case of a packing integer program where the underlying constraint system has a well-defined graph structure. We use dynamic programming to obtain a polynomial time solution on ladders of width \( k = O(1) \), and a polynomial time approximation scheme for trees and grids. We also investigate the difficulty of developing approximation algorithms for this problem on low-treewidth graphs and planar graphs.
Dedication

This work is dedicated to my mother, Erika. She has done far more for me than I can ever express, and no son ever had a better mother. I love you, mom.
I have many people to thank for helping make this dissertation possible.

First and foremost, I want to thank the best advisor in the world, Dr. Brian Dean. Without his infinite knowledge, patience, guidance, and friendship, this dissertation would never have become what it is today, or even existed at all, for that matter. There’s no one I respect more than Dr. Dean, and I am extremely fortunate to have him as my advisor. I’d like to acknowledge the rest of my committee as well, and thank them for serving: Dr. Wayne Goddard, Dr. David Jacobs, and Dr. Pradip Srimani.

I would be remiss if I did not acknowledge Dr. Ojas Parekh, whose collaboration was absolutely invaluable. Ben Cousins and Adam Griffis provided a great deal of collaborative effort and help, as well. I also want to thank the entire Applied Algorithms Group for their help and support, especially during the final weeks before my submission deadline.

I can’t forget to mention three amazing friends of mine: Thomas Grindinger, Chris Mitchell, and Joshua Hetherington. You guys are absolutely the best, and I can’t thank you enough. Without your friendship and good advice all these years, I would never have finished this journey. That’s a fact.

Finally, I want to thank my godfather Will Moore and my undergraduate advisor Dr. Dee Parks. You are family, and without you I would never have even started this journey.
# Table of Contents

Title Page ................................................................. i
Abstract ................................................................. ii
Dedication ............................................................... iv
Acknowledgments ......................................................... v
List of Tables ............................................................. viii
List of Figures ............................................................ ix

1 Introduction ........................................................... 1
  1.1 Multiway Cut ...................................................... 2
  1.2 $k$-hurdle Multiway Cut .......................................... 4
  1.3 $k$-hurdle Multicut ................................................. 5
  1.4 Network Knapsack .................................................. 6
  1.5 Dissertation Organization ......................................... 9

2 Related Work .......................................................... 10
  2.1 Network Fortification .............................................. 10
  2.2 Network Interdiction .............................................. 12
  2.3 Cut Problems ...................................................... 13
  2.4 Knapsacks and Alliances ......................................... 15

3 Multiway Cut .......................................................... 17
  3.1 Algorithm .......................................................... 17
  3.2 Algorithm Analysis ............................................... 20

4 Multiway Cut Computational Experiments ............................. 26
  4.1 Motivation and Background ........................................ 26
  4.2 Experimental Avenues ............................................. 32

5 $k$-hurdle Multiway Cut .............................................. 47

6 $k$-hurdle Multicut .................................................... 52
  6.1 A 2-Approximation for Trees ..................................... 54
  6.2 $k$-Hurdle Vertex Multicut in Trees .............................. 57
  6.3 The Ultrametric Case ............................................. 57

7 Network Knapsack .................................................... 61
  7.1 PTAS for Trees .................................................... 61
8 Conclusion and Open Problems ........................................... 68

Bibliography ................................................................. 71
List of Tables

3.1 This table lists the approximation ratios $d^*$ of our multiway cut algorithm for various values of $r$, the number of terminals. For each value of $r$, this table lists the minimum approximation ratio $d^*$ achievable by setting $\alpha$ and $\beta$ to the listed optimal values. .................................. 25

4.1 4-terminal bad graphs for 3, 4, and 5 discretizations, listing the weight of each canonical edge. We represent each edge in the table with the coordinates of its adjacent vertices. The $i$th coordinate of a vertex is its distance from the face opposite terminal $i$. These edge weights are calculated using our row generation technique. The edge weights for the $n = 5$ case do not seem to have the clear proportions of the $n = 3$ and $n = 4$ edge weights. See Figure 4.4 for an illustration of $n = 3$. ................................................................. 35
List of Figures

3.1 Figure 3.1(a) at the top left depicts a corner cut in a 3-simplex. The dashed lines depict the boundary between the central ball region of width $\beta$ and the corner regions of width $1 - \beta$. Figure 3.1(b) on the top right depicts a ball cut, consisting of two slices intersecting at point $c$. The first (slanted) slice cuts across the entire simplex, but the second (horizontal) slice only cuts the region not yet captured by the first slice. Figure 3.1(c) at the bottom left shows a 1-2 aligned edge $(u, v)$ as well as strip 1 and strip 2; only a ball cut with $c$ in the shaded region can cut edge $(u, v)$. Figure 3.1(d) at the bottom right depicts a helpful ball cut with center point $c$; the 1-2 aligned edge $(u, v)$ is safe due to the helpful slice from terminal 1.

4.1 A bad graph for the 3-simplex with $n = 96$ discretizations. One terminal (at the top corner of the simplex) is shown much closer to our viewpoint than the other two terminals, at the bottom left and right corners. Edge weight from the LP solution is proportional to the thickness of the edge in the figure. The edges closest to a particular terminal are tinted a unique color (gold, purple, or green) for clarity. Observe that edges of high weight occur on the outer boundary of the simplex, and that uniformly weighted edges fill a hexagonal region in the center of the simplex.

4.2 An example of a discretized 4-simplex, depicting a (non-SPARC) multiway cut. Edges included in the cut are outlined in black, and each component of the result is depicted with a unique color. For clarity, only edges lying on the front faces of the simplex are shown, but the back faces and interior are also filled with the triangular mesh. One outer face of this 4-simplex is, itself, a 3-simplex.

4.3 In Figure 4.3(a) at the top, we see an illustration of canonical edges on the 3-simplex with 4 discretizations. Each simplex has emphasized all copies of a unique canonical edge, displayed more thickly than other simplex edges. Figure 4.3(b) at the bottom is an illustration of canonical vertices in the same simplex. Each simplex has emphasized all copies of a single canonical vertex, displayed larger than the other simplex vertices.

4.4 An optimal bad graph for the 4-simplex with 3 discretizations, found with the row generation technique. We have displayed only one edge of each canonical type, labeled with its weight for clarity. Each vertex's visual size is inversely proportional to our viewing distance, so vertices in the back appear smaller than in the front. See Figure 4.1 for a tabular list of the edge weights of this bad graph.

4.5 Figure 4.5(a) on the left shows a 3-simplex path of height 3, consisting of three segments of three edges each. The optimal 3-terminal bad graph from [51] is composed entirely of paths of this form with varying heights. Figure 4.5(b) depicts a 4-simplex path consisting of five segments. The weight assigned to this path is one variable of an instance of LP9.
4.6 Figure 4.6(a) is a visualization of a solved $r = 3$ instance of LP10: a raw cutting scheme consisting of a distribution of weight on each SPARC cut of a 3-simplex with 72 discretizations. Slice distance from the two terminals form the $x$ and $y$ axes; we use whole number slice distances from 1 to $n$ for simplicity. The darkness of each pixel $(x, y)$ is directly related to the weight (or probability of selecting) the SPARC with corresponding slice distances $x$ and $y$. There is significant degeneracy present. Figure 4.6(b) is a raw cutting scheme output for the same instance, with the added constraints that all corner cuts must have equal weight and all cuts that are neither a ball cut nor a corner cut must have 0 weight. Figure 4.6(c) adds the additional constraint of symmetry about the $x = y$ axis in the ball region. Figure 4.6(d) adds another constraint, enforcing symmetry about the $x + y = n^2/3 - 1$ diagonal, which is perpendicular to the $x = y$ axis in the ball region. Notice the degeneracy is significantly less than Figure 4.6(a), yet the resulting distribution still provides only a fleeting glimpse of the structure of the true optimal analytical cutting scheme shown in Figure 4.7.

4.7 An illustration of the $\frac{n^2}{4}$ cutting scheme of Karger et al. [51] in the same format as Figure 4.6. Their algorithm is equivalent to the cut distribution depicted here. The bottom right corner containing solid black corresponds to their corner cut, and is equivalent to the diagonal line in the bottom right corner of Figure 4.6(b), due to a valid speculative constraint. The black star shape on the top left corresponds to their ball cut.

4.8 This figure depicts a raw cutting scheme output from an instance of LP10 with $r = 4$ and $n = 36$. At the top left corner, we see a three-dimensional view, where each point $(x, y, z)$ is gray if a SPARC with slice distances $x, y, z$ has probability $> 0$ in the solution. In the top right, bottom left, and bottom right corners, we see the three two-dimensional marginals.

4.9 This figure depicts a raw cutting scheme output from an instance of LP10 with $r = 4$ and $n = 36$, where we also enforce the speculative constraint that the only corner cuts with nonzero weight are those with all slices at the same distance. At the top left corner, we see a three-dimensional view, where each point $(x, y, z)$ is gray if a SPARC with slice distances $x, y, z$ has probability $> 0$ in the solution. In the top right, bottom left, and bottom right corners, we see the three two-dimensional marginals.

4.10 This figure depicts a raw cutting scheme output from an instance of LP10 with $r = 4$ and $n = 36$, where we enforce two speculative constraints: 1) that the only corner cuts with nonzero weight are those with all slices at the same distance, and 2) an axis of symmetry about the $x = y$ axis in the ball region. At the top left corner, we see a three-dimensional view, where each point $(x, y, z)$ is gray if a SPARC with slice distances $x, y, z$ has probability $> 0$ in the solution. In the top right, bottom left, and bottom right corners, we see the three two-dimensional marginals.
Chapter 1

Introduction

This dissertation discusses several problems which all fall into one of two categories. First, we have network fortification problems (also called network upgrading, network improvement, network augmentation, and network reinforcement). Second, we have network interdiction problems (also called network inhibition, network obstruction, and network degrading). Together, they are a broad field, applicable in many areas of practice, as the graphs in question may represent communication, transportation, utility, surveillance, military, or other types of real-world graph-representable networks. In these types of problems, we desire to optimally reinforce a network (in a network fortification problem) or impede a network (in a network interdiction problem), with respect to criteria such as shortest path distances, connectivity, or other measures. We either optimize the reinforcement or impediment of the graph given a budget constraint, or we optimize the cheapest way to achieve a specific amount of effect.

In this dissertation, we discuss several specific network interdiction problems related to the well-known s-t min-cut problem; these problems allow us to analyze the cost of eliminating or reducing network connectivity between special vertices called terminals. We also discuss a network fortification problem, which is a network-related generalization of the well-studied 0-1 knapsack problem, which considers simultaneously satisfying knapsack packing constraints on each vertex of a graph.

As mentioned, many of the results in this dissertation are regarding generalizations of the s-t min-cut problem, popularized by Ford and Fulkerson [26] in the 1960’s. In this problem, we are given a graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges with costs $c : E \to \mathbb{R}^+$, and
two special vertices \( s, t \in V \) called the source, \( s \), and the sink, \( t \). We seek to separate \( s \) and \( t \) into different components of \( G \) by selecting a minimum cost set of edges \( S \subseteq E \) called the cut set. It is easily solvable in polynomial time by exploiting its well-studied strong relationship with its dual problem, the \( s-t \) max-flow problem, in which we wish to maximize the amount of flow we can send from \( s \) to \( t \) while respecting capacity constraints on edges.

Some parts of this dissertation pertaining to various \( k \)-hurdle problems, including primarily results on the \( k \)-hurdle multiway cut problem and the \( k \)-hurdle multicut problem, are reproduced from the author’s work in [18] and [17].

### 1.1 Multiway Cut

The first problem discussed in this dissertation is the multiway cut problem (also called the multiterminal cut problem) which is similar to the \( s-t \) min-cut problem, except instead of \( s \) and \( t \), we are given a set \( T \subseteq V \) containing \( r = |T| \geq 2 \) special vertices called terminals. These terminals function similarly to \( s \) and \( t \) earlier, as we must remove a minimum cost set of edges from \( G \) such that no two elements of \( T \) end up in the same connected component. Alternately, this can be viewed as cutting every terminal-terminal path at least once. The multiway cut problem is formulated by the following integer program,

\[
\begin{align*}
\text{OPT} = \text{Minimize} & \quad \sum_{e \in E} x(e)c(e) \\
\text{Subject to} & \quad \sum_{e \in p} x(e) \geq 1 \quad \forall p \in P \\
& \quad x(e) \in \{0, 1\} \quad \forall e \in E,
\end{align*}
\]

where \( x(e) = 1 \) iff edge \( e \in E \) is cut and 0 otherwise, and where \( P \) denotes the set of all terminal-terminal paths. As with most of the integer programs presented in this dissertation, the standard linear relaxation for IP1 (which we call LP1) is similar, but allows \( x(e) \in [0, 1] \) to be continuous in its range. The multiway cut problem is \( APX \)-hard for \( r \geq 3 \) terminals, meaning that there exists some specific constant factor beyond which it is NP-hard to approximate this problem. However, its precise value is unknown for general \( r \), though any proven approximation ratio provides an upper bound.

It is simple to achieve an approximation ratio of \( 2 - \frac{2}{r} \) with an isolating cut heuristic, first attributable to Dahlhaus et al. in 1994 [15], whereby we union together \( r - 1 \) two-terminal \( s-t \) cuts,
which separate one designated "source" terminal $s$ from the other $r - 1$ terminals, thereby "isolating" $s$. A performance bound of 2 is also achievable via LP rounding, since the natural LP relaxation of the multiway cut problem is known to be half-integral, which means we can construct an optimal solution to LP1 where each variable $x(e) \in \{0, \frac{1}{2}, 1\}$. Rounding all $x(e)$ variables of value $\frac{1}{2}$ up to 1 will at most double the solution value, giving a 2-approximation. This half-integrality also carries over to the $k$-hurdle generalization, discussed later.

The multiway cut problem is well-studied for $r = 3$; in this case a strengthened version of the above multiway cut LP relaxation (LP1) is known, meaning that it has a smaller integrality gap than LP1. An integrality gap represents a fixed lower bound on any possible approximation ratio using a particular LP relaxation. It has been proven ([12, 51]) that the integrality gap of this strengthened LP is $\frac{12}{11}$, and an optimal approximation algorithm with the same ratio of $\frac{12}{11}$ is also known [12, 51]. This $\frac{12}{11}$ bound is therefore considered tight. In this dissertation, we consider the general case with $r$ terminals, where a tight integrality gap is unknown for all values of $r \geq 4$.

We present an approximation algorithm for general $r$ with a simple analysis guaranteeing an approximation ratio $\leq 1.4647 - \varepsilon_r$, where $r$ is a small constant related to the number of terminals $r$. This is better than the $1.5 - \frac{1}{r}$ approximation guarantee from Călinescu et al. [9], and less complicated than the $1.3438 - \varepsilon_r$ result of Karger et al. [51]. Although the approximation bound we achieve is not as strong as in [51], we believe the analysis of our algorithm is simpler and potentially interesting in its own right.

Many of the algorithms developed for the multiway cut problem, in this dissertation as well as by other authors, depend on the notion of a simplex embedding, first developed by Călinescu et al. in 2000 [9]. Călinescu et al. prove any input graph $G$ can be embedded into a simplex (a multi-dimensional “triangle” defined precisely in Chapter 3), and that the simplex embedding of $G$ is equivalent to the strengthened version of the multiway cut LP (LP1) discussed earlier. Thus, by using a simplex embedding, we gain a stronger LP while also taking advantage of the embedding’s added geometric structure.

In this dissertation, we also perform computational experiments to obtain empirical results for the multiway cut problem on small discretized versions of the 4-simplex. Further details on these graphs and surrounding concepts are unrelated to other results in this dissertation, so we introduce them in Chapter 4, which is devoted to these experiments.
1.2 $k$-hurdle Multiway Cut

A different way to generalize the $s$-$t$ min-cut problem is to require multiple layers of cuts between the two terminals $s$ and $t$. This gives us the $k$-hurdle cut problem (also called the minimum $k$-cut problem), where the objective is to choose a minimum-cost subset of the edges of the graph $G$ that cuts every $s$-$t$ path at least $k \geq 1$ times. This problem can model a situation where we wish to install multiple layers of checkpoints for goods flowing between a set of important facilities in a network.

We can write the $k$-hurdle cut problem as the following integer program,

$$
\text{OPT} = \text{Minimize} \quad \sum_{e \in E} x(e)c(e)
$$

Subject to

$$
\sum_{e \in P} x(e) \geq k \quad \forall p \in P_{st}
$$

$$
x(e) \in \{0, 1\} \quad \forall e \in E,
$$

where $P_{st}$ denotes the set of all $s$-$t$ paths, each of which we assume has length at least $k$ edges or else there is no feasible solution. Since the $k$-hurdle cut problem is polynomial-time solvable [36], we will focus on NP-hard generalizations of this problem, such as a combination of the $k$-hurdle cut problem with the multiway cut problem, called the $k$-hurdle multiway cut problem. The $k$-hurdle multiway cut problem takes as input a set of $r \geq 2$ terminals $t_1 \ldots t_r$ and asks us to compute a minimum-cost subset of edges that cuts every terminal-to-terminal path at least $k \geq 1$ times. It can be formulated as the following integer program,

$$
\text{OPT} = \text{Minimize} \quad \sum_{e \in E} x(e)c(e)
$$

(IP2) Subject to

$$
\sum_{e \in P} x(e) \geq k \quad \forall p \in P
$$

$$
x(e) \in \{0, 1\} \quad \forall e \in E.
$$

Since the multiway cut problem is the special case where $k = 1$, the $k$-hurdle multiway cut problem is similarly APX-hard for $r \geq 3$ just as is the multiway cut problem.

We provide two approximation results for this problem. The first is a pseudo approximation algorithm that outputs a multiway cut with at least $k - 1$ hurdles for $k \geq 2$, whose cost is no larger than the optimal cost of a $k$-hurdle multiway cut, and the other is a true approximation algorithm with guarantee $2(1 - \frac{1}{r})$. Our approach is based on half-integrality, where we simplify and extend the work of Garg et al. [31] to the $k$-hurdle case. As opposed to the classical ($k = 1$) multiway cut problem, where stronger approximation ratios are possible, we show via an approximation-preserving
reduction from vertex cover that it may be difficult to improve on our $2(1 - \frac{1}{r})$ ratio for the $k$-hurdle variant.

1.3 $k$-hurdle Multicut

Another generalization of the $k$-hurdle cut problem, the $k$-hurdle multicut problem, takes as input $r$ terminal pairs $(s_1, t_1) \ldots (s_r, t_r)$ called commodities and asks us to select a minimum-cost subset of edges that cuts each $s_i$-$t_i$ path at least $k_i$ times, where the hurdle count $k_i$ can now vary by commodity. It can be formulated as the following integer program,

$$OPT = \text{Minimize } \sum_{e \in E} x(e)c(e)$$

(IP3) Subject to $\sum_{e \in p} x(e) \geq k_i \forall p \in P_i, \forall i$

$x(e) \in \{0, 1\} \forall e \in E,$

where $P_i$ is the set of all paths connecting the source-sink terminal pair $(s_i, t_i), 1 \leq i \leq r$.

The $k$-hurdle multicut problem seems somewhat more difficult to approximate than $k$-hurdle multiway cut if we wish to find a solution containing all of the required hurdles. For any constant $\varepsilon > 0$, we show how to compute a solution that provides a $1 - \varepsilon$ fraction of the required hurdles (rounded up) for each commodity, whose cost is at most $O(\log n)$ times that of an optimal solution containing all hurdles ($\varepsilon$ appears as a hidden, constant multiplier to the runtime). If $k_i = O(1)$ for every commodity $i$, we therefore obtain an $O(\log n)$-approximation. For the special case of $k$-hurdle multicut in a tree, we obtain a 2-approximation algorithm; this algorithm additionally applies with little alteration to the vertex version of the problem, which we call the vertex $k$-hurdle multicut problem. In this problem, we seek to remove a set $T \in V$ of vertices instead of edges, which also implies the removal of all edges incident to vertices in $T$.

This dissertation also considers a hybrid problem of $k$-hurdle multiway cut and $k$-hurdle multicut, in which different hurdle counts $k_{ij}$ exist between each pair of terminals $(i, j)$, and all form an ultrametric, where for every triple of terminals $x, y,$ and $z$, the inequality $k_{xz} \leq \max(k_{xy}, k_{yz})$ holds. Given this restriction, it is not as general a problem as the standard multicut problem, which is difficult to approximate well. However, this restricted problem is still more general than the $k$-hurdle multiway cut problem (where all $k_{ij}$ are equal to a single value $k$), yet we still obtain a 2-approximation via half-integrality.
1.4 Network Knapsack

In order to motivate our study of what we call “network knapsack” problems in this dissertation, let us first consider the general problem of covering integer programming (CIP). We are given \( m \) elements with a rational coverage vector \( b \in [1, \infty]^m \) describing how many times each element must be covered. To do so, we are given a collection of \( n \) sets of elements to choose from with a rational cost vector \( c \in [0, 1]^n \). We also have a rational \( m \times n \) constraint matrix \( A \), where \( A_{ij} \) denotes the multiplicity of element \( i \) in set \( j \), and we wish to cover all elements to the required extent at minimum cost. More formally, covering integer programming is stated as

\[
\text{OPT} = \text{Minimize} \quad c^T x
\]

(IP4) \quad \text{Subject to} \quad Ax \geq b
\]

\[ x \in \{0,1\}^n. \]

The problem of packing integer programming (PIP) is similar. Instead of a cost vector \( c \) we have a value vector \( v \), the \( b \) vector now describes maximum coverage restrictions, and we wish to obtain maximum value from the selected sets. Formally, the problem is stated as

\[
\text{OPT} = \text{Maximize} \quad v^T x
\]

(IP5) \quad \text{Subject to} \quad Ax \leq b
\]

\[ x \in \{0,1\}^n. \]

These problems are notoriously difficult to approximate well, even in restricted cases. For example, if we restrict \( b = [1]^m \) and restrict \( A \) to a binary matrix, we obtain the well-known minimum cost set cover and maximum value set packing problems. Minimum cost set cover is known to be inapproximable within any factor better than \( (1 - \varepsilon) \ln(n) \) for any \( \varepsilon > 0 \), unless \( \text{NP} \subset \text{DTIME}(n^{\log \log n}) \) [25]. Maximum value set packing is even more difficult to approximate; it is known to be inapproximable to within any factor better than \( m^{\frac{3}{2} - \varepsilon} \) for any \( \varepsilon > 0 \), unless \( \text{NP} = \text{ZPP} \) [43] (their proof also applies to the maximum clique problem). The best known approximation ratios for minimum cost set cover [49] and for maximum value set packing [40] closely match their respective inapproximability bounds, and both use greedy algorithms where only locally optimal choices are made. Approximation results for the general PIP and CIP problems in the literature are somewhat weaker [78].

If the non-zero structure of the constraint matrix \( A \) corresponds to the adjacency matrix of a graph \( G(A) \) of a special type, however (such as a tree, a grid, or a planar graph), then it may
be possible to achieve much stronger approximation results. This is the motivation for the network knapsack problem introduced in this dissertation, which can be interpreted as a special case of the PIP or CIP problems where the graph \( G(A) \) underlying our system of linear constraints has special structure.

Many other problems are much easier to solve on many of the graph types we consider for network knapsack, such as trees, grids, low-treewidth graphs\(^1\), and planar graphs. For example, the multicut problem mentioned earlier in this dissertation admits a 2-approximation for trees, yet the best known approximation ratio for general graphs is \( O(\log r) \) [29]. Another example that motivates our study of the network knapsack problem is the similar yet much simpler task of solving a linear system of equations of the form \( Ax = b \), where \( A \)'s non-zero elements represent the adjacency matrix of a graph \( G(A) \). We can use standard Gaussian elimination requiring \( O(n^3) \) time, or even the Coppersmith-Winograd algorithm requiring \( O(n^{2.376}) \) time [13]. However, in the special case that \( G(A) \) is a grid or a planar graph, we can perform Gaussian elimination in \( O(n^{1.5}) \) time via “nested dissection” [59]. If \( G(A) \) instead has treewidth \( k \), we can solve the system in only \( O(k^2 n) \) time [71], which is linear for all \( k = O(1) \).

We can also arrive at the network knapsack problem by generalizing the classical 0-1 knapsack problem. In the 0-1 knapsack problem, we are given a set \( S \) of \( n \) items \( i = 1 \ldots n \), each with a value denoted by \( \text{value}(i) \) and a size denoted by \( \text{size}(i) \). We also know a size capacity \( C \) which we cannot exceed; by appropriately scaling each item’s size by \( 1/C \), we can state without loss of generality that \( C = 1 \). We seek to find a maximum-value set \( T \subseteq S \), such that \( \sum_{i \in T} \text{size}(i) \leq C = 1 \) and such that \( \sum_{i \in T} \text{value}(i) \) is maximized. The items in \( T \) can be interpreted as being packed into a knapsack of known capacity. This problem is well-known to be solvable with a fully polynomial-time approximation scheme (FPTAS) [46, 52]; this means the optimal solution value can be approximated to within any desired constant factor of \( (1 - \varepsilon) \), in polynomial time with respect to the problem size \( n \) as well as with respect to \( \frac{1}{\varepsilon} \) (by contrast, a polynomial-time approximation scheme (PTAS) is not necessarily polynomial with respect to \( \frac{1}{\varepsilon} \)).

In order to describe the network knapsack problem as a generalization of the 0-1 knapsack problem, we begin with a graph \( G = (V, E) \), such that each vertex \( i \) in the graph has a value, a size,
and a “knapsack” capacity which only i’s closed neighborhood \( N[i] \) can fill (that is, the set of vertices neighboring \( i \) as well as \( i \) itself). However, when a vertex is selected, it contributes its size towards the capacity of every vertex in its closed neighborhood. We seek to obtain a maximum-value subset of items that respects all capacities. More formally, we state the network knapsack problem as

\[
OPT = \text{Maximize} \sum_{i \in V} \text{value}(i)x(i)
\]

\[
\text{Subject to} \sum_{j \in N[i]} \text{size}(j)x(j) \leq \text{capacity}(i) \quad \forall i \in V
\]

\[
x(i) \in \{0, 1\} \quad \forall i \in V,
\]

which, as mentioned earlier, is a special case of the PIP problem (IP5) where the graph underlying IP5’s constraint matrix \( A \) has some special structure, such as a tree, grid, low-treewidth graph, or planar graph. In this dissertation, we develop an optimal solution in \( O(8^k n) \) time (linear time for all \( k = O(1) \)) for the network knapsack problem on ladders of width \( k \) (grids of dimension \( k \times \frac{n}{k} \)), and PTAS solutions to approximate the problem on trees and grids. We also investigate the difficulty of developing approximation algorithms for this problem on low-treewidth graphs and planar graphs.

The network knapsack problem is a packing problem, but the closely related covering variant (related to the CIP problem, IP4) is perhaps more closely interpretable as a network fortification problem. Each vertex, instead of having a maximum capacity, has instead a minimum required amount of coverage it must be provided. Instead of values on items, we have costs, and we wish to choose a minimum-cost set of items such that each vertex is reinforced (or covered) to the required extent. The covering variant is thus also viewable as a kind of weighted alliance\(^2\) problem with costs, but where each vertex \( i \) has its own unique fortification requirement indicated by \( \text{capacity}(i) \).

The covering variant of the network knapsack problem can be stated as

\[
OPT = \text{Minimize} \sum_{i \in V} \text{cost}(i)x(i)
\]

\[
\text{Subject to} \sum_{j \in N[i]} \text{size}(j)x(j) \geq \text{capacity}(i) \quad \forall i \in V
\]

\[
x(i) \in \{0, 1\} \quad \forall i \in V.
\]

\(^2\)The two simplest types of alliances are the defensive alliance and the offensive alliance. An offensive alliance for a graph \( G = (V, E) \) is set \( S \subseteq V \) whose neighboring vertices each have at least as many neighbors inside \( S \) as there are outside \( S \). A defensive alliance is a set \( S \subseteq V \) for which each member has more neighbors in \( S \) (counting itself) than outside of \( S \). See Chapter 2 for further discussion of these and other types of alliances, as well as related literature.
1.5 Dissertation Organization

The remainder of this dissertation is organized as follows. In Chapter 2, we present a selection of literature generally related to network interdiction and fortification, and then literature specifically related to the problems considered in this dissertation. Following that chapter, we detail our results in one problem area at a time. Chapter 3 discusses theoretical results and proofs for the multiway cut problem, Chapter 4 discusses computational experiments performed to investigate the multiway cut problem, Chapter 5 discusses results obtained for the \(k\)-hurdle multiway cut problem, Chapter 6 discusses results related to the \(k\)-hurdle multicut problem, and Chapter 7 discusses our algorithms for the network knapsack problem on various graph types. In Chapter 8, we offer concluding remarks and a summary of open problems, followed by a list of references.
Chapter 2

Related Work

In this chapter, we discuss works that are broadly related to network interdiction and fortification, as well as works more closely related to the problems considered in this dissertation. We begin with a broad overview of various types of network fortification problems, interdiction problems, as well as other very similar and related problems. Following that, we discuss works related more closely to the multicut and multiway cut problems considered in Chapters 3 through 6. Finally, we discuss works related to the network knapsack problem, which we investigate in Chapter 7.

2.1 Network Fortification

The domain of network interdiction and fortification is a widely studied and multifarious field, where for example we might want to build multiple redundant layers of checkpoints for inspecting goods being shipped through a network, or we might want to disable multiple layers of edges in a network to inhibit the movement of a malicious adversary. A variety of papers discussing traditional fortification and interdiction papers have appeared over the last few decades.

Though the exact terminology has changed a lot in the past several decades, the sub-field of connectivity augmentation is a popular one in the literature. Some early papers from the late 1970s including [76, 21] show how to make a graph biconnected (i.e. connected, and remaining connected even after the removal of any vertex) in linear time by adding a minimum number of edges. However, the cost-based generalization is more difficult to solve; [21] shows that given edge costs, finding a
minimum cost augmentation is NP-hard. Continuing into the 1980s, Frederickson and Ja’Ja’ [27] take a broader view on this type of problem, and consider approximation algorithms for several problems which they refer to as “graph augmentation” problems. This term is a very broad one as used here; these problems involve modifying or adding a selected set of edges in order for a given graph (sometimes directed or weighted) to satisfy a given property, for example biconnectivity (as in [76, 21]) or strong connectivity. Such a broad definition is also a precursor to many newer types of inhibition and other related problems, many of which we will mention. Another seminal network fortification paper is from 1985 by Cunningham [14], and discusses algorithms for reinforcing a network against attacks, where edge weights signify the effort required on the part of the attacker to disable the edge. He provides algorithms for computing the strength of the network against such attacks, and for reinforcing the network at minimum cost to achieve a desired strength.

We see an expansion of the variety of papers on these topics in the 1990s. Paik and Sahni [67] consider various problems on communication networks that involve reducing communication delays of graph edges or vertices (i.e. “upgrading” them) to increase the performance of the network with respect to various criteria. They show that several variations of problems of this kind are NP-hard, for example (1) upgrading the minimum number of edges so that the communication delays on the shortest (or longest, in another problem variant) paths between every pair of vertices in the graph are below a given threshold, and (2) upgrading a minimum number of vertices (which will reduce the communication delay on all incident edges) so that no edge has a delay greater than a given threshold.

Krumke et al. in 1998 [54] consider optimal network improvement and upgrading problems with the added constraint of a budget, where we seek to optimally improve the network by reducing either the cost or the communication delays of either edges or vertices subject to some optimization criteria (such as minimizing either the bottleneck delay or the cost of a spanning tree), and also subject to a budget constraint. They show inapproximability results as well as approximation algorithms for several variations. For an overview of more problems in this family, see the 1999 survey of Noltemeier et al. [66], a brief but illuminating look at approximation algorithms for various network design problems, network upgrade problems, network improvement problems, and multicriteria problems.

More recently, a paper by Zhang et al. in 2004 [84] explores a network improvement problem of reducing edge lengths in a network to satisfy upper bounded distance requirements from a special
source terminal $s$. They seek to shorten a minimum cost subset of edges so that $||d||$ drops to at most some desired threshold, where $d$ denotes the vector of distances from $s$ to all other vertices, and $||\cdot||$ denotes a specific vector norm (a way of measuring distance) such as $L_1$, $L_2$ (standard euclidian distance), and $L_\infty$. From the $L_1$ norm (where distance is a simple sum of the components) arises the problem of minimizing the sum of the distances from $s$ to the other vertices. From the $L_\infty$ norm (where the distance between two points is simply the maximum of the components) arises an even simpler problem: minimize the maximum distance from $s$ to any other terminal. The authors show a strongly polynomial-time algorithm for their network improvement problem in the $L_\infty$ norm, but for both the $L_1$ and weighted $L_2$ norms, they give NP-hardness as well as $\Omega(\log(n))$ inapproximability results.

2.2 Network Interdiction

Network interdiction/inhibition has seen an increasing amount of research in the past two decades, but some of the earliest papers on these topics appear in 1970. For example, Ghare et al. \cite{ghare1970network} as well as McMasters and Mustin \cite{mcmasters1975military} consider military applications of inhibiting $s$-$t$-planar graphs, a class of graphs that are not only planar, but have a single source and sink both lying on the outer face. They provide algorithms for the interdictor to expend limited resources to minimize the maximum $s$-$t$ flow through the network.

Beginning with two papers from 1993, we see an increasing number of publications which take a broader and less restrictive view of network inhibition and interdiction problems. In a paper by Wood et al. \cite{wood1993network}, an adversary once again wishes to minimize the maximum flow in a general network. Integer programming models are presented and strengthened with additional inequalities to tighten the linear programming relaxation. Phillips \cite{phillips1993network} discusses the network inhibition problem where we wish to minimize the maximum flow or equivalently the minimum cut. She provides proofs of strong NP-completeness on general graphs, even of degree at most 3, and provides proofs of weak NP-completeness for series-parallel graphs, grids, graphs of bandwidth $\leq 3$, and Halin graphs. She also provides some approximation algorithms including an FPTAS for planar graphs. Phillips along with Swiler \cite{phillips1993network} take this sort of problem in a different direction, providing an entire analysis system for network vulnerability using a graph-based tool to analyze networks with repeated histories of attacks. It assigns probabilities of attack to edges based on past attack profiles, to find attack
paths with the highest probability of successful interdiction by the attacker. Burch et al. [7] give an interesting pseudo approximation result, based on a linear programming relaxation, to the strongly NP-hard problem of inhibiting the source-sink flow in an edge-capacitated graph by expending a limited budget to pay to reduce or remove edge capacities, to minimize the resulting source-sink max flow. Their algorithm returns either a \((1, 1 + \varepsilon)\)-approximation or a \((1 + \varepsilon, 1)\)-pseudo approximation (based on a given error parameter \(\varepsilon\)), but we do not know which beforehand.

Very recently, some authors have considered the differences between solving network improvement and inhibition variants of the same core problem, such as [32]. They consider budget constrained network improvement and degrading problems all based on the 1-center problem: to minimize (or maximize) the weighted distance between the designated 1-center vertex and all the others, by decreasing (or increasing) vertex weights under a budget. Given a distance matrix, they show the improvement problem is polynomial-time solvable with a \(O(n^2)\) algorithm, whereas they show that the degrading problem is strongly NP-hard on general graphs but admits a polynomial-time \(O(n^2)\) algorithm on trees.

The \(k\)-hurdle cut problem, on which much of this dissertation is based, can be viewed as a special case of a “shortest path” network interdiction problem. Phrased in this way, we can describe the \(k\)-hurdle cut problem as paying \(c(e)\) per unit length to increase the length of edge \(e\), with the goal of increasing the shortest \(s-t\) path length to at least \(k\) at minimum cost; see also [47] for a further discussion of shortest path network interdiction.

### 2.3 Cut Problems

The multiway cut problem mentioned in the introduction is APX-hard for \(r \geq 3\) terminals [15], but can be approximated fairly well. There are several ways to obtain a \(2(1 - \frac{1}{r})\)-approximation bound, the first being a simple “isolating cut” heuristic due to Dahlhaus et al. [15]. Călinescu et al. [9] developed an elegant \((1.5 - \frac{1}{r})\)-approximation algorithm, and this approach (explained further in Chapter 3) was improved by Karger et al. [51] to obtain a guarantee of 1.3438, and Karger et al. as well as Cheung et al. [12] independently obtained a guarantee of \(\frac{12}{11}\) for the special case of \(r = 3\). The general geometric embedding technique used in these papers has been widely applied, and used to solve many other discrete graph optimization problems (see e.g. [2, 22, 35, 56, 58]). A recent thesis by Damon Mosk-Aoyama [63] explores the special case of \(k = 4\) to improve via computational
experiments on the best known analytical proofs. The author first isolates two terminals from the other two, reducing the problem to a state similar to two separate s-t min-cut instances.

The multicut problem is APX-hard for \( r \geq 3 \) and can be approximated to within an \( O(\log r) \) factor using the prominent “region-growing” approach of Garg et al. [29]. In a tree, one can obtain a 2-approximation algorithm using the primal-dual algorithm of Garg et al. [30], or a more recent approach independently discovered by Golovin et al. [37] as well as Levin and Segev [57].

Many authors [8, 64, 65, 81, 82] have studied the \( k \)-hurdle cut problem, and several polynomial-time solution algorithms for it are known. Its natural LP relaxation, whereby the variables are allowed to take on values in the range \([0,1]\), can be solved in polynomial time using the ellipsoid method (using a shortest path algorithm as a separation oracle), a technique noteworthy since it allows us to solve the LP in polynomial time even though it has an exponential number of constraints; alternatively, the LP can also be restated using a polynomial number of constraints. Burch et al. [8] show that as a consequence of total unimodularity, one can always find an optimal integer-valued solution to the LP relaxation. This is true also for the slightly more general case where we have multiple sources and sinks, where each source must be separated by \( k \) layers of cuts from each sink — this problem can be reduced back to the single-source, single-sink variant by introducing a new super-source vertex \( s \) linked to all sources by infinite-cost edges, and a new super-sink vertex \( t \) linked from all sinks by infinite-cost edges. There are also several ways to round an optimal fractional solution \( x \) to the LP relaxation of the \( s-t \) \( k \)-hurdle cut problem to obtain an integer solution of the same cost. For example, consider a geometric embedding of \( G \) onto the interval \([0,k]\) such that vertex \( v \) is embedded at position \( d_x(s, v) \), the shortest path distance from \( s \) to \( v \) using edge lengths \( x \). Choose any \( \alpha \in (0,1) \), and make \( k \) successive cuts at distances \( \alpha, 1 + \alpha, \ldots, (k-1) + \alpha \) away from \( s \). It is easy to show (see [8]) that the set of all edges crossing these cuts gives us an optimal \( k \)-hurdle solution.

The dual of the \( s-t \) \( k \)-hurdle cut problem is known in the literature as the \( k \)-maximum flow problem [82]. Since it can be expressed as a minimum cost circulation problem with unit costs, we can solve it (and hence also the \( k \)-hurdle cut problem) in \( \tilde{O}(mn) \) time [36], where \( \tilde{O} \) hides logarithmic factors. Problems of this flavor are often found in network upgrading and improvement applications, where we want to maximize the amount of additional flow one can send from \( s \) to \( t \) (typically subject to a budget constraint), where the capacity of certain edges can be upgraded at a price; see [28, 55, 62] for further details. For a more general and complete reference on network
flows, see the textbook of Ahuja et al. [1].

2.4 Knapsacks and Alliances

Beginning with the initial work of Dantzig in 1957 [16], the field of knapsack literature is vast. The first PTAS for the 0-1 knapsack problem is due to Sahni [77], and the first FPTAS comes from Ibarra and Kim [46], both in 1975. There are many writings discussing the breadth of knapsack problems, including the book of Martello and Toth [60], which discusses many classical problem variants, an article of Dudzinski and Walukiewicz [20] which discusses many generalizations of the classical problems, and the thesis of Pisinger [70] which explores several knapsack problems. Knapsack problems also frequently occur with linear programming relaxation, as detailed in various textbooks on the subject [79, 44, 45].

As mentioned in the introduction, the network knapsack problem is related to the concept of alliances. Alliances are a relatively new field of research, which began with the work of Kristiansen, Hedetniemi and Hedetniemi in 2002 [53], and now includes several studies of the properties and complexity of computing various types of alliances. Favaron et al. [23, 24] explored the concepts of the offensive alliance, a set $S$ whose neighboring vertices each have at least as many neighbors inside $S$ as there are outside $S$, and the strong offensive alliance, in which the above inequality is strict. An offensive alliance $S$ can informally be thought of as having the “offensive” capability to outnumber and therefore successfully attack any vertex neighboring a set member. They bound the minimum nonempty size a set $S$ must have in order to meet those definitions. Haynes et al. [41, 42] studied a slightly different type of alliance called a defensive alliance, a set $S$ for which each member has more neighbors in $S$ (counting itself) than outside of $S$. A defensive alliance can, similarly, be thought of as having the “defensive” capability to outnumber and therefore defend against an attack from any vertex neighboring a set member. They study the case where an alliance $S$ either contains or neighbors every vertex in the graph, making $S$ also a dominating set; this is referred to as a global defensive alliance, and a similar definition exists for global offensive alliances as well. In 2006, Cami et al. [10] demonstrate the NP-hardness of finding optimal instances of global alliances, including the offensive, defensive, and powerful (i.e. simultaneously offensive and defensive) varieties; this was accomplished via transformations from the related dominating set problem. In 2007, a paper [19] and a Ph.D thesis [48] from Jamieson explore the weighted generalization of alliances, providing
algorithmic and complexity results for many of the problems above, where sums of neighborhood vertex weights are compared instead of cardinalities.
Chapter 3

Multiway Cut

In this chapter, we discuss our theoretical results for the multiway cut problem. Specifically, we show an algorithm for the problem and an accompanying analysis showing a performance ratio \( \leq 1.4647 - \varepsilon_r \), where \( \varepsilon_r \) is a small constant related to the number of terminals \( r \). This algorithm improves on the \( 1.5 - \frac{1}{r} \) algorithm of Călinescu et al. [9] without the complicated analysis required to prove the \( 1.3438 - \varepsilon_r \) result of Karger et al. [51]. The structure of our approach is essentially the same as these authors’, but our goal in this chapter is to shed additional light on the multiway cut problem through a new analysis. The result of Karger at al. [51] is the method of choice if one wishes to use the algorithm with the best known approximation guarantee for general \( r \).

3.1 Algorithm

The algorithm we present for the multiway cut problem on \( r \) terminals makes use of the \( r \)-simplex embedding, introduced by Călinescu et al. [9]. It is a construction that embeds an instance of the multiway cut problem into a convex \((r - 1)\)-dimensional polytope in \( r \)-dimensional space given by the set of points \( x = (x_1, ..., x_r) \) with nonnegative coordinates such that \( \sum_{i=1}^{r} x_i = 1 \); we call this set of points the \( r \)-simplex, denoted by \( \Delta_r \). We denote by \( \Delta_r(v) \) the location where vertex \( v \in V \) is embedded in the \( r \)-simplex. Each terminal \( t_i \) is embedded at a separate “corner” \( e_i \) of the simplex, a point \( x \), such that \( x_i = 1 \) and \( x_j = 0, \forall j \neq i \). In this setting, the natural way to measure distance is half the standard \( L_1 \) norm:
Due to this way of measuring distance, our simplex has a length of 1 unit on a side, measured from one terminal to another or from a terminal to the opposite face. The simplex embedding has been shown to be equivalent \([9]\) to a strengthened form of the basic linear program for the problem \((\text{LP}1)\) where additional constraints are added. This allows us to create approximation algorithms with better guarantees than \(2(1 - \frac{1}{r})\) using the simplex embedding \(\text{LP}\), which can is formulated as

\[
\text{Minimize } \sum_{e \in E} c(e)||e||
\]

\((\text{LP7})\) \text{ Subject to } \Delta_r(v) \in \Delta_r \quad \forall v \in V \\
\Delta_r(t_i) = e_i \quad \forall \text{ terminals } i, 1 \leq i \leq r.

Our approximation algorithm is known as a \textit{SPARC} (Side PARallel Cut) cutting scheme, because it employs only side-parallel \textit{slices} through the simplex in order to induce a cut. Each slice \(s\) is defined simply by the index of a terminal \(i\), and a distance \(d\) from the face opposite \(i\); \(s\) is parallel to this face. Every edge that crosses such a slice is cut (i.e., edges with one adjacent vertex embedded into the simplex at a point \(x\) such that \(x_i < d\), and the other vertex embedded at a point \(y\) such that \(y_i > d\)). A collection of these slices together form a SPARC, which induces a multiway cut of the simplex by separating each terminal from the others.

Any optimal SPARC cutting scheme can be interpreted as having the following form (proven in \([51]\)). First, choose \(r\) slice distances \((d_1, \ldots, d_r)\). Then, apply \(r - 1\) slices (in that order) at the first \(r - 1\) distances to a uniformly random permutation of the terminals. Each one of these terminal-distance pairs will be used to define a side-parallel slice isolating one terminal from the rest. In this way, \(r - 1\) terminals will be isolated, inducing a multiway cut of the simplex. We use a “ball-corner” scheme, somewhat similar to the one described in \([9]\). Our scheme has two types of cuts we define: a \textit{ball cut} and a \textit{corner cut}.

With probability \(\alpha\), \(0 < \alpha < 1\), we perform a ball cut (see Figure 3.1(b)) by choosing a center point \(c = (c_1, \ldots, c_r)\) from the range \([0, \beta]^r\), where \(\beta \geq 1/2\). Note that this center point may or may not be inside the simplex. We then place \(r - 1\) side-parallel slices, all intersecting this center point, according to a uniformly random permutation of the terminals. Each slice \(i\) from terminal \(t_i\) captures all points \(x\) in the simplex with \(x_i > c_i\), thus any edges in this region will not be cut by subsequent slices, as shown in Figure 3.1(b). Otherwise, with probability \(1 - \alpha\), we perform a
Figure 3.1: Figure 3.1(a) at the top left depicts a corner cut in a 3-simplex. The dashed lines depict the boundary between the central ball region of width \( \beta \) and the corner regions of width \( 1 - \beta \). Figure 3.1(b) on the top right depicts a ball cut, consisting of two slices intersecting at point \( c \). The first (slanted) slice cuts across the entire simplex, but the second (horizontal) slice only cuts the region not yet captured by the first slice. Figure 3.1(c) at the bottom left shows a 1-2 aligned edge \((u, v)\) as well as strip 1 and strip 2; only a ball cut with \( c \) in the shaded region can cut edge \((u, v)\). Figure 3.1(d) at the bottom right depicts a helpful ball cut with center point \( c \); the 1-2 aligned edge \((u, v)\) is safe due to the helpful slice from terminal 1.
corner cut (see Figure 3.1(a)) by choosing each of our \( r - 1 \) slice distances to be in the range \((\beta, 1]\).

We restrict \( \beta \geq \frac{1}{2} \) since otherwise the corner cut slices might overlap within the simplex.

The cutting scheme we use, and those used by Karger et al. and Călinescu et al., vary only in the choice of \( \alpha \) and \( \beta \), but the analyses vary in complexity. We now provide an analysis of our SPARC cutting scheme leading to an approximation ratio \( \leq 1.4647 - \varepsilon_r \), where \( \varepsilon_r \) is a small constant related to \( r \). This improves upon the \( 1.5 - \frac{1}{r} \) result from Călinescu et al. [9] without the extremely complicated proof of the \( 1.3438 - \varepsilon_r \) result of Karger et al. [51]

### 3.2 Algorithm Analysis

To analyze our cutting scheme, we analyze the density of an edge \( e \), \( \operatorname{density}(e) = \Pr[e \text{ is cut}]/||e|| \).

This will be maximized in the case where \( e \) is of infinitesimal length. The maximum density(e) over all edges \( e \) we denote by \( d^* \), which gives the approximation guarantee of our algorithm. To justify this, linearity of expectation tells us that the expected cost of our solution is

\[
\sum_{e \in E} c(e) \Pr[e \text{ is cut}] \leq \sum_{e \in E} c(e)||e||d^* = d^* \sum_{e \in E} c(e)||e|| \leq d^* \operatorname{OPT},
\]

(3.1)

since \( \sum_{e \in E} c(e)||e|| \) is the objective of the relaxation given by the simplex embedding LP (LP7).

We proceed to analyze our algorithm by considering two cases. For the first case, consider an edge \( e = (u, v) \) lying in the “ball” region (i.e. with both endpoints in the range \([0, \beta]^r \); see Figure 3.1(a) depicting the boundaries of the ball region). Edge \( e \) can only be cut by a ball cut with a center point \( c \) chosen uniformly at random from the range \([0, \beta]^r \). We know \( 0 \leq u_i, v_i \leq \beta \), for all \( 1 \leq i \leq r \). We assume without loss of generality, as the authors of [9] do, that edge \( e \) is 1-2 aligned. This means that the coordinates of \( u \) and \( v \) differ only by their distances to terminals 1 and 2, but they are exactly the same distance to all other terminals; see Figure 3.1(c) for an illustration of a 1-2 aligned edge \((u, v)\). The coordinates of the points \( u, v \), and \( p \) at the center of edge \( e \) are as follows.

Note for all three points that not only are all coordinates after the first 2 exactly identical, but the first 2 coordinates are nearly identical due to the infinitesimal length of \( e \).

\[
\begin{align*}
u &= (u_1, u_2, u_3, u_4, \ldots, u_r) \\
v &= (v_1, v_2, u_3, u_4, \ldots, u_r) \\
p &= (p_1 = (u_1 + v_1)/2 , \ p_2 = (u_2 + v_2)/2 , \ u_3, u_4, \ldots, u_r).
\end{align*}
\]
Next, we define strip 1, corresponding to terminal 1, as the set of all points \( \{ c \mid c_1 \in [u_1, v_1] \} \) and strip 2, corresponding to terminal 2, similarly as \( \{ c \mid c_2 \in [u_2, v_2] \} \); see Figure 3.1(c) for an illustration. Note that \( e \) cannot be cut by any corner cut, and can only be cut by a ball cut where \( c \) is in strip 1 or strip 2. By itself, this observation yields a 2-approximation, but our analysis does not stop here.

Given that \( c \) lies within strip 1 or strip 2, we wish to compute \( \text{Pr}[e \text{ is cut}] \), and to do so we will first compute \( \text{Pr}[e \text{ is safe}] \). In order to define this event, consider the first slice \( i \) in the slicing order that can cut \( e \). This is either the slice from terminal 1 or the slice from terminal 2, having slicing distance \( d_i \) from the opposite face. \( \text{Pr}[e \text{ is safe}] \) represents the likelihood that either 1) \( i \) is in the last position \( r \) in the slice ordering and is therefore not used (and thus no slice can cut \( e \)), or 2) \( i \) captures \( e \), meaning that \( d_i < p_i \) (and thus no future slice can cut \( e \)); see Figure 3.1(d) for an illustration of slice \( i \) capturing \( e \). In either of the above cases, if \( e \) is safe then it will not be cut, thus

\[
\text{Pr}[e \text{ is safe}] \leq 1 - \text{Pr}[e \text{ is cut}],
\]

though this is not necessarily an equality.

We will assume without loss of generality that \( c \) lies within strip 1, and will analyze \( \text{Pr}[e \text{ is cut}] \) for edge \( e \) of infinitesimal length. The analysis for the other situation, where \( c \) lies within strip 2, is similar and completely symmetric. We define the center point \( c \) as helpful if \( \exists j, 2 \leq j \leq r \) such that \( c_j < p_j \), and refer to \( j \) as the helpful slice (though there may be others). See Figure 3.1 depicting a helpful center point \( c \) and a helpful slice. From the definition, we can see that if \( c \) is not helpful, then regardless of the rest of the slice ordering, no slice will capture \( e \). Conversely, if \( c \) is helpful, then some slice ordering(s) will cause \( e \) to be captured and therefore not be cut. Note that there is only one slice \( i \) (specifically, the slice from terminal number 1) that can cut \( e \).

We first wish to compute \( \text{Pr}[c \text{ is helpful}] \), and we will do this by first finding \( \text{Pr}[c \text{ is not helpful}] \).
For $c$ to not be helpful, then $p_k \leq c_k, \forall i, 2 \leq k \leq r$.

\[
\Pr[c \text{ is not helpful}] = \prod_{k=2}^{r} \Pr[p_k \leq c_k] \tag{3.3}
\]

\[
= \prod_{k=2}^{r} \frac{\beta - p_k}{\beta} \tag{3.4}
\]

\[
= \prod_{k=2}^{r} 1 - \frac{p_k}{\beta} \tag{3.5}
\]

\[
\leq \prod_{k=2}^{r} e^{-p_k/\beta} \tag{3.6}
\]

\[
= e^{(1/\beta) \sum_{k=2}^{r} p_k} \tag{3.7}
\]

\[
\leq e^{(-1/\beta)(1-p_1)} \tag{3.8}
\]

\[
= e^{(p_1-1)/\beta} \tag{3.9}
\]

Therefore,

\[
\Pr[c \text{ is helpful}] \geq 1 - e^{(p_1-1)/\beta}. \tag{3.10}
\]

Based on this bound for $\Pr[c \text{ is helpful}]$, we now wish to compute $\Pr[e \text{ is safe}]$. We will use conditional probability, and consider all the situations where an edge $e$ is safe using a collection of four events, based on the single slice $i$ which can cut $e$. First, we have the event that $i$ is last in the slice ordering, which is one of the two conditions we originally used to define the term safe, earlier. This case occurs with probability $\frac{1}{r}$, and in this case $\Pr[e \text{ is safe}] = 1$, thus it contributes a total of $\frac{1}{r}$ to the conditional probability. Second, we have the event that $i$ is not the last in the slice ordering, and $c$ is not helpful. In this case, $\Pr[e \text{ is safe}] = 0$, thus this case contributes nothing to the conditional probability. Third, we have the event that $i$ is not last, $c$ is helpful, and the helpful slice $j$ is the last slice. In this case, the helpful slice is unused, therefore $\Pr[e \text{ is safe}] = 0$, and this case contributes nothing to the conditional probability. Fourth and finally, we have the event that $i$ is not last, $c$ is helpful, and the helpful slice $j$ is not the last slice. In this case, $\Pr[e \text{ is safe}] \geq \frac{1}{2}$, since we have slice $j$ (and possibly other slices) to capture $e$ and exactly one slice $i$ to cut $e$, each having an equal chance to occur before the others in the slice ordering. To calculate the probability of being in this case, we use $\Pr[i \text{ is not last}], \Pr[j \text{ is not last}], \text{ and } \Pr[c \text{ is helpful}]$ to obtain $(1 - \frac{2}{r})\Pr[c \text{ is helpful}]$. So, only the first and fourth cases contribute to the conditional
probability, and we can use the bound given by equation (3.10) to state \( \Pr[e \text{ is safe}] \) as

\[
\Pr[e \text{ is safe}] \geq \frac{1}{r} + \frac{1}{2} - \left( 1 - \frac{1}{r} \right) \left( \frac{1}{2} - e^{-\frac{1}{r}} \right) \left( 1 - e^{(p_1-1)/\beta} \right)
\]

(3.11)

\[
= \frac{1}{2} - \left( \frac{1}{2} - \frac{1}{r} \right) e^{(p_1-1)/\beta}.
\]

(3.12)

Combining equations (3.2) and (3.12), we have

\[
\Pr[e \text{ is cut}] \leq \frac{1}{2} + \left( \frac{1}{2} - \frac{1}{r} \right) e^{(p_1-1)/\beta},
\]

(3.13)

under the assumption that \( e \) is in strip 1. By a similar and symmetric analysis, if we assume that \( e \) is in strip 2, we will find that

\[
\Pr[e \text{ is cut}] \leq \frac{1}{2} + \left( \frac{1}{2} - \frac{1}{r} \right) e^{(p_2-1)/\beta}.
\]

(3.14)

Since \( e \) can only be cut when \( e \) is in strip 1 or \( e \) is in strip 2, we sum equations (3.13) and (3.14) to obtain

\[
\Pr[e \text{ is cut}] \leq 1 + \left( \frac{1}{2} - \frac{1}{r} \right) \left( e^{(p_1-1)/\beta} + e^{(p_2-1)/\beta} \right),
\]

(3.15)

for the general case where \( e \) lies in the ball region. \( p_1 \) and \( p_2 \) are each in the range \([0, \beta]\) and sum to at most 1. Subject to these constraints, we can upper bound (3.15) by setting \( p_1 = \beta \) and \( p_2 = 1 - \beta \), giving

\[
\Pr[e \text{ is cut}] \leq 1 + \left( \frac{1}{2} - \frac{1}{r} \right) \left( e^{1-1/\beta} + e^{-1} \right).
\]

(3.16)

With that, we can bound density\((e)\) as

\[
density(e) \leq \frac{\alpha}{\beta} \left[ 1 + \left( \frac{1}{2} - \frac{1}{r} \right) \left( e^{1-1/\beta} + e^{-1} \right) \right]
\]

(3.17)

for this case, where \( e \) lies in the ball region. Equation (3.17) contains the multiplicative term \( \frac{\alpha}{\beta} \) since \( \beta \) is the width of the ball region, and we choose a ball cut with probability \( \alpha \).

For the second case, we consider an edge \( e \) lying in a “corner” region, where both endpoints \( u \) and \( v \) have exactly one component \( i \) such that \( u_i, v_i \geq \beta \), and all other components \( u_j, v_j < \beta, \forall j \neq i \).
Since $e$ can be cut by both corner cuts and ball cuts, we calculate density($e$) in equation (3.18) below as the sum of two quantities: the contribution due to corner cuts (wherein all $r - 1$ slice distances are each chosen uniformly at random from the range ($\beta, 1$)), and the contribution due to ball cuts. The density due to corner cuts is inversely dependent on the fact that corner cuts can slice through the area of the simplex at distance $< \beta$ from each terminal (equivalent to $1 - \beta$ from the opposite face), and directly dependent on the fact that we choose a corner cut with probability $1 - \alpha$. We must also multiply the density due to corner cuts by the probability that a corner is one of the $r - 1$ sliced corners. Regarding the density due to ball cuts, the ($\alpha \beta$) term appears for the same reason as in equation (3.17), but here this term is halved since only one of the two relevant slices (corresponding to terminals 1 and 2) cuts $e$, while the other captures $e$ and prevents $e$ from being cut. From these facts, we can express the density of edge $e$ as

$$\text{density}(e) \leq \left(1 - \frac{1}{r}\right)\left(\frac{1 - \alpha}{1 - \beta}\right) + \frac{\alpha}{2\beta}. \text{ (corner)}$$

$$\text{density}(e) \leq \left(1 - \frac{1}{r}\right)\left(1 - \beta\right) + \frac{\alpha}{2\beta}. \text{ (ball)}$$

for the case that $e$ lies in the corner region.

Now that we have analyzed density($e$) in both cases of the location of edge $e$, we know that the maximum density $d^*$ of any edge in our simplex is given by

$$d^* = \max \left(\frac{\alpha}{\beta} \left[1 + \left(\frac{1}{2} - \frac{1}{r}\right)\left(e^{1-1/\beta} + e^{-1}\right)\right], \left(1 - \frac{1}{r}\right)\left(\frac{1 - \alpha}{1 - \beta}\right) + \frac{\alpha}{2\beta}\right). \text{ (3.19)}$$

Now, since $d^*$ is our approximation ratio, we seek to minimize the above expression of $d^*$ over all possibilities of $\alpha$ and $\beta$, for each value of $r \geq 3$. See Figure 3.1 for a table of minimized approximation ratios for various values of $r$, along with the corresponding values of $\alpha$ and $\beta$ to achieve that ratio. As $r$ approaches $\infty$, $d^*$ is minimized at $\beta = \frac{1}{2}$ and $\alpha = \frac{2}{3+2\sqrt{e}}$. This gives us an approximation ratio as $r$ approaches $\infty$ of $2 - \frac{2}{3+2\sqrt{e}}$, which is $< 1.4647 - \varepsilon_r$, where $r$ is a small constant related to the number of terminals $r$.

It is important to note that our algorithm is the same as that of Karger et al. [51], except that they use a value of 0.667186 for $\alpha$, a value of $\frac{6}{11}$ for $\beta$, and they provide a different analysis. Though
<table>
<thead>
<tr>
<th>$r$</th>
<th>$d^*$</th>
<th>$\alpha$</th>
<th>$\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.1598</td>
<td>0.562</td>
<td>0.550</td>
</tr>
<tr>
<td>4</td>
<td>1.2377</td>
<td>0.561</td>
<td>0.544</td>
</tr>
<tr>
<td>5</td>
<td>1.2838</td>
<td>0.548</td>
<td>0.526</td>
</tr>
<tr>
<td>6</td>
<td>1.3143</td>
<td>0.547</td>
<td>0.523</td>
</tr>
<tr>
<td>7</td>
<td>1.3361</td>
<td>0.544</td>
<td>0.518</td>
</tr>
<tr>
<td>8</td>
<td>1.3523</td>
<td>0.539</td>
<td>0.511</td>
</tr>
<tr>
<td>9</td>
<td>1.3649</td>
<td>0.538</td>
<td>0.509</td>
</tr>
<tr>
<td>10</td>
<td>1.3749</td>
<td>0.545</td>
<td>0.517</td>
</tr>
<tr>
<td>11</td>
<td>1.3832</td>
<td>0.538</td>
<td>0.508</td>
</tr>
<tr>
<td>12</td>
<td>1.3900</td>
<td>0.540</td>
<td>0.510</td>
</tr>
<tr>
<td>99</td>
<td>1.4557</td>
<td>0.535</td>
<td>0.500</td>
</tr>
</tbody>
</table>

Table 3.1: This table lists the approximation ratios $d^*$ of our multiway cut algorithm for various values of $r$, the number of terminals. For each value of $r$, this table lists the minimum approximation ratio $d^*$ achievable by setting $\alpha$ and $\beta$ to the listed optimal values.

our approximation guarantee is strictly weaker, it can nonetheless be viewed as an intermediate result between the $1.5 - \frac{1}{r}$ guarantee of [9] and the $1.3438 - \varepsilon_r$ guarantee of [51]. However, unlike the latter, we provide a closed-form expression for our approximation guarantee, and a simpler analysis using different techniques.
Chapter 4

Multiway Cut Computational Experiments

In addition to the algorithm and analysis presented earlier for the multiway cut problem, we have performed computational experiments to gain further insight into the problem. Though most of the results obtained by our experiments are inconclusive or negative, we nonetheless offer some interesting observations that may be useful for future research in the area. We will begin this chapter by motivating our computational study, describing its theoretical underpinnings, and introducing the central concepts that apply specifically to these experiments. Afterward, we will describe the three computational avenues we have explored.

4.1 Motivation and Background

Central to all our experimental investigations is the notion of a bad graph, a distribution of weights on the edges of a given graph G summing to a fixed constant, and characterized by giving the highest minimum-cost multiway cut over all possible cuts of G. A linear program for finding the multiway cut bad graph weight distribution is
Figure 4.1: A bad graph for the 3-simplex with \( n = 96 \) discretizations. One terminal (at the top corner of the simplex) is shown much closer to our viewpoint than the other two terminals, at the bottom left and right corners. Edge weight from the LP solution is proportional to the thickness of the edge in the figure. The edges closest to a particular terminal are tinted a unique color (gold, purple, or green) for clarity. Observe that edges of high weight occur on the outer boundary of the simplex, and that uniformly weighted edges fill a hexagonal region in the center of the simplex.

Maximize \( \lambda \)

(LP8) Subject to \[ \sum_{e \in C} c(e) \geq \lambda \quad \forall \text{ multiway cuts } C \]
\[ \sum_{e \in E} c(e) = 1 \]
\[ c(e) \geq 0 \quad \forall e \in E. \]

The bad graph (i.e., the set of edge weights) obtained by solving this LP is useful because it is a concrete worst case for the multiway cut problem. However, this LP can be quite challenging to solve, as we discuss shortly.

All of our experiments take place on a special type of graph called a discretized \( r \)-simplex, which is a discretized version of the \( r \)-simplex \( \Delta_r \) into which the simplex embedding LP embeds the graph \( G \) (LP7, introduced in Chapter 3). A discretized 3-simplex, used to study the 3-terminal case, is a 2-dimensional regular triangular mesh with some constant number \( n \) edges on a side; as \( n \) approaches infinity, the behavior of the discretized simplex approaches that of the theoretical
3-simplex. In the limit as \( n \) approaches \( \infty \) (i.e., if the discretization is “fine-grained” enough), the worst approximation ratio for a simplex embedding-based approximation algorithm (such as [9], [51], or our algorithm presented in Chapter 3) will occur on a bad graph in the shape of a simplex with \( n \) discretizations. Figure 4.1 is an example of a highly discretized 3-simplex, depicting a 3-terminal bad graph. The discretized 3-simplex is an important experimental tool, used by Karger et al. to discover their \( \frac{12}{11} \) approximation algorithm [51] along with matching integrality gap proof for the \( r = 3 \) case. A discretized 4-simplex is a 3-dimensional regular triangular mesh; see Figure 4.2 for an illustration. Higher-order simplexes for \( r \geq 5 \) are \( r - 1 \geq 4 \) dimensional, making them somewhat more challenging to visualize.

Regarding the bad graph LP, LP8, it is important to note that its topmost constraints,

\[
\sum_{e \in C} c(e) \geq \lambda, \forall \text{ multiway cuts } C, \tag{4.1}
\]
are exponential in number due to the sheer number of possible multiway cuts. To address this issue, we could consider using the ellipsoid algorithm [38], which can solve any LP with an exponential number of constraints in only polynomial time, provided we have an “oracle” to verify in polynomial time whether or not a given solution either violates a constraint or satisfies all constraints. In our case, however, this approach can be difficult to use, since the oracle itself would need to solve the multiway cut problem.

Partly because of this, one approach we investigate is to restrict ourselves to a polynomially-sized class of cuts, namely SPARCs, thus keeping the LP manageably small. As mentioned in Chapter 3, a **SPARC** (Side PARallel Cut) is a multiway cut of the simplex consisting of side-parallel *slices* through the simplex. We will devote some of our experiments to finding a so-called **SPARC bad graph** for $r = 4$, the edge weight distribution of $G$ which gives the highest minimum-cost multiway cut over all possible SPARC cuts. SPARCs are a very popular class of cuts, since nearly every published cutting scheme for this problem (with a notable exception from Mosk-Aoyama’s master’s thesis [63]) either restricts itself entirely to SPARCs or is provably equivalent to a SPARC cutting scheme. In fact, the $\frac{12}{11}$ approximation ratio we mentioned earlier for the $r = 3$ case from Karger et al. [51] uses a SPARC cutting scheme, and the tightness of this result proves that SPARCs are sufficient to optimally cut the 3-simplex. However, the 3-simplex is somewhat “better behaved” than higher-order simplexes. For example, one can replace the exponentially many constraints of a 3-simplex bad graph LP (LP8) with a polynomially sized set of constraints encoding a shortest-path problem in the planar dual graph $G'$ of the discretized (planar) simplex $G$; these dual constraints are known to be polynomial in number [51]. Thus, planar graph duality allows us to quickly solve for an optimal multiway cut by considering shortest paths in the dual. The planarity of the $r = 3$ case is one of its most important properties, helping researchers (e.g., Karger et al.) develop strong results for that case. Sadly, with $r \geq 4$, $G$ is no longer planar, and the problem of solving the bad graph LP (LP8) is substantially more computationally challenging. This is true even if we consider only SPARC cuts, since even in this case LP8 has $O(n^{r-1})$ variables and constraints.

We can also motivate our reasons for seeking an optimal bad graph weight distribution for $G$, and show this problem’s relationship with that of finding a cutting scheme for $G$, by considering them from the point of view of two opposing players in a zero-sum game. In a zero-sum game, two opposing players $A$ and $B$ each wish to obtain a maximum number of points from the other player. First, player $A$ picks a strategy from a finite list of choices. Then, player $B$ picks a strategy in
response, from a separate finite list of choices. Based on the strategies selected by player A and player B, an outcome occurs, and player A gives player B the specified amount of points (an outcome below zero means player A gains points from player B instead of losing them). The term “zero-sum game” comes from the fact that any time player A gains (or loses) points, player B loses (or gains) the exact same number of points; thus, the sum total of all points held by the players always remains at 0 (though one player may have a negative number of points), and the points gained and lost by a single exchange always sum to zero. Consider a two-dimensional matrix $M$ wherein we label each row with a choice of player A, and each column with a choice of player B. Then, the entry in the matrix $M(i, j)$ corresponding to a specific row $i$ and column $j$ shows the number of points lost by player A (and therefore gained by player B) when A chooses strategy $i$ and B chooses strategy $j$. We call this matrix a zero-sum table.

In the above description, the strategies used by the players are called pure strategies, because a single option is chosen with probability 1. A mixed strategy is the more general case, where a player may use a probability distribution over all possible choices he has available, thus giving each one a probability of being selected.

For the purpose of our experiments with the multiway cut problem, each row of the table represents a multiway cut and each column represents an edge. The value $M(i, j)$ of a particular entry is either 1 or 0 depending on whether cut $i$ includes edge $j$. Player A chooses a mixed strategy $a$ represented by a vector indicating the probability for each $a_i$ of choosing cut $i$ (i.e. row $i$ of the table). (Alternatively, each $a_i$ can be viewed as the probability of choosing cut $i$ randomly from a distribution of potential cuts.) Player B chooses a strategy $b$ represented by a vector indicating the probability $b_j$ of choosing edge $j$ (i.e. column $j$ of the table). Note that $a$ corresponds to a cutting scheme, and $b$ corresponds to the distribution of edge weights in a bad graph.

Using this representation, the problem from player A’s perspective is to find

$$\min_a \max_b (a^T M b),$$

(4.2)

over all probability distributions $a$ and $b$. This is a mixed strategy that defines an optimal cutting scheme as a distribution over all possible cuts. Similarly, the problem from player B’s perspective
is to find

\[
\max_b \min_a (a^T Mb),
\]  

over all probability distributions \(a\) and \(b\). This mixed strategy defines an optimal bad graph as a distribution over edge weights, and is equivalent to the bad graph LP (LP8).

The minimax theorem of John von Neumann [80] plays an important role here, to help motivate our study of bad graphs. This famous theorem tells us that if and only if two problems are strong duals (as the bad graph problem and the cutting scheme problem are), then their optimal values will be equal. Thus, we can find both 1) an optimal set of bad graph edge weights \(b\) and 2) an optimal cutting scheme \(a\), such that both solution values (4.2) and (4.3) are equal to \(a^T Mb\). As a consequence, we know that an optimal cutting scheme is optimal for all graphs, not just the bad graph it was computed to optimally cut. Therefore, we focus our experiments not only on finding the bad graph, but also on the dual problem of finding the optimal cutting scheme.
One can always find a pair of solutions to the optimal cutting scheme problem and the optimal bad graph problem that are symmetric about the simplex (so that the solution values do not change by permuting the terminals), by averaging together all $r!$ possible symmetries of an optimal solution. Hence, we can develop more compact formulations of these problems by using canonical edges, canonical vertices, and canonical cuts within discretized simplexes. A canonical edge represents all edges in a given discretized simplex that are indistinguishable by symmetry. For example, in any 3-simplex or 4-simplex, we consider all edges directly incident to terminals to be copies of a single canonical edge having multiplicity equal to the number of copies. All these edges are structurally identical due to symmetry and are therefore treated as copies of a single canonical edge. See Figure 4.3 for an illustration of canonical edges as well as canonical vertices, which are directly analogous. Canonical cuts are similarly analogous; all cuts whose slice distances form the same unordered set are considered to be copies of a single canonical cut; the order in which terminals are cut does not matter, structurally.

4.2 Experimental Avenues

We now discuss each of the three avenues we have explored in our experiments. First, we discuss a row generation technique used to obtain bad graphs for the $r = 4$ case for 3, 4, and 5 discretizations. Though this technique is computationally intensive, the bad graphs we do obtain in this fashion are optimal. Second, we discuss our attempts to decompose the bad graph LP into paths instead of edges; this reduces the number of variables considerably, but unfortunately we obtain non-optimal bad graphs using this technique. Third and finally, we discuss our efforts to find SPARC bad graphs by considering the dual problem of finding an optimal SPARC cutting scheme.

4.2.1 Row Generation

Our investigation of row generation is motivated by the fact that the bad graph LP (LP8) is difficult to solve due to the number of constraints. So, our goal is to begin with a restricted (e.g. SPARC) version of the problem with a relatively small number of constraints, then use a row generation technique to add additional constraints one at a time as needed until we reach an optimal solution over all possible cuts. Here and in the rest of this dissertation, we use ILOG CPLEX 10 in situations like this to solve all instances of LPs and IPs.
Our row generation technique generates a linear program (an instance of the bad graph LP, LP8) with only SPARC constraints, to solve for the canonical edge weights of a particular discretized simplex. The optimal solution to this problem we obtain from CPLEX gives us the optimal SPARC bad graph of this simplex. We then create a corresponding integer program to directly solve the multiway cut problem (an instance of IP1) with respect to the edge costs of the SPARC bad graph we obtained. Finally, we solve this IP using CPLEX to obtain the optimal integral multiway cut for the bad graph with the given edge costs.

Keep in mind that this optimal cut we obtain here is only optimal with respect to the original SPARC bad graph we first computed with our instance of the bad graph LP, LP8. So, we employ row generation to gradually refine our SPARC bad graph into a true bad graph with respect to all possible multiway cuts instead of merely SPARCs. This is accomplished by taking the minimum multiway cut which determined the solution to the IP and adding this cut as a new cut constraint in the LP. We then re-solve this updated LP and use the bad graph edge weights of that solution to recreate and resolve an updated IP. We repeat the process until the minimum multiway cut costs from successive iterations converge, at which point we have arrived at the optimal bad graph and corresponding optimal multiway cut. Figure 4.1 lists the optimal bad graph edge weights we have obtained with this technique, for \( r = 4 \) and for 3, 4, and 5 discretizations; Figure 4.4 visually illustrates the \( n = 3 \) optimal bad graph.

Unfortunately, this process is computationally expensive, much more so than we had originally hoped. Directly solving an IP in the above manner using CPLEX is a brute-force endeavor with exponential runtime, and is only computationally feasible for a very small number of discretizations. CPLEX performs particularly slowly in our case, because many cuts are close to or tied with an optimal cut, due to the specific “worst case” structure of the bad graph. Thus, it is comparatively rare for CPLEX to be able to prune subtrees from consideration in its branch and bound strategy. Computing the bad graphs for 3 and 4 discretizations using row generation required only a few minutes of CPLEX computation for the minimum cut cost to converge. However, computing the bad graph for 5 discretizations took roughly two weeks of continuous CPLEX computation.

Our original intent with this direction of research was to compute optimal bad graphs for the largest values of \( n \) as are feasibly possible, then look for patterns in the edge weight distributions that could lead to an analytical description of the optimal cutting scheme over all possible cuts, not merely SPARCs. However, since it has proven computationally infeasible to compute bad graphs
Figure 4.4: An optimal bad graph for the 4-simplex with 3 discretizations, found with the row generation technique. We have displayed only one edge of each canonical type, labeled with its weight for clarity. Each vertex’s visual size is inversely proportional to our viewing distance, so vertices in the back appear smaller than in the front. See Figure 4.1 for a tabular list of the edge weights of this bad graph.
### Table 4.1: 4-terminal bad graphs for 3, 4, and 5 discretizations, listing the weight of each canonical edge. We represent each edge in the table with the coordinates of its adjacent vertices. The $i$th coordinate of a vertex is its distance from the face opposite terminal $i$. These edge weights are calculated using our row generation technique. The edge weights for the $n = 5$ case do not seem to have the clear proportions of the $n = 3$ and $n = 4$ edge weights. See Figure 4.4 for an illustration of $n = 3$.

<table>
<thead>
<tr>
<th>$n = 3$</th>
<th>$n = 4$</th>
<th>$n = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge</td>
<td>Wt</td>
<td>Edge</td>
</tr>
<tr>
<td>$(0, 0, 0, 1)$</td>
<td>2</td>
<td>$(0, 0, 0, 1)$</td>
</tr>
<tr>
<td>$(0, 0, \frac{1}{3}, \frac{2}{3})$</td>
<td>0</td>
<td>$(0, 0, \frac{1}{3}, \frac{2}{3})$</td>
</tr>
<tr>
<td>$(0, 0, \frac{2}{3}, \frac{1}{3})$</td>
<td>0</td>
<td>$(0, 0, \frac{2}{3}, \frac{1}{3})$</td>
</tr>
<tr>
<td>$(\frac{1}{3}, 0, \frac{2}{3}, 0)$</td>
<td>3</td>
<td>$(\frac{1}{3}, 0, \frac{2}{3}, 0)$</td>
</tr>
<tr>
<td>$(\frac{1}{3}, \frac{2}{3}, 0, 0)$</td>
<td>3</td>
<td>$(\frac{1}{3}, \frac{2}{3}, 0, 0)$</td>
</tr>
<tr>
<td>$(0, 0, \frac{1}{2}, \frac{1}{2})$</td>
<td>18</td>
<td>$(0, 0, \frac{1}{2}, \frac{1}{2})$</td>
</tr>
<tr>
<td>$(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$</td>
<td>2</td>
<td>$(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$</td>
</tr>
<tr>
<td>$(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$</td>
<td>2</td>
<td>$(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$</td>
</tr>
</tbody>
</table>

in this fashion for $n > 5$, and since the edge weight distributions we have obtained for $n = 5$ (see Figure 4.1) seem to lack any simple structure, this avenue of research seems not to lead in a fruitful direction. Still, we can make one more important observation from these row generation experiments, an observation also made in the 2002 thesis of Mosk-Aoyama [63]. The rows (i.e. new constraints) we add to the initial SPARC bad graph are convincing evidence that SPARCs are insufficient to characterize an optimal bad graph for the $r = 4$ case, even though SPARCs are previously known to be sufficient for $r = 3$ [51].

### 4.2.2 Path Decomposition

Our next line of experimentation is motivated by the fact that there exist optimal bad graphs that are *path decomposable* in the 3-terminal case; that is, they are accurately representable solely with terminal-terminal paths instead of individual edges. This is evidenced by the optimal 3-terminal bad graph found by [51], which Karger et al. describe as a collection of terminal-terminal
paths. Figure 4.5(a) depicts one such path of which their bad graph is composed; we describe them in more detail in a moment. The goal of our experiments here is to investigate the possibility of describing the \( r = 4 \) bad graph as a collection of similarly shaped terminal-terminal paths (e.g. Figure 4.5(b)).

We begin by describing an LP for finding the bad graph edge weights of a discretized simplex decomposed into terminal-terminal paths, such that the LP variables are (canonical) paths instead of edges, as follows.

\[
\text{Maximize} \quad \lambda \\
\text{(LP9) Subject to} \quad \sum_{p \in C} m(p, C)c(p) \geq \lambda \quad \forall \text{ multiway cuts } C \\
\sum_{p \in P} l(p)c(p) = 1 \\
c(p) \geq 0 \quad \forall p \in P,
\]

where \( P \) is the set of all terminal-terminal paths, and each \( C \subseteq P \) is a multiway cut consisting of a set of terminal-terminal paths. Each \( m(p, C) \) is a coefficient denoting how many times canonical path \( p \) intersects a canonical cut \( C \) in the instance. Each \( l(p) \) is a coefficient denoting the number of edges (i.e. the length) of canonical path \( p \), multiplied by the number of symmetries of \( p \). Note that each \( c(p) \) is a single value denoting the cost (or weight) assigned to the path variable \( p \), and does not
represent $\sum_{e \in p} c(e)$; the only variables in LP9 are path variables, which we use instead of the edge variables $c(e)$ from the original bad graph LP (LP8). LP9 also illustrates another reason we have investigated path decomposition: there are a factor of $O(n)$ fewer variables here than in the original bad graph LP (LP8). This makes each instance much smaller and therefore potentially much faster to solve, which in turn allows us to feasibly solve for bad graphs at higher levels of discretization.

Though we do not use edge variables in LP9, it is nonetheless useful to analytically describe a path in terms of the simplex edges of which the path consists. We will begin by describing the paths of the 3-simplex bad graph of [51], then describe the paths of LP9 as a higher dimensional generalization to the $r = 4$ case. The terminal-terminal paths $p$ constituting the 3-terminal bad graph of [51] each consist of three segments, each of which is the set of all edges $e$ that are $i,j$-aligned to a particular pair of terminals $(i, j)$. These edges are all adjacent and appear as a straight line. As illustrated in Figure 4.5(a), the first segment of an $i - j$ path between terminals $i$ and $j$ of height $h \leq n$ contains $h$ edges starting at terminal $i$, the middle segment contains the $i,j$-aligned edges going through the interior of the simplex, and the third segment contains $h$ edges ending at terminal $j$; visually, these three segments appear as three sides of a trapezoid.

As illustrated in Figure 4.5(b), the $i, j$ paths we use for the 4-simplex contain up to 5 segments, and each path corresponds to a unique vertex $x$ on the face opposite terminal $j$. To carry over the canonical idea to this context, we include only canonical paths in our LP by removing those paths that duplicate the shape of an existing one, and we also only include paths between a single pair of terminals $i, j$ instead of between all pairs. Starting at terminal $i$, the first segment of a 4-simplex path is analogous to the first segment of a 3-simplex path. The second segment goes through the interior of the outer face opposite terminal $j$ to a special node $x$, from which the third segment passes through the interior of the 4-simplex to the other side, ending at node $y$ that corresponds to $x$ on the face opposite terminal $i$. From there, the fourth and fifth segments mirror the first and second, taking the path the remainder of the way to terminal $j$. Note that some of these paths do not pass through the interior of the 4-simplex and contain only 3 (or even 2) segments, thus they look like 3-simplex paths, depicted in Figure 4.5(a).

These $i, j$ paths can be defined inductively for all $r \geq 3$ using the paths of the bad graph in [51] as a base case for $r = 3$. The first half of the path (from $i$ to the middle segment) all travel along the outer face of the $r$-simplex. The middle segment passes through the interior of the $r$-simplex from one outer face containing $i$ to another face containing $j$. The rest of the path connects the
middle segment to \( j \) by exactly mirroring the first half of the path, but now on a different face. Our experiments, however, are conducted only on the \( r = 4 \) case.

Solving LP9 gives us bad graph edge costs that represent the simplex with the highest minimum SPARC multiway cut cost arising from a simplex decomposed into the 4-simplex paths we describe above. This method of computation is much faster than the row generation technique, allowing us to quickly obtain 4-terminal bad graphs up to approximately \( n = 25 \) on a single workstation. However, the bad graphs we obtain in this way are not optimal for \( r = 4 \). This is evident because the minimum multiway cut costs of these path-decomposed bad graphs are in each case slightly smaller than those of the corresponding bad graphs generated using the edge based formulation, which we know is optimal. It remains an open question whether we need another type of path in our LP, or whether optimal 4-simplex bad graphs (unlike the 3-simplex) are in general not decomposable into path structures.

As mentioned earlier, one of our motivations for considering a path based formulation is that it contains \( O(n) \) fewer variables than the edge based formulation. Our hope for this formulation was not only to speed up computation time, but also to help overcome the degeneracy evident in the bad graphs we obtained with an edge based formulation, even in small instances. This leads us to our next avenue of research, where we focus more directly on the issue of coping with the degeneracy and artifacts of discretization in our LP solutions.

### 4.2.3 The Bad Graph LP Dual Problem

Our final approach is to examine the dual of the bad graph LP for SPARCs, which is the problem of finding an optimal SPARC cutting scheme. In this problem, we have variables on cuts, constraints on edges, and we wish to assign a weight to each SPARC to minimize the maximum edge density. It can be stated formally as

\[
\text{Minimize} \quad \mu \\
\text{(LP10)} \quad \text{Subject to} \quad \sum_{c \ni e} w(c) \leq \mu \quad \forall e \in E \\
\sum_{c \in C} w(c) = 1 \\
w(c) \geq 0 \quad \forall c \in C,
\]

where \( w(c) \) represents the weight assigned to SPARC cut \( c \), and \( C \) is the set of all such cuts. When we solve an instance of this problem, we obtain a cutting scheme in its most basic form: a 2D (for 3-
(for 4-terminal) or 3D (for 4-terminal) array of weights on every possible SPARC cut of the simplex. Since this cutting scheme is described only as a probability distribution over cuts, rather than with an analytical description (e.g., the cutting scheme from Chapter 3), we call this a raw cutting scheme. See Figures 4.6(a) and 4.8 for illustrations of raw cutting scheme output from a \( r = 3 \) and a \( r = 4 \) instance of LP10, respectively. Our ultimate goal is to deduce an analytical cutting scheme, using these raw cutting schemes to provide hints about the structure our cutting scheme needs.

Other authors, particularly Mosk-Aoyama [63], have performed extensive computational experiments on cutting scheme LPs for the \( r = 4 \) multiway cut problem. Mosk-Aoyama introduces new classes of hyperplane slices of the \( r \)-simplex with are not side-parallel. These include the pair-isolating slice, which separates two terminals from the other two, and the edge-perpendicular slice, which separates the remaining region of the simplex on one side of a pair-isolating slice into two regions of the same shape. He uses both SPARCs and these new types of slices to achieve a cutting scheme yielding an approximation ratio for \( r = 4 \) of \( \frac{56455889}{48000000} < 1.1762 \), which is better than the \( < 1.189 \) approximation guarantee of the \( r = 4 \) case of the Karger et al. [51] algorithm.

There are at least two main problems with “interpreting” raw cutting schemes to determine what the underlying analytical cutting scheme looks like. First of all, there are sometimes artifacts due to discretization, which presumably diminish if we can solve instances for large enough \( n \). Second, there is substantial degeneracy in raw cutting schemes (see Figures 4.6(a) and 4.8). This causes many solutions to tie for optimal, and CPLEX may choose one that has no apparent structure. In our cutting scheme experiments with LP10, we sought to minimize the degeneracy of our LP solutions by adding additional constraints that we call speculative constraints, which represent symmetries or other properties that appear to be true in an optimal cutting scheme, based on inspection of raw LP solutions. These constraints also significantly improve runtime, and represent significant insight into the nature of optimal cutting schemes. Coded separately for 3 and the 4 terminal cases, these speculative constraints include axes of symmetry (where each primary axis is the cutting distance of a different side-parallel slice in the SPARC) as well as specifying all-zero regions of the cutting scheme’s array. Since the solution value is essentially unchanged regardless of the presence of these speculative constraints (modulo some small artifacts affecting the solution value by a relative amount < 0.001, due to discretization), these symmetries are inherent “properties” that optimal solutions to LP10 exhibit without loss of generality, furthering our characterization of optimal cutting schemes for the multiway cut problem on 4 terminals. By including speculative constraints, we reduce the
degeneracy of our solutions by eliminating as many solutions as possible which would otherwise be tied for optimal, leaving us with more well-structured solutions.

We will now detail each of the speculative constraints. The first two are simple, and are illustrated in Figure 4.6(b) for \( r = 3 \) and Figure 4.9 for \( r = 4 \). One constraint ensures that we only consider ball cuts and corner cuts (discussed in detail in Chapter 3), which is consistent with the proof in [51] that there exists an optimal cutting scheme taking this form. This constraint ensures that all cuts in neither of these two regions have weight 0 (i.e., white pixels); it is only implemented for \( r = 3 \). Another constraint ensures that the only corner cuts with nonzero weight are those with all slices at the same distance. This constraint causes the black diagonal line (indicating high probability) in the corner cut region in Figure 4.6(b) for \( r = 3 \), surrounded by white pixels indicating zero probability, and causes the same diagonal line for corner cuts in Figure 4.9 for \( r = 4 \). The next constraint we add, illustrated in Figure 4.6(c) for \( r = 3 \) and Figure 4.10 for \( r = 4 \), is symmetry about the \( x = y \) axis in the ball region, and is self-evident in light of our discussions of canonical cuts. In the \( r = 4 \) case, we have implemented an axis of symmetry about the \( x = y \) plane of the ball region, but not the \( x = z \) or \( y = z \) planes. The final constraint we add is illustrated in Figure 4.6(d) for \( r = 3 \). It is another axis of symmetry in the ball region: \( x + y = n \frac{2}{3} - 1 \). Though the formula may seem unintuitive, this axis is perpendicular to the \( x = y \) axis inside the ball region, intersecting at the center of the ball region at the point \((\frac{2}{3}n, \frac{2}{3}n)\). As seen in Figure 4.6(d), there are a total of two perpendicular axes of symmetry implemented. In the \( r = 4 \) case, we have not implemented any axes of symmetry to correspond to the \( x + y = n \frac{2}{3} - 1 \) symmetry of \( r = 3 \), as they caused a suspicious increase in solution value (e.g., for \( n = 36 \), they caused an increase from 1.132405 to 1.145872). This leads us to suspect in the absence of any errors in our code that such constraints are never present in optimal \( r = 4 \) solutions, and are therefore “invalid” speculative constraints in that case.

One of our goals with this avenue of research was to develop an analytical cutting scheme for \( r = 4 \) based on the raw cutting schemes output by this dual method. This is precisely the way the cutting scheme used to prove the \( \frac{12}{11} \) result of Karger et al. [51] was developed for \( r = 3 \). They used a raw \( r = 3 \) cutting scheme (such as the one in Figure 4.6(a)) to deduce an optimal analytical cutting scheme description (illustrated in Figure 4.7). In an effort to follow this line of research in the \( r = 4 \) case, we have implemented speculative constraints as discussed above for \( r = 3 \) and the \( r = 4 \). Unfortunately, even with all speculative constraints added, our \( r = 4 \) solutions still seem to lack obvious structure (e.g., Figure 4.10). So, we have been unable to analytically describe an
optimal cutting scheme for \( r = 4 \) based on raw cutting schemes from running instances of LP10; this remains an open problem. Nonetheless, our speculative constraints are a set of properties that partially describe optimal cutting schemes for \( r = 4 \).

As demonstrated by various authors (e.g., [9, 51]), the simplex embedding approach provides very useful structure for the \( r = 3 \) case. It is disappointing that our experiments and those of others have not found such comparatively nice structure for \( r \geq 4 \). Perhaps the structure of the problem becomes significantly more complicated beyond the \( r = 3 \) case, or perhaps the simplex embedding LP needs to be strengthened with additional constraints in order to provide sufficient structure to provide tight approximation results for \( r \geq 4 \).
Figure 4.6: Figure 4.6(a) is a visualization of a solved $r = 3$ instance of LP10: a raw cutting scheme consisting of a distribution of weight on each SPARC cut of a 3-simplex with 72 discretizations. Slice distance from the two terminals form the x and y axes; we use whole number slice distances from 1 to $n$ for simplicity. The darkness of each pixel $(x, y)$ is directly related to the weight (or probability of selecting) the SPARC with corresponding slice distances $x$ and $y$. There is significant degeneracy present. Figure 4.6(b) is a raw cutting scheme output for the same instance, with the added constraints that all corner cuts must have equal weight and all cuts that are neither a ball cut nor a corner cut must have 0 weight. Figure 4.6(c) adds the additional constraint of symmetry about the $x = y$ axis in the ball region. Figure 4.6(d) adds another constraint, enforcing symmetry about the $x+y = n\frac{2}{3} = \frac{1}{2}$ diagonal, which is perpendicular to the $x = y$ axis in the ball region. Notice the degeneracy is significantly less than Figure 4.6(a), yet the resulting distribution still provides only a fleeting glimpse of the structure of the true optimal analytical cutting scheme shown in Figure 4.7.
Figure 4.7: An illustration of the \( \frac{12}{17} \) cutting scheme of Karger et al. [51] in the same format as Figure 4.6. Their algorithm is equivalent to the cut distribution depicted here. The bottom right corner containing solid black corresponds to their corner cut, and is equivalent to the diagonal line in the bottom right corner of Figure 4.6(b), due to a valid speculative constraint. The black star shape on the top left corresponds to their ball cut.
Figure 4.8: This figure depicts a raw cutting scheme output from an instance of LP10 with $r = 4$ and $n = 36$. At the top left corner, we see a three-dimensional view, where each point $(x, y, z)$ is gray if a SPARC with slice distances $x, y,$ and $z$ has probability $> 0$ in the solution. In the top right, bottom left, and bottom right corners, we see the three two-dimensional marginals.
Figure 4.9: This figure depicts a raw cutting scheme output from an instance of LP10 with $r = 4$ and $n = 36$, where we also enforce the speculative constraint that the only corner cuts with nonzero weight are those with all slices at the same distance. At the top left corner, we see a three-dimensional view, where each point $(x, y, z)$ is gray if a SPARC with slice distances $x$, $y$, and $z$ has probability $> 0$ in the solution. In the top right, bottom left, and bottom right corners, we see the three two-dimensional marginals.
Figure 4.10: This figure depicts a raw cutting scheme output from an instance of LP10 with $r = 4$ and $n = 36$, where we enforce two speculative constraints: 1) that the only corner cuts with nonzero weight are those with all slices at the same distance, and 2) an axis of symmetry about the $x = y$ axis in the ball region. At the top left corner, we see a three-dimensional view, where each point $(x, y, z)$ is gray if a SPARC with slice distances $x$, $y$, and $z$ has probability $> 0$ in the solution. In the top right, bottom left, and bottom right corners, we see the three two-dimensional marginals.
Chapter 5

$k$-hurdle Multiway Cut

In this chapter, we present results on the $k$-hurdle multiway cut problem, reproduced from the author’s work in [18] and later in [17]. First, we demonstrate a $(1, k - 1)$-pseudo approximation result, obtaining a $(k - 1)$-hurdle solution whose cost is no more than the optimal cost of a $k$-hurdle solution. We also demonstrate the half-integrality of our $k$-hurdle multiway cut LP, and extend that to a true approximation algorithm for the problem, guaranteeing all hurdles with an approximation ratio of $2(1 - \frac{1}{r})$. Finally, we show that it may be difficult to improve upon this approximation result by showing an approximation-preserving reduction from the well-studied vertex cover problem.

To begin, let $t_1 \ldots t_r$ denote a set of terminals, and let $P$ denote the set of all terminal-to-terminal paths. We reproduce the NP-hard $k$-hurdle multiway cut problem’s IP,

$$\text{OPT} = \text{Minimize} \sum_{e \in E} x(e)c(e)$$

(IP2) Subject to $x(p) \geq k \quad \forall p \in P$

$$x(e) \in \{0, 1\} \quad \forall e \in E,$$

where $x(p)$ denotes $\sum x(e)$ of the edges $e$ on a path $p$. The corresponding LP relaxation we denote by $LP2$. We begin our study of the $k$-hurdle multiway cut problem by showing the following pseudo approximation result.

**Theorem 1.** In polynomial time, one can obtain a $(k - 1)$-hurdle solution of cost at most $OPT$, where $OPT$ denotes the cost of an optimal solution with $k$ hurdles.

Let $x$ be an optimal solution to $LP2$ of cost $z_{LP2}$, and let $d_x(u, v)$ denote the shortest path distance from $u$ to $v$ using edge lengths $x$. Choose $\alpha$ uniformly at random from $(0, 1)$ and consider
making concentric cuts around each terminal as follows. We define \( E_{i,\rho} \) as the set of edges \( uv \in E \) such that \( \rho \in \left[ \min\{d_x(t_i, u), d_x(t_i, v)\}, \max\{d_x(t_i, u), d_x(t_i, v)\} \right] \). Note that \( E_{i,\rho} \) contains all edges crossing the boundary of a ball of radius \( \rho \) around terminal \( t_i \); for \( \rho \leq k/2 \), the set \( E_{i,\rho} \) is therefore nonempty, or else our graph would contain no path from \( t_i \) to the other terminals. Let \( E_i \) denote the union of \( E_{i,\rho} \) over all \( \rho \in \{\alpha, 1 + \alpha, 2 + \alpha, \ldots\} \cap [0, [k/2]] \). The “cutset” \( E_i \) contains the edges chosen by \( t_i \) for inclusion in our cut. Visually, we think of \( E_i \) as defined by a set of concentric rings emanating out from \( t_i \) at distances \( \alpha, 1 + \alpha \), and so on. These rings do not overlap, since they extend to a maximum distance of \( k/2 \) from each terminal (whereas all terminals lie at distance at least \( k \) from each-other). However, if \( k \) is even, an edge \( e \) straddling the frontier at mutual distance \( k/2 \) from two terminals \( t_i \) and \( t_j \) could be included in both \( E_i \) and \( E_j \); otherwise, \( e \) will belong to at most one cutset \( E_i \), since otherwise we would have \( x(e) \geq 1 \), as each pair of consecutive concentric rings is spaced one unit apart. We now show that the set \( S = \bigcup_{i=1}^r E_i \) is a \((k-1)\)-hurdle multiway cut whose cost is at most \( z_{LP2} \leq OPT \) in expectation.

**Lemma 2.** The set \( S \) is a \((k-1)\)-hurdle multiway cut.

*Proof.* Consider any path \( p \in P \) connecting some terminal \( t_i \) to some other terminal \( t_j \). Let \( p_i = p \cap E_i \) and \( p_j = p \cap E_j \). Suppose first that \( k \) is odd, in which case \( |p_i| \geq (k-1)/2 \), \( |p_j| \geq (k-1)/2 \), and \( p_i \cap p_j = \emptyset \), from which it follows that \( |p \cap S| \geq k-1 \). On the other hand, if \( k \) is even, then \( |p_i| \geq k/2 \) and \( |p_j| \geq k/2 \) but one edge \( e \) in \( p \) might appear in \( p_i \cap p_j \), so again \( |p \cap S| \geq k-1 \).

**Lemma 3.** \( E[c(S)] \leq z_{LP2} \).

*Proof.* Consider an edge \( e = uv \) for which \( d_x(t_i, u) < k/2 \) and \( d_x(t_i, v) < k/2 \) for some terminal \( i \) (i.e., the edge \( e \) belongs to the ball of radius \( k/2 \) surrounding \( t_i \)). Assuming \( d_x(t_i, u) \leq d_x(t_i, v) \), the triangle inequality gives us \( \Pr[e \in S] = d_x(t_i, v) - d_x(t_i, u) \leq x(e) \). It is also true that \( \Pr[e \in S] \leq x(e) \) in the special case where \( e \) belongs to two different cutsets \( E_i \) and \( E_j \); this is justified by splitting \( e \) into two virtual edges \( e' \) and \( e'' \) with \( x(e') + x(e'') = x(e) \) such that \( e' \) and \( e'' \) belong to a single cutset. The analysis above then gives us \( \Pr[e \in S] \leq \Pr[e' \in S] + \Pr[e'' \in S] \leq x(e) \). Therefore, \( E[c(S)] = \sum e c(e) \Pr[e \in S] \leq \sum e c(e) x(e) = z_{LP2} \). 

We note that this approach can be easily derandomized using standard techniques. For every vertex \( v \) and terminal \( t_i \) such that \( d_x(t_i, v) \) is minimized, we call the value \( d_x(t_i, v) \) critical (we also define 0 and 1 as critical values). It suffices to try one choice for \( \alpha \) between every consecutive
pair of critical values, since all such $\alpha$ values result in the same cutsets. Since expected solution cost is at most $z_{LP2}$, one of our selections for $\alpha$ must yield a solution $S$ with $c(S) \leq z_{LP2}$.

A slight variation on the argument above allows us to prove that $LP2$ is half-integral, thereby giving an immediate 2-approximation algorithm for the $k$-hurdle multiway cut problem.

**Theorem 4.** $LP2$ is half-integral.

**Proof.** Let $x$ be an optimal fractional solution to $LP2$. Select $\alpha \in (0, 1/2)$ uniformly at random. For each terminal $i$, construct $k$ sets of edges $E_{i,\rho}$ as above for $\rho \in \{\alpha, 1-\alpha, 1+\alpha, 2-\alpha, 2+\alpha, 3-\alpha, \ldots\} \cap [0, k/2]$. Set $x^*(e)$ to half the total number of sets $E_{i,\rho}$ in which edge $e$ appears. We claim that the half-integral solution $x^*$ is optimal for $LP2$. To show that $x^*$ is feasible for $LP2$, note that $x^*(e) \leq 1$ for every edge $e$. This is clear for the case of an edge $e = uv$ where both $u$ and $v$ belong to the same cutset $E_i$, since otherwise we would have $e \in E_{i,\rho} \cap E_{i,\rho'}$ for $|\rho - \rho'| \geq 1$, so $x(e) > 1$. Essentially the same justification applies in the case that $u$ and $v$ belong to different cutsets $E_i$ and $E_j$. If $e \in E_{i,\rho}$ and $e \in E_{j,\rho'} \cap E_{j,\rho''}$ with $\rho' < \rho''$, then the key observation is that $|(k/2 - \rho) + (k/2 - \rho')| \geq 1$, so $x(e) > 1$ by the fact that the balls of radius $k/2$ around $t_i$ and $t_j$ are disjoint. Observe also that $x^*(p) \geq k$ for any $p \in P$ connecting terminal $t_i$ to $t_j$, since each of the $2k$ total sets $E_{i,\rho}$ and $E_{j,\rho}$ contributes $1/2$. By linearity of expectation, we have $E[x^*(e)] \leq x(e)$ for each edge $e$, so letting $C$ denote the cost of the solution $x^*$, we have $E[C] = E[\sum_e c(e)x^*(e)] \leq \sum_e c(e)x(e) = z_{LP2}$. However, since $C \geq z_{LP2}$ always holds, we conclude that $C = z_{LP2}$ irrespective of $\alpha$. \hfill \Box

We remark that if we replace edges with vertices, this same proof also establishes half-integrality for the analogous LP relaxation of the vertex $k$-hurdle multiway cut problem. Garg et al. [31] have previously proved half-integrality for the vertex version of the standard multiway cut problem, so our argument above not only gives a simpler alternative proof of this result, but it also provides a generalization to multiple layers of hurdles. We use the half-integrality of $LP2$ to obtain a improved approximation algorithm for small values of $r$.

**Theorem 5.** In polynomial time, one can obtain a $k$-hurdle solution of cost at most $2(1 - \frac{1}{r})z_{LP2}$.

We construct $r$ different $k$-hurdle solutions, $S_1, \ldots, S_r$, such that at least one solution has cost at most $2(1 - \frac{1}{r})z_{LP2}$. Let $x$ be an optimal half-integral solution to $LP2$; an edge $e$ is called a $1/2$-edge if $x_e = 1/2$. We employ a cutting scheme similar to the ones used above except that our solutions are constructed deterministically. Denote by $A_i$ the union of $E_{i,\rho}$ over all $\rho \in \{0, 1, 2, \ldots, \lfloor (k-1)/2 \rfloor\}$;
similarly, define $B_i$ as the union of $E_{i, \rho}$ over all $\rho \in \{1/2, 3/2, 5/2, \ldots, \lfloor k/2 \rfloor - 1/2\}$ ($B_i = \emptyset$ when $k = 1$). We set, for $1 \leq i \leq r$,

$$S_i = B_i \cup \bigcup_{j \neq i} A_j,$$

and proceed to show that each $S_i$ is a feasible solution and that at least one has the desired cost.

**Lemma 6.** Each edge set $S_i$ is a $k$-hurdle multiway cut.

**Proof.** Call an edge $uv$ internal to a terminal $t_i$ if $\max\{d_x(t_i, u), d_x(t_i, v)\} \leq k/2$. Observe that if $uv$ is internal to $t_i$ than it may not belong to any set $A_l$ or $B_l$ for $l \neq i$. Our argument proceeds along the lines of Lemma 2, considering any path $p \in P$ connecting terminals $t_i$ and $t_j$. Each set $S_i$ is comprised of $r$ sets where $r - 1$ of the sets lie in $\{A_1, \ldots, A_r\}$. Thus it suffices to show that $|p \cap S| \geq k$ where $S = A_i \cup A_j$ or $S = B_i \cup A_j$. First suppose $k$ is even, in which case $|p \cap A_i| = |p \cap A_j| = |p \cap B_i| = k/2$. When $k$ is even, $A_i$ may only contain edges internal to $t_i$, hence $A_i \cap A_j = B_i \cap A_j = \emptyset$, giving us the desired result.

If $k$ is odd, then $|p \cap A_i| = |p \cap A_j| = (k + 1)/2$, and $|p \cap B_i| = (k - 1)/2$; this time $B_i$ contains only internal edges to $t_i$, yielding $|p \cap (B_i \cup A_j)| = k$. To deduce that $|p \cap (A_i \cup A_j)| \geq k$, it suffices to observe that at most one edge on $p$ lies in both $A_i$ and $A_j$. \qed

**Lemma 7.** Let $\chi(F)$ be the incidence vector of an edge set $F \subseteq E$. We may use the integral solutions $S_1, \ldots, S_r$ to construct an approximate convex decomposition of $x$:

$$2 \left(1 - \frac{1}{r}\right) \cdot x \geq \sum_{1 \leq i \leq r} \frac{1}{r} \cdot \chi(S_i).$$

**Proof.** The claim is equivalent to $(r - 1)(2 \cdot x) \geq \sum_{1 \leq i \leq r} \chi(S_i)$, and it suffices to show that each 1/2-edge appears in at most $r - 1$ of $S_1, \ldots, S_r$. Note that edges $e$ with $x_e = 1$ are free to appear in each $S_i$ since $2 \cdot x_e = 2 \geq r/(r - 1)$ for $r \geq 2$.

Since $x$ is half-integral, for each terminal $t_i$ and vertex $v$ we have that $d_x(t_i, v)$ is also half-integral. Consequently each 1/2-edge must be internal to some terminal. Coupled with the fact that each 1/2-edge may belong to at most one of $A_i$ and $B_i$ for any $i$, we see that each such edge belongs to at most one set among $A_1, \ldots, A_r, B_1, \ldots, B_r$. If a 1/2-edge $e$ belongs to $A_j$ for some $j$, then it is missed by the solution $S_j$. On the other hand if $e \in B_j$, then $e$ is missed by all $S_i$ with $i \neq j$. \qed

It follows directly from Lemma 7 that there exists a solution $S_i$ with $c(S_i) \leq 2(1 - \frac{1}{r})(c \cdot x) = \ldots$
2(1−\frac{1}{r})z_{LP}. We can indeed find such a solution in polynomial time by calculating \(\arg\min_{1 \leq i \leq r} c(B_i - A_i)\) and selecting \(S_l\) as the solution where \(l\) is a minimizer.

It is tempting to think that one may be able to substantially improve the \(2(1 - \frac{1}{r})\) approximation guarantee by designing a stronger relaxation in the same vein as the standard multiway cut problem. However, we show by a simple observation that, perhaps surprisingly, this is likely not the case.

**Proposition 8.** There is an approximation preserving reduction from vertex cover to \(k\)-hurdle multiway cut for any \(k > 1\).

**Proof.** Suppose we are given an instance of weighted vertex cover \(G = (V, E, w)\). From \(G\) we construct \(G' = (V', E', T', c)\), an instance of \(k\)-hurdle multiway cut where we map the vertices of \(G\) to terminals of \(G'\), setting \(T' = V\). Moreover, for each \(t_i \in T'\), we include an additional vertex \(v_i\) and edge \((t_i, v_i)\) of cost \(w_i\). For each \(ij \in E\), we include in \(G'\) a path of length \(k - 1\) (adding new vertices and edges as demanded) between \(v_i\) and \(v_j\), where each edge of the path is assigned a cost of 0. This completes the construction.

We may restrict our attention to \(k\)-hurdle solutions in \(G'\) which select all the 0 cost edges not incident upon some terminal. Consider an edge of \(G\), \(ij \in E\); in order to satisfy the \(k\)-hurdle requirement between \(t_i\) and \(t_j\) in \(G'\), any feasible \(k\)-hurdle solution must select at least one of \((t_i, v_i)\) or \((t_j, v_j)\). On the other hand if \(i, j \in V\) but \(ij \notin E\), any \(t_i\)-\(t_j\) path contains at least \(k\) edges of 0 cost. Associating \((t_i, v_i) \in E'\) with \(i \in V\), we see that there is a cost preserving correspondence between vertex covers in \(G\) and \(k\)-hurdle multiway cut solutions in \(G'\).

We note also that the technique of embedding a graph into a simplex, pioneered by Călinescu et al. [9] for the \(k = 1\) case, seems not to extend in a straightforward manner for \(k > 1\) hurdles (in the early, conference version [18] of the author’s paper on this subject, it was erroneously claimed that such an embedding could lead to an alternative means of achieving a \(2(1 - \frac{1}{r})\)-approximation algorithm).
Chapter 6

$k$-hurdle Multicut

In this chapter, we present results on the $k$-hurdle multicut problem, some of which are reproduced from the author’s work in [18] and later in [17]. First, we show a $(\log(n), [(1-\varepsilon)k_{\max}])$-pseudo approximation algorithm for this problem based on the “region-growing” algorithm of Garg et al. [29], where $k_{\max} = \max_{i,j} k_{ij}$, the maximum hurdle count. With an approximation guarantee of $O(\log(n))$ and a runtime with polynomial dependence on $1/\varepsilon$, we ensure that at most an $\varepsilon$ fraction of the requested $k_{ij}$ hurdles are missing between each terminal pair $(i, j)$ in the solution. If each $k_{ij} = O(1)$, the algorithm satisfies all hurdle requirements and becomes a true $O(\log(n))$-approximation algorithm for this case. Next, we show a 2-approximation algorithm for trees, and demonstrate that this algorithm also works with little modification in the vertex variant of the multicut problem, where we seek to remove vertices instead of edges. Finally, we consider another special case of the multicut problem, where the hurdle counts $k_{ij}$ form an ultrametric, and provide a 2-approximation algorithm for this case based on half-integrality. As may be implied by comparing the stated results of this chapter with the previous one, the $k$-hurdle multicut problem seems somewhat more difficult to approximate than the $k$-hurdle multiway cut problem (or the classical multicut problem), particularly if we seek a solution that obtains all the required hurdles.

We begin by reproducing the IP formulation of the $k$-hurdle multicut problem,

\[
OPT = \text{Minimize} \quad \sum_{e \in E} x(e) c(e)
\]

subject to

\[
x(p) \geq k_i \quad \forall i \in \{1, \ldots, r\}, p \in P_i
\]

\[
x(e) \in \{0, 1\} \quad \forall e \in E,
\]
where $P_i$ denotes the set of all paths connecting the source and sink of commodity $i$ (we use one subscript to denote a particular hurdle count $k_i$, except in the ultrametric special case discussed later in this chapter). We denote IP3’s natural LP relaxation by $LP3$, with optimal solution cost $z_{LP3}$. Our pseudo approximation algorithm is as follows:

1. Solve our instance $I$ of $LP3$ to obtain an optimal fractional solution $x$ (this can be done in polynomial time with the ellipsoid algorithm).

2. Set $\delta = \varepsilon^2$ and let $x = x/\delta$.

3. Generate a new instance $I'$ by adding a source-sink pair $(u, v)$ for each $u, v \in V$ such that $|d_\pi(s_i, u) - d_\pi(s_i, v)| \geq 1$ for some existing source-sink terminal pair $(s_i, t_i)$. Note that these pairs are unordered; the order will not affect the algorithm’s outcome.

4. Use the region-growing algorithm of Garg, Vazirani, and Yannakakis [29] to round $x$ to an integer solution $x'$ for instance $I'$. Return $x'$.

Note that $x$ would be an optimal fractional solution for $LP3$ for the instance $I'$ only if $x_e \leq 1$ for all $e \in E$. However, the region-growing algorithm of Garg, Vazirani, and Yannakakis is not adversely affected by the fact that $x_e > 1$ for some edges $e$ — it still returns an integer solution of cost at most $O(\log R)$ times the objective value of our initial LP solution $\pi$, where $R$ is the number of commodities in the instance. In our case, since $R \leq n^2$ for the instance $I'$, we obtain a solution of cost at most $O(\log n) \times z_{LP3}/\delta = z_{LP3}/\varepsilon^2)$. The approximation ratio we obtain can therefore be written more precisely as $O(\log n/\varepsilon^2)$; as a result, any choice of $\varepsilon$ larger than a constant will negatively impact our approximation guarantee.

**Theorem 9.** The algorithm above returns an integer solution $x'$ for which $x'(p) \geq \lceil (1 - \varepsilon)k_i \rceil$ for every $i \in \{1, \ldots, r\}$ and $p \in P_i$.

**Proof.** We use the following fact about the region-growing algorithm of Garg, Vazirani, and Yannakakis: for each commodity $(s_i, t_i)$, it makes a cut at some radial distance $\rho_i \leq 1$ from either $s_i$ or $t_i$. Consider now any commodity $i \in \{1, \ldots, r\}$ and any path $p \in P_i$. As we walk along $p$ from $s_i$ to $t_i$, we acquire at least one hurdle for every $1 + 1/\delta$ units of distance traveled. More precisely, we show how to obtain at least $\lceil (\pi(p) - 1)/(1 + 1/\delta) \rceil$ hurdles. Since $\pi(p) \geq k_i/\delta$ and $\delta = \varepsilon^2$, we have

$$\frac{\pi(p) - 1}{1 + 1/\delta} \geq \frac{k_i/\delta - 1}{1 + 1/\delta} \geq \frac{1 - \delta}{1 + \delta} k_i = \frac{(1 - \varepsilon)(1 + \varepsilon)}{1 + \varepsilon^2} k_i \geq (1 - \varepsilon) k_i,$$
so we obtain at least \(\lceil (1 - \varepsilon)k_i \rceil\) hurdles.

Define \(q = \lceil \frac{\pi(p)}{(1 + 1/\delta)} \rceil\). Let \(v_0 = s_i\) and \(v_q = t_i\), and define \(v_i\) for \(i \in \{1, \ldots, q - 1\}\) as the farthest vertex along \(p\) from \(s_i\) such that \(s_i\) and \(v_i\) are no more than \(q(1 + 1/\delta)\) units of distance apart on \(p\). Let \(p_i, i = 1 \ldots q\), denote the subpath of \(p\) from \(v_{i-1}\) up to \(v_i\). We claim that \(x'(p_i) \geq 1\) for each \(i \in \{1 \ldots q - 1\}\), and that \(x'(p_q) \geq 1\) if \(\pi(p_q) \geq 1\). Consider any \(i \in \{1, \ldots, q - 1\}\). Since \(\pi(e) \leq 1/\delta\) for all \(e \in E\), we must have \(\pi(p_i) \geq 1\); otherwise, the first edge in \(p_{i+1}\) would rightly belong to the end of \(p_i\), since the length of \(p_i\) in this case would still be at most \(1 + 1/\delta\). Finally, if \(\pi(p_i) \geq 1\), then \(p_i\) must be cut by the algorithm of Garg, Vazirani, and Yannakakis, since a radial cut at distance \(\leq 1\) will be made from at least one endpoint of \(p_i\).

By setting \(\varepsilon\) appropriately, we obtain a true \(O(\log n)\)-approximation algorithm for the special case where \(k_i = O(1)\) for all commodities \(i\), for example a case in which we wish to set up only two or three redundant layers of “checkpoints” for inspecting traffic through a network, which we suspect may occur in many practical situations.

### 6.1 A 2-Approximation for Trees

We now focus on the special case of a tree. We note with some interest that the well-known primal-dual algorithm of Garg et al. [30] does not seem to generalize in a straightforward fashion to the \(k\)-hurdle case, especially for the “non-uniform” case where \(k_i\) can vary by commodity. For the uniform case where all \(k_i\) are equal, the authors tried a natural generalization of the primal-dual approach. However, it seems unable to match the 2-approximation algorithm we obtain by generalizing a more recent approach of Golovin et al. [37] and Levin and Segev [57] for the classical multiway cut problem in a tree (see also [11] and [33] for earlier instances of this general technique applied to different problems). We can state the \(k\)-hurdle multicut problem for trees as

\[
\text{OPT} = \text{Minimize} \sum_{e \in E} x(e)c(e) \\
\text{Subject to} \quad x(P_i) \geq k_i \quad \forall 1 \leq i \leq r \\
x(e) \in \{0, 1\} \quad \forall e \in E,
\]

where \(P_i\) denotes the unique path in the tree connecting the source and sink of commodity \(i\), and \(x(P_i)\) denotes \(\sum_{e \in P_i} x(e)\). The natural LP relaxation of \(\text{IP11}\) we denote by \(\text{LP11}\). Our approach is based on the following key property, mentioned in [37, 57].
Lemma 10. If each commodity \((s_i, t_i)\) is unidirectional (\(s_i\) and \(t_i\) having an ancestor-descendant relationship), then LP11 is totally unimodular.

We can therefore compute an integer optimal solution to LP11 in polynomial time if all commodities are unidirectional. In fact, we can compute such a solution in strongly polynomial time, since the dual of LP11 can be stated as a minimum-cost flow problem. In the unit cost case, we can even use a simple combinatorial greedy algorithm in lieu of solving LP11 [5]: root the tree and perform a postorder scan over its edges, setting \(x(e) = 1\) if \(|P_i \cap P_e| - x(P_i \cap P_e) = k_i - x(P_i)\) for any commodity \(i\) (here, \(P_i\) denotes the path from \(e\) up to the root).

Let \(x\) be an optimal solution to LP11 of cost \(z_{LP11}\). Although \(x\) may in general be non-integral, we can use \(x\) to construct a new instance of LP11, NLP, whose commodities are all unidirectional, and whose optimal (integral) solution gives us a 2-approximate solution to IP. For each commodity \(i\) that is not unidirectional, let \(u_i\) be the lowest common ancestor of \(s_i\) and \(t_i\), and replace \(i\) with two commodities \(i'\) and \(i''\), having source-sink pairs \((s_i', t_i') = (s_i, u_i)\) and \((s_i'', t_i'') = (u_i, t_i)\) and hurdle demands \(k_i' = \text{round}(x(P_{i'}))\) and \(k_i'' = \text{round}(x(P_{i''}))\) The function \(\text{round}(x)\) evaluates to \([x]\) if the fractional part of \(x\) is at least 0.5, and \([x]\) otherwise. This approach can be viewed as a strict generalization of [37, 57] for the simpler unit hurdle case of \(k_i = 1\), where \(i'\) and \(i''\) are each included in NLP only if \(x(P_{i'}) \geq 0.5\) or \(x(P_{i''}) \geq 0.5\), respectively.

Consider now an optimal integer-valued solution \(x^*\) to NLP. We know that \(x^*\) is feasible for IP since \(x^*(P_i) = x^*(P_{i'}) + x^*(P_{i''}) \geq k_i' + k_i'' = \text{round}(x(P_{i'})) + \text{round}(x(P_{i''})) \geq \lfloor x(P_i) \rfloor \geq k_i\) for each original commodity \(i\). We now only need to show that the cost of \(x^*\) is at most 2OPT. In [37, 57], this is easily achieved since LP11 in the unit hurdle case does require constraints of the form \(x(e) \leq 1\), so \(2x\) is a feasible solution for NLP, and therefore \(z_{NLP} \leq 2z_{LP11} \leq 2OPT\). In our case, however, \(2x\) may not be feasible for NLP, since doubling any \(x(e) > 0.5\) will violate the constraint that \(x(e) \leq 1\). However, by first truncating 2x, we can show an analogous result.

Lemma 11. The solution \(\pi = \min(2x, 1)\) is feasible for NLP.

Proof. Consider a particular unidirectional commodity \(j\) in NLP (so \(j = i'\) or \(j = i''\) for some commodity \(i\) in the original instance). We will show that \(\pi(P_j) \geq k_j\). Note that \(k_j\) was obtained by rounding \(x(P_j)\) up or down. If \(k_j\) was obtained by rounding \(x(P_j)\) down, then clearly \(\pi(P_j) \geq x(P_j) \geq k_j\). Henceforth, let us therefore assume \(k_j\) was obtained by rounding \(x(P_j)\) up, which implies that \([x(P_j)] - x(P_j) \leq 0.5\). Let \(L_j\) denote the set of “large” edges \(e \in P_j\) with \(x(e) \geq 0.5\)
and let $S_j$ denote the “small” edges $e \in P_j$ with $x(e) < 0.5$. We can express $\pi(P_j) = |L_j| + 2x(S_j)$. If $x(S_j) = 0$, then $\pi(P_j) = |L_j| \geq \lceil x(L_j) \rceil = \lceil x(P_j) \rceil = k_j$. Therefore, we assume $x(S_j) > 0$ and consider two cases:

1. $[x(S_j)] - x(S_j) \leq 0.5$. Here, since $x(S_j) \geq [x(S_j)] - 0.5$ and $x(S_j) \geq 0.5$, we have $2x(S_j) \geq [x(S_j)]$. Therefore, $\pi(P_j) = |L_j| + 2x(S_j) \geq |L_j| + [x(S_j)] = \lceil |L_j| + x(S_j) \rceil \geq \lceil x(L_j) + x(S_j) \rceil = \lceil x(P_j) \rceil = k_j$.

2. $[x(S_j)] - x(S_j) > 0.5$. By expanding out our assumption that $[x(P_j)] - x(P_j) \leq 0.5$, we obtain $0.5 \geq [x(S_j) + x(L_j)] - (x(S_j) + x(L_j))$, which implies that $x(L_j) > [x(S_j) + x(L_j)] - [x(S_j)]$. Since $|L_j| \geq x(L_j) > [x(S_j) + x(L_j)] - [x(S_j)]$ and $|L_j|$ is an integer, we have $|L_j| \geq [x(S_j) + x(L_j)] - [x(S_j)]$. Therefore, $\pi(P_j) = |L_j| + 2x(S_j) \geq |L_j| + [x(S_j)] \geq [x(S_j) + x(L_j)] = \lceil x(P_j) \rceil = k_j$.

Since $\pi$ is feasible for $NLP$ and its cost is at most $2z_{LP11}$, we have $z_{NLP} \leq 2z_{LP11} \leq 2OPT$, so our integer optimal solution $x^*$ to $NLP$ is a 2-approximation.

Note that the multicut problem on a tree $T$ is often phrased as the set cover problem in the special case of a tree-representable set system, which is sometimes called the tree-representable set cover problem [39]. In this special case of set cover, elements correspond to paths in $T$, sets correspond to edges of $T$, and each set contains the paths cut by the corresponding edge. The $k$-hurdle multicut problem on a tree admits a similar “tree-representable” interpretation for the general covering problem, yielding Theorem 12 below. The general covering problem is similar to minimum cost set cover, except we wish to cover all the elements $i$ not just once, but some number $b_i \geq 1$ times (analogous to the hurdle counts of the $k$-hurdle multicut problem). The formal description of the general covering problem is identical to that of general covering integer programming (IP5), except that in this case $A$ is a binary matrix.

**Theorem 12.** There exists a 2-approximation algorithm for the general covering problem in the special case where $A$ encodes a tree-representable set system.
6.2 $k$-Hurdle Vertex Multicut in Trees

The 2-approximation bound for the edge version of $k$-hurdle multicut in a tree will be challenging to improve, as there is a straightforward approximation-preserving reduction from vertex cover [30] that holds even in the case of $k = 1$ hurdle, unit edge weights, and unit-depth trees. For the case of vertex multicuts, however, things are slightly more interesting. The unweighted case for $k = 1$ can be solved in polynomial time with a simple greedy algorithm: perform a postorder scan of the tree, selecting each vertex $v$ if it is the only remaining vertex capable of separating an $(s_i, t_i)$ pair that is not yet cut ($v$ will be the LCA of $s_i$ and $t_i$). The weighted case for $k = 1$ is NP-hard, since we can transform a weighted edge multicut problem into a weighted vertex multicut problem by assigning every vertex a weight of $+\infty$ (thus ensuring they cannot be selected in any optimal solution), and by adding a “dummy” vertex in the middle of each edge with weight equal to the original weight of the edge. For $k = 2$, however, even the unweighted vertex problem is NP-hard (and challenging to approximate to within a factor better than 2), by a trivial extension of the approximation-preserving reduction in [30]. On the other hand, our 2-approximation algorithm above generalizes quite easily to the weighted vertex case for arbitrary $k$. In this case, when we split a commodity $(s_i, t_i)$ into two unidirectional commodities $(u_i, s_i)$ and $(u_i, t_i)$, we simply remove $u_i$ from one of them, making them vertex-disjoint. Our algorithm then proceeds as before.

6.3 The Ultrametric Case

Since the $k$-hurdle multicut problem has proven to be difficult to approximate well, we consider a restricted case of the problem. Consider the case where all hurdle counts $k_{uv}$ (denoting source terminal $i$ and sink terminal $j$) have some added structure. For example, these hurdle counts may form a metric, whereby the triangle inequality holds for any set of 3 terminals $x$, $y$, and $z$: $k_{xz} \leq k_{xy} + k_{yz}$. This restriction is still quite broad; we investigate a more restrictive special case where the hurdle counts form an ultrametric, whereby the triangle inequality on each triple of terminals $x$, $y$, and $z$ is replaced by a more restrictive inequality: $k_{xz} \leq \max(k_{xy}, k_{yz})$. This special case of the $k$-hurdle multicut problem is provably half-integral, thereby implying a 2-approximation. The metric case does not yet appear to have any such structure that we can exploit in a similar way, and remains an open problem.
6.3.1 Half Integrality

We seek to show the half-integrality of the natural LP relaxation of the multicut problem, in the case that all hurdle counts \( k_{ij} \) form an ultrametric. We begin with a randomized algorithm guaranteeing a half-integral solution with expected cost equal to the cost of the given linear solution. Afterward, we derandomize the algorithm to obtain a worst-case result. Our algorithm is inspired by Kruskal’s famous minimum spanning tree algorithm [50], which begins with a separate component for each vertex in the graph. During Kruskal’s algorithm, these components merge with new edges from the graph until, in the end, every vertex in the graph is connected together in a single connected component. Analogously, in our algorithm each terminal begins in a unique location in the graph, and over the course of the algorithm we gradually contract them together while rounding the edges to achieve half-integrality. At the end, all terminals have been contracted into one location and our half-integral solution is complete.

Theorem 13. The multicut problem admits a half-integral optimal solution, if the hurdle counts \( k_{uv} \) are integer and form an ultrametric.

Proof. We begin with a graph \( G = \{V, E\} \) and an optimal solution \( x \) of the multicut LP, with a cost of \( C = \text{cost}(x) = \sum_{e \in E} c(e)x(e) \). We seek to construct a solution \( x' \) such that \( \mathbb{E}[\text{cost}(x')] = C \) and \( x'(e) \in \{0, \frac{1}{2}, 1\}, \forall e \in E \). We will do this by rounding each \( x(e) \) to a half-integral value as described below. We will use a parameter \( \alpha \in (0, \frac{1}{2}) \), chosen uniformly at random. We proceed through a number of rounds equal to \( \max_{u,v} k_{uv} \); a round affects the \( x(e) \) values near each terminal \( t \), and consists of a subdivision step, a cutting step, and finally a rounding step. Once a particular round has completed at each terminal, all \( k_{uv} \) will be reduced by 1. We detail the three steps of a round, from the perspective of a terminal \( t \).

As we just mentioned, the first step of a round is to perform subdivision. Each edge \( e = (i, j) \) with exactly one vertex at distance \( \leq \frac{1}{2} \) from some terminal \( t \) (with respect to the \( x(e) \) values of all edges) is subdivided into two edges \((i, a)\) and \((a, j)\), which as a consequence introduces a new vertex \( a \) at distance \( \frac{1}{2} \) from \( t \) and removes edge \( e \). After subdividing in this way, we add a new constraint

\[
x(i, a) + x(a, j) = x(e), \tag{6.1}
\]

and set the new costs \( c(i, a) \) and \( c(a, j) \) both to \( c(e) \), to ensure that the total cost of our solution,
cost(x), remains unchanged.

As the next step in the round, we perform the cut. We set $\alpha = \frac{1}{2} - \alpha$ and place a concentric cut emanating from each terminal $t$ at distance $\alpha$; in this way, $\alpha$ alternates between two values, from one round to the next. Any edge with one endpoint having shortest path distance $< \alpha$ from $t$ and the other $> \alpha$ from $t$ is cut in this step.

Finally, we perform the rounding step, whereby all edges $e$ with both endpoints at distance $\leq 1/2$ from $t$ have $x'(e)$ set to half the number of times $e$ was cut in the previous step. Since, after subdivision, any edge can be cut at most once, an edge $e = (i, j)$ that was replaced through subdivision by two edges $(i, a)$ and $(a, j)$ will have $x'(e) \in \{0, \frac{1}{2}, 1\}$ due to the added constraint (6.1). To justify why an edge $e$ will never be subdivided into more than two edges, consider the situation where our problem instance contains only two terminals: $t$ and the closest other terminal $q$ to $e$. The subdivision of $e$ would be unaffected, and since we have only one hurdle count $k_{tq}$, the 1/2-integrality of $k$-hurdle multiway cut from Theorem 4 applies. These entirely half-integral $x'(e)$ values for these edges $e$ are saved, externally, for inclusion in the final half-integral solution $x'$. Then, we set all such $x'(e) = 0$ for the purposes of completing the algorithm, effectively “contracting” the region of radius $1/2$ around the terminal $t$ into a single point at $t$. As a consequence of completing a round at every terminal $t$, the shortest-path distance between every terminal pair is reduced by 1. Thus, we then subtract 1 from all $k_{uv}$, and the next round can begin.

When $k_{uv}$ reaches 0 for some pair of terminals $u$ and $v$, these terminals are contracted together into a single terminal $p$. Any such terminal $v$ contracted to $p$ will have all of its hurdle counts $k_{vq}$ to other terminals $q$ replaced by new hurdle counts $k_{pq}$ of the same value. The ultrametric property of our hurdle counts guarantees $k_{vq} = k_{pq}$, for any such terminal $v$ contracted to $p$. In this way, we uncover the laminar, hierarchical tree structure of our hurdle counts as a side effect of the algorithm.

After a number of rounds equal to $\max_{u,v} k_{uv}$, all $k_{uv} = 0$ and all terminals have been contracted together, so the algorithm terminates. The saved $x'(e)$ values from all the original edges $e = (i, j) \in E$ constitute our half-integral solution $x'$.

Consider any edge $e$ replaced by subdivided edges, and consider any edge $d$ created by subdivision of $e$. $\mathbb{E}[\# \text{ times } d \text{ is cut}] \leq 2x(d)$, since $\alpha \in (0, \frac{1}{2})$. Since this holds for $d$, is also holds for $e$ by linearity of expectation, thus we know $\mathbb{E}[\# \text{ times } e \text{ is cut}] \leq 2x(e), \forall e \in E$. Now, we can state
\[ E[\text{cost}(x')] = \sum_{e \in E} c(e)x'(e) = \frac{1}{2} \sum_{e \in E} c(e)E[\# \text{ times } e \text{ is cut}] \leq \sum_{e \in E} c(e)x(e) = C. \]

Thus, we have shown how to construct a half-integral solution \( x' \), with expected cost at most that of the given optimal LP solution \( x \).

\[ \square \]

**Corollary 14.** A half-integral optimal solution to the multicut problem can be constructed deterministically, if the hurdle counts \( k_{uv} \) are integer and form an ultrametric.

**Proof.** We show that \( \forall \alpha \in (0, \frac{1}{2}) \) in the construction of the above theorem, \( \text{cost}(x') = C \). Thus, the choice of \( \alpha \) does not matter; any arbitrary choice of \( \alpha \in (0, \frac{1}{2}) \) suffices, and no randomization is needed to achieve a half-integral solution.

By the above theorem, \( E[\text{cost}(x')] = C \). We note that \( \forall \alpha \), \( \text{cost}(x') \geq C \) since \( C \) is the optimal, lowest possible solution value. So, it suffices to show \( \forall \alpha \), \( \text{cost}(x') \leq C \). Suppose by way of contradiction that for some value \( \alpha \in (0, \frac{1}{2}) \), \( \text{cost}(x') > C \). Since \( \not\exists \alpha \) such that \( \text{cost}(x') < C \), and yet \( \exists \alpha \) such that \( \text{cost}(x') > C \), it directly follows that \( E[\text{cost}(x')] > C \), which contradicts the fact that \( E[\text{cost}(x')] = C \).

\[ \square \]
Chapter 7

Network Knapsack

In this chapter, we discuss the network knapsack problem. We use dynamic programming (DP) to obtain a PTAS algorithm approximating the network knapsack problem on trees, a polynomial time algorithm that optimally solves the problem on ladders, and a PTAS approximating the problem on grids. Then, we investigate the difficulty of approximating the problem on increasingly complicated graph types, in particular low-treewidth graphs and planar graphs. Our results in this chapter also apply with little modification to the covering variant of the problem, mentioned in Chapter 1.

7.1 PTAS for Trees

For our PTAS for network knapsack in a tree $T = (V, E)$, we denote the optimal solution value $OPT$, and guess a value of $W$ such that $\frac{OPT}{2} \leq W \leq OPT$. In order to do so, we first bound $OPT$ from below by $x$, the value of the largest single vertex $i$ such that $\text{size}(i) < \text{capacity}(j), \forall j \in N[i]$. Using this lower bound, we can trivially bound $OPT$ from above by $xn$. Now, we proceed through a process of successive doubling, trying values $x, 2x, 4x, ..., xn$ for $W$. For each such value of $W$, we run the entire algorithm that follows, and at the end we take the best solution. This successive doubling contributes a $O(\log n)$ factor to the runtime.

Given a particular value of $W$, we proceed to rescale all item values by setting $\text{value}(i)$ for each $i$ to $\lfloor \frac{\text{value}(i) \cdot n}{2W} \rfloor$. If some item $i$ has $\text{value}(i) > \frac{2n}{\epsilon}$ after this rounding procedure, then therefore $\text{value}(i) > (\frac{2n}{\epsilon})(\frac{2W}{n}) = 2W$. So, we know $OPT \geq \text{value}(i) > 2W$, thus our guess for $W$ is
too small. In this case we terminate the algorithm, return a solution value of \( \infty \) for this \( W \), and try another \( W \). Otherwise, we may safely assume each value \( (i) \) is an integer in the set \( \{1, 2, \ldots, \frac{2W}{n}\} \), and proceed to solve the problem optimally with respect to these rounded values. By performing this rescaling procedure, each of the \( n \) items loses at most \( \frac{2W}{n} \) value by being rounded down. Therefore, we lose \( \leq n \frac{2W}{n} = \varepsilon W \leq \varepsilon OPT \) total value from any solution. Since our algorithm returns an optimal solution on the rounded values, no other value is lost from \( OPT \), and our algorithm returns a final solution of value \( \geq (1 - \varepsilon)OPT \).

Consider now a given vertex \( i \), having a parent \( h \) and having \( k \) children \( j_1, \ldots, j_k \) which are the roots of the subtrees \( \ldots k \) of \( i \). We define \( B_p^- (i, v) \) as the minimum size contributed to vertex \( i \)'s capacity such that \( i \) is not selected (indicated by the superscript \(-\)) and exactly \( v \) units of total value are present in subtrees \( \ldots p \) of \( i \). Similarly, we define \( B_p^+ (i, v) \) as the minimum size contributed to vertex \( i \)'s capacity such that \( i \) is selected (indicated by the superscript \(+\)) and exactly \( v \) units of total value are present in \( i \) along with subtrees \( \ldots p \) of \( i \).

Now, we can define boolean variables

\[
A^-_p (i, v) = \begin{cases} 
T & \text{if } B_k^- (i, v) \leq \text{capacity}(i) \\
F & \text{otherwise},
\end{cases} \quad (7.1)
\]

\[
A^+_p (i, v) = \begin{cases} 
T & \text{if } B_k^- (i, v) + \text{size}(h) \leq \text{capacity}(i) \\
F & \text{otherwise},
\end{cases} \quad (7.2)
\]

\[
A^-_p (i, v) = \begin{cases} 
T & \text{if } B_k^+ (i, v) \leq \text{capacity}(i) \\
F & \text{otherwise}, \text{ and}
\end{cases}
\]

\[
A^+_p (i, v) = \begin{cases} 
T & \text{if } B_k^+ (i, v) + \text{size}(h) \leq \text{capacity}(i) \\
F & \text{otherwise},
\end{cases} \quad (7.4)
\]

where \( i \) is a vertex in our tree \( T \), and where \( v \) is the total value of the vertices chosen in \( i \)'s subtree, an integer. The superscript \(+\) (or \(-\)) on \( A \) indicates that the parent \( h \) of \( i \) belongs (or does not belong) to the chosen set, and similarly the subscript \(+\) (or \(-\)) on \( A \) indicates that \( i \) belongs (or does not belong) to the chosen set.

We must calculate \( A^-_p (i, v) \), \( A^+_p (i, v) \), \( A^-_p (i, v) \), and \( A^+_p (i, v) \) for all \( i \in V \) and all \( v \in \).
\( \{1, 2, ..., 2^n\} \), a total of \( O(n^2) \) subproblems since \( i = O(n) \) and \( v = O(\frac{n}{2}) \). Note that since the root \( r \) of \( T \) has no parent vertex, we will not calculate \( A_-(r, v) \) nor \( A_+(r, v) \). For our solution, we find a maximum value of \( v \) such that either \( A_-(r, v) \) or \( A_+(r, v) \) is true.

We use the following DP formulations to calculate \( B_p^-(i, v) \) and \( B_p^+(i, v) \) for all \( i \in V \) and all \( v \in \{1, 2, ..., 2^n\} \):

\[
B_p^-(i, v) = \min \left( \begin{array}{l}
\text{size}(j_p) + \min_{0 \leq x \leq v} \text{s.t. } A_-(j_p, x) B_{p-1}^-(i, v-x), \\
\text{min}_{0 \leq x \leq v} \text{s.t. } A_-(j_p, x) B_{p-1}^-(i, v-x)
\end{array} \right)
\]

\[
(7.5)
\]

\[
B_p^+(i, v) = \min \left( \begin{array}{l}
\text{size}(j_p) + \min_{0 \leq x \leq v} \text{s.t. } A_+(j_p, x) B_{p-1}^+(i, v-x), \\
\text{min}_{0 \leq x \leq v} \text{s.t. } A_+(j_p, x) B_{p-1}^+(i, v-x)
\end{array} \right)
\]

\[
(7.6)
\]

As base cases, we use the following:

\[
B_0^-(i, v) = \begin{cases} 
0 & \text{if } v = 0 \\
+\infty & \text{otherwise}
\end{cases}
\]

\[
(7.7)
\]

\[
B_0^+(i, v) = \begin{cases} 
\text{size}(i) & \text{if } v = \text{value}(i) \\
+\infty & \text{otherwise.}
\end{cases}
\]

\[
(7.8)
\]

### 7.1.1 Runtime Analysis

The runtime of the above algorithm for trees is \( O(\frac{n^3}{2} \log n) \). To derive this, we note that the runtime is dominated by the time taken to calculate all values of \( B_p^-(i, v) \) and \( B_p^+(i, v) \). Since there are \( n - 1 \) total values \((i, p)\) (one for each edge in the tree), and \( O(\frac{4}{2}) \) values of \( v \), there are a total of \( O(\frac{n^3}{2}) \) such subproblems. For each one, we must calculate a minimum over \( O(\frac{n}{2}) \) values of \( v \). We also recall that the runtime is multiplied by \( O(\log n) \) due to guessing \( W \). Multiplying all these factors together, we arrive at the final runtime of \( O(\frac{n^3}{2} \log n) \).
7.2 Polynomial Algorithm for Ladders

Our algorithm for ladders is reminiscent of the above algorithm for trees. We begin with a ladder $G = (V, E)$ with width $k = O(1)$ and height $d = \frac{n}{k}$; $V = \{1, 2, ..., d\} \times \{1, 2, ..., k\}$. We proceed by dynamic programming from one end (the “bottom”) of the ladder to the other end (the “top”). We seek to define the DP formulae for a given row $i$ of the ladder consisting of vertices $(i, 1), ..., (i, k)$. The row $i + 1$ “above” $i$ is the row of subproblems to be computed next after row $i$, analogous to the “parent” vertex $h$ in the algorithm for trees. The row $i − 1$ “below” $i$ is the row of subproblems computed before row $i$, and row $i − 2$ is below that. As a starting point, we compute all possible solutions to the bottom two rows of the ladder directly with brute force. With only $2k = O(1)$ vertices in these two rows, this takes $O(1)$ time. Henceforth, we can assume that subproblems on rows $i − 1$ and $i − 2$ are already computed when considering any row $i$. In order to maintain the invariant that there is a row $i + 1$ above row $i$, we will create a row of “dummy” vertices above the top row of the ladder, each constituent vertex having infinite capacity, infinite size, and zero value; thus, they will not affect the solution.

We define the set $R_i$ for each $1 \leq i \leq d$ as the subset of vertices selected on row $i$. Notationally, we use value($R_i$) to mean $\sum_{(i,j) \in R_i} \text{value}(i, j)$. Next, we define the value $f_{R_i}^{R_{i-1}}(i)$ to be the maximum value possible from selected vertices in rows $1, ..., i$ such that $R_i$ and $R_{i-1}$ are the subsets of vertices selected on rows $i$ and $i − 1$, respectively.

With these definitions we can state the DP formula for calculating $f_{R_i}^{R_{i-1}}(i)$ as follows.

$$f_{R_{i-1}}^{R_i}(i) = \max_{R_{i-2}} \left( f_{R_{i-1}}^{R_i}(i - 1) + \text{value}(R_i) \right), \quad (7.9)$$

where we only maximize over $f_{R_{i-2}}^{R_{i-1}}(i - 1)$ for which all capacities of vertices on rows $i - 1$ and $i - 2$ are respected. As a final solution value, we wish to find the maximum $f_{R_{d-1}}^{R_d}(d)$ such that all capacities are respected on row $d$.

7.2.1 Runtime Analysis

The runtime of the above algorithm for ladders is $O(8^k n)$, which is polynomial for ladders of width $k = O(1)$. To derive this runtime, we note that there are a total of $4^k d = 4^k \left( \frac{n}{k} \right)$ subproblems to solve of the form $f_{R_i}^{R_{i-1}}(i)$, which comes from the $d = \frac{n}{k}$ rows of $G$ and the $4^k$ possible combinations of $R_i$ and $R_{i-1}$. In order to solve each of those subproblems, we must take a maximum over $2^k$
possibilities of $R_{i-2}$, each of which requires $O(k)$ time to verify its feasibility. Multiplying these factors together, we get $O(4^k(n^{2})2^k) = O(8^k n)$.

7.3 PTAS for Grids

The network knapsack problem admits a $(1 - \varepsilon)$-approximation algorithm on a grid $G$ by breaking the grid into ladders. Every $2/\varepsilon$ rows of $G$, we will remove two consecutive rows of vertices. This procedure removes an $\varepsilon$ fraction of the $n$ vertices, leaving the remaining vertices in a collection of connected components, each of which is a ladder of width at most $2/\varepsilon - 2 = O(1)$. There are $1/(2\varepsilon) = O(1)$ possibilities for which rows to remove, so we will run this algorithm separately for each possibility, which involves removing rows $x, x + 1, x + 2/\varepsilon, x + 3/\varepsilon + 1, x + 4/\varepsilon, x + 5/\varepsilon + 1$, etc, for a unique value $x \in \{0, 1, 2, ..., \lfloor 2/\varepsilon \rfloor - 1\}$. The least costly of these possibilities will remove $\leq \varepsilon$OPT value from $G$. The remainder of the graph consists of ladders of width at most $2/\varepsilon - 2 = O(1)$, each of which is solved optimally with the above algorithm for ladders. Breaking $G$ into ladders in this fashion is an application of Baker’s technique [4], where $G$ is not only planar but also a grid; for additional discussion of this technique, see the section on planar graphs where we apply it again.

7.4 Low-Treewidth Graphs

Though we have been unsuccessful so far in developing an algorithm for network knapsack on low-treewidth graphs, we can nonetheless discuss our observations regarding what makes this problem challenging to approximate. To attempt to apply the network knapsack problem to low-treewidth graphs, we begin with a graph $G$ of treewidth $k' = O(1)$. We can perform a tree decomposition on $G$, for instance using a method from [72] and improved in [6], which creates a tree decomposition $G'$ of treewidth $k \leq 3k' + 2$; each piece (vertex of the tree decomposition $G'$) includes $\leq k + 1$ vertices from $G$.

We wish to use a dynamic programming solution on the tree $G'$ similar to the algorithm we have described for trees, proceeding from the leaf pieces of $G'$ to the root piece. In order to do this, we need to encapsulate the state of our DP into a polynomial-size configuration, to contain all necessary information required to compute the solution at a piece $i$ based on the configurations of the children pieces of $i$. 

65
However, the nature of the tree decomposition itself presents significant challenges to this goal. A particular vertex $x$ in $G$ may be present in as many as $O(n)$ pieces of $G$, a non-locality issue causing difficulty for the designing of a DP. In order to determine whether a solution is feasible at any given vertex $x$ of $G$, we need to observe each of its neighbors in $G$, sum up the sizes of the neighbors in the selected set, and compare that to the capacity of $x$. But, the neighbors of $x$ in $G$ may each be present in a distinct $O(n)$-size subset of the pieces of $G'$, including up to $O(n)$ pieces which the DP has not reached yet. Conditioning on all possibilities in all such pieces is prohibitively exponential.

The inherent difficulties of the network knapsack problem on low-treewidth graphs is surprising, given the relative ease of computing a wide range of other graph properties on low-treewidth graphs. The dominating set problem, for example, is relatively straightforward; a vertex $x$ can be dominated by a single neighboring vertex. Even alliance problems are approachable in this context, by conditioning the DP on the cardinality of neighbors of each vertex in the current piece of $G'$, which is polynomially sized for $k = O(1)$. In the network knapsack problem, however, the sum of the sizes of all neighboring vertices must be simultaneously considered in order to respect a single capacity constraint. This size-based, continuous measure of feasibility over neighborhoods works directly against the nature of the tree decomposition, which can spread the neighborhood of a vertex of $G$ all throughout its tree decomposition $G'$, beyond the limit of the number of possibilities upon which the DP can be conditioned in polynomial time. Moreover, rounding item sizes (as opposed to item values, which we rounded in the algorithm for trees) also seems problematic. Rounding sizes up may cause some previously feasible solutions to now be considered infeasible, and rounding sizes down may cause the algorithm to return slightly infeasible solutions, though in this case a pseudo approximation with respect to size may be possible. Obtaining an approximation result for the low-treewidth case, particularly a true approximation algorithm that does not allow slightly infeasible solutions, remains a challenging open problem.

### 7.5 Planar Graphs

Based on a working PTAS for low-treewidth graphs, one can easily construct a PTAS for all planar graphs. This is based on the concept of an $O(1)$-outerplanar graph: a graph with a planar embedding containing disjoint cycles nested at most $O(1)$ levels deep. By noting that all
$O(1)$-outerplanar graphs are $O(1)$-treewidth, we may invoke the construction of Baker [4] to achieve an algorithm for any planar graph. This technique involves removing vertices from the given planar graph, such that only a disconnected collection of $O(1)$-outerplanar (and therefore $O(1)$-treewidth) graphs remain. Those removed vertices are selected intelligently such that no more than an $\varepsilon$ fraction of the solution value is lost. An algorithm for the low-treewidth case can then be invoked separately on each remaining component of the graph. See Baker’s work for further details [4].
Chapter 8

Conclusion and Open Problems

This dissertation discusses several problems from the family of network interdiction and network fortification. This is a large area of research applicable to many real-world scenarios, including network surveillance, reinforcement, and obstruction. The \( k \)-hurdle cut problem, on which many of our considered problems are based, can be viewed as a special case of “shortest path” network interdiction, where we pay a cost to increase the length of edges in the graph, ensuring that all source-sink paths are at least length \( k \). The lengthening of these edges can conceptually represent slowing or partially inhibiting flow on a network; the \( k \) hurdles can also represent the installation of \( k \) independent checkpoints for goods passing from \( s \) to \( t \).

The first problem this dissertation addresses is the multiway cut problem, a generalization of the well-known \( s \)-\( t \) min-cut problem. In the multiway cut problem, we must separate \( r \geq 2 \) terminals from each other by cutting edges from the graph to ensure each terminal is in a unique component. For \( r = 2 \), the problem reduces to the \( s \)-\( t \) min cut problem, and is thus polynomial-time solvable by exploiting its duality with the \( s \)-\( t \) max-flow problem. For \( r \geq 3 \), the problem is APX-hard but is well-studied for the case of \( r = 3 \); the integrality gap of the 3-simplex LP formulation is known to be exactly \( \frac{12}{11} \), due to an integrality gap proof and matching approximation ratio [51, 12]. This dissertation considers the case of general \( r \). Previous results based on side parallel cuts (SPARCs) of the simplex are known to provide approximation ratios of \( 1.5 - \frac{1}{2} \) (due to Călinescu et al. [9]) and, with an extended analysis, \( 1.3438 - \varepsilon \) (due to Karger et al. [51]). This dissertation provides another analysis of these SPARC cutting schemes, leading to an approximation guarantee \( \leq 1.4647 - \varepsilon_r \), which improves upon the result of Călinescu et al. without resorting to the complicated techniques.
used by Karger et al.

This dissertation also performs computational experiments on the $r = 4$ case, in an effort to find an optimal bad graph and an optimal cutting scheme for small, discretized 4-simplexes. These experiments follow three general avenues. First, we use a row-generation technique to obtain optimal bad graph weight distributions for discretized 4-simplexes of 3, 4, and 5 discretizations. These represent concrete worst cases for the multiway cut problem on these instances, however the exponential run-time seems prohibitive. Second, we attempt to shrink the size of our LP formulations by decomposing the discretized 4-simplex into terminal-terminal paths instead of edges. We obtain path-decomposed SPARC bad graphs for several different levels of discretization, however these bad graphs are not optimal. Third and finally, we explore the dual of the the optimal bad graph problem, and find optimal raw cutting schemes. However, degeneracy prevents us from analytically describing them, which remains an open problem.

This dissertation also considers a generalization of the multiway cut problem called the $k$-hurdle multiway cut problem, where we must cut every terminal-terminal path at least $k \geq 1$ times. For $k = 1$, it reduces to the multiway cut problem described earlier. For $k \geq 2$, this dissertation provides several results for this problem including a $(1, k - 1)$-pseudo approximation result guaranteeing all but one hurdle, at a cost no greater than the optimal solution value for $k$ hurdles. Guaranteeing that final hurdle, however, is expensive; we show via an approximation preserving reduction from the well-studied vertex cover problem that the $2(1 - \frac{1}{r})$-approximation algorithm we describe (guaranteeing all $k$ hurdles) may be difficult to improve upon.

A different generalization of s-t min-cut problem considered in the dissertation is the multicut problem, which is arguably more difficult to approximate well. Based on the “region-growing” approach of Garg et al. [29] for the $k = 1$ case, we describe a $(\log(n), [(1 - \varepsilon)k_{\max}])$-pseudo approximation algorithm; if all $k = O(1)$, this becomes a true approximation algorithm guaranteeing all hurdles with an approximation ratio of $O(\log(n))$. Obtaining a true approximation bound guaranteeing all hurdles for $k = \omega(1)$, and obtaining approximation bounds with respect to $r$ instead of $n$, remain open questions. We show a 2-approximation result for trees, generalizing an approach of Garg et al. [30] for the $k = 1$ tree case. It remains an open problem whether other methods for the $k = 1$ case, such as the primal-dual algorithm of [30], can be generalized to the $k$-hurdle case and yield a 2-approximation. Our 2-approximation also works with little modification in the vertex variant, which we call $k$-hurdle vertex multicut in a tree. Finally, we investigate a special
case of the $k$-hurdle multicut problem where the hurdle counts form an ultrametric. Viewable as an intermediate level of generalization between $k$-hurdle multiway cut and $k$-hurdle multicut, we derive a 2-approximation via half-integrality. Many more open problems can be considered by taking $k$-hurdle generalizations of additional graph cut problems from the literature, for example the multi-multiway cut problem [3].

The final problem considered in this dissertation is the network knapsack problem, a generalization of the 0-1 knapsack problem and also a special case of the packing integer programming problem. Each vertex has not only a size (or equivalently a weight) but also a value, and we wish to find a maximum-value subset of vertices that respects individual capacity constraints at each vertex. The covering variant of the problem is similar to many alliance problems studied in the last decade, but here each vertex has not only a weight but a value (instead of a cost as in the covering variant), and each vertex has its own minimum required sum of included neighboring vertex weight. We give dynamic programming solutions to the network knapsack problem on increasingly complicated graph types including trees (providing a PTAS), ladders (providing an optimal algorithm with runtime $O(8^k n)$), and grids (providing a PTAS). Obtaining a solution to the network knapsack problem on more general graph types such as low-treewidth graphs remains an open problem, though we do show how to obtain an algorithm for planar graphs based on one for low-treewidth graphs.
Bibliography


