PRIZE-COLLECTING NETWORK DESIGN

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by

Jiawei Qian

January 2012

PRIZE-COLLECTING NETWORK DESIGN

Jiawei Qian, Ph.D.

Cornell University 2012

Network design is an active research area in discrete optimization that focuses on problems arising from the construction of communication networks. The prize-collecting version of these problems allow some connectivity requirements to be violated in exchange for paying a penalty. In this dissertation, we consider prize-collecting network design problems in two settings, in which inputs for the problem are either known in advance or revealed over time.

In the first setting, we give a 3-approximation algorithm for the prize-collecting Steiner forest problem using iterative rounding. In the second setting, we give an $O(\log n)$-competitive algorithm for the constrained forest problem with 0-1 proper connectivity requirement functions using the primal-dual method and extend our algorithm to solve its prize-collecting version. Computational experiments are carried out to compare this online algorithm with the corresponding offline optimal solutions on a set of random generated large-scale instances for the special case of the prize-collecting Steiner tree problem.

In addition, we study the problem of finding the worst-case integrality gap between the traveling salesman problem and its subtour LP relaxation. We restrict ourselves to the special case in which costs between cities are either one or two. We give a proof of upper bound of $\frac{106}{81}$ for this integrality gap. By carrying out computational experiments, we find the worst-case integrality gap to be $\frac{10}{9}$ for small number of cities, $n \leq 12$.

BIOGRAPHICAL SKETCH

Jiawei Qian was born on September 21, 1983 in Beijing, China. After finishing his studies from the High School Affiliated to Renmin University of China, he went to Canada for his undergraduate studies in the School of Computer Science under Faculty of Mathematics at University of Waterloo.

During his undergraduate studies, he was introduced to the fascinating world of algorithm design and programming. After doing software engineering internships at Lenovo Group and Microsoft, he quickly realized his desire for more quantitative knowledge and graduate studies. Upon returning to school, he started exploring research areas among the fields of computer science and optimization. He received his Bachelor of Mathematics degree with double majors in Computer Science and Combinatorics & Optimization in May 2007.

He then moved to United States of America to pursue his doctorate degree in the School of Operations Research and Information Engineering at Cornell University. His research is focused on the design of approximation algorithms in the area of discrete optimization, which is an intersection of his favorite fields, computer science and optimization. While enjoying the beautiful Ithaca area and sometimes struggling with his work, he also had some great opportunities to escape and travel around the world for research activities at Berlin, Germany and Zurich, Switzerland, and internships at Nomis Solutions, San Francisco and Bank of America Merrill Lynch, Chicago. Thanks to his advisor's support, he was able to meet and work with great people around the world. It opened his eyes to quite different fields that utilize his research area.

To my grandfather, Nanhai Qian, and my family.

ACKNOWLEDGEMENTS

Working toward a Ph.D. at Cornell's School of Operations Research and Information Engineering is the greatest experience of my life so far. I am thankful for the education system, research environment, and support that I have received over the past few years. In particular, I am very grateful to the guidance and support from my advisor Prof. David Williamson. He is a terrific mentor and advisor who is always available for advice, not only for research work but many other aspects of life as well.

I would also like to thank my other two committee members Prof. Shane Henderson and Prof. Robert Kleinberg, who give me excellent directions for this thesis. In addition, Prof. Leslie Trotter, who hired me as his teaching assistant five times and served as our director of graduate studies during my second and third years, has always been supportive and encouraging to me.

I feel lucky to have many friends in our department over the years. Tuohua Wu, Fan Zhu, Yuemeng Sun and Collin Chan have always been helpful and resourceful. I enjoy hanging out and studying together with Maurice Cheung, Sophia Liu, Chao Ding, and Juan Li. A lot of fun comes from playing basketball and poker games with Mathew McLean, Rolf Waeber, Zack Rayfield, Gabriel Zayas, and Bradford Westgate.

Last, I thank my parents, grandparents and Weiwei for their love and for those many days and things when they needed me the most but I was not around. They deserve the most.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Network design is an active research area in *discrete optimization* that focuses on problems arising from the construction of communication networks. In a typical setting, we are given a directed or undirected graph $G = (V, E)$ with node set V, edge set E and non-negative edge costs c_e for every edge e in E, and our goal is to find a minimum-cost subgraph H of G that satisfies some connectivity requirements. The nodes of the graph G can be seen as homes, cities or other types of destinations, and the edges in G can be seen as cables or routes that connect destinations. We want to set up communication between nodes via either data transmission or physical transportation through edges in the graph, and we must choose a subset of edges in the most cost-effective way subject to the connectivity requirements. A direct application of this problem is for telecommunication companies, such as AT&T, to construct fiber-optic networks to provide services to homes and cities across the country. Other examples include the design of transportation networks and integrated circuit chips.

The *prize-collecting* versions of these problems allow some connectivity constraints to be violated in exchange for paying a penalty. In these variants, our goal becomes minimizing the sum of the edge costs plus the penalties. In the fiber-optic network example above, AT&T can choose to not serve some destinations if connecting them is so costly that outsourcing the services to other vendors generates more profit after the network construction cost. The penalties for not serving some of its customers can be seen as loss of revenues or payments for outsourcing.

Many practically relevant instances of network design problems are *NP-hard* and so are their prize-collecting versions. Thus it is believed that there are no efficient algorithms that can always find optimal solutions. In this dissertation, we focus on the design of *approximation algorithms* that efficiently compute solutions with objective values guaranteed to be close to the best possible objective value among all solutions. More precisely, an *α-approximation algorithm* for a given problem runs in polynomial time and finds a solution with objective value at most α times the objective of an optimal solution. We will refer α as the *approximation ratio* or *performance guarantee* of the respective approximation algorithm.

We consider several variants of network design problems and their corresponding prize-collecting versions in this dissertation.

1.1 The Steiner Problems and Their Prize-Collecting Versions

Given an undirected graph $G = (V, E)$ with non-negative edge costs and a set of source-sink (s_i, t_i) pairs of nodes in G, the goal of the *Steiner forest problem* is to find a minimum-cost set of edges $F \subseteq E$ such that each pair (s_i, t_i) are connected in (V, F). This problem is, as its name implies, a generalization of the *Steiner tree problem*, in which all pairs have the same source. The Steiner tree problem is well known to be NP-hard, and thus so is the Steiner forest problem. In fact, the decision version of the Steiner tree problem is one of Karp's original 21 NP-complete problems [26]. Chlebík and Chlebíková [10] showed that it is NP-hard to approximate the Steiner tree problem within any ratio better than $\frac{96}{95}$. The best known approximation algorithm up to the time of this dissertation for the Steiner tree problem is a randomized LP-rounding algorithm with an approximation ratio of 1.39 due to Byrka et al. [9].

An important technique to solve the Steiner problems is the *primal-dual* method. After first modeling the problem as a primal integer program and formulating the dual of a corresponding linear programming relaxation, a primal-dual algorithm usually starts with an empty primal solution and a dual solution of zero. It will iteratively improve the primal and dual solutions until a primal feasible solution is obtained. The performance guarantee is obtained by comparing the cost of primal and dual solution and by using the cost of dual solution as a lower bound on the cost of the optimal solution. Agrawal, Klein and Ravi [1] first introduced a 2-approximation algorithm for the Steiner forest problem. Goemans and Williamson [17] generalized Agrawal et al.'s algorithm to give a 2-approximation algorithm for a broad class of network design problems.

The *iterative rounding* scheme is another elegant technique. A typical iterative rounding algorithm repeatedly does the following until a feasible solution is obtained: it solves the linear programming relaxation with the values of some variables fixed in previous iterations, then fixes some additional variables in the solution to integer values. The performance guarantee is usually obtained by analyzing the basic solutions of the linear programming relaxation solved in each iteration. This method was first introduced by Jain [24] to solve the *generalized Steiner network problem*, which is a generalization of the Steiner forest problem to higher connectivities, i.e. each terminal pair can be required to have any non-negative number of edge-disjoint paths in the solution. Jain showed that any basic feasible solution to the linear programming relaxation of this problem must have some variable of value at least $\frac{1}{2}$. This leads to a performance guarantee of 2.

In the prize-collecting version of the Steiner forest problem, each pair (s_i, t_i) is assigned additionally a non-negative penalty π_i. We need to choose either connect (s_i, t_i) in the set of solution edges F or pay the penalty π_i. Our goal becomes minimizing the sum of total edge costs in F plus the total penalties for pairs that are not connected in F.

For the prize-collecting Steiner tree problem, Goemans and Williamson showed that their algorithm in [17] can be extended to solve this problem with the same approximation ratio of 2. Haijiaghayi and Jain [21] gave a primal-dual 3-approximation algorithm for the prize-collecting Steiner forest problem and a LP rounding 2.54-approximation algorithm. It was conjectured by the authors of [37] that the iterative rounding technique is a promising way to give an better approximation ratio for the forest case. They suggested extending Jain's 2-approximation algorithm for the generalized Steiner network problem to solve the prize-collecting Steiner forest problem.

In Chapter 2, we show that we can indeed extend Jain's algorithm to solve the prize-collecting Steiner forest problem using iterative rounding, but the approximation ratio is 3 and tight. Around the same time, Konemann et al. [27] and Hajiaghayi and Nasri [22] both independently gave similar iterative rounding 3-approximation algorithms for prize-collecting versions of Steiner forest and generalized Steiner network problems respectively. They also use similar examples as ours to show the approximation factor is tight.

1.2 The Constrained Forest Problem and Online Algorithms

All algorithms mentioned so far assume that the inputs of the problems are given in advance. In practice, it is often the case that inputs of problems are only partially known at the beginning and more inputs will be revealed over time. In the case of AT&T fiber-optic network construction, customers may sign contracts with the company over time or the company may expand its network from some initial regions subsequently to others. We refer to algorithms that solve problems with inputs revealed in such a serial fashion as *online algorithms*. In contrast, algorithms solving problems with all inputs known in advance are referred to as *offline algorithms*. The problems themselves can be categorized as offline and online versions with the same criteria.

The hardness of online algorithms lies in the difficulty of planning for an unknown future. For example, in the online version of the Steiner forest problem, instead of knowing all source-sink pairs in advance, pairs arrive one by one over time. For each new pair, we start a new *phase* and need to augment edges in current solution to connect this new pair. Consider Figure 1.1 below: there are three terminal pairs (s_1, t_1), (s_2, t_2), and (s_3, t_3), and four non-terminal nodes n_1, n_2, n_3 and n_4. All edges have cost 1. Suppose (s_1, t_1) is the first arriving pair. We can choose either path (s_1, n_1, n_2, t_2) or path (s_1, n_3, n_4, t_3) in phase 1 to connect them. Without loss of generality, we will choose path (s_1, n_1, n_2, t_2). Then in phase 2, if (s_3, t_3) is the second arriving pair, we could have saved a cost of one if we had chosen the other path in phase 1. However, even if we did that, (s_2, t_2) could be the second arriving pair and we would face the same problem. Therefore, without knowing the subsequent inputs, it is impossible to make optimal decisions based on current data.

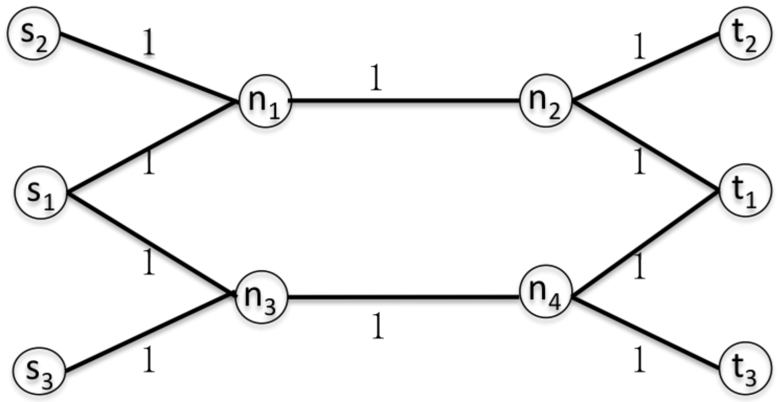


Figure 1.1: Example of the online Steiner forest problem

The quality of an online algorithm is often measured in terms of its *competitive ratio*: an *α-competitive algorithm* is one such that at any time step, the value of current solution is within a factor of α of the value of an optimal offline solution. In terms of the online Steiner forest problem, the set of edges constructed by the algorithm at current phase is at most α times the cost of the optimal value on the set of source-sink pairs that have arrived thus far.

Imase and Waxman [23] gave a greedy $O(\log n)$-competitive algorithm to solve the online Steiner tree problem, where $n = |V|$ is the number of nodes in the graphs. They also showed a lower bound of $\frac{1}{2}\log n$ on the competitive ratio of this problem. Awerbuch, Azar and Bartal [2] gave an $O(\log^2 n)$-competitive algorithm for the online Steiner forest problem. Subsequently, Berman and Coulston [4] gave an $O(\log n)$-competitive algorithm for this problem, where the competitive ratio is tight up to a constant factor by Imase and Waxman's result.

Goemans and Williamson [17] first introduced the offline constrained forest problem with a 0-1 proper connectivity function, for which they gave a primal-dual offline 2-approximation algorithm. The 0-1 proper function induces the connectivity requirements by specifying which cuts must have an edge in the solution, where a cut on a subset of nodes S is the set of edges having exactly one endpoint in S. Their model of constrained forest problems can be seen as a generalization of many related network design problems such as Steiner forest, T-join, point-to-point connection, and others. Their framework and algorithm have been adopted or extended by others to give approximation algorithms for many problems in the field of discrete optimization; see (for example) [21] and [37].

In the online version of constrained forest problem, instead of having one proper function, a series of proper functions are revealed over time, representing subsequent connectivity requirements. For example, the online Steiner forest problem described above is a special case of the constrained forest problem, since the connectivity requirement of each arriving terminal pair can be described in a form of the proper function as we mentioned in the offline case. In Chapter 3, we give a primal-dual $O(\log n)$-competitive algorithm for the online constrained forest problem by reinterpreting Berman and Coulston's online algorithm for the Steiner forest problem and melding it with Goemans and Williamson's offline algorithm for constrained forest problems. We also show that our algorithm can be extended to solve various prize-collecting network design problems with the same competitive ratio. For approximation algorithms, it is often the case that its performance is better than the theoretical worst case analysis. By implementing this online algorithm and using a package by Ljubic et al. [28] to solve the corresponding offline problem, we compare the quality of this online algorithm with the offline optimal solutions on a set of random generated large-scale inputs similar to real-world instances for the special case of the prize-collecting Steiner tree problem. We find that the average competitive ratio is 1.62 among 46 instances with up to 400 nodes.

1.3 The Traveling Salesman Problem and Its Integrality Gap

The Traveling Salesmen Problem (TSP) is one of the most intensively studied problems in discrete optimization. Given a list of cities and their pairwise distances, the goal of the problem is to find the shortest possible tour that visits each city exactly once. The TSP and its modifications have many applications in a variety of fields, such as planning, logistics, microchip manufacturing, and DNA sequencing. When the distance between two cities is the same in each direction, the problem is called the symmetric TSP, otherwise it is called the asymmetric TSP. Whether symmetric or asymmetric, the TSP can be modeled as a subclass of network design problem, where the connectivity requirement for general network design problem is the tour requirement of the TSP.

The TSP is NP-hard and its decision version is NP-complete. In fact, Sahni and Gonzalez [35] showed that no TSP approximation algorithm exists with constant performance guarantee unless P = NP. But if the distance measure is metric (i.e. it satisfies the triangle inequality) and symmetric, the problem becomes APX-hard. Christofides [11] gave a simple and elegant $\frac{3}{2}$-approximation algorithm for this case and no one has been able to improve upon this algorithm for over three decades. Many heuristics with no provable performance guarantee have been designed to solve medium to large TSP instances in practice with good success.

A natural direction for trying to obtain better approximation algorithms for the symmetric TSP is to use linear programming. The following linear programming relaxation of the traveling salesman problem was used by Dantzig, Fulkerson, and Johnson [13] in 1954. For simplicity of notation, we let $G = (V, E)$

be a complete undirected graph on n vertices. In the LP relaxation, we have a variable $x(e)$ for all $e = (i, j)$ that denotes whether we travel directly between cities i and j on our tour. Let $c(e) = c(i, j)$, and let $\delta(S)$ denote the set of all edges with exactly one endpoint in $S \subseteq V$. Then the relaxation is

$$
\begin{array}{llll}
 & \text{Min} & \sum_{e \in E} c(e)x(e) & \\
(SUBT) \quad & \text{subject to:} & \sum_{e \in \delta(i)} x(e) = 2, & \forall i \in V, \qquad (1.1) \\
 & & \sum_{e \in \delta(S)} x(e) \geq 2, & \forall S \subset V,\ 3 \leq |S| \leq |V| - 3 \quad (1.2) \\
 & & 0 \leq x(e) \leq 1, & \forall e \in E. \qquad (1.3)
\end{array}
$$

The first set of constraints (1.1) are called the *degree constraints*. The second set of constraints (1.2) are sometimes called *subtour elimination constraints* or sometimes just *subtour constraints*, since they prevent solutions in which there is a subtour of just the vertices in S. As a result, the linear program is sometimes called the *subtour LP*. Wolsey [41] (and later Shmoys and Williamson [38]) showed that Christofides' algorithm finds a tour of length at most $\frac{3}{2}$ times the optimal value of the subtour LP. This implies that the *integrality gap*, the worst case ratio of the length of an optimal tour divided by the optimal value of the LP, is at most $\frac{3}{2}$. However, no examples are known that show that the integrality gap can be as large as $\frac{3}{2}$; in fact, no examples are known for which the integrality gap is greater than $\frac{4}{3}$. A well known conjecture states that the integrality gap is indeed $\frac{4}{3}$, see (for example) Goemans [15].

Recently, progress has been made in several directions, both in improving the best approximation guarantee and in determining the exact integrality gap of the subtour LP for certain special cases of the symmetric TSP. In the *graph*-TSP, the costs $c(i, j)$ are equal to the shortest path distance in an underlying unweighted graph. Oveis Gharan, Saberi, and Singh [33] showed that the graph-

TSP can be approximated to within $\frac{3}{2} - \epsilon$ for a small constant $\epsilon > 0$. Boyd, Sitters, van der Ster and Stougie [8] gave a $\frac{4}{3}$-approximation algorithm if the underlying graph is cubic. Mömke and Svensson [31] improved these results by giving a 1.461-approximation for the graph-TSP and an $\frac{4}{3}$-approximation algorithm if the underlying graph is subcubic. Their results also imply upper bounds on the integrality gap of 1.461 and $\frac{4}{3}$ in these cases. Mucha [32] improved the bound for graph-TSP to $\frac{13}{9}$.

A 2-matching of a graph is a set of edges such that no edge appears twice and each node has degree two, i.e. an integer solution to *SUBT* with only constraints (1.1) and (1.3). Note that a minimum-cost 2-matching thus provides a lower bound on the length of the optimal TSP tour. A minimum-cost 2-matching can be found in polynomial time using a reduction to a certain minimum-cost matching problem. Boyd and Carr [7] conjectured that the worst case ratio of the cost of a minimum-cost 2-matching and the optimal value of the subtour LP is at most $\frac{10}{9}$. This conjecture was proved to be true by Schalekamp, Williamson and van Zuylen [36] and examples are known that show this result is tight.

The importance of integrality gap is that it either translates into the approximation ratio of an approximation algorithm or gives a hint on the best approximation ratio can be achieved by an LP-based approach. In this dissertation, we focus on the integrality gap of the special case of the symmetric 1,2-TSP, where distances between cities are either 1 or 2. The symmetric 1,2-TSP is NP-hard as well and naturally satisfies the triangle inequality. We prove that this integrality gap is at most $\frac{106}{81} \approx 1.31 < \frac{4}{3}$. For small number of cities, $n \leq 12$, we find the worst-case integrality gap to be $\frac{10}{9}$ by carrying out computational experiments.

CHAPTER 2

A FACTOR 3 ITERATIVE ROUNDING ALGORITHM FOR PRIZE-COLLECTING STEINER FOREST

2.1 Introduction

For the prize-collecting Steiner forest problem (PCSF problem), we are given as input an undirected graph $G = (V, E)$ with non-negative edge costs $c_e \geq 0$ for all $e \in E$, a set of terminal pairs $T = \{\ i \mid (s_i, t_i)$ with s_i and $t_i \in V\ \}$, and a penalty $\pi_i \geq 0$ for each terminal pair $i \in T$. For each terminal pair (s_i, t_i), we either need to build some edges to connect them or to pay its penalty. Let $F \subseteq E$ be subset of edges we build and let $Q \subseteq T$ be the set of terminal pairs for which we pay the penalty. Our goal is to minimize the total cost of edges in F plus the total penalties for terminal pairs in Q. As we mentioned in Chapter 1, a direct application of this problem is for telecommunication companies like AT&T to construct fiber-optic networks to serve customers at different locations while having the choice to outsource part of the service to other vendors.

The special case in which all terminal pairs have the same source, i.e. $s_i = r$ for all i, is the prize-collecting Steiner tree problem. Goemans and Williamson [17] gave a primal-dual 2-approximation algorithm for this problem. Hajiaghayi and Jain [21] gave a primal dual 3-approximation algorithm for the general form of the PCSF problem, where they used a novel integer programming formulation with a doubly-exponential number of variables. They also gave a randomized LP rounding algorithm with an approximation ratio of 2.54 for this problem. Their approach has been generalized by Sharma, Swamy, and Williamson [37] with the same approximation ratio for PCSF problem with more general connectivity constraints and penalty functions.

Jain [24] introduced the iterative rounding technique to give a 2-approximation algorithm for the generalized Steiner network problem, which is a generalization of the Steiner Forest problem (or the PCSF problem with penalty = ∞ for all terminal pairs) to higher connectivity, i.e. each terminal pair can be required to have any non-negative number of edge disjoint paths in the solution. A typical iterative rounding algorithm repeatedly does the following until a feasible solution is obtained: it solves the the linear programming relaxation with the values of some variables fixed in previous iterations, then fixes some additional basic variables in the solution to integer values. The performance guarantee is usually obtained by analyzing the basic solutions of the linear programming relaxation solved in each iteration. Jain showed that any basic feasible solution to the linear programming relaxation of this problem must have some variable of value at least $\frac{1}{2}$ and this leads to the performance guarantee of 2.

It was conjectured by the authors of [37] that the iterative rounding technique is a promising way to give a better approximation ratio for the PCSF problem. In this chapter, we show that we can indeed solve the prize-collecting Steiner forest problem by extending ideas from Jain [24]. However, the approximation ratio we obtain is 3 and is tight.

2.2 Preliminaries

Consider the PCSF problem with undirected graph $G = (V, E)$, non-negative edge costs $c_e \geq 0$ for all $e \in E$, terminal pairs $T = \{ i \mid (s_i, t_i) \in (V, V)$ and penalties

$\pi_i \geq 0$ for all $i \in T$. We define a *cut-terminal pair* to be a pair (A, i) with $A \subseteq V$ and $i \in T$. We also define a class of functions to be *extended weakly supermodular* as follows.

Definition 2.2.1 *A function $f : (2^V, T) \to \{0, 1\}$ is* extended weakly supermodular *if it satisfies $f(\emptyset, i) = f(V, i) = 0$ for all $i \in T$, and for any two cut-terminal pairs $(A, i), (B, j) \in (2^V, T)$, one of the following six conditions holds:*

1. $f(A, i) + f(B, j) \leq f(A \cup B, i) + f(A \cap B, j)$
2. $f(A, i) + f(B, j) \leq f(A \cup B, j) + f(A \cap B, i)$
3. $f(A, i) + f(B, j) \leq f(A - B, i) + f(B - A, j)$
4. $f(A, i) + f(B, j) \leq f(A - B, j) + f(B - A, i)$
5. $2f(A, i) + 2f(B, j) \leq f(A \cup B, i) + f(A \cap B, i) + f(A - B, j) + f(B - A, j)$
6. $2f(A, i) + 2f(B, j) \leq f(A \cup B, j) + f(A \cap B, j) + f(A - B, i) + f(B - A, i)$

Define $\delta(S)$ to be the subset of edges with exactly one endpoint in $S \subseteq V$ and the other endpoint not in S. Then we can model the PCSF problem as a special case of the following integer program (IP) with extended weakly supermodular function $f(S, i) = 1$ if and only if $|S \cap (s_i, t_i)| = 1$ for some $i \in T$,

$$
(IP) \qquad \begin{array}{lll}
\text{Min} & \displaystyle\sum_{e \in E} c_e x_e + \sum_{i \in T} \pi_i y_i & \\
 & \displaystyle\sum_{e \in \delta(S)} x_e + y_i \geq f(S, i), & \forall S \subseteq V \quad \forall i \in T, \\
 & x_e \in \{0, 1\}, & \forall e \in E, \\
 & y_i \in \{0, 1\}, & \forall i \in T,
\end{array}
$$

where $f(S, i)$ is extended weakly supermodular.

Lemma 2.2.2 *The function $f(S,i) = 1$ if and only if $|S \cap \{s_i, t_i\}| = 1$ for some $i \in T$ is extended weakly supermodular.*

Proof: Clearly, $f(\emptyset, i) = f(V, i) = 0$ for all $i \in T$. We observe that $f(S,i) = f(V-S,i)$ for any $S \subseteq V$ and $i \in T$. Also, we have for any disjoint $A, B \subseteq V$ and $i \in T$, $f(A \cup B, i) \leq \max(f(A,i), f(B,i))$. Then for any two cut-terminal pairs $(A,i), (B,j) \in (2^V, T)$, f satisfies the following four inequalities:

1. $f(A,i) \leq \max(f(A-B,i), f(A \cap B, i)))$,
2. $f(A,i) = f(V-A,i) \leq \max(f(B-A,i), f(V-(A \cup B),i)) = \max(f(B-A,i), f(A \cup B,i))$,
3. $f(B,j) \leq \max(f(B-A,j), f(A \cap B, j)))$,
4. $f(B,j) = f(V-B,j) \leq \max(f(A-B,j), f(V-(A \cup B),j)) = \max(f(A-B,j), f(A \cup B,j))$.

Consider the four of $f(A-B,i)$, $f(B-A,i)$, $f(A \cap B,i)$, $f(A \cup B,i)$, $f(B-A,j)$, $f(A-B,j)$, $f(A \cap B,j)$, and $f(A \cup B,j)$ that attain maximum of $\max(f(A-B,i), f(A \cap B,i))$, $\max(f(B-A,i), f(A \cup B,i))$, $\max(f(B-A,j)$,$f(A \cap B,j))$, and $\max(f(A-B,j), f(A \cup B,j))$. If the four maximums are $f(A-B,i)$, $f(B-A,i)$, $f(A \cap B,j)$ and $f(A \cup B,j)$, or $f(B-A,j)$, $f(A-B,j)$, $f(A \cap B,i)$ and $f(A \cup B,i)$, then condition 5 or 6 in Definition 2.2.1 is satisfied by summing all four inequalities above. In all other cases, the sum of two above inequalities must imply one of conditions 1 to 4 in Definition 2.2.1; for example, if $f(A-B,i)$ and $f(B-A,j)$ attain maximum of first and third inequality above, then we have $f(A,i) + f(B,j) \leq f(A-B,i) + f(B-A,j)$. □

Lemma 2.2.3 *Let $G = (V,E)$ be a graph and $\delta_x(S) = x \cap \delta(S)$ for any $x \in \{0,1\}^E$. Then* *the function $|\delta_x(\cdot)|$ satisfies $|\delta_x(\emptyset)| = |\delta_x(V)| = 0$ and for every two sets $A, B \subseteq V$, the following two statements hold:*

1. $|\delta_x(A)| + |\delta_x(B)| \geq |\delta_x(A \cup B)| + |\delta_x(A \cap B)|$
2. $|\delta_x(A)| + |\delta_x(B)| \geq |\delta_x(A - B)| + |\delta_x(B - A)|$

Proof: Let $S_1 = A - B$, $S_2 = A \cap B$, $S_3 = B - A$ and $S_4 = V - (A \cup B)$ and let $\delta_x(S_i, S_j)$ denote the set of edges in x which have one endpoint in S_i and the other in S_j. The lemma follows by observing,

1. $|\delta_x(A)| = |\delta_x(S_1, S_3)| + |\delta_x(S_1, S_4)| + |\delta_x(S_2, S_3)| + |\delta_x(S_2, S_4)|$,
2. $|\delta_x(B)| = |\delta_x(S_1, S_2)| + |\delta_x(S_1, S_3)| + |\delta_x(S_2, S_4)| + |\delta_x(S_3, S_4)|$,
3. $|\delta_x(A - B)| = |\delta_x(S_1, S_2)| + |\delta_x(S_1, S_3)| + |\delta_x(S_1, S_4)|$,
4. $|\delta_x(B - A)| = |\delta_x(S_1, S_3)| + |\delta_x(S_2, S_3)| + |\delta_x(S_3, S_4)|$,
5. $|\delta_x(A \cap B)| = |\delta_x(S_1, S_2)| + |\delta_x(S_2, S_3)| + |\delta_x(S_2, S_4)|$,
6. $|\delta_x(A \cup B)| = |\delta_x(S_1, S_4)| + |\delta_x(S_2, S_4)| + |\delta_x(S_3, S_4)|$.

$\square$

Lemma 2.2.4 *Let $x \in \{0, 1\}^E$. If $f : (2^V, T) \rightarrow \{0, 1\}$ is an extended weakly supermodular function, then $f(S, i) - |\delta_x(S)|$ is also an extended weakly supermodular function.*

Proof: This proof is straightforward. By Lemma 2.2.3, we have $|\delta_x(\emptyset)| = |\delta_x(V)| = 0$, and for every two sets $A, B \subseteq V$,

1. $|\delta_x(A)| + |\delta_x(B)| \geq |\delta_x(A \cup B)| + |\delta_x(A \cap B)|$,
2. $|\delta_x(A)| + |\delta_x(B)| \geq |\delta_x(A - B)| + |\delta_x(B - A)|$,

3. $2|\delta_x(A)| + 2|\delta_x(B)| \geq \delta_x(A \cup B)| + |\delta_x(A \cap B)| + \delta_x(A - B)| + |\delta_x(B - A)|$.

Consider any extended weakly supermodular function f, for $(A, i), (B, j) \in (2^V, T)$, one of the six inequalities in Definition 2.2.1 must hold. Assume $f(A, i) + f(B, j) \leq f(A \cup B, i) + f(A \cap B, j)$ holds; the other cases are similar. Then by condition 1, we know that

$$f(A, i) + f(B, j) - |\delta_x(A)| - |\delta_x(B)| \leq f(A \cup B, i) - |\delta_x(A \cup B)| + f(A \cap B, j)| - |\delta_x(A \cap B)|.$$

Last, since $f(S, i) - |\delta_x(S)| = 0$ when $S = \emptyset$ or $S = V$, we have shown that $f(S, i) - |\delta_x(S)|$ is again extended weakly supermodular. □

2.3 A Factor 3 Iterative Rounding Algorithm

Now consider the linear programming relaxation of (IP), where $0 \leq x_e \leq 1$ for all $e \in E$ and $y_i \geq 0$ for all $i \in T$; call the relaxation (LP). We obtain a 3-approximation algorithm for the PCSF problem using the following theorem.

Theorem 2.3.1 *For any basic solution $[x\ y]$ to the linear programming relaxation (LP), there exists $x_e \geq \frac{1}{3}$ for some edge $e \in E$ or $y_i \geq \frac{1}{3}$ for some $i \in T$.*

We will prove this main theorem in Section 2.4. Let us first see how to use this theorem to give a factor 3 iterative rounding algorithm for the PCSF problem. Our algorithm is formally stated in Figure 2.1. Starting with $F = \emptyset$ and $Q = \emptyset$, in the kth iteration of the algorithm, we solve (LP) on edge set $E - F$ and terminal pairs $T - Q$ with function f_k, where $f_k(S, i) = f(S, i) - |\delta_F(S)|$. Since this linear program is trivially bounded and feasible, there must be a basic optimal solution

Algorithm

```
$F \leftarrow \emptyset, Q \leftarrow \emptyset$
$k \leftarrow 1$
While $(F, Q)$ is not a feasible solution
    a) Solve $LP$ on edge set $E - F$ and terminal set $T - Q$ with function $f_k$
       where $f_k(S, i) = f(S, i) - |\delta_F(S)|$
    b) $F_k = \{e : x_e \geq 1/3\}$ and $Q_k = \{i : y_i \geq 1/3\}$
    c) $F \leftarrow F \cup F_k$, $Q \leftarrow Q \cup Q_k$
    d) $k \leftarrow k + 1$
Return $(F, Q)$
```

Figure 2.1: Facter 3 iterative rounding algorithm for the prize-collecting Steiner forest problem

to it. Then by Theorem 2.3.1, we must have $x_e \geq \frac{1}{3}$ for some edge $e \in E$ or $y_i \geq \frac{1}{3}$ for some $i \in T$. We round up all $x_e \geq \frac{1}{3}$ and $y_i \geq \frac{1}{3}$ to 1, and put these edges and/or terminal pairs into sets F and Q separately. We iteratively round up solutions to (LP) until we find a feasible solution for the original problem (IP), i.e. all terminal pairs in T are either connected by edges in F or are included in Q.

To show that the algorithm works, each function f_k needs to be again extended weakly supermodular, and this is shown by Lemma 2.2.4. Therefore, Theorem 2.3.1 applies for all iterations. Then F_k and Q_k are not both empty in each iteration, and thus there are at most $|E| + |T|$ iterations before the algorithm terminates. In addition, it is well known that we can find a basic optimal solution of (LP) efficiently using ellipsoid method and a separation oracle (see [19]).

We can now show that Theorem 2.3.1 implies that the algorithm of Figure 2.1 is a 3-approximation algorithm.

Theorem 2.3.2 *Given Theorem 2.3.1, the algorithm of Figure 2.1 is a 3-approximation* *algorithm for prize-collecting Steiner Forest problem.*

Proof: We will use induction on the number of iterations of the algorithm. Let $[x\ y]$ be the solution to (LP) with function $f_1 = f$ for any extended weakly supermodular function f. The base case is straightforward: if after one iteration, $(F = F_1, Q = Q_1)$ is a feasible solution, then since $F_1 = \{e \in E : x_e \geq 1/3\}$ and $Q_1 = \{i \in T : y_i \geq 1/3\}$, it is clear that $c(F)+\pi(Q) \leq 3\sum_{e\in E} c_e x_e + 3\sum_{i\in T} \pi_i y_i \leq 3 \cdot OPT$.

Now suppose that the statement holds if the algorithm takes t iterations, and we show that it holds if the algorithm takes $t+1$ iterations. By induction, the cost of all edges we decide to add plus the penalty of all terminal pairs we decide to pay from the second iterations onward is no more than three times the value of the LP solution on $E - F_1$ and $T - Q_1$ with the extended weakly supermodular function f_2; that is, if $[x'\ y']$ is the solution found in the second iteration for the LP on $E - F_1$ and $T - Q_1$ with function f_2, then $c(F - F_1) + \pi(Q - Q_1) \leq 3\sum_{e\in E-F_1} c_e x'_e + 3\sum_{i\in T-Q_1} \pi_i y'_i$ by induction hypothesis, since the algorithm finds a solution for second iteration onwards in t iterations. For $e \in F_1$ and $i \in Q_1$, we know that $c(F_1) + \pi(Q_1) \leq 3\sum_{e\in F_1} c_e x_e + 3\sum_{i\in Q_1} \pi_i y_i$, since $x_e \geq 1/3$ for all $e \in F_1$ and $y_i \geq 1/3$ for all $i \in Q_1$. To complete the proof, we will show that $[x\ y]$ is a feasible solution on the edges $E - F_1$ and terminal pairs $T - Q_1$ for the function f_2. Thus,

$$\sum_{e\in E-F_1} c_e x'_e + \sum_{i\in T-Q_1} \pi_i y'_i \leq \sum_{e\in E-F_1} c_e x_e + \sum_{i\in T-Q_1} \pi_i y_i,$$

so that

$$\begin{aligned} c(F) + \pi(Q) &= c(F - F_1) + \pi(Q - Q_1) + c(F_1) + \pi(Q_1) \\ &\leq 3 \textstyle\sum_{e \in E - F_1} c_e x'_e + 3 \sum_{i \in T - Q_1} \pi_i y'_i + 3 \sum_{e \in F_1} c_e x_e + 3 \sum_{i \in Q_1} \pi_i y_i \\ &\leq 3 \textstyle\sum_{e \in E - F_1} c_e x_e + 3 \sum_{i \in T - Q_1} \pi_i y_i + 3 \sum_{e \in F_1} c_e x_e + 3 \sum_{i \in Q_1} \pi_i y_i \\ &= 3 \textstyle\sum_{e \in E} c_e x_e + 3 \sum_{i \in T} \pi_i y_i \\ &= 3 \cdot OPT. \end{aligned}$$

To see that $[x\ y]$ is feasible for the LP with function f_2 on $E - F_1$ and $T - Q_1$, we need to show $x(\delta(S) \cap (E - F_1)) + y_i \geq f_2(S)$ for all $S \subseteq V$ and for all $i \in T - Q_1$. This is easy to see as for all $S \subseteq V$ and for all $i \in T - Q_1$, we have

$$\begin{aligned} x(\delta(S) \cap (E - F_1)) + y_i &= x(\delta(S)) - x(\delta(S) \cap F_1) + y_i \\ &\geq f_1(S) - x(\delta(S) \cap F_1) \\ &\geq f_1(S) - |\delta(S) \cap (F_1)| \\ &= f_2(S), \end{aligned}$$

where the first inequality follows from $x(\delta(S)) + y_i \geq f_1(S)$ by the feasibility of $[x\ y]$ on the LP with f_1 and the fact that $i \in T - Q_1$ implies $i \in T$. For the second inequality, we use the fact that $x_e \leq 1$. $\square$

2.4 Proof of Main Theorem

We now turn to the proof of Theorem 2.3.1. We assume without loss of generality that $x_e < 1$ for all $e \in E$ and $y_i < 1$ for all $i \in T$, since if some x_e or y_i equal to 1, we can proceed to next iteration. Let E' be the set of edges with $x_e > 0$ and V' be the set of terminal pairs with $y_i > 0$. We will need some more definitions and lemmas.

Definition 2.4.1 *For $x \in \mathcal{R}^{|E|}$ and a subset of edges F, we define $x(F) = \sum_{e \in F} x_e$.*

Definition 2.4.2 *We say two sets A and B are* intersecting *if $A \cap B$, $A - B$, and $B - A$ are all nonempty. Two cut-terminal pairs (A, i) and (B, j) are* intersecting *if the corresponding cut sets A and B are intersecting.*

Definition 2.4.3 *For a solution $[x\ y]$ to LP, we say a* cut-terminal pair *(S, i) is* tight *if $x(\delta(S)) + y_i = f(S, i) = 1$, where $S \subseteq V$ and $i \in T$.*

Definition 2.4.4 *We say a collection of sets $\mathcal{S}$ is* laminar *if no pair of sets $A, B \in \mathcal{S}$ are intersecting. A collection of cut-terminal pairs is* extended-laminar *if the collection of distinct sets S of its (S, i)-pairs is laminar.*

Definition 2.4.5 *For a subset of edges $F \subseteq E$, the* characteristic vector *of F is $\Delta_F \in \{0, 1\}^{|E|}$, where $\Delta_F(e) = 1$ if $e \in F$ and 0 otherwise. Similiarly, for a subset of terminal pairs $Q \subseteq T$, the* characteristic vector *of Q is $\Theta_Q \in \{0, 1\}^{|T|}$, where $\Theta_Q(i) = 1$ if $i \in Q$ and 0 otherwise. We define the* characteristic vector *of a cut-terminal pair (S, i), $S \subseteq V$ and $i \in T$, to be $\chi_{(S,i)} = [\Delta_{\delta(S)}\ \Theta_{\{i\}}]$.*

We are now able to state the following theorem and its corollary, which we will need to prove Theorem 2.3.1.

Theorem 2.4.6 *For each basic feasible solution $[x\ y]$ to LP, there exists a collection $\mathcal{L}$ of cut-terminal pairs with the following properties:*

1. *For all $(S, i) \in \mathcal{L}$, (S, i) is tight.*
2. *The characteristic vectors $\chi_{(S,i)}$ for $(S, i) \in \mathcal{L}$ are linearly independent.*

3. $|\mathcal{L}| = |E'| + |V'|$.
4. *The collection $\mathcal{L}$ is extended-laminar.*

The first three properties follow from the fact that $[x\ \ y]$ is a basic solution. A basic solution is formed by taking $|E'| + |V'|$ linearly independent constraints from the linear program, setting them at equality, and solving the resulting linear system. This is precisely what the first two properties state. The third property states that the number of constraints is equal to the number of non-zero variables.

To show that $\mathcal{L}$ is extended-laminar, we will need the following lemma. The basic idea is that we start with a collection of cut-terminal pairs $\mathcal{S}$ that may not be extended-laminar, and as long as we have two pairs $(A, i), (B, j) \in \mathcal{S}$ that are intersecting, we show that we can "uncross" them and replace them with other non-intersecting pairs.

Lemma 2.4.7 *If (A, i) and (B, j) are two tight cut-terminal pairs such that A and B are intersecting, then one of the following statments must hold:*

1. *$(A \cup B, i)$ and $(A \cap B, j)$ are tight and* $\chi_{(A,i)} + \chi_{(B,j)} = \chi_{(A\cup B,i)} + \chi_{(A\cap B,j)}$
2. *$(A \cup B, j)$ and $(A \cap B, i)$ are tight and* $\chi_{(A,i)} + \chi_{(B,j)} = \chi_{(A\cup B,j)} + \chi_{(A\cap B,i)}$
3. *$(A - B, i)$ and $(B - A, j)$ are tight and* $\chi_{(A,i)} + \chi_{(B,j)} = \chi_{(A-B,i)} + \chi_{(B-A,j)}$
4. *$(A - B, j)$ and $(B - A, i)$ are tight and* $\chi_{(A,i)} + \chi_{(B,j)} = \chi_{(A-B,j)} + \chi_{(B-A,i)}$
5. *$(A \cup B, i), (A \cap B, i), (A - B, j)$ and $(B - A, j)$ are tight and*
   $2\chi_{(A,i)} + 2\chi_{(B,j)} = \chi_{(A\cup B,i)} + \chi_{(A\cap B,i)} + \chi_{(A-B,j)} + \chi_{(B-A,j)}$
6. *$(A \cup B, j), (A \cap B, j), (A - B, i)$ and $(B - A, i)$ are tight and*
   $2\chi_{(A,i)} + 2\chi_{(B,j)} = \chi_{(A\cup B,j)} + \chi_{(A\cap B,j)} + \chi_{(A-B,i)} + \chi_{(B-A,i)}$

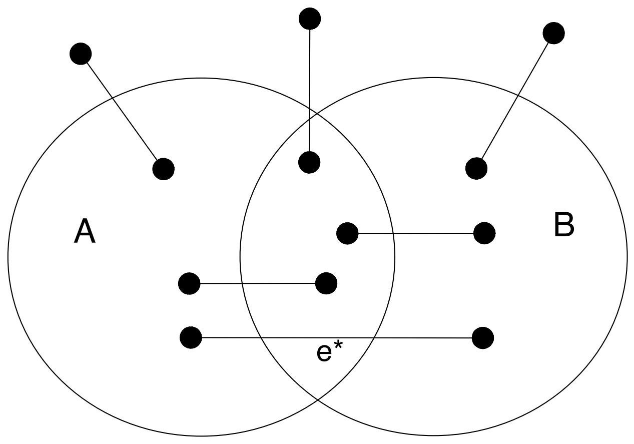


Figure 2.2: Proof of Lemma 2.4.7

Proof: First, by a counting argument, we have

1. $x(\delta(A)) + x(\delta(B)) \geq x(\delta(A \cup B)) + x(\delta(A \cap B))$,
2. $x(\delta(A)) + x(\delta(B)) \geq x(\delta(A - B)) + x(\delta(B - A))$,
3. $2x(\delta(A)) + 2x(\delta(B)) \geq x(\delta(A \cup B)) + x(\delta(A \cap B)) + x(\delta(A - B)) + x(\delta(B - A))$.

We give proof of the first case and the other cases are very similar. Consider Figure 2.2, which contains all possible types of edges in $\delta(A)$ and $\delta(B)$, where e^* contributes only to left-hand side of the inequality and all others contribute equally to both sides. So $x(\delta(A)) + x(\delta(B)) \geq x(\delta(A \cup B)) + x(\delta(A \cap B))$.

Second, since f is extended weakly supermodular, one of the six inequalities in Definition 2.2.1 must hold. Assume $f(A,i) + f(B,j) \leq f(A \cup B, i) + f(A \cap B, j)$ holds and the other cases are similar. So we have

$$\begin{aligned} x(\delta(A)) + y_i + x(\delta(B)) + y_j &= f(A,i) + f(B,j) \\ &\leq f(A \cup B, i) + f(A \cap B, j) \\ &\leq x(\delta(A \cup B)) + y_i + x(\delta(A \cap B)) + y_j. \end{aligned}$$

The second inequality holds by the feasibility of $[x\ y]$ on constraints $(A \cup B, i)$ and $(A \cap B, j)$. Therefore, we have $x(\delta(A)) + x(\delta(B)) = x(\delta(A \cup B)) + x(\delta(A \cap B))$. So both $(A \cup B, i)$ and $(A \cap B, j)$ are tight and the two sets do not intersect. It also must be the case that $\chi_{(A,i)} + \chi_{(B,i)} = \chi_{(A \cup B,i)} + \chi_{(A \cap B,j)}$, since we have assumed that all edges in E' have $x_e > 0$. □

Note that the characteristic vectors include parts corresponding to y_i and y_j, and are matched on both sides of the equality. This is one of the differences from the iterative rounding results in Jain [24]. We now give the proof of Theorem 2.4.6.

Proof: As we showed previously, there exists a collection $\mathcal{S}$ of cut-terminal pairs that have the first three properties of the theorem. Let $span(\mathcal{S})$ be the span of the set of vectors $\{\chi_{(S,i)} : (S,i) \in \mathcal{S}\}$. Let $\mathcal{L}$ be a maximal collection of cut-terminal pairs that have all four properties. We will show that $|\mathcal{L}| = |\mathcal{S}|$; suppose otherwise. Then there must be a tight pair (S,i) such that $\chi_{(S,i)} \in span(\mathcal{S})$ and $\chi_{(S,i)} \notin span(\mathcal{L})$; we choose a pair (S,i) such that there is no other such pair intersecting fewer sets in $\mathcal{L}$. Note that such a pair (S,i) must be intersecting with at least one pair in $\mathcal{L}$, otherwise, $\mathcal{L}$ is not maximal.

Now pick a pair $(T, j) \in \mathcal{L}$ such that S and T intersect. By Lemma 2.4.7, one of the its six cases must hold. Let us consider the last case, i.e. $(S \cup T, j), (S \cap T, j), (S - T, i)$ and $(T - S, i)$ are tight and $2\chi_{(S,i)} + 2\chi_{(T,j)} = \chi_{(S \cup T,j)} + \chi_{(S \cap T,j)} + \chi_{(S-T,i)} + \chi_{(T-S,i)}$; the other cases are similar. Since $\chi_{(S,i)} \notin span(\mathcal{L})$ and $\chi_{(T,j)} \in span(\mathcal{L})$, at least one of $\chi_{(S \cup T,j)}, \chi_{(S \cap T,j)}, \chi_{(S-T,i)}, \chi_{(T-S,i)}$ must not be in $span(\mathcal{L})$. We will show that each of the pairs $(S \cup T, j), (S \cap T, j), (S - T, i)$ and $(T - S, i)$ intersects fewer pairs in $\mathcal{L}$, contradicting to the choice of S.

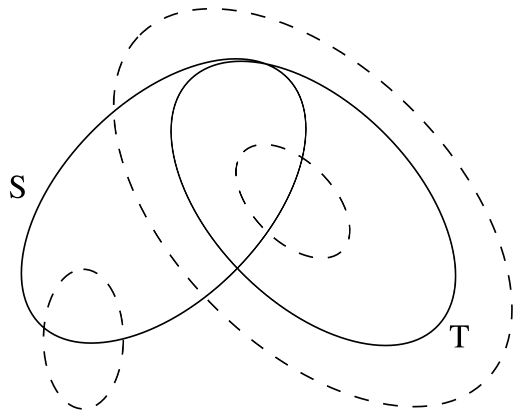


Figure 2.3: Proof of Lemma 2.4.6

Figure 2.3 illustrates the three ways in which a pair $(T', k) \in \mathcal{L}$ can intersect one of these four sets without intersecting T itself (T' is shown dotted). In all cases, T' intersects S as well. In addition, S intersects T, while the sets $S \cup T$, $S \cap T$, $S - T$, and $T - S$ do not intersect T, so they must intersect fewer sets in $\mathcal{L}$ than S. This contradicts the choice of S.

□

Note that we might have cut-terminal pairs (S_1, i), (S_2, j), $i \neq j$ both belong to $\mathcal{L}$ but $\delta(S_1) = \delta(S_2)$. In this case, we must have $y_i = y_j$ since $x(\delta(S_1)) + y_i = x(\delta(S_2)) + y_j = 1$. We need to eliminate this case in order to apply a technique similar to Jain's to prove Theorem 2.3.1.

Corollary 2.4.8 *For each basic feasible solution $[x \ \ y]$ to LP, there exists a collection $\mathcal{L}' \subseteq \mathcal{L}$ of cut-terminal pairs with the following properties:*

1. *There are no sets S_1, S_2 with terminals pairs $i \neq j$ and $\delta(S_1) = \delta(S_2)$ such that both (S_1, i) and (S_2, j) are in $\mathcal{L}'$,*
2. *For all $(S, i) \in \mathcal{L}'$, (S, i) is tight,*

3. *The characteristic vectors $\chi_{(S,i)}$ for $(S,i) \in \mathcal{L}$ are linearly independent,*
4. $|\mathcal{L}'| = |E'| + |V''|$*, where* $V'' = \{i \mid (S,i) \in \mathcal{L}'\}$*,*
5. *The collection $\mathcal{L}'$ is extended-laminar.*

Proof: Suppose there are S_1, S_2 with terminals pairs $i \neq j$ and $\delta(S_1) = \delta(S_2)$ such that both (S_1, i) and (S_2, j) are in $\mathcal{L}'$. It cannot be the case that we have (S_3, i), (S_4, j) with terminals pairs $i \neq j$ and $\delta(S_3) = \delta(S_4)$ such that both S_3, S_4 are in $\mathcal{L}'$. Otherwise, we have $\chi_{(S_1,i)} - \chi_{(S_2,j)} = \chi_{(S_3,i)} - \chi_{(S_4,j)}$, so that the characteristic vectors are linearly dependent, which contradicts the linear independence property of $\mathcal{L}$.

So for any two cut-terminal pairs (S_1, i) and (S_2, j) in $\mathcal{L}'$ with $i \neq j$ and $\delta(S_1) = \delta(S_2)$, we can replace y_i with y_j (or y_j with y_i), which will produce exactly one duplicate constraint. We can remove the duplicate constraint so that both the number of constraints and number of variables are reduced by exactly one. If we repeatedly do so until no such two cut-terminal pairs exist, we get a collection $\mathcal{L}' \subseteq \mathcal{L}$ satisfying all conditions in the corollary. □

We are now ready to prove Theorem 2.3.1. Our ideas come from Jain's original argument in [24] with added fractional penalty variables y. In addition, we need to consider locations of terminals in some of the cases.

Proof: We consider an extended laminar collection $\mathcal{L}'$ described in Corollary 2.4.8. Then we consider only edges in E' and terminals in V''. For cut-terminal pairs in $\mathcal{L}'$, we will call (R, i) the *parent* of (C, j) if R is the smallest set strictly containing C and call (C, j) a *child* of (R, i). A parentless node is called a *root*, and a childless node is called a *leaf*.

Consider (R,i) and its children. We define four kinds of edges: *children-out* (CO) edges which have one endpoint in a child C and the other endpoint outside of R, *parent-out* (PO) edges which have one endpoint in R and the other endpoint outside of R, *children-children* (CC) edges which have endpoints in two different children, and *children-parent* (CP) edges which have one endpoint in a child C and the other endpoint in R but not in any child. We will use subscripts to identify a particular child in the notation when necessary. For example, (C_1C_2) is the set of edges having one endpoint in C_1 and the other endpoint in C_2.

Lemma 2.4.9 *Consider any cut-terminal pair (R,i) and its children $(C_1,j_1)...,(C_k,j_k)$ in $\mathcal{L}'$, if $t_i, t_{j_1},..,t_{j_k}$ are all outside of R, then we must have $y_i = \min\{y_{j_1},..,y_{j_k}\}$ and s_i is in C_l for the y_{j_l} attaining the minimum of $y_{j_1},..,y_{j_k}$.*

Proof: Suppose there exists $y_{j_l} < y_i$. Since $x(\delta(R)) + y_i = 1 = f(R,i)$, we have $x(\delta(R)) + y_{j_l} < 1 = f(R,j_l)$, which contradicts the feasibility of $[x\ y]$ since $s_{j_l} \in R$ and $t_{j_l} \notin R$. Therefore, $y_i = \min\{y_{j_1},..,y_{j_k}\}$.

Similarly, suppose $s_i \in C_l$ and $y_{j_l} > y_i$. Then we have $x(\delta(C_l))+y_{j_l} = 1 = f(C_l,j_l)$ but $x(\delta(C_l))+y_i < 1$, which is again a contradiction. So s_i is in C_l with y_{j_l} attaining the minimum of $y_{j_1},..,y_{j_k}$. □

Lemma 2.4.10 *Consider any cut-terminal pair (R,i) and its children $(C_1,j_1)...,(C_k,j_k)$ in $\mathcal{L}'$. If $k>1$, $t_i, t_{j_1},..,t_{j_k}$ are all outside of R, and there is no endpoint of an edge of E' in $R - \bigcup_{l=1}^{l=k} C_l$, then there must be no child of (R,i) with only children-out edges.*

Proof: Since there is no endpoint of an edge of E' in $R - \bigcup_{l=1}^{l=k} C_l$, we can only have children-out (CO) and children-children (CC) edges for (R,i) and its children.

Since $k > 1$, without loss of generality, assume C_1 has only children-out edges. Let $C' = \bigcup_{l=2}^{l=k} C_l$. Then there must be no (C_1C') edges. Since $t_i, t_{j_1}, .., t_{j_k}$ are all outside of R, then by feasibility $x(C_1O) + y_1 \geq 1$ and $x(C'O) + y_j \geq 1$ for any terminal pair j in C'. Hence, we must also have $x(C_1O) > 0$ and $x(C'O) > 0$, since we assumed all y variables have value less than $\frac{1}{3}$.

If $s_i \in C_1$, we have $x(C_1O) + x(C'O) + y_i = 1$ and $x(C_1O) + y_i \geq 1$ (by feasibility). This contradicts the fact that $x(C'O) > 0$. Similarly, if $s_i \in C'$, we have $x(C_1O) + x(C'O) + y_i = 1$ and $x(C'O) + y_i \geq 1$. This contradicts the fact that $x(C_1O) > 0$. □

We say an endpoint is *incident* to a cut-terminal pair (S, i) in $\mathcal{L}'$, if for all cut-terminal paris in $\mathcal{L}'$, S is the smallest set containing that endpoint. So an endpoint is incident to one cut-terminal pair only. We define the *degree* of (S, i) to be the number of edges in $\delta(S)$, i.e. $|\delta(S)|$. We also say a terminal pair i is *contained* in a cut-terminal pair (S, j) in $\mathcal{L}'$ if S is the smallest set contains both terminals s_i and t_i. For any cut-terminal pair $(S, i) \in \mathcal{L}'$, we always assume with loss of generality that $s_i \in S$ and $t_i \notin S$.

Our proof is a contradiction-based proof. Assume Theorem 2.3.1 fails, i.e. $x_e < \frac{1}{3}$ for all e in E and $y_i < \frac{1}{3}$ for all i in T. Consider edges in E' and terminals in V''. We will distribute one token to cut-terminal pair $(S, i) \in \mathcal{L}'$ for each endpoint of E' incident to (S, i) and distribute two tokens to (S, i) for each terminal pair $i \in V''$ contained in (S, i). We will show that we can redistribute tokens such that every cut-terminal pair in $\mathcal{L}'$ gets at least two tokens and some pairs get strictly more than two tokens. This contradicts the equality $|\mathcal{L}'| = |E'| + |V''|$.

We define a subtree rooted at cut-terminal pair (R, i) to be a subset of $\mathcal{L}'$ that consists of (R, i) and all its descendants. We will do the redistribution of tokens inductively on every rooted subtree.

Lemma 2.4.11 *For any rooted subtree of $\mathcal{L}'$, we can redistribute the tokens in it such that every cut-terminal pair of its descendants gets at least 2 tokens, and the root gets at least 3. Furthermore, the root gets exactly 3 tokens only if its degree is 3.*

Proof: Consider the base case in which the subtree is just a leaf pair (R, i). If $y_i = 0$, it must have degree at least four otherwise some edge e in $\delta(R)$ must have $x_e \geq \frac{1}{3}$, contradicting to our assumption. So (R, i) must have at least four endpoints incident to it, i.e. at least four tokens. Similarly, if $y_i > 0$, it must have at least three endpoints incident to it, i.e. at least three tokens, otherwise some x_e or y_i is greater or equal to $\frac{1}{3}$. Furthermore, (R, i) gets exactly 3 tokens only if its degree is 3. So the lemma is true for the base case.

For the inductive step, consider a subtree rooted at (R, i). We have four cases depending on the number of children of (R, i).

Case (a): (R, i) has four or more children.

Each of them has at least three tokens by induction hypothesis, so we can easily redistribute four tokens to (R, i).

Case (b): (R, i) has three children (C_1, j_1), (C_2, j_2) and (C_3, j_3).

We are done if one of them has more than three tokens. If all three children have exactly three tokens, then each child has degree exactly three by our induction hypothesis. We need to consider the locations of terminals. If any terminal s_l is in $R - C_1 - C_2 - C_3$, then (R, i) gets at least three tokens since by feasibility of $[x\ y]$ the cut-terminal pair $(\{s\}, l)$ must have $x(\delta(\{s\})) + y_l \geq 1$ and the degree of

$(\{s\}, l)$ is at least three by same argument in the base case. If any terminal pair of j_1, j_2, and j_3 has both terminals in R, then the terminal pair is contained in (R, i) and (R, i) gets two tokens as well. In both cases, (R, i) can get at least four tokens by having one token from two of its children.

We can then consider only cases in which no terminal is in $R - C_1 - C_2 - C_3$ and all terminals $t_i, t_{j_1}, t_{j_2}, t_{j_3} \notin R$. Now consider endpoints of edges in E'. If there is a non-terminal endpoint v of an edge e in E' that is in $R - C_1 - C_2 - C_3$, then there must be at least another edge e' in E' that also has v as its endpoint, giving at least two tokens to (R, i). Otherwise, the edge e is either a parent-out (PO) edge or a children-parent (CO) edge. If it is a (PO) edge, then we have $x(\delta(R)) = x(\delta(R - \{v\})) + x_e$. We know $x_e > 0$ and $x(\delta(R)) + y_i = 1$, which implies that $x(\delta(R - \{v\})) + y_i < 1$, contradicting the feasibility of $[x\ y]$. A similar argument applies to the case that e is a (CP) edge since all terminals $t_{j_1}, t_{j_2}, t_{j_3} \notin R$. So (R, i) can get at least four tokens by having one token from two of its children in this case as well.

Hence, we can now further assume that there is no non-terminal endpoint of an edge of E' in $R - C_1 - C_2 - C_3$. Then only possible edges in $\delta(R)$, $\delta(C_1)$, $\delta(C_2)$, $\delta(C_3)$ are children-out (CO) edges and children-children (CC) edges. Consider the degree of (R, i). We have $|\delta(R)| = |\delta(C_1)| + |\delta(C_2)| + |\delta(C_3)| - 2|CC| = 9 - 2|CC|$. So by parity, (R, i) has odd degree. Also, the degree of (R, i) is at least three by the same argument for the base case. Therefore, the degree of (R, i) is three, five, seven or nine. We are done if the degree of (R, i) is exactly three and we will show all other cases are not possible.

By Lemma 2.4.10, no child can have all three edges be children-out edges. This implies the degree of (R, i) cannot be seven or nine.

If degree if (R, i) is five and no child has three children-out edges, then we can assume without loss of generality that we have one edge in (C_1O), two edges in (C_2O) and (C_3O), and one edge in (C_1C_2) and (C_1C_3). By Lemma 2.4.9, we must also have $y_i = \min\{y_{j_1}, y_{j_2}, y_{j_3}\}$ and s_i is in the child C_{j_l} with y_{j_l} attained this minimum.

If $s_i \in C_1$, then $y_i = y_{j_1} = \min\{y_{j_1}, y_{j_2}, y_{j_3}\}$. We have $x(C_1O)+x(C_2O)+x(C_3O)+y_i = 1$ and $x(C_1O) + x(C_1C_2) + x(C_1C_3) + y_{j_1} = 1$, which implies $x(C_1C_2) + x(C_1C_3) = x(C_2O) + x(C_3O)$. Then either $x(C_1C_2) \geq x(C_2O)$ or $x(C_1C_3) \geq x(C_3O)$. Assume it is $x(C_1C_2) \geq x(C_2O)$. Since $x(C_1C_2) + x(C_2O) + y_{j_2} = 1$, the single edge e of C_1C_2 has $x_e \geq \frac{1}{3}$ since $y_{j_2} < \frac{1}{3}$, contradicting our assumption.

If $s_i \in C_2$ (case for $s_i \in C_3$ is identical), then $y_i = y_{j_2} = \min\{y_{j_1}, y_{j_2}, y_{j_3}\}$. A similar argument as above shows this case is not possible as well. We have $x(C_1O) + x(C_2O) + x(C_3O) + y_i = 1$ and $x(C_2O) + x(C_1C_2) + y_i = 1$, which implies $x(C_1C_2) = x(C_1O) + x(C_3O)$. We must have $x(C_1C_2) < \frac{1}{3}$ by our assumption since it is a single edge in (C_1C_2). Then we have $x(C_3O) < \frac{1}{3}$ which implies the single edge e in (C_1C_3) has $x_e > \frac{1}{3}$, since $x(C_1C_3) + x(C_3O) + y_{j_3} = 1$ and $y_{j_3} < \frac{1}{3}$.

Case (c): (R, i) has two children (C_1, j_1) and (C_2, j_2).

We are done if both children have at least four tokens. Assume without loss of generality (C_1, j) has exactly three tokens. As in case (b), we can consider only the case that $t_i, t_{j_1}, t_{j_2} \notin R$, $s_{j_1} \in C_1$, $s_{j_2} \in C_2$, s_i is in one of C_1, or C_2 and no terminal or non-terminal endpoint of E' is in $R - C_1 - C_2$, since otherwise (R, i) can get at least four tokens by having two tokens from itself and one token from each of its children. Also, by Lemma 2.4.9, we must also have $y_i = \min\{y_{j_1}, y_{j_2}\}$ and s_i is in the child C_{j_l} with y_{j_l} attaining this minimum.

If $s_i \in C_1$, then $y_i = y_{j_1} \leq y_{j_2}$. By Lemma 2.4.10, not all three edges of (C_1, j_1) can be children-out (C_1O) edges. If it has two C_1O edges and one C_1C_2 edge, then we have $x(C_1C_2) + x(C_1O) + y_{j_1} = 1$ and $x(C_1O) + x(C_2O) + y_i = 1$. Since $y_i = y_{j_1}$, we have $x(C_1C_2) = x(C_2O)$. But we also have $x(C_1C_2) + x(C_2O) + y_{j_2} = 1$ and $y_{j_2} < \frac{1}{3}$. So the single edge e of C_1C_2 has $x_e \geq \frac{1}{3}$, contradicting our assumption. If (C_1, j_1) has one C_1O edges and two C_1C_2 edges, we have $x(C_1O) + x(C_2O) + y_i = 1$ and $x(C_1C_2) + x(C_2O) + y_{j_2} = 1$. Since $y_i \leq y_{j_2}$, we have $x(C_1O) \geq x(C_1C_2)$. But we also have $x(C_1O) + x(C_1C_2) + y_{j_1} = 1$ and $y_{j_1} \leq \frac{1}{3}$. So the single edge e of C_1O has $x_e \geq \frac{1}{3}$, again, a contradiction. If (C_1, j_1) has three (C_1C_2) edges, then we have $x(C_1C_2) + x(C_2O) + y_{j_2} = 1$ and $x(C_2O) + y_i = 1$. Since $y_i \leq y_{j_2}$, we must have $x(C_1C_2) = 0$ which contradicts our assumption that all edges in E' have $x_e > 0$. If $s_i \in C_2$, then $y_i = y_{j_2} \leq y_{j_1}$, and a similar argument as above shows this case is not possible as well.

Case (d): (R, i) has one child (C_1, j).

First, (R, i) must have at least one endpoint v of E' in $R - C$, otherwise δ_R and δ_C are the same, which is not possible by the linear independence property of $\mathcal{L}'$. By same argument in case (b), (R, i) has at least three tokens if v is a terminal and at least two tokens if v is a non-terminal endpoint. We are done if the surplus token(s) from (C_1, j) can leave (R, i) with four tokens. The only case left is when (R, i) has exactly one endpoint in $R - C$, (C, j) has exactly three tokens, and $t_j \notin (R, i)$. In this case, we must have one children-parent edge, one parent-out edge and two children-out edges for (R, i) and (C, j), so the degree of (R, i) is exactly three. Then (R, i) can take one token from (C, j) so it has exactly three tokens.

□

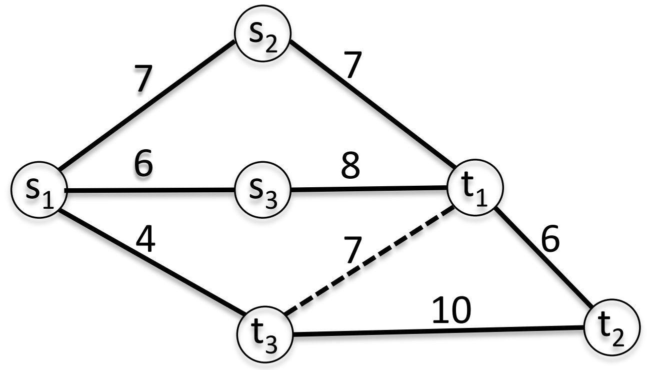


Figure 2.4: Tight example for iterative rounding algorithm. Penalties for each terminal pairs are $\pi_1 = 10, \pi_2 = 9, \pi_3 = 6$. The unique optimal solution is $y_1 = 0, y_2 = \frac{1}{3}, y_3 = \frac{1}{3}$, all solid edges have $x_e = \frac{1}{3}$, and the dotted edge has $x_e = 0$.

Therefore, it cannot be the case that $|\mathcal{L}'| = |E'| + |V''|$, contradicting our assumption that Theorem 2.3.1 fails. □

2.5 Remarks

First, notice that our definition of an extended weakly supermodular function and Theorem 2.4.6 can both be extended to the higher connectivity case of the prize-collecting Steiner forest problem, i.e. $f(S, i) = r_i$ for any integer $r_i \geq 0$. However, our proof for Theorem 2.3.1 relies on the fact that f is a 0-1 function.

Second, Figure 2.4 shows a unique optimal solution to an instance of PCSF problem where all variables have value less than or equal to $\frac{1}{3}$. This implies that our algorithm is tight with approximation factor 3 and the iterative rounding approach cannot be used directly to obtain a performance guarantee better than 3.

CHAPTER 3

AN $O(\text{LOG } N)$-COMPETITIVE ALGORITHM FOR ONLINE CONSTRAINED FOREST

3.1 Introduction

Given an undirected graph $G = (V, E)$, edge costs $c_e \geq 0$ for all $e \in E$, and a set of l source-sink pairs s_i-t_i, the goal of the *generalized Steiner tree problem* (also known as the *Steiner forest problem*) is to find a minimum-cost set of edges $F \subseteq E$ such that for each i, s_i and t_i are connected in (V, F). This problem is (as its name implies) a generalization of the *Steiner tree problem*: in Steiner tree problem, we are given an undirected graph with edge costs as above, and also a set $R \subseteq V$ of *terminals*. The goal of the Steiner tree problem is to find a minimum-cost tree T that spans all the terminals R. If we choose one of the terminals $r \in R$ arbitrarily, and set $s_i = r$ for all i and the sink vertices t_i are the remaining vertices in R, then clearly a Steiner tree instance can be expressed as a generalized Steiner tree problem instance. In the 1990s, Agrawal, Klein, and Ravi [1] gave a 2-approximation algorithm for the generalized Steiner tree problem.

At about the same time, online algorithms were being proposed for online versions of the Steiner tree problem, and later, the generalized Steiner tree problem. In the online version of the Steiner tree problem, terminals arrive over time. At each time step we must give a set of edges F that connects all of the terminals that have arrived thus far; we are not allowed to remove any edges from F in future iterations. The quality of an online algorithm for this problem is measured in terms of its *competitive ratio*: an α-competitive algorithm is one such that at any time step, the set of edges constructed by the algorithm is within a factor of α of the cost of an optimal Steiner tree on the set of terminals that have arrived thus far. Similarly, in the online generalized Steiner tree problem, source-sink pairs arrive in each time step, and we must find a set of edges F such that each s_i-t_i pair that has arrived thus far is connected in (V, F). Imase and Waxman [23] gave a greedy $O(\log n)$-competitive algorithm for the online Steiner tree problem, where $n = |V|$; when a terminal arrives, it finds the shortest path from the terminal to the tree already constructed, and adds that set of edges to its solution. Imase and Waxman also showed that the competitive ratio of any online algorithm must be at least $\frac{1}{2} \log n$. Awerbuch, Azar, and Bartal [2] then showed that a similar greedy algorithm for the online generalized Steiner tree problem has competitive ratio $O(\log^2 n)$. In 1997, Berman and Coulston [4] devised a more complicated algorithm that is an $O(\log n)$-competitive algorithm for the online generalized Steiner tree problem, matching the lower bound of Imase and Waxman to within constant factors.

Also in the 1990s, Goemans and Williamson [17] extended the offline algorithm of Agrawal, Klein, and Ravi to a large class of problems they called constrained forest problems; in doing so, they cast the algorithm of Agrawal et al. as a primal-dual algorithm. A constrained forest problem is defined by a function $f : 2^V \to \{0, 1\}$; for any set $S \subseteq V$ such that $f(S) = 1$, a feasible solution must have selected at least one edge in $\delta(S)$, the set of edges with exactly one endpoint in S. The Goemans-Williamson algorithm works when the function f is *proper*: that is, when $f(S) = f(V - S)$ for all $S \subseteq V$, and for all disjoint sets $A, B \subseteq V$, $f(A \cup B) \leq \max(f(A), f(B))$. For instance, for the case of the generalized Steiner tree problem $f(S) = 1$ if and only if there exists some i such that $|S \cap \{s_i, t_i\}| = 1$, and this function is proper. Another example of constrained forest problems given in [17] is the nonfixed point-to-point connection problem, in which a subset C of the vertices are sources, a disjoint subset D of vertices are destinations, and we must find a minimum-cost set of edges such that each connected component has the same number of sources and destinations; this is modelled by having $f(S) = 1$ if $|S \cap C| \neq |S \cap D|$. Yet another example given in [17] is that of partitioning specified vertices D into connected components such that the number of vertices of D in each connected component C is divisible by some parameter k. This problem is given the proper function f such that $f(S) = 1$ if $|S \cap D| \not\equiv 0 (\text{mod } k)$.

In this chapter, we show that by melding the ideas of Goemans and Williamson with those of Berman and Coulston, we can obtain an $O(\log n)$-competitive algorithm for any *online* constrained forest problem. In an online constrained forest problem, in each time step i we are given a proper function f_i. We must choose a set of edges F such that for all $S \subseteq V$, if $\max_{j=1,\ldots,i} f_j(S) = 1$, then $|\delta(S) \cap F| \geq 1$. This yields, for example, online algorithms for online variants of the nonfixed point-to-point connection problem and the partitioning problems given above.

Our techniques also extend to give an $O(\log n)$-competitive algorithm for an online version of the prize-collecting Steiner tree problem and its generalizations. In the offline version of the prize-collecting Steiner tree problem, we are given an undirected graph $G = (V, E)$, edge costs $c_e \geq 0$ for all $e \in E$, a root vertex $r \in V$, and penalties $\pi_v \geq 0$ for all $v \in V$. The goal is to find a tree T spanning the root vertex that minimizes the cost of the edges in the tree plus the penalties of the vertices not spanned by the tree; that is, we want to minimize $\sum_{e \in T} c_e + \sum_{v \in V - V(T)} \pi_v$, where $V(T)$ is the set of vertices spanned by T. In the online version of the problem, initially every vertex v has penalty $\pi_v = 0$. At each step in time, for one vertex v its penalty π_v is increased from 0 to some positive value. We then must either connect the vertex to the root by adding edges to our current solution or pay the penalty π_v for each remaining time step of the algorithm even if it is connected to the root later on. The competitive ratio of the algorithm compares the cost of our solution in each step with the cost of the optimal solution of the instance at that point in time.

The basic idea of the Berman-Coulston algorithm (BC) is that it constructs many different families of nonoverlapping balls around terminals as they arrive; in the jth family, balls are limited to have radius at most 2^j. Each family of balls is a lower bound on the cost of an optimal solution to the generalized Steiner tree problem. When balls from two different terminals touch, the algorithm buys the set of edges connecting the two terminals, and balls from one of the two terminals (in some sense the 'smaller' one) can be charged for the cost of the edges, leaving the balls from the other terminal (the 'larger' one) uncharged and able to pay for future connections. One can show that the $O(\log n)$ largest families are essentially all that are relevant for the charging scheme, so that the largest of these $O(\log n)$ families is within an $O(\log n)$ factor of the cost of the constructed solution, thereby giving the competitive ratio. Our algorithm replaces each family of balls with an analogous solution to the dual of the linear programming relaxation of the constrained forest problem, as used by Goemans and Williamson. We need somewhat more complicated dual solutions than the balls used by BC. However, we can then largely follow the outline of the BC analysis to obtain our $O(\log n)$ competitive ratio.

The rest of this chapter is structured as follows. In Section 3.2, we introduce the online constrained forest problem more precisely and define some concepts we will need for our algorithm. In Section 3.3, we give the algorithm and its analysis. In Sections 3.4, 3.5 and 3.6, we extend the algorithm and analysis to the online prize-collecting Steiner tree problem and its generalizations.

3.2 Preliminaries

Given an undirected graph $G = (V, E)$, edges costs $c_e \geq 0$ and a $\{0, 1\}$-proper function $f : 2^V \to \{0, 1\}$, the offine constrained forest problem studied in Goemans and Williamson [17] is to find a set of edges F of minimum cost that satisfies a connectivity requirement function $f : 2^V \to \{0, 1\}$; the function is *satisfied* if for each set $S \subseteq V$ with $f(S) = 1$, we have $|\delta_F(S)| \geq 1$, where $\delta(S)$ is the set of edges with exactly one endpoint in S, and $\delta_F(S) = \delta(S) \cap F$. In the online version of this problem, we have a sequence of connectivity functions $f_1, f_2, ..., f_i$, arriving one by one. Starting with $F = \emptyset$, for each *time step* $i \geq 1$, function f_i arrives and we need to add edges to F to satisfy function f_i. Let $g_i(S) = \max\{f_1(S), ..., f_i(S)\}$ for all $S \subseteq V$ and $i \geq 1$. Then our goal is to a find a minimum-cost set of edges F that satisfies function g_i, that is, all connectivity requirements given by $f_1, ..., f_i$ that have arrived thus far. We require that each function f_i be a proper function, as defined above. It is easy to see that function g_i is also proper.

Call a vertex v a *terminal* if $f_l(\{v\}) = 1$ for some $l \leq i$. Let $R_i = \{s \in V \mid g_i(\{s\}) = 1\}$ be the set of terminals defined by function g_i; that is, R_i is the set of all terminals that have arrived by time i. A special case of this problem is the online generalized Steiner tree problem where terminal pairs $(s_1, t_1), ..., (s_i, t_i)$ arrive one at a time. In this case, $f_i(S) = 1$ if $|S \cap \{s_i, t_i\}| = 1$ and (s_i, t_i) is the pair of terminals arrive in time step i; then $R_i = \{s_j, t_j : j \leq i\}$. Berman and Coulston [4] give an

$O(\log |R_i|)$-competitive algorithm for the online generalized Steiner tree problem.

Let (IP_i) be the integer program corresponding to the online proper constrained forest problem with set of functions $f_1, ..., f_i$ that have arrived thus far and the corresponding function g_i. The integer programming formulation of (IP_i) is

$$(IP_i) \qquad \begin{array}{lll} \text{Min} & \displaystyle\sum_{e \in E} c_e x_e & \\ & \displaystyle\sum_{e \in \delta(S)} x_e \geq g_i(S), & \forall S \subseteq V, \\ & x_e \in \{0,1\}, & \forall e \in E. \end{array}$$

We let (LP_i) denote the corresponding linear programming relaxation in which the constraints $x_e \in \{0,1\}$ are replaced with $x_e \geq 0$. The dual of this linear program, (D_i), can be described as

$$(D_i) \qquad \begin{array}{lll} \text{Max} & \displaystyle\sum_{S \subseteq V} g_i(S) y_S & \\ & \displaystyle\sum_{S : e \in \delta(S)} y_S \leq c_e, & \forall e \in E, \\ & y_S \geq 0, & \forall S \subseteq V. \end{array}$$

We now define a number of terms that we will need to describe our algorithm. We will keep an infinite number of feasible dual solutions y^j to bound the cost of edges in F over all time steps; we call this the dual solution for *level* j. For each level j, we will maintain that for any terminal s that has arrived thus far, $\sum_{S \subseteq V : s \in S} y_S^j \leq 2^j$. So we say that the *limit* of the dual in level j is 2^j, and we say that a dual variable y_S^j *reaches its limit* if the inequality for level j is tight for any terminal $s \in S$. As a matter of algorithmic implementation, we don't need to maintain levels $j < -1$, or $j > \lceil \log(\max_{u,v \in V} d(u,v)) \rceil$, where $d(u,v)$ is the distance in G between u and v using edge costs c_e.

An edge $e \in E$ is *tight* in level j for dual vector y^j if the corresponding constraint in dual problem (D_i), $\sum_{S : e \in \delta(S)} y_S^j \leq c_e$, holds with equality. A path $p \subseteq E$ is *tight* in level j if every edge in the path is tight in level j.

Let $\bar{F}^j$ denote the set of edges that are tight in level j. To avoid confusion with connected components in F, we will use the term *moat* to refer to a connected component S^j in $\bar{F}^j$ and use y_S^j to refer the dual variable associated with S^j.

A set $S \subseteq V$ is a *violated* set for function g_i by edges B if $|\delta_B(S)| < g_i(S)$; that is, if $g_i(S) = 1$ but $\delta_B(S) = \emptyset$. A *minimal violated set* is a violated set with every strict subset not violated. The connectivity requirement function g_i is satisfied by edges B if every set $S \subseteq V$ is not a violated set for g_i by B.

During time step i, a terminal $s \in R_i$ is *active* if for some set S, we have $s \in S$ and S is a violated set for function g_i by current solution F. Let A be the set of active terminals at any time of the algorithm. We define the set of *active* moats as the moats S^j of the lowest level j that satisfy the following three conditions: (i) S^j contains some active terminal $s \in A$; (ii) S^j is a minimum violated set for g_i by edges $\bar{F}^j$; (iii) y_S^j has not yet reached its limit in level j. We denote the current set of active moats by $\mathcal{M}$. Last, we say a dual variable y_S^j is *active* if its corresponding moat S^j is active.

3.3 The Algorithm and Its Analysis

3.3.1 The Primal-Dual Online Algorithm

Our algorithm (see Fig. 3.1) is a dual ascent algorithm in which we grow active dual variables. We say two disjoint moats S_1^j and S_2^j *collide* in level j during our dual growing process if both of them have been active at some point and a path connecting two terminals $s_1 \in S_1^j$ and $s_2 \in S_2^j$ becomes tight in level j. In order for this to happen, at least one of S_1^j and S_2^j must currently be active.

Our algorithm starts with $F = \emptyset$ and $y_S^j = 0$ for all j and all $S \subseteq V$. At the beginning of each time step i, the function f_i arrives and some non-terminal nodes in V may become terminals. We will update active terminal set A and active moat set $\mathcal{M}$. In each time step i, while there are still some active terminals in A, our algorithm will grow uniformly all active dual variables until: (1) an active y_S^j with reaches its limit in level j; (2) an edge $e \in E$ becomes tight in level j; we then add e to $\bar{F}^j$; (3) two disjoint moats S_1^j and S_2^j collide in level j; we then let p be the path connecting two terminals $s_1 \in S_1^j$ and $s_2 \in S_2^j$ in level j, and we build path p in F if s_1 and s_2 are not yet connected in F, and update the set A of active terminals. At the end of each iteration, we update the set $\mathcal{M}$ of active moats. We output F as the solution for (IP_i).

3.3.2 The Analysis

Now, we will now state our main theorem and a few lemmas that we need to prove it.

Algorithm

```
$F = \emptyset$, $\bar{F}^j = \emptyset$ for all $j$, and $y_S^j = 0$ for all $j$ and $S \subseteq V$
For each $\{0,1\}$-proper function $f_i$ that arrives
    Update active terminals $A$, and active moats $\mathcal{M}$
    While $|A| > 0$
        Grow uniformly all active dual variables $y_S^j$ until
        1) An active $y_S^j$ with reaches limit in level $j$
        2) An edge $e \in E$ becomes tight in level $j$, then
            $\bar{F}^j = \bar{F}^j \cup \{e\}$
        3) Two disjoint moats $S_1^j$ and $S_2^j$ collide in level $j$, then
            Let $p \subseteq E$ be the corresponding path that becomes tight in level $j$
            Let $s_1$ and $s_2$ be the two terminals connected by $p$ in $\bar{F}^j$
            If $s_1$ and $s_2$ are yet connected in $F$
                $F = F \cup \{p\}$, i.e. build edges $p - F$
                Update $A$
        Update $\mathcal{M}$
```

Figure 3.1: Primal-dual algorithm for the online proper constrained forest problem

Theorem 3.3.1 *Our algorithm gives an $O(\log |R_i|)$ competitive ratio for the online proper constrained forest problem (IP_i).*

Lemma 3.3.2 *At the end of time step i of our algorithm, F is a feasible solution to (IP_i) and each dual vector y^j is a feasible solution to (D_i).*

Proof: Our algorithm terminates each time step i when there are no active terminals in A. By definition of active terminals, this implies that there is no violated set for g_i for the solutions F; that is, F is a feasible solution to (IP_i).

We need to show our algorithm always terminates in each time step. Notice that if there are no active moats in $\mathcal{M}$, then there must be no active terminals in A, since there is always a level j for sufficiently large j that conditions (ii) and (iii) are satisfied in the definition of $\mathcal{M}$. Similarly, if there is still an active terminal, there must be an active moat that contains it. Then our algorithm will continue to grow duals at progressively higher levels; eventually all pairs of active terminals must be connected.

By construction of the algorithm each dual solution y^j is feasible for (D_i) since we stop growing a dual y_S^j if it would violate a dual constraint. □

In order to give a bound on the total cost of edges in F, we create an account for each connected component X in F, denoted $\text{Account}(X)$. We will define a shadow algorithm to credit potential to accounts as dual grows and remove potential from accounts to pay for building edges. We will show that the total cost of edges in F plus the total unused potential remaining in all accounts is always equal to the sum of all dual variables over all levels, i.e. $\sum_j \sum_S y_S^j$.

We need the following lemma before we describe the shadow algorithm.

Lemma 3.3.3 *Any active dual variable y_S^j has a unique connected component X in F that contains all terminals in its corresponding moat S^j.*

Proof: It is sufficient to show that all terminals in an active moat S^j are connected in F. Suppose not; then we have terminals s_1 and s_2 both in S^j and not connected in F. Then either s_1 and s_2 have no path in $\bar{F}^j$ connecting them or at least one of s_1 and s_2 was not a terminal when the path in $\bar{F}^j$ connecting s_1 and s_2 became tight. The first case contradicts the fact that y_S^j is active so that S^j must be a moat, i.e. a connected component in $\bar{F}^j$. The second case cannot happen by the construction of our algorithm since we grow duals from lowest levels possible. When s_1 and s_2 became terminals, some level j' with $j' < j$ small enough will have s_1 and s_2 in different moats, and a path p between them becomes tight in some level between j' and $j-1$ by our structure of limits in each level (i.e. dual growth in one level can be no larger than two times of dual growth in one level below). Then path p is built in F and s_1 and s_2 will be connected in F before the algorithm grows dual in level j again. □

Now the shadow algorithm is as follows. First, whenever we grow an active dual variable y_S^j, we will credit the same amount of potential to Account(X), where X is the unique connected component in F that contains all terminals in S^j. Second, whenever the algorithm builds a path p in F connecting two terminal s_1 and s_2, it must be the case that two disjoint moats S_1^j and S_2^j collide in some level j with $s_1 \in S_1^j$ and $s_2 \in S_2^j$ not yet connected in F. Let X_k be connected component in F that contains s_k for $k = 1, 2$. As a result of building edges $p - F$, $X_3 = X_1 \cup X_2 \cup \{p - F\}$ will become a connected component in F. We will merge unused potential remaining in Account(X_1) and Account(X_2) into Account(X_3) and remove potential from Account(X_3) to pay for the cost of building edges in $p - F$.

At any time of the algorithm, for each connected component X, define the class of X to be the highest level j with a dual variable already grown that credits X; we denote this as as Class(X) and sometimes refer to it as the *top level* of X. Define TopGrowth(X) to be the maximum total dual growth of a terminal in X in level Class(X), i.e.

$$\text{TopGrowth}(X) = \max_{s \in X}\{\sum_{S \subseteq V : s \in S} y_S^{\text{Class}(X)} \text{ and } s \text{ is a terminal}\}.$$

We know that TopGrowth(X) $\leq 2^j$ by dual limit on level j.

Lemma 3.3.4 *At any time of the algorithm, the following two invariants hold:*

1. *Every connected component X of F has*

$$\text{Account}(X) \geq 2^{\text{Class}(X)} + \text{TopGrowth}(X);$$

2. $\sum_{e \in F} c_e + \sum_{X \in F} \text{Account}(X) = \sum_j \sum_S y_S^j$.

Proof: Invariant 1 ensures that for a component X, Account(X) stores at least 2^j total potential for each level $j < \text{Class}(X)$ plus the maximum total dual growth of a terminal in X at the top level, which gives $2^{\text{Class}(X)-1} + 2^{\text{Class}(X)-2} + ... = 2^{\text{Class}(X)}$ plus TopGrowth(X).

We prove the first invariant by induction on the algorithm. It is easy to see that this invariant holds when no edges are added to F since the algorithm grows dual variables in level j until some active dual variable reaches limit 2^j; it then grows duals in next higher level. Account(X) is credited 2^j for each level below the level Class(X) while getting TopGrowth(X) for current level.

Now, assume invariant 1 holds just before we add edges to F. The algorithm builds a path p in F connecting two terminals s_1 and s_2 only if there are two disjoint moats S_1^j and S_2^j that collide in some level j with $s_1 \in S_1^j$ and $s_2 \in S_2^j$ not yet connected in F. Let X_k be connected component in F that contains s_k for $k = 1, 2$. We know at least one of S_1^j and S_2^j must be active. Without loss of generality, let it be S_1^j. Then we know $\text{Class}(X_1) = j$ and $\text{Class}(X_2) = j' \geq j$ since we only grow active dual variables in the top level of each component in F. Then by assumption, $\text{Account}(X_1) \geq 2^j + \text{TopGrowth}(X_1)$ and $\text{Account}(X_2) \geq 2^{j'} + \text{TopGrowth}(X_2)$. After building edges $p - F$, $X_3 = X_1 \cup X_2 \cup \{p - F\}$ will be a connected component in F. Our shadow algorithm merges the unused potential remaining in Account(X_1) and Account(X_2) into Account(X_3), and removes potential from Account(X_3) to pay for the cost of building edges $p - F$. Since $\text{Class}(X_3) = \max\{j, j'\} = j'$, we need to show Account(X_3) $\geq 2^{j'}$ + TopGrowth(X_3).

If $j = j'$, we know TopGrowth(X_1) + TopGrowth(X_2) $\geq \sum_{e \in p} c_e \geq \sum_{e \in p-F} c_e$ since S_1^j and S_2^j collide in level j, and the cost of the path from s_1 to s_2 cannot be more than the total dual containing s_1 and s_2 in level j. Also, TopGrowth(X_3) $\leq 2^j$ by the dual limit on level j. So we have

$$\begin{aligned}
\text{Account}(X_3) &= \text{Account}(X_1) + \text{Account}(X_2) - \textstyle\sum_{e \in p-F} c_e \\
&\geq 2^j + \text{TopGrowth}(X_1) + 2^j + \text{TopGrowth}(X_2) - \textstyle\sum_{e \in p-F} c_e \\
&\geq 2^j + 2^j \geq 2^{j'} + \text{TopGrowth}(X_3).
\end{aligned}$$

If $j < j'$, we know TopGrowth(X_1) + $2^j \geq \sum_{e \in p} c_e \geq \sum_{e \in p-F} c_e$ since S_1^j and S_2^j collide in level j and the cost of the path from s_1 to s_2 cannot be more than the total dual containing s_1 and s_2 in level j. Also, TopGrowth(X_3) = TopGrowth(X_2) since $j < j'$. So, we have

$$\begin{aligned}
\text{Account}(X_3) &= \text{Account}(X_1) + \text{Account}(X_2) - \textstyle\sum_{e \in p-F} c_e \\
&\geq 2^j + \text{TopGrowth}(X_1) + 2^{j'} + \text{TopGrowth}(X_2) - \textstyle\sum_{e \in p-F} c_e \\
&\geq 2^{j'} + \text{TopGrowth}(X_3).
\end{aligned}$$

Therefore, the invariant 1 holds at any time of the algorithm. Furthermore, since accounts get credited for dual growth and debited exactly the cost of edges in F, we also have that invariant 2 holds at any time of the algorithm. $\square$

Lemma 3.3.5 *Let the dual vector y^j with the maximum total dual $\sum_S y_S^j$ be $y^{\max}$. At the end of time step i, we have $\sum_{e \in F} c_e \leq (\log |R_i| + 2) \sum_S y_S^{max}$.*

Proof: By invariant 2 of Lemma 3.3.4, $\sum_{e \in F} c_e = \sum_j \sum_S y_S^j - \sum_{X \in F}$ Account(X). So it suffices to show $\sum_j \sum_S y_S^j - \sum_{X \in F}$ Account(X) $\leq (\log |R_i| + 2) \sum_S y_S^{max}$.

At the end of time step i, let X^* be a connected component in F of highest class and let $m = \text{Class}(X^*)$. We know $\text{Account}(X^*) \geq 2^m$ by invariant 1 of Lemma 3.3.4. So the total unused potential remaining in all accounts (that is, $\sum_{X\in F} \text{Account}(X)$) is at least 2^m.

Let $\lambda = \lceil \log_2 |R_i| \rceil$. Each terminal s has total dual $\sum_{S\subseteq V: s\in S} y_S^{-\lambda} \leq 2^{-\lambda} \leq 1/|R_i|$ in level $-\lambda$ by the dual limit, so that $\sum_S y_S^{-\lambda} \leq 1$. Similarly, we have $\sum_S y_S^{m-\lambda-j-1} \leq 2^{m-j-1}$. Consider all dual vectors of the form $y^{m-\lambda-j-1}$ with $j \geq 0$; then we have $\sum_{j\geq 0} \sum_S y_S^{m-\lambda-j-1} \leq 2^{m-1} + 2^{m-2} + 2^{m-3} + ... = 2^m \leq \sum_{X\in F} \text{Account}(X)$.

Then, if we consider the dual solutions $y^{m-\lambda}, \ldots, y^m$, we have

$$\sum_j \sum_S y_S^j - \sum_{X\in F} \text{Account}(X) \leq \sum_{k=0}^{\lambda} \sum_S y_S^{m-\lambda+k} \leq (\log |R_i| + 2) \sum_S y_S^{max}.$$

The lemma follows. □

Now, we are ready to prove Theorem 3.3.1.

Proof of Theorem 3.3.1: By Lemma 3.3.2, at the end of time step i of the algorithm, F is a feasible solution to (IP_i). We have

$$\begin{array}{rcll} \sum_{e\in F} c_e & = & \sum_j \sum_S y_S^j - \sum_{X\in F} \text{Account}(X) & \text{by Lemma 3.3.4} \\ & \leq & (\log |R_i| + 2) \sum_S y_S^{\max} & \text{by Lemma 3.3.5} \\ & \leq & (\log |R_i| + 2) OPT_i & \text{by Lemma 3.3.2} \end{array}$$

where OPT_i is the optimal value of (IP_i) and the last inequality follows by the fact that the cost of a feasible dual solution to (D_i) is a lower bound on OPT_i. Therefore, our algorithms is an $O(\log |R_i|)$-competitive algorithm for the online proper constrained forest problem. Note that we have $|R_i| = O(n)$, where is n is the number of nodes in G. □

3.4 The Online Prize-Collecting Steiner Tree Problem

We now show that our algorithm can be extended to solve the prize-collecting Steiner tree problem. Define the online prize-collecting Steiner tree problem as follows: we are given a root node r in G, and a penalty of zero for each non-root node. In each time step i, a terminal $s_i \neq r$ arrives with a new penalty $\pi_i > 0$. We have a choice to either connect s_i to root r or pay a penalty π_i for not connecting it (for time step i and each future time step); in the latter case, we mark the terminal. Our goal is to find a set of edges F that minimizes the sum of edge costs in F plus sum of penalties for all marked terminals.

The integer programming formulation of (IP_i) is

$$(IP_i) \qquad \begin{array}{lll} \text{Min} & \sum_{e\in E} c_e x_e + \sum_{s_l \in R_i} \pi_l z_l & \\ & \sum_{e\in\delta(S)} x_e + z_l \geq 1, & \forall S \subseteq V - \{r\}, s_l \in S \cap R_i, \\ & x_e \in \{0,1\}, & \forall e \in E, \\ & z_l \in \{0,1\}, & \forall s_l \in R_i. \end{array}$$

Let (LP_i) denote the corresponding linear programming relaxation in which the constraints $x_e \in \{0,1\}$ and $z_l \in \{0,1\}$ are replaced with $x_e \geq 0$ and $z_l \geq 0$. The dual of this linear program, (D_i), is

$$(D_i) \qquad \begin{array}{lll} \text{Max} & \sum_{S\subseteq V-\{r\}} y_S & \\ & \sum_{S: e\in\delta(S)} y_S \leq c_e, & \forall e \in E, \\ & \sum_{S\subseteq U} y_S \leq \sum_{s_l\in U\cap R_i} \pi_l, & \forall U \subset V, r \notin U, \\ & y_S \geq 0, & \forall S \subseteq V - \{r\}. \end{array}$$

For the each dual problem (D_i), call the constraints of type $\sum_{S:e\in\delta(S)} y_S \leq c_e$ the edge cost constraints and call the constraints of type $\sum_{S\subseteq U} y_S \leq \sum_{s_l\in U\cap R_i} \pi_l$ the penalty constraints. A penalty constraint corresponding to a set U^j is *tight* in level j if the left-hand side of the inequality is equal to the right-hand side.

A terminal is *active* during time step i if it is *unmarked* and it is not yet connected to root r in current solution F. A terminal is *marked* by our algorithm if we decide to pay its penalty. A moat S^j is *active* during time step i, if it is on the lowest level j that satisfies the following three conditions: (i) S^j contains some active terminal $s \in A$; (ii) y_S^j has not yet reached its limit in level j; (iii) S^j does not contain root r. We denote the current set of active moats by $\mathcal{M}$.

The rest of the definitions are the same as in the main algorithm.

We extend our main algorithm to give an $O(\log |R_i|)$-competitive algorithm for the prize-collecting Steiner tree problem. For each time step i, a new terminal s_i with penalty π_i arrives. With the modified definitions of active terminals and actives moats, we follow along the same lines of the main algorithm to grow dual variables, with the same conditions (1)-(3) in that algorithm, but additionally: (4) When a path p connecting a terminal s to root r becomes tight in level j, we buy the path if s and r are not yet connected in F and update active terminal set A; (5) when a penalty constraint corresponding to a set U^j becomes tight in level j, we mark all terminals in U^j to pay their penalties and update active terminal set A. We let a set Q be all the vertices marked by our algorithm in current time step and in previous time steps. At the end of time step i, our algorithm outputs F and the set of terminals Q marked to pay penalties.

Theorem 3.4.1 *Our extended algorithm gives an $O(\log |R_i|)$-competitive algorithm for the online prize-collecting Steiner tree problem (IP_i).*

Proof: To bound total edge costs and penalties, we need to bound the cost of

```
Extended Algorithm

$F = \emptyset$, $\bar{F}^j = \emptyset$ for all $j$ and $y_S^j = 0$ for all $j$ and $S \subseteq V$
For each terminal $s_i$ that arrives
    Update active terminals $A$, and active moats $\mathcal{M}$
    While $|A| > 0$
        Grow uniformly all active dual variables $y_S^j$ until
        (1) An active $y_S^j$ with reaches limit in level $j$
        (2) An edge $e \in E$ becomes tight in level $j$, then
            $\bar{F}^j = \bar{F}^j \cup \{e\}$
        (3) Two disjoint moats $S_1^j$ and $S_2^j$ collide in level $j$, then
            Let $p \subseteq E$ be the corresponding path that becomes tight in level $j$
            Let $s_1$ and $s_2$ be the two terminals connected by $p$ in $\bar{F}^j$
            If $s_1$ and $s_2$ are not yet connected in $F$
                $F = F \cup \{p\}$, i.e. build edges $p - F$
            Update $A$
        (4) A path $p$ connecting a terminal $s$ to root $r$ becomes tight in level $j$
            If $s$ and $r$ are not yet connected in $F$
                $F = F \cup \{p\}$, i.e. build edges $p - F$
            Update $A$
        (5) A penalty constraint corresponding to set $U^j$ becomes tight in level $j$
            Mark all terminals in $U^j$ to pay penalties
            Update $A$
        Update $\mathcal{M}$
    EndWhile
EndFor
Let $Q$ be the set of terminals marked to pay penalties
Output $F$ and $Q$
```

Figure 3.2: Primal-dual algorithm for the online prize-collecting Steiner tree problem

edges built by conditions (3) and (4), and penalties paid by condition (5).

Consider edges in F. Let P be the set of paths we built in F by condition (4) of the extended algorithm, i.e. paths that connect a component in $F - P$ to root r. Then $F - P$ is the set of edges built by condition (3) of the extended algorithm. By invariant 2 of Lemma 3.3.4, we have $\sum_{e \in F-P} c_e = \sum_j \sum_S y_S^j - \sum_{X \in F-P} \text{Account}(X)$. By condition (4) of the algorithm, for each path p that connects a terminal s to root r, there must be a moat S^j such that $\sum_{e \in p} c_e \leq \sum_{S' \subseteq S^j : s \in S'} y_{S'}^j$. Let X_p be the component in $F - P$ that is connected to the root by p. All $S' \subseteq S^j$ with $s \in S'$ must credit Account(X_p). Also, j must be equal to Class(X_p) since every terminal in X_p is connected to root r after building p in F; after X_p connects to r, our algorithm does not grow any dual that contains a terminal in X_p. By definition of TopGrowth, we know $\sum_{e \in p} c_e \leq \text{TopGrowth}(X_p)$. Therefore, we have $\sum_{e \in P} c_e \leq \sum_{X_p : p \in P} \text{TopGrowth}(X_p)$ so that

$$\begin{aligned}
\sum_{e \in F} c_e &= \sum_{e \in F-P} c_e + \sum_{e \in P} c_e \\
&\leq \sum_j \sum_S y_S^j - \sum_{X \in F-P} \text{Account}(X) + \sum_{X_p : p \in P} \text{TopGrowth}(X_p) \\
&\leq \sum_j \sum_S y_S^j - \sum_{X \in F-P : X = X_p, p \in P} (\text{Account}(X_p) - \text{TopGrowth}(X_p)) \\
&\qquad - \sum_{X \in F-P : X \neq X_p \forall p \in P} \text{Account}(X) \\
&\leq (\log |R_i| + 2) \sum_S y_S^{max}.
\end{aligned}$$

The last inequality follows by same argument as Lemma 3.3.5 since for the component X^* in $F - P$ with highest class, whether it has a path that connects it to root or not, $\text{Account}(X^*) - \text{TopGrowth}(X^*) \geq 2^{\text{Class}(X^*)}$ by invariant 1 of Lemma 3.3.4.

Next, we use a new copy of dual variables to bound penalties of terminals in Q, i.e. $\sum_{s_l \in Q} \pi_l$. For each dual solution y^j, let $\mathcal{S}^j$ be the set of moats in $\bar{F}^j$ that correspond to a tight penalty constraint. It must the case that $\sum_{s_l \in S^j \cap Q} \pi_l = \sum_{S' \subseteq S^j} y_{S'}^j$ for any $S^j \in \mathcal{S}^j$ by condition (5) of the algorithm. By construction, each dual variable is charged to pay a penalty at most once. Also, since we keep growing same set of dual variables in all time steps, our algorithm continues to pay penalties corresponding to tight penalty constraints over all time steps. To bound the total penalty, we know that a terminal can be marked to pay a penalty only by condition (5), so that

$$\sum_{s_l \in Q} \pi_l \leq \sum_j \sum_{S^j \in \mathcal{S}^j} \sum_{S' \subseteq S^j} y_{S'}^j \leq \sum_j \sum_S y_S^j.$$

Let X^* be a component in F of highest class and let $m = \text{Class}(X^*)$. Consider $\sum_j \sum_S y_S^j \leq 2\sum_j \sum_S y_S^j - \sum_j \sum_S y_S^j$, by invariant 1 of Lemma 3.3.4, we know $\sum_j \sum_S y_S^j \geq \text{Account}(X^*) \geq 2^m$. By similar technique in Lemma 3.3.5, let $\lambda = \lceil \log_2 |R_i| \rceil$, we know $\sum_S y_S^{m-\lambda-k-2} \leq 2^{m-k-2}$. Consider all dual vectors of the form $y^{m-\lambda-k-2}$ with $k \geq 0$, we have $2\sum_{k\geq 0} \sum_S y_S^{m-\lambda-k-2} \leq 2^m$. Then, for dual solutions $y^{m-\lambda-1}, \ldots, y^m$, we have

$$2\sum_{k=0}^{k=\lambda+1} \sum_S y_S^{m-\lambda-1+k} \leq 2(\log |R_i| + 3) \sum_S y_S^{max}.$$

Therefore,

$$\begin{aligned} \textstyle\sum_{e\in F} c_e + \sum_{s_l \in Q} \pi_l &\leq [(\log |R_i| + 2) + 2(\log |R_i| + 3)] \textstyle\sum_S y_S^{max} \\ &\leq O(\log |R_i|) OPT_i. \end{aligned}$$

□

3.5 The Online Prize-Collecting Steiner Forest Problem

We now show that our algorithm can be extended to solve the prize-collecting Steiner forest problem. Define the online prize-collecting Steiner forest problem as follows: given an undirected graph G, and zero penalty for each pair of nodes. In each time step i, a terminal pair (k, l) arrives with a new penalty $\pi_{kl} > 0$. We have a choice to either connect k to l or pay a penalty π_{kl} for not connecting them (for time step i and all future time steps). Our goal is to find a set of edges F that minimizes the sum of edge costs in F plus sum of penalties for terminals pairs not connected by F.

For a set $S \subseteq V$, we denote $|\{k, l\} \cap S| = 1$ by $S \odot (k, l)$. Define the proper function $f_i(S) = 1$ if $S \odot (k, l)$ for any arrived terminal pair (k, l) and $f_i(S) = 0$ otherwise. Let $g_i(S) = \max\{f_1(S), ..., f_i(S)\}$ for all $S \subseteq V$ and $i \geq 1$.

The integer programming formulation of (IP_i) is

$$
(IP_i) \qquad
\begin{array}{lll}
\text{Min} & \sum_{e \in E} c_e x_e + \sum_{(k,l) \in (V \times V)} \pi_{kl} z_{kl} & \\
& \sum_{e \in \delta(S)} x_e + z_{kl} \geq g_i(S), & \forall S \subseteq V, (k,l) \in (V \times V), S \odot (k,l) \\
& x_e \in \{0,1\}, & \forall e \in E, \\
& z_{kl} \in \{0,1\}, & (k,l) \in (V \times V).
\end{array}
$$

Let (LP_i) denote the corresponding linear programming relaxation in which the constraints $x_e \in \{0,1\}$ and $z_{kl} \in \{0,1\}$ are replaced with $x_e \geq 0$ and $z_{kl} \geq 0$. By applying Farkas Lemma, Hajiaghayi and Jain [21] introduced a dual of this linear program, (D_i),

$$
(D_i) \qquad
\begin{array}{lll}
\text{Max} & \sum_{S \subseteq V} g_i(S) y_S & \\
& \sum_{S : e \in \delta(S)} y_S \leq c_e, & \forall e \in E, \\
& \sum_{S \subseteq \mathcal{S}} y_S \leq \sum_{(k,l) \in (V \times V), S \in \mathcal{S} : S \odot (k,l)} \pi_{kl}, & \forall \mathcal{S} \subseteq 2^{2^V} \\
& y_S \geq 0, & \forall S \subseteq V.
\end{array}
$$

For dual problem (D_i), call the constraints of type $\sum_{S : e \in \delta(S)} y_S \leq c_e$ the edge cost constraints and the constraints of type $\sum_{S \subseteq \mathcal{S}} y_S \leq \sum_{(k,l) \in (V \times V), S \in \mathcal{S} : S \odot (k,l)} \pi_{kl}$ the penalty constraints. A penalty constraint corresponding to a family $\mathcal{S}^j$ is *tight* in level j if the left-hand side of the inequality is equal to the right-hand side.

We extend our main algorithm to give an $O(\log |R_i|)$-competitive algorithm for the prize-collecting Steiner forest problem. For this problem, a terminal is *active* during time step i if it is *unmarked* and every set S containing it is not a violated set for function g_i by current solution F. A terminal is *marked* by our algorithm if we decide to pay the penalty associated with its pair. The rest of the definitions follow as in the main algorithm.

We will follow the same lines of the main algorithm to grow dual variables, with the same conditions (1)-(3) in that algorithm, but additionally: (4) when a penalty constraint corresponding to a family $\mathcal{S}^j$ becomes tight in level j, we mark all terminals pairs (k, l) that have $S \odot (k, l)$ for $S \in \mathcal{S}^j$ to pay their penalties and update active terminal set A. We let Q be all terminal pairs marked by our algorithm in current time step and in previous time steps. At the end of time step i, our algorithm outputs F and the set of marked terminal pairs Q. Note that to find the next tight constraint in level j for the dual problem efficiently, we need to apply the algorithm in Section 4 of [21].

Theorem 3.5.1 *Our extended algorithm gives an $O(\log |R_i|)$-competitive algorithm for the online prize-collecting Steiner Forest problem (IP_i).*

Proof: To bound total edge costs and penalties, we need to bound the cost of edges built by conditions (3) and penalties paid by condition (4).

By Lemma 3.3.5, we have $\sum_{e \in F} c_e \leq (\log |R_i| + 2) \sum_S y_S^{max}$. We need to use a new copy of dual variables to bound penalties of terminal pairs in Q, i.e. $\sum_{(k,l) \in Q} \pi_{kl}$. For each dual solution y^j, let $\mathcal{X}^j$ be the collection of families in level j that correspond to a tight penalty constraint. It must the case that $\sum_{(k,l) \in Q, S^j \odot (k,l), S^j \in \mathcal{S}^j} \pi_{kl} = \sum_{S^j \subseteq \mathcal{S}^j} y_S^j$ for any $\mathcal{S}^j \in \mathcal{X}^j$ by condition (4) of the algorithm. By construction, each dual variable is charged to pay a penalty at most once. Also, since we keep growing the same set of dual variables in all time steps, our algorithm continues to pay the penalties corresponding to tight penalty constraints over all time steps. To bound the total penalty, we know that a terminal

```
Algorithm

$F = \emptyset$, $\bar{F}^j = \emptyset$ for all $j$, and $y_S^j = 0$ for all $j$ and $S \subseteq V$
For each terminal pair that arrives
    Update active terminals $A$, and active moats $\mathcal{M}$
    While $|A| > 0$
        Grow uniformly all active dual variables $y_S^j$ until
        1) An active $y_S^j$ with reaches limit in level $j$
        2) An edge $e \in E$ becomes tight in level $j$, then
            $\bar{F}^j = \bar{F}^j \cup \{e\}$
        3) Two disjoint moats $S_1^j$ and $S_2^j$ collide in level $j$, then
            Let $p \subseteq E$ be the corresponding path that becomes tight in level $j$
            Let $s_1$ and $s_2$ be the two terminals connected by $p$ in $\bar{F}^j$
            If $s_1$ and $s_2$ are not yet connected in $F$
                $F = F \cup \{p\}$, i.e. build edges $p - F$
                Update $A$
        (4) A penalty constraint w.r.t. family $\mathcal{S}^j$ becomes tight in level $j$
            Mark all terminals pairs $(k, l)$ that have $S \odot (k, l)$ and $S \in \mathcal{S}^j$
                Update $A$
        Update $\mathcal{M}$
Let $Q$ be the set of terminal pairs marked to pay penalties
Output $F$ and $Q$
```

Figure 3.3: Primal-dual algorithm for the online prize-collecting Steiner forest problem

can be marked to pay a penalty only by condition (4), so that

$$\sum_{(k,l)\in Q} \pi_{kl} \leq \sum_j \sum_{\mathcal{S}^j \in \mathcal{X}^j} \sum_{S^j \subseteq \mathcal{S}^j} y_S^j \leq \sum_j \sum_S y_S^j.$$

Let X^* be a component in F of highest class and let $m = Class(X^*)$. Consider $\sum_j \sum_S y_S^j \leq 2\sum_j \sum_S y_S^j - \sum_j \sum_S y_S^j$, by invariant 1 of Lemma 3.3.4, we know $\sum_j \sum_S y_S^j \geq \text{Account}(X^*) \geq 2^m$. By a similar technique as in Lemma 3.3.5, let $\lambda = \lceil \log_2 |R_i| \rceil$, we know $\sum_S y_S^{m-\lambda-k-2} \leq 2^{m-k-2}$. Consider all dual vectors of the form $y^{m-\lambda-k-2}$ with $k \geq 0$, we have $2\sum_{k\geq 0} \sum_S y_S^{m-\lambda-k-2} \leq 2^m$. Then, for dual solutions $y^{m-\lambda-1}, \ldots, y^m$, we have

$$2 \sum_{k=0}^{k=\lambda+1} \sum_S y_S^{m-\lambda-1+k} \leq 2(\log |R_i| + 3) \sum_S y_S^{max}.$$

Therefore,

$$\begin{aligned}\textstyle\sum_{e\in F} c_e + \sum_{s_l\in Q}\pi_l &\le [(\log|R_i|+2)+2(\log|R_i|+3)]\textstyle\sum_S y_S^{max}\\ &\le O(\log|R_i|)OPT_i.\end{aligned}$$

□

3.6 The Online Prize-Collecting Constrained Forest Problem with Submodular Penalty Functions

Sharma, Swamy and Williamson [37] introduced a prize-collecting version of constrained forest problems with an arbitrary 0-1 connectivity requirement function $g : 2^V \to \{0,1\}$, and a submodular and monotone penalty function $\pi : 2^{2^V} \to \mathbb{Z}_{\geq 0}$. We will further show that our algorithm can be extended to solve this problem as well.

This framework generalized the prize-collecing Steiner forest framework of Hajiaghayi and Jain [21] to incorporate more general connectivity requirements and penalty functions. They show that if the return of a feasible solution for the connectivity function f is enforced, then f must be proper.

We can define an online version of the prize-collecting constrained forest problems with proper connectivity requirement function and submodular penalty function in the following way. We start with connectivity requirement function g_0 and penalty function π_0, where $g_0(S) = 0$ for all $S \subseteq V$ and $\pi_0(\mathcal{S}) = 0$ for all $\mathcal{S} \subseteq 2^V$. In each time step i, a connectivity requirement function g_i and a penalty function π_i arrive with the following properties:

1. $g_i : 2^V \to \{0,1\}$ is a proper function,

2. $g_i(S) \geq g_{i-1}(S)$ for all $S \subseteq V$,

3. $\pi_i : 2^{2^V} \to Z_{\geq 0}$ is submodular, monotone and satisfies all other properties described in [37],

4. $\pi_i(\mathcal{S}) \geq \pi_{i-1}(\mathcal{S})$ for all $\mathcal{S} \subseteq 2^{2^V}$.

The integer programming formulation of (IP_i) is

$$
(IP_i) \qquad
\begin{array}{lll}
\text{Min} & \sum_{e \in E} c_e x_e + \sum_{\mathcal{S}} \pi_i(\mathcal{S}) z_{\mathcal{S}} & \\
& \sum_{e \in \delta(S)} x_e + \sum_{S : S \in \mathcal{S}} z_{\mathcal{S}} \geq g_i(S), & \forall S \subseteq V, \\
& x_e \in \{0,1\}, & \forall e \in E, \\
& z_{\mathcal{S}} \in \{0,1\}, & S \subseteq 2^{2^V}.
\end{array}
$$

Let (LP_i) denote the corresponding linear programming relaxation in which the constraints $x_e \in \{0,1\}$ and $z_{\mathcal{S}} \in \{0,1\}$ are replaced with $x_e \geq 0$ and $z_{\mathcal{S}} \geq 0$. The dual of this linear program, (D_i), is

$$
(D_i) \qquad
\begin{array}{lll}
\text{Max} & \sum_{S \subseteq V} g_i(S) y_S & \\
& \sum_{S : e \in \delta(S)} y_S \leq c_e, & \forall e \in E, \\
& \sum_{S : S \in \mathcal{S}} y_S \leq \pi_i(\mathcal{S}), & \forall \mathcal{S} \subseteq 2^{2^V} \\
& y_S \geq 0, & \forall S \subseteq V.
\end{array}
$$

For dual problem (D_i), call the constraints of type $\sum_{S : e \in \delta(S)} y_S \leq c_e$ the edge cost constraints and the constraints of type $\sum_{S : S \in \mathcal{S}} y_S \leq \pi(\mathcal{S})$ the penalty constraints. A penalty constraint corresponding to a family $\mathcal{S}^j$ is *tight* in level j if the left-hand side of the inequality is equal to the right-hand side.

We extend our main algorithm to give an $O(\log |R_i|)$-competitive algorithm for the online prize-collecting constrained forest problem with submodular penalty functions. For this problem, a terminal is *active* during time step i if it is *unmarked* and every set S containing it is not a violated set for function g_i by current solution F. Unlike all the previous prize-collecting results, our algorithm may choose not to pay the penalty for a collection in the future iterations, just the current iteration. The rest of the definitions follow as in the main algorithm.

We will follow the same lines of main algorithm to grow dual variables, with the same conditions (1)-(3) in that algorithm, but additionally: (4) when a penalty constraint corresponding to a family $\mathcal{S}^j$ becomes tight in level j, in which case we mark all terminals s with $s \in S^j \in \mathcal{S}^j$, mark family $\mathcal{S}^j$ to pay its penalty and update active terminal set A. Let Q be the collection of all families marked by our algorithm currently. At the beginning of time step i, we unmark family $\mathcal{S}$ in Q if its corresponding penalty constraints are no longer tight with the new penalty function π_i, and unmark all terminals contained in a set S in $\mathcal{S}$ if S is a violated set for function g_i. (Notice that this allows us to change our decisions to pay penalty for some families in earlier time steps to not pay penalty. This is needed because $\pi_i(\mathcal{S}^j)$ may be strictly greater than $\pi_{i-1}(\mathcal{S}^j)$ for some family $\mathcal{S}^j$.) At the end of time step i, our algorithm outputs F and the collection of marked families Q. Note that to find the next dual constraint to go tight in level j efficiently, we need to apply algorithm described in [37], which uses submodular function minimization.

Theorem 3.6.1 *Our extended algorithm gives an $O(\log |R_i|)$-competitive algorithm for the online prize-collecting Steiner Forest problem (IP_i).*

```
Algorithm
$F = \emptyset$, $\bar{F}^j = \emptyset$ for all $j$, and $y_S^j = 0$ for all $j$ and $S \subseteq V$
For each connectivity requirement function $g_i$ and penalty function $\pi_i$ that arrive
    Update active terminals $A$, active moats $\mathcal{M}$, and $Q$
    While $|A| > 0$
        Grow uniformly all active dual variables $y_S^j$ until
        1) An active $y_S^j$ with reaches limit in level $j$
        2) An edge $e \in E$ becomes tight in level $j$, then
            $\bar{F}^j = \bar{F}^j \cup \{e\}$
        3) Two disjoint moats $S_1^j$ and $S_2^j$ collide in level $j$, then
            Let $p \subseteq E$ be the corresponding path that becomes tight in level $j$
            Let $s_1$ and $s_2$ be the two terminals connected by $p$ in $\bar{F}^j$
            If $s_1$ and $s_2$ are not yet connected in $F$
                $F = F \cup \{p\}$, i.e. build edges $p - F$
                Update $A$
        (4) A penalty constraint w.r.t. family $\mathcal{S}^j$ becomes tight in level $j$
            Mark all terminals $s$ with $s \in S^j \in \mathcal{S}^j$
            Mark family $\mathcal{S}^j$ to pay its penalties
            Update $A$
        Update $\mathcal{M}$
Let $Q$ be the collection of families marked to pay penalties
Output $F$ and $Q$
```

Figure 3.4: Primal-dual algorithm for the online prize-collecting constrained forest problem with submodular penalty functions

Proof: Since $\pi_i(\mathcal{S}) \geq \pi_{i-1}(\mathcal{S})$ for all $\mathcal{S} \subseteq 2^{2^V}$ and $i \geq 1$, each dual solution y^j that is feasible at the end of time step i will remain feasible at the beginning of time step $i+1$. Together with Lemma 3.3.2, each dual solution y^j is feasible for (D_i) and F is feasible for (IP_i) at the end of phase i.

To bound total edge costs and penalties, we need to bound the cost of edges built by conditions (3) and penalties paid by condition (4). By Lemma 3.3.5, we have $\sum_{e \in F} c_e \leq (\log |R_i| + 2) \sum_S y_S^{max}$. We need to use a new copy of dual variables to bound the penalties of families in Q, i.e. $\sum_{\mathcal{S} \in Q} \pi_i(\mathcal{S})$. For each dual solution y^j, let $\mathcal{X}^j$ be the collection of families in level j that correspond to a tight penalty constraint. It must the case that $\sum_{\mathcal{S}^j \in Q} \pi_i(\mathcal{S}^j) = \sum_{S^j \subseteq \mathcal{S}^j} y_S^j$ for any $\mathcal{S}^j \in \mathcal{X}^j$ by condition (4) of the algorithm. By construction, each dual variable is charged to pay a penalty at most once. Also, since we keep growing the same set of dual variables in all time steps, our algorithm continues to pay penalties corresponding to tight penalty constraints over all time steps. To bound the total penalty, we know that a terminal can be marked to pay a penalty only by condition (4), so that

$$\sum_{\mathcal{S}\in Q} \pi_i(\mathcal{S}) \leq \sum_j \sum_{\mathcal{S}^j \in \mathcal{X}^j} \sum_{S^j \subseteq \mathcal{S}^j} y_S^j \leq \sum_j \sum_S y_S^j.$$

Let X^* be a component in F of highest class and let $m = Class(X^*)$. Consider $\sum_j \sum_S y_S^j \leq 2\sum_j \sum_S y_S^j - \sum_j \sum_S y_S^j$, by invariant 1 of Lemma 3.3.4, we know $\sum_j \sum_S y_S^j \geq \text{Account}(X^*) \geq 2^m$. By similar technique to that in Lemma 3.3.5, let $\lambda = \lceil \log_2 |R_i| \rceil$, we know $\sum_S y_S^{m-\lambda-k-2} \leq 2^{m-k-2}$. Consider all dual vectors of the form $y^{m-\lambda-k-2}$ with $k \geq 0$, we have $2\sum_{k\geq 0}\sum_S y_S^{m-\lambda-k-2} \leq 2^m$. Then, for dual solutions $y^{m-\lambda-1}, \dots, y^m$, we have

$$2\sum_{k=0}^{k=\lambda+1} \sum_S y_S^{m-\lambda-1+k} \leq 2(\log|R_i| + 3)\sum_S y_S^{max}.$$

Therefore,

$$\begin{aligned}\textstyle\sum_{e\in E} c_e x_e + \sum_{\mathcal{S}} \pi_i(\mathcal{S}) z_{\mathcal{S}} &\leq [(\log|R_i| + 2) + 2(\log|R_i| + 3)]\textstyle\sum_S y_S^{max} \\ &\leq O(\log|R_i|)OPT_i.\end{aligned}$$

□

3.7 Computational Results

In this section, we give computational results of our algorithm in the special case of the online prize-collecting Steiner tree problem.

We implemented our online algorithm using MATLAB R2010b [29] and a Macintosh desktop computer with a 2.5GHz Intel Quad-Core i5 processor and 4 GB of memory. We found solutions on two sets of randomly generated instances from Jonhson et al. [25]. In the so-called $\mathcal{P}$ group, instances are unstructured and designed to have constant expected degree and penalty to weight ratio. The $\mathcal{K}$ group consists of random geometric instances designed to have a structure similar to street maps. We considered a total of 46 instances with up to 400 vertices, 1500 edges, and 100 terminals. See [25] for a detailed description of the generators for these instances. For six instances, we tested on two different random permutations. They are suffixed with .R1 and .R2 after their original labels.

For each instance $G = (V, E)$, there is a file containing information on nodes in V with corresponding nonnegative penalties and edges in E with corresponding nonnegative edge costs. A terminal is a node with a positive penalty. We used the terminal with maximum penalty as the root. We generated a random permutation of the remaining terminals and used it as the sequence of arrival terminals. It was shown by Garg et al. [14] that for the special case of the Steiner tree problem, using a random permutation of the terminals still has an $\Omega(\log n)$ lower bound on the competitive ratio. For non-root terminals, we used our algorithm in Figure 3.2 to decide which edges to buy or which penalties to pay. To find the competitive ratio, i.e. the ratio between cost of online algorithm solution and the cost of the optimal offline solution, we used a package by Ljubic et al. [28] to find the optimal solutions of the corresponding offline problems. Tables 3.1 and 3.2 show the results of 46 instances we tested. The range of the competitive ratios after all terminals arrived is from 1.194 to 2.584 for the instances we tested, with an average ratio of 1.62.

Instance	Online Objective	Offline Optimal	Competitive Ratio
K100	208416	135511	1.538
K100.1	173007	124108	1.394
K100.2.R1	359671	200262	1.796
K100.2.R2	267150	200262	1.334
K100.3	212194	115953	1.830
K100.4.R1	134659	87498	1.539
K100.4.R2	118385	87498	1. 353
K100.5	298171	119078	2.584
K100.6.R1	173283	132886	1.304
K100.6.R2	197439	132886	1.486
K100.7	334222	172457	1.938
K100.8.R1	382221	210869	1.813
K100.8.R2	365640	210869	1.734
K100.9	154507	122917	1.257
K100.10.R1	187395	133567	1.403
K100.10.R2	252709	133567	1.892
K200.R1	816049	329211	1.879
K200.R2	769520	329211	1.938
K400	552797	350093	1.579
K400.1	686031	490771	1.438
K400.2	617332	477073	1.294
K400.3	896693	415328	2.159
K400.4	588850	389451	1.512
K400.5	953330	519526	1.835
K400.6	469686	374849	1.253
K400.7	694144	474466	1.463
K400.8	690294	418614	1.649
K400.9	545542	383105	1.424
K400.10	600353	394191	1.523
K100.red	174846	113132	1.546
K200.red	723228	296935	2.436
K400.red	733838	322470	2.276

Table 3.1: Competitive ratios for online prize-collecting Steiner tree algorithm on 32 instances from group $\mathcal{K}$.

For several instances, we also computed the competitive ratios after the arrival of each terminal in the random permutation (see Figure 3.6 for example). We found that our algorithm performs optimally for the first few steps in which

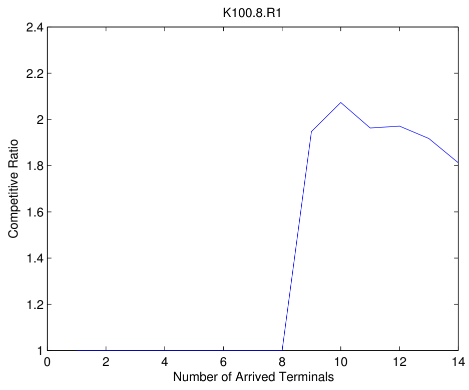
Instance	Online Objective	Offline Optimal	Competitive Ratio
P100	1086865	803300	1.353
P100.1	1431038	926238	1.545
P100.2	563904	401641	1.404
P100.3	1281688	659644	1.943
P100.4	1012761	827419	1.224
P200	1892467	1317874	1.436
P400	2981404	2459904	1.212
P400.1	6018487	2808440	2.143
P400.2	3407635	2518577	1.353
P400.3	5487257	2951725	1.859
P400.4	4407817	2852956	1.545
P100.red	674380	564753	1.194
P200.red	1760141	1257107	1.400
P400.red	2918481	2255191	1.294

Table 3.2: Competitive ratios for online prize-collecting Steiner tree algorithm on 14 instances from group $\mathcal{P}$.

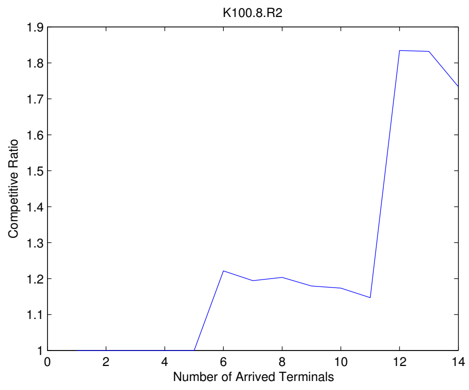
the online algorithm may choose to buy the same set of edge-distinct shortest paths as the optimal solution from terminals to the root or other terminals. Later on, our online algorithm may make a "mistake" compared to the optimal offline solution and pay a lot more on edge costs or penalties. So the competitive ratio may increase a lot in one step. As an example, in Figure 3.6(c), after the 30th terminal arrived, the optimal value of the offline algorithm does not change from the previous phase with 29 terminals. But the online algorithm pays a large cost to buy a path from the 30th terminal to the root, increasing the competitive ratio from 1.66 to 2.81 in one step. In other cases, the amount of phase-by-phase costs increase may be about the same for the online algorithm and the offline algorithm. This usually reduces the competitive ratio. Overall, the competitive ratio will spike up whenever our online algorithm makes a "big mistake" while its impact on the competitive ratio is decreasing over the sequence of the terminals since the cost of the online and offline solutions are increasing. As an illustration, in Figure 3.6(e) and 3.6(c), the competitive ratio is mostly trending down eventually.

As an illustration, Figure 3.7 shows snapshots of online algorithm and optimal solutions on instance K100.8.R1. Circles are the nodes in V, arrived terminals are marked with asterisks in addition, and built edges are drawn between their two endpoints. After 9 terminals have arrived, the online algorithm already built edges to connect 5 terminals to the root while the offline optimal solution only built one edge (a tiny edge on the graph due to scale). With 11 arrived terminals, the online algorithm built a lot more edges than the offline optimal solution. But for the last two terminals, the competitive ratio is decreasing as the online algorithm "paid off" from previous built edges, making a "smaller mistake" comparing the previous phases.

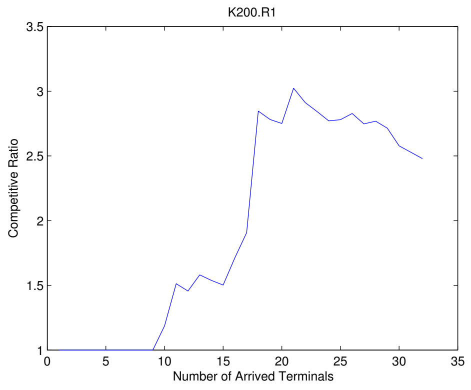
In addition, it appears that there is no pattern in the competitive ratios between instances in the $\mathcal{P}$ group and those in the $\mathcal{K}$ group, nor on the same instance but different permutations (See Figures 3.6 and 3.5, for example). The reason for this appears to be the fact that the competitive ratio depends on when and how large the online algorithm makes a "mistake", which does not solely depend on the type of an instance nor the permutation of the arriving terminals.



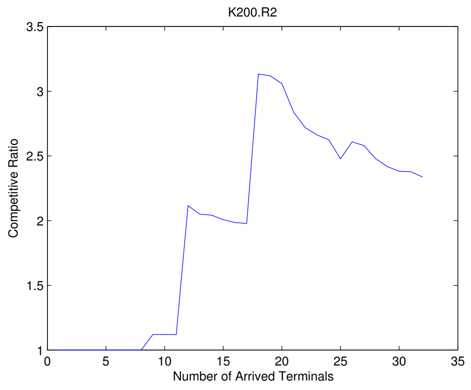
(a) K100.8.R1



(b) K100.8.R2

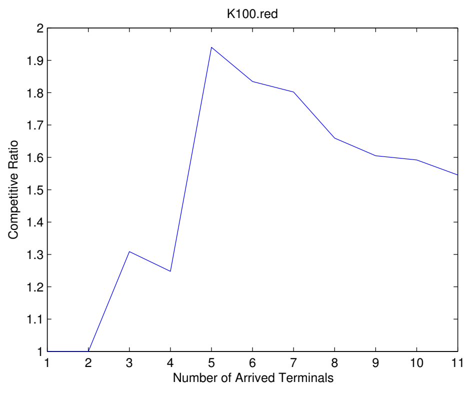


(c) K200.R1

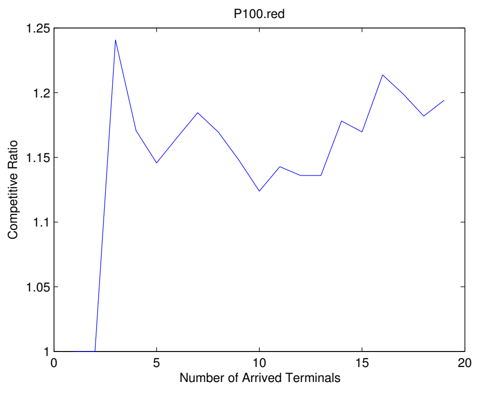


(d) K200.R2

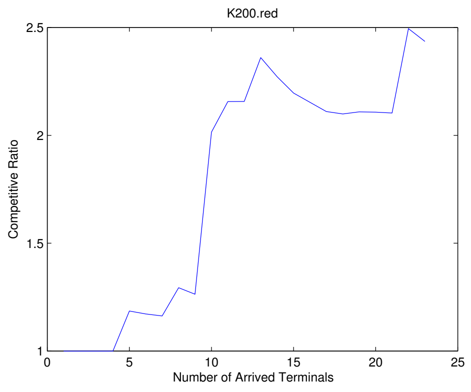
Figure 3.5: Illustration of step-by-step competitive ratios on two instances with two different permutations each. The horizontal axis represents the number of arrived terminals and the vertical axis represents the competitive ratio.



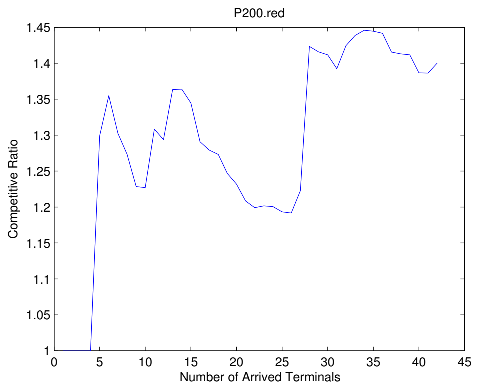
(a) K100red



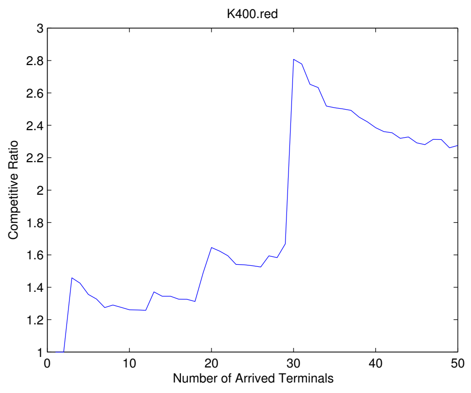
(b) P100red



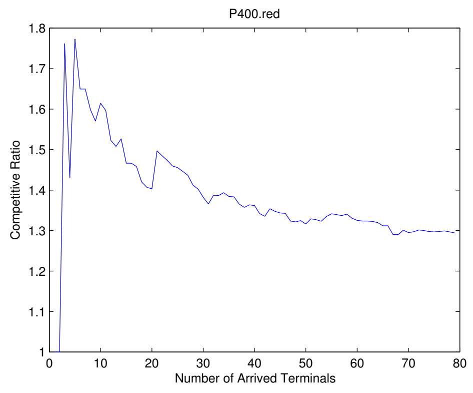
(c) K200red



(d) P200red

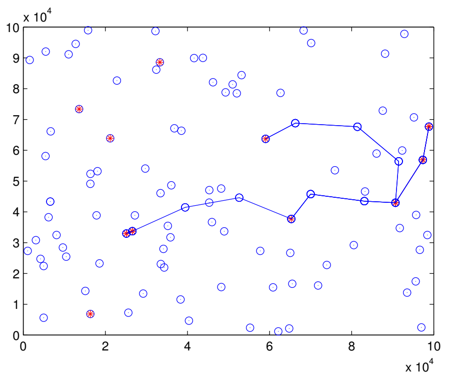


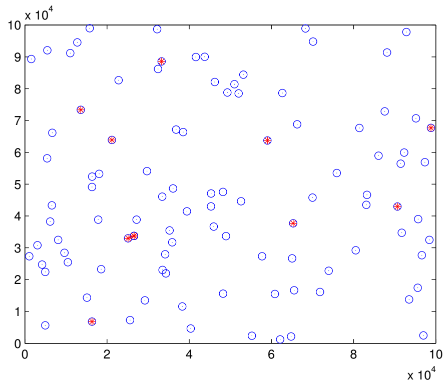
(e) K400red

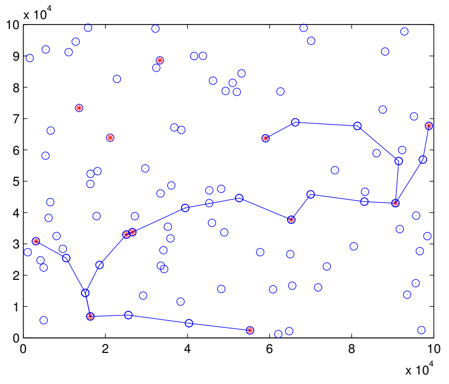


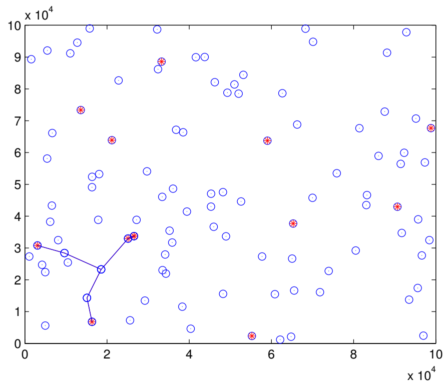
(f) P400red

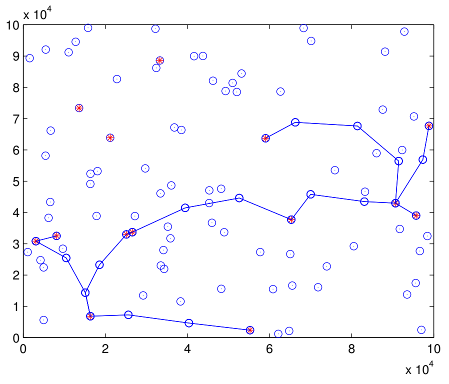
Figure 3.6: Illustration of step-by-step competitive ratios on six instances. The horizontal axis represents the number of arrived terminals and the vertical axis represents the competitive ratio.


(a) Online 9 terminals


(b) Offline 9 terminals


(c) Online 11 terminals


(d) Offline 11 terminals


(e) Online 13 terminals

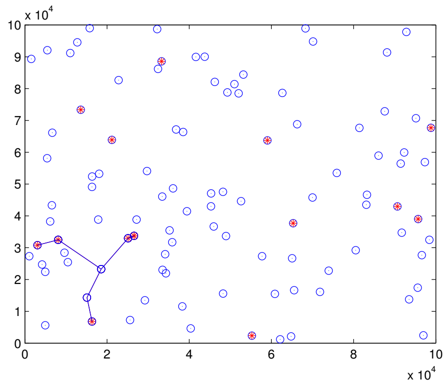

(f) Offline 13 terminals

Figure 3.7: Snapshots of online algorithm and optimal solutions on instance K100.8.R1. Arrived terminals are marked in red.

CHAPTER 4

ON THE INTEGRALITY GAP OF THE SUBTOUR LP FOR 1,2-TSP

4.1 Introduction

As we mentioned in Chapter 1, the Traveling Salesman Problem (TSP) is one of the most well studied problems in combinatorial optimization. Given a set of cities $\{1, 2, \ldots, n\}$, and distances $c(i, j)$ for traveling from city i to j, the goal is to find a tour of minimum length that visits each city exactly once. An important special case of the TSP is the case when the distance forms a metric, i.e., $c(i, j) \leq c(i, k) + c(k, j)$ for all i, j, k, and all distances are symmetric, i.e., $c(i, j) = c(j, i)$ for all i, j. The symmetric TSP is known to be APX-hard, even if $c(i, j) \in \{1, 2\}$ for all i, j [34]; note that such instances trivially obey the triangle inequality.

The symmetric TSP that is also metric can be approximated to within a factor of $\frac{3}{2}$ using an algorithm by Christofides [11] from 1976. The algorithm combines a minimum spanning tree with a matching on the odd-degree vertices to get an Eulerian graph that can be shortcut to a tour; the analysis shows that the minimum spanning tree and the matching cost no more than the optimal tour and half the optimal tour respectively. Better results are known for several special cases, but, surprisingly, no progress has been made on approximating the general symmetric TSP in more than thirty years. A natural direction for trying to obtain better approximation algorithms is to use linear programming. The following linear programming relaxation of the traveling salesman problem was used by Dantzig, Fulkerson, and Johnson [13] in 1954. For simplicity of notation, we let $G = (V, E)$ be a complete undirected graph on n vertices. In the LP relaxation, we have a variable $x(e)$ for all $e = (i, j)$ that denotes whether we travel directly between cities i and j on our tour. Let $c(e) = c(i, j)$, and let $\delta(S)$ denote the set of all edges with exactly one endpoint in $S \subseteq V$. Then the relaxation is

$$
\begin{array}{llll}
 & \text{Min} & \displaystyle\sum_{e \in E} c(e)x(e) & \\
(SUBT) \quad \text{subject to:} & & \displaystyle\sum_{e \in \delta(i)} x(e) = 2, & \forall i \in V, \qquad (4.1) \\
 & & \displaystyle\sum_{e \in \delta(S)} x(e) \geq 2, & \forall S \subset V,\ 3 \leq |S| \leq |V| - 3, \quad (4.2) \\
 & & 0 \leq x(e) \leq 1, & \forall e \in E. \qquad (4.3)
\end{array}
$$

The first set of constraints (4.1) are called the *degree constraints*. The second set of constraints (4.2) are sometimes called *subtour elimination constraints* or sometimes just *subtour constraints*, since they prevent solutions in which there is a subtour of just the vertices in S. As a result, the linear program is sometimes called the *subtour LP*. Wolsey [41] (and later Shmoys and Williamson [38]) showed that Christofides' algorithm finds a tour of length at most $\frac{3}{2}$ times the optimal value of the subtour LP; these proofs show that the minimum spanning tree and the matching on odd-degree nodes can be bounded above by the value of the subtour LP, and half the value of the subtour LP, respectively. This implies that the integrality gap, the worst case ratio of the length of an optimal tour divided by the optimal value of the LP, is at most $\frac{3}{2}$. However, no examples are known that show that the integrality gap can be as large as $\frac{3}{2}$; in fact, no examples are known for which the integrality gap is greater than $\frac{4}{3}$. A well known conjecture states that the integrality gap is indeed $\frac{4}{3}$; see Goemans [15].

Recently, progress has been made in several directions, both in improving the best approximation ratio and in determining the exact integrality gap of the subtour LP for certain special cases of the symmetric TSP. In the *graph*-TSP, the costs $c(i, j)$ are equal to the shortest path distance in an underlying unweighted graph. Oveis Gharan, Saberi, and Singh [33] showed that the graph-TSP can be approximated to within $\frac{3}{2} - \epsilon$ for a small constant $\epsilon > 0$. Boyd, Sitters, van der Ster and Stougie [8] gave a $\frac{4}{3}$-approximation algorithm if the underlying graph is cubic. Mömke and Svensson [31] gave a 1.461-approximation for the graph-TSP and an $\frac{4}{3}$-approximation algorithm if the underlying graph is subcubic. Their results also imply upper bounds on the integrality gap of 1.461 and $\frac{4}{3}$ in these cases. Mucha [32] improved the bound for graph-TSP to $\frac{13}{9}$.

Schalekamp, Williamson and van Zuylen [36] resolved a related conjecture. A 2-matching of a graph is a set of edges such that no edge appears twice and each node has degree two, i.e., it is an integer solution to the LP $(SUBT)$ with only constraints (4.1) and (4.3). Note that a minimum-cost 2-matching thus provides a lower bound on the length of the optimal TSP tour. A minimum-cost 2-matching can be found in polynomial time using a reduction to a certain minimum-cost matching problem. Boyd and Carr [7] conjectured that the worst case ratio of the cost of a minimum-cost 2-matching and the optimal value of the subtour LP is at most $\frac{10}{9}$. This conjecture was proved to be true by Schalekamp et al. and examples are known that show this result is tight.

Unlike the techniques used to obtain better results for the graph-TSP, the techniques of Schalekamp et al. work on general weighted instances that are symmetric and obey the triangle inequality. However, their results only apply to 2-matchings and it is not clear how to enforce global connectivity on the solution obtained by their method. A potential direction for progress on resolving the integrality gap for the subtour LP is a conjecture by Schalekamp et al. that the worst-case integrality gap is attained for instances for which the subtour elimination constraints are redundant.

In this chapter, we turn our attention to the 1,2-TSP, where $c(i, j) \in \{1, 2\}$ for

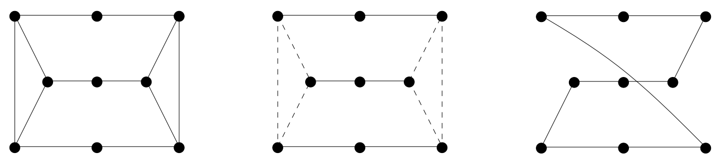


Figure 4.1: Illustration of the worst example known for the integrality gap for the 1,2-TSP. The figure on the left shows all edges of cost 1. The figure in the center gives the subtour LP solution, in which the dotted edges have value $\frac{1}{2}$, and the solid edges have value 1; this is also an optimal fractional 2-matching. The figure on the left gives the optimal tour and the optimal 2-matching.

all i, j. Papadimitriou and Yannakakis [34] showed how to approximate 1,2-TSP within a factor of $\frac{11}{9}$ starting with a minimum-cost 2-matching. In addition, they showed a ratio of $\frac{7}{6}$ with respect to the the minimum-cost 2-matching that has no cycles of length 3. Bläser and Ram [6] improved this ratio and the best known approximation of $\frac{8}{7}$ is given by Berman and Karpinski [5].

We do not know a tight bound on the integrality gap of the subtour LP even in the case of the 1,2-TSP. As an upper bound, we appear to know only that the gap is at most $\frac{3}{2}$ via Wolsey's result. There is an easy 9 city example showing that the gap must be at least $\frac{10}{9}$; see Figure 4.1. This example has been extended to a class of instances on $9k$ nodes for any positive integer k by Williamson [39]. The contribution of this work is to begin a study of the integrality gap of the 1,2-TSP, and to improve our state of knowledge for the subtour LP in this case. We are able to give the first bound that is strictly less than $\frac{4}{3}$ for these instances. This is the first bound on the integrality gap for the subtour LP with value less than $\frac{4}{3}$ for a natural class of TSP instances. Under an analog of a conjecture of Schalekamp et al. [36], we show that the integrality gap is at most $\frac{7}{6}$, and with an additional uniqueness assumption, we can improve this bound to $\frac{10}{9}$. We describe these results in more detail below.

We start by giving a bound on the subtour LP in the general case of 1,2-TSP. All the known approximation algorithms since the initial work of Papadimitriou and Yannakakis [34] on the problem start by computing a minimum-cost 2-matching. However, the example of Figure 4.1 shows that an optimal 2-matching can be as much as $\frac{10}{9}$ times the value of the subtour LP for the 1,2-TSP, so we cannot directly replace the bound on the optimal solution in these approximation algorithms with the subtour LP in the same way that Wolsey did with Christofides' algorithm in the general case. Using the result of Schalekamp, Williamson, and van Zuylen [36] and some additional work, we are able to show that an algorithm of Papadimitriou and Yannakakis [34] obtains a bound on the subtour LP for the 1,2-TSP of $\frac{7}{9} \cdot \frac{10}{9} + \frac{4}{9} = \frac{106}{81} \approx 1.3086$.

Next, we show stronger results in some cases. A fractional 2-matching is an optimal solution to the LP (*SUBT*) with only constraints (4.1) and (4.3). Schalekamp et al. [36] have conjectured that the worst-case integrality gap for the subtour LP is obtained when the subtour elimination constraints are redundant. We show that if this is the case for 1,2-TSP, i.e. in the worst case there is an optimal solution to the subtour LP that is an optimal solution to the fractional 2-matching problem, we can find a tour of cost at most $\frac{7}{6}$ the cost of the fractional 2-matching, implying that the integrality gap is at most $\frac{7}{6}$ in these cases. We then show that if this optimal solution to the fractional 2-matching problem is the *unique* optimal fractional 2-matching, then we can find a tour of cost at most $\frac{10}{9}$ times the cost of the fractional 2-matching, implying that the integrality gap is at most $\frac{10}{9}$ in these cases. Figure 4.1 shows that this result is tight.

The results above all lead to polynomial-time algorithms, though we do not state the exact running times.

We anticipate that substantially stronger bounds on the integrality gap can be shown. In particular, we conjecture that the integrality gap is in fact exactly $\frac{10}{9}$. We perform computational experiments that show that this conjecture is true for $n \leq 12$.

The remainder of this chapter is structured as follows. Section 4.2 contains preliminaries and a general bound on the integrality gap for the 1,2-TSP. We show how to obtain stronger bounds in some cases in Section 4.3. We describe our computational experiments in Section 4.4.

4.2 Preliminaries and A First Bound on The Integrality Gap

We will work extensively with 2-matchings and fractional 2-matchings; that is, extreme points x of the LP (*SUBT*) with only constraints (4.1) and (4.3), where in the first case the solutions are required to be integer. For convenience we will abbreviate "fractional 2-matching" by F2M and "2-matching" by 2M. F2Ms have the following well-known structure (attributed to Balinski [3]). Each connected component of the support graph (that is, the edges e for which $x(e) > 0$) is either a cycle on at least three vertices with $x(e) = 1$ for all edges e in the cycle, or consists of odd-sized cycles with $x(e) = \frac{1}{2}$ for all edges e in the cycle connected by paths of edges e with $x(e) = 1$ for each edge e in the path (the center figure in Figure 4.1 is an example). We call the former components *integer components* and the latter *fractional components*. In a fractional component, we call a path of edges e with $x(e) = 1$ a *1-path*. The edges e with $x(e) = \frac{1}{2}$ in cycles are called *cycle edges*. An F2M with a single component is called *connected*, and we call a component *2-connected* if the sum of the x-values on the edges crossing any cut is at least 2.

As mentioned in the introduction, Schalekamp, Williamson, and van Zuylen [36] have recently shown the following.

Theorem 4.2.1 (Schalekamp et al. [36]) *If edge costs obey the triangle inequality, then the cost of an optimal 2-matching is at most $\frac{10}{9}$ times the value of the subtour LP.*

We now show that applying an algorithm of Papadimitriou and Yannakakis [34] to this 2-matching will produce a tour of cost at most $\frac{106}{81}$ times the value of the subtour LP.

Theorem 4.2.2 *The integrality gap of the subtour LP is at most $\frac{106}{81}$ for 1,2-TSP.*

Proof: We show that there exists a tour of cost at most $\frac{7}{9}$ times the cost of the optimal 2M plus $\frac{4}{9}$ times the value of the subtour LP if all edge costs are either 1 or 2. Using Theorem 4.2.1, this implies that the tour has cost at most $\frac{7}{9} \times \frac{10}{9} + \frac{4}{9} = \frac{106}{81}$ times the value of the subtour LP.

Papadimitriou and Yannakakis [34] observe that we can assume without loss of generality that the optimal 2M solution consists of a number of cycles with only edges of cost 1 ("pure" cycles) and at most one cycle which has one or more edges of cost 2 (the "non-pure" cycle). This is because we can always patch two non-pure cycles into one cycle without increasing the cost of the solution. Moreover, if i is a node in the non-pure cycle which is incident on an edge of cost 2 in the cycle, then there can be no edge of cost 1 connecting i to a node in a pure cycle (since otherwise, we can merge the non-pure cycle with a pure cycle without increasing the cost).

The algorithm of Papadimitriou and Yannakakis solves the following bipartite matching problem: On one side we have a node for every pure cycle, and on the other side, we have a node for every node in the instance. There is an edge from pure cycle C to node i, if $i \notin C$ and there is an edge of cost 1 from i to some node in C. Let r be the number of pure cycles that are unmatched in the maximum cardinality bipartite matching. Papadimitriou and Yannakakis show how to "patch together" the matched cycles, and finally how to combine the resulting cycles into a tour of cost at most $\frac{7}{9}OPT(2M) + \frac{4}{9}|V| + \frac{1}{3}r$, where $OPT(2M)$ is the cost of an optimal 2M. We claim that

$$OPT(SUBT) \geq |V| + r,$$

where $OPT(SUBT)$ denotes the cost of the optimal subtour LP solution. Given the claim and Theorem 4.2.1, we have that the cost of the tour is then at most

$$\frac{7}{9} \cdot \frac{10}{9} OPT(SUBT) + \frac{4}{9} OPT(SUBT).$$

To prove the claim, we note that for a bipartite matching instance, there always exists a vertex cover that has size equal to the size of the maximum matching. We use this fact to construct a feasible dual solution to the subtour LP that has value $|V| + r$.

The dual of the subtour LP $(SUBT)$ is

$$\begin{array}{llll} & \text{Max} & 2\sum_{S \subset V} y(S) + 2\sum_{i \in V} y(i) - \sum_{e \in E} z(e) & \\ (D) & \text{subject to:} & \sum_{S \subset V : e \in \delta(S)} y(S) + y(i) + y(j) - z(e) \leq c(e), & \forall e = (i, j), \\ & & y(S) \geq 0, & \forall S \subset V,\ 3 \leq |S| \leq |V| - 3, \\ & & z(e) \geq 0, & \forall e \in E. \end{array}$$

We begin by setting $z(e) = 0$ for each $e \in E$, $y(i) = \frac{1}{2}$ for each $i \in V$, and for each pure cycle on vertices C, we set $y(C) = \frac{1}{2}$. Now, given a maximum matching in the bipartite graph constructed by the algorithm of Papadimitriou and Yannakakis, and a vertex cover of the same size, note that the vertex cover contains nodes and pure cycles of the 2M solution. We decrease the dual value for these objects to 0. Note that the dual objective for this solution is $|V| + r$.

It remains to show that the dual constructed is feasible. Define the *load* on an edge $e = (i, j)$ of solution (y, z) to be $\sum_{S \subset V : e \in \delta(S)} y(S) + y(i) + y(j) - z(e)$. For any edge of cost 1 inside a cycle of the 2M, the load on the edge is at most 1. For an edge (i, j) where $i \in C$ and $j \in C' \neq C$, the load is at most 2. Suppose (i, j) has cost 1, and the cycles C and C' are both pure cycles. Then the edge occurs twice in the bipartite matching instance (namely, once going from i to C and once going from j to C') and hence the dual of at least two of the four objects i, j, C and C' has been reduced to 0. The total load on edge (i, j) is thus at most 1. Now, suppose C' is the non-pure cycle, then $y(C') = 0$, since we only increased the dual variables for the pure cycles. Moreover, at least one endpoint of the (j, C) edge in the bipartite matching instance must be in the vertex cover, so the load on edge (i, j) is again at most 1. □

We note that the bound obtained on the integrality gap seems rather weak, as the best known lower bound on the integrality gap is only $\frac{10}{9}$. Schalekamp, Williamson, and van Zuylen [36] have conjectured that the integrality gap (or worst-case ratio) of the subtour LP occurs when the solution to the subtour LP is a fractional 2-matching. That is, the worst-case ratio for the subtour LP occurs for costs c such that an optimal subtour LP solution for c is the same as an optimal fractional 2-matching for c. Schalekamp et al. call such costs *c fractional 2-matching costs* for the subtour LP.

Conjecture 1 (Schalekamp et al. [36]) *The integrality gap for the subtour LP is at-*

tained for a fractional 2-matching cost for the subtour LP.

In the next section, we show that we can obtain better bounds on the integrality gap of the subtour LP in the case that the optimal solution is a fractional 2-matching.

4.3 Better Bounds if The Optimal Solution is A Fractional 2-Matching

If the optimal solution to the subtour LP is a fractional 2-matching, then a natural approach to obtaining a good tour is to start with the edges with cost 1 and x-value 1, and add as many edges of cost 1 and x-value $\frac{1}{2}$ as possible, without creating a cycle on a subset of the nodes. In other words, we will propose an algorithm that creates an acyclic spanning subgraph (V, T) where all nodes have degree one or two. We will call an acyclic spanning subgraph in which all nodes have degree 1 or 2 a partial tour. A partial tour can be extended to a tour by adding $d/2$ edges of cost 2, where d is the number of degree 1 nodes. The cost of the tour is $c(T) + d$, where $c(T) = \sum_{e \in T} c_e$.

We will prove the following lemma.

Lemma 4.3.1 *Let $G = (V, T)$ be a partial tour. Let A be a set of edges not in T that form an odd cycle or a path on $V' \subset V$, where the nodes in V' have degree one in T. We can find $A' \subset A$ such that $(V, T \cup A')$ is a partial tour, and*

- $|A'| \geq \frac{1}{3}|A|$ *if A is a cycle,*

- $|A'| \geq \frac{1}{3}(|A| - 1)$ *if A is a path,*

We postpone the proof of the lemma and first prove the implication for the bound on the integrality gap if the optimal subtour LP solution is a fractional 2-matching.

Theorem 4.3.2 *There exists a tour of cost at most α times the cost of an F2M solution if $c(i,j) \in \{1,2\}$ for all i, j, where α equals*

- $\frac{7}{6}$ *if the F2M solution is 1-connected;*
- $\frac{10}{9}$ *if the F2M solution is 2-connected and every 1-path that is adjacent to four unit cost cycle edges has cost at least 2.*

Proof: Let $P = \{e \in E : x(e) = 1\}$ (the edges in the 1-paths of x). We will start the algorithm with $T = P$. Let $R = \{e \in E : x(e) = \frac{1}{2} \text{ and } c(e) = 1\}$ (the edges of cost 1 in the cycles of x). Note that the connected components of the graph (V, R) consist of paths and odd cycles. The main idea is that we consider these components one by one, and use Lemma 4.3.1 to show that we can add a large number of the edges of each path and cycle, where we keep as an invariant that T satisfies the conditions of the lemma. Note that by Lemma 4.3.1, the number of edges added from each path or cycle A is at least $|A|/3$, except for the paths for which $|A| \equiv 1 \pmod 3$. Let $\mathcal{P}_1$ be this set of paths. We would like to claim that we add a third of the edges from *each* component, and we therefore preprocess the paths in $\mathcal{P}_1$, where we attempt to add one edge (either the first or last edge from each path in $\mathcal{P}_1$) to T, and remove this edge and its neighboring edge in R (if any) from R. After the preprocessing, we use Lemma 4.3.1 to process each of the components in (V, R).

More precisely, in the preprocessing step we (try to) add one edge, either the first or last edge, from each path in $\mathcal{P}_1$ to T, while maintaining that T is a partial tour. We call a path A in $\mathcal{P}_1$ "eared" if the 1-paths that are incident on the first and last node of the path are such that they go between two neighboring nodes of A. We claim that we can add an edge from at least half of the paths in $\mathcal{P}_1$ that are not eared: If we cannot add either the first or the last edge from a path A in $\mathcal{P}_1$, and A is not eared, then it must be the case that either the first or the last edge forms a cycle with an edge that was added to T from another path in $\mathcal{P}_1$. We remove the edges that were added to T, and their neighboring edge (if any) from R.

We now iterate through the connected components in (V, R) and add edges to T while maintaining that T is a partial tour. By Lemma 4.3.1, the number of edges added from each path or cycle A is at least $|A|/3$, except for the paths in $\mathcal{P}_1$. Note that for a path A in $\mathcal{P}_1$ that is not eared, and for which we had already added an edge to T in the preprocessing step, will have added a total of at least $1 + (|A| - 2 - 1)/3 = |A|/3$ edges. For a path in $\mathcal{P}_1$ for which we did not add an edge to T in the preprocessing stage, we have added at least $(|A| - 1)/3$ edges. Now, recall that a path A in $\mathcal{P}_1$ has $|A| \equiv 1 \pmod 3$, and that the number of edges added is an integer, so in the first case, the number of edges added is at least $|A|/3 + \frac{2}{3}$ and in the second case it is $|A|/3 - \frac{1}{3}$.

Let z be the number of eared paths in $\mathcal{P}_1$, and recall that we added an edge in the preprocessing stage for at least half of the paths in $\mathcal{P}_1$ that are not eared. Hence, the total number of edges from R added can be lower bounded by $\frac{1}{3}|R| - \frac{1}{3}z$. We now give an upper bound on the number of nodes of degree one in T.

Let k be the number of cycle nodes in x, i.e. $k = \#\{i \in V : x(i, j) =$

$\frac{1}{2}$ for some $j \in V\}$, and let p be the number of cycle edges of cost 2 in x, i.e. $p = \#\{e \in E : x(e) = \frac{1}{2}$ and $c(e) = 2\}$. Initially, when T contains only the edges in the 1-paths, all k nodes have degree one, and there are $k - p$ edges in R. We argued that we added at least $\frac{1}{3}|R| - \frac{1}{3}z = \frac{1}{3}k - \frac{1}{3}p - \frac{1}{3}z$ edges to T. Each edge reduces the number of nodes of degree one by two, and hence, the number of nodes of degree one at the end of the algorithm is at most $k - 2(\frac{1}{3}k - \frac{1}{3}p - \frac{1}{3}z) = \frac{1}{3}k + \frac{2}{3}p + \frac{2}{3}z$. Recall that $c(P)$ denotes the cost of the 1-paths, and the total cost of T at the end of the algorithm is equal to $c(P) + \frac{1}{3}k - \frac{1}{3}p - \frac{1}{3}z$. Since at most $\frac{1}{3}k + \frac{2}{3}p + \frac{2}{3}z$ nodes have degree one in T, we can extend T into a tour of cost at most $c(P) + \frac{2}{3}k + \frac{1}{3}p + \frac{1}{3}z$.

The cost of the solution x can be expressed as $c(P) + \frac{1}{2}k + \frac{1}{2}p$. We will now consider two cases which give different approximation guarantees.

We first consider the case where we only assume that the F2M solution x is connected. Note that each 1-path connects two cycle nodes, hence $c(P) \geq \frac{1}{2}k$. Moreover, an eared path A is incident to one (if $|A| = 1$) or two (if $|A| > 1$) 1-paths of length two, since the support graph of x is simple. Therefore we can lower bound $c(P)$ by $\frac{1}{2}k + z$. Therefore, $\frac{7}{6}\left(c(P) + \frac{1}{2}k + \frac{1}{2}p\right) \geq c(P) + \frac{1}{12}k + \frac{1}{6}z + \frac{7}{12}k + \frac{7}{12}p \geq c(P) + \frac{2}{3}k + \frac{1}{3}z + \frac{1}{3}p$, where $p \geq z$ is used in the last inequality.

If the F2M solution x is 2-connected, then $z = 0$, since if there is a 1-path connecting two nodes connected by a cycle edge $\{i, j\}$, then $\{i, j\}$ is a cut in x with only two cycle edges crossing the cut, and hence x is not 2-connected. Suppose that, in addition, x is such that every 1-path that is adjacent to four unit-cost cycle edges has cost at least 2. Then $c(P) \geq k - 2p$. Then $\frac{10}{9}\left(c(P) + \frac{1}{2}k + \frac{1}{2}p\right) \geq c(P) + \frac{1}{9}k - \frac{2}{9}p + \frac{5}{9}k + \frac{5}{9}p = \frac{2}{3}k + \frac{1}{3}p + c(P)$. □

Proof of Lemma 4.3.1: The basic idea behind the proof of the lemma is the following: We go through the edges of A in order, and try to add them to T if this does not create a cycle or node of degree three in T. If we cannot add an edge, we simply skip the edge and continue to the next edge. Since the edges in T form a collection of disjoint paths and each node in A has degree one in T, we can always add either the first edge or the second edge of A: if the first edge cannot be added, then adding it to T must create a cycle, and since the edges in T form a collection of node disjoint paths, adding the second edge of the path or cycle to T cannot create a cycle. Similarly, we need to skip at most two edges between two edges that are successfully added to T: first, an edge is skipped because otherwise we create a node of degree three in T, and if a second edge is skipped, then this must be because adding that edge to T would create a cycle. But then, adding the next edge on the path cannot create a cycle in T. Hence, the number of edges from we can add from each path or cycle A is at least $(|A| - 1)/3$, if A is a path, and $\lfloor |A|/3 \rfloor$, if A is a cycle, where $|A|$ denotes the number of edges in A.

We now show that by being a little more careful, we can in fact add $|A|/3$ edges if A is a cycle. Note that the number of nodes in A is odd, and hence there must be some j such that the path in T that starts in u_j ends in some node $v \notin A$. We claim that if we consider the edges in A starting with either edge $\{u_{j-1}, u_j\}$ or edge $\{u_j, u_{j+1}\}$, we are guaranteed that for at least one of these starting points, we can add both the first and the third edge to T.

Clearly, neither $\{u_{j-1}, u_j\}$ nor $\{u_j, u_{j+1}\}$ can create a cycle if we add it to T. So suppose that $T \cup \{u_{j-1}, u_j\} \cup \{u_{j+1}, u_{j+2}\}$ contains a cycle. This cycle does not contain the node u_j, because the path in T that starts in u_j ends in some node $v \notin C$. Hence T contains a path that starts in u_{j+1} and ends in u_{j+2}. But then

$T \cup \{u_j, u_{j+1}\} \cup \{u_{j+2}, u_{j+3}\}$ does not have a cycle, since if it did, T must have a path starting in u_{j+2} and ending in u_{j+3} which is only possible if $u_{j+1} = u_{j+3}$. Since the number of nodes in A is at least three, this is not possible. □

4.4 Computational Results

In the case of the 1,2-TSP, for a fixed n we can generate all instances as follows. For each value of n, we first generate all nonisomorphic graphs on n nodes using the software package NAUTY [30]. We let the cost of edges be one for all edges in G and let the cost of all other edges be two. Then each of the generated graph G gives us an instance of 1,2-TSP problem with n nodes, and this covers all instances of the 1,2-TSP for size n up to isomorphism.

In fact, we can do slightly better by only generating biconnected graphs. We say that a graph $G = (V, E)$ is *biconnected* if it is connected and there is no vertex $v \in V$ such that removing v disconnects the graph; such a vertex v is a *cut vertex*. We show that there must be a biconnected graph that gives rise to the worst case instances for the integrality gap for the 1,2-TSP.

For each instance of size n, we solve the subtour LP and the corresponding integer program using CPLEX 12.1 [12] and a Macintosh laptop computer with dual core 2GHz processor and 1GB of memory. It is known that the integrality gap is 1 for $n \leq 5$, so we only consider problems of size $n \geq 6$. The results are summarized in Table 4.1.

For $n = 11$, the number of nonisomorphic biconnected graphs is nearly a billion and thus too large to consider, so we turn to another approach. For $n = 11$

n	6	7	8	9	10	11	12
IP/LP ratio	8/7.5	8/7.5	9/8.5	10/9	11/10	12/11	13/12
# graphs	56	468	7,123	194,066	9,743,542	900,969,091	–

Table 4.1: The subtour LP integrality gap for 1,2-TSP for $6 \le n \le 12$, along with the number of nonisomorphic biconnected graphs for $6 \le n \le 11$.

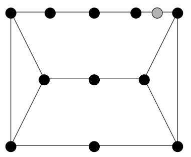


Figure 4.2: Illustration of the instances with integrality gap at least $\frac{12}{11}$ for $n = 11$ (without the grey node) and $\frac{13}{12}$ for $n = 12$ (with the grey node) for the 1,2-TSP. All edges of cost 1 are shown.

and $n = 12$, we use the fact that we know a lower bound on the integrality gap of $\frac{n+1}{n}$, namely for the instances depicted in Figure 4.2. The claimed lower bounds on the integrality gap for these instances follow readily from the integrality gap for the example in Figure 4.1. We then check whether this is the worst integrality gap for each vertex of subtour LP. A list of non-isomorphic vertices of the subtour LP is available for $n = 6$ to 12 at Sylvia Boyd's website `http://www.site.uottawa.ca/~sylvia/subtourvertices`. In order to check whether the lower bound on the integrality gap is tight, we solve the following integer programming problem for each vertex x of the polytope for $n = 11$ and $n = 12$, where now the costs $c(e)$ are the decision variables, and x is fixed:

$$\text{Max} \quad z - \alpha_n \sum_{e \in E} c(e)x(e)$$

subject to:

$$\sum_{e \in T} c(e) \ge z, \qquad\qquad \forall \text{ tours } T,$$

$$c(e) \in \{1, 2\}, \qquad\qquad \forall e \in E.$$

Note that α_n is the lower bound on the integrality gap for instances of n nodes. If the objective is nonpositive for all of the vertices of the subtour LP, then we know that α_n is the integrality gap for a particular value of n.

Since the number of non-isomorphic tours of n nodes is $(n-1)!/2$, the number of constraints is too large for CPLEX for $n = 11$ or 12. We overcome this difficulty by first solving the problem with only tours that have at least $n - 1$ edges in the support graph of the vertex x, and repeatedly adding additional violated tours. Our results shows that the worst case integrality gap for $n = 11$ is $\frac{12}{11}$ and for $n = 12$ is $\frac{13}{12}$.

We now show that the worst-case integrality gap for the subtour LP for the 1,2-TSP can be found on graphs of cost 1 edges that are biconnected. Let $OPT(G)$ and $SUBT(G)$ be the cost of the optimal tour and the value of the subtour LP (respectively) when G is the graph of cost 1 edges. We start by proving that the worst case is obtained on a connected graph.

Lemma 4.4.1 *Let G be the graph of cost 1 edges in a 1,2-TSP instance. Then if G is not connected, there exists a connected graph G' such that $OPT(G)/SUBT(G) \leq OPT(G')/SUBT(G')$.*

Proof: Suppose G has more than one connected component. We create $G' = (V', E')$ by adding a new vertex i^* to the graph, and adding edges from all $j \in V$ to i^* so that $V' = V \cup \{i^*\}$ and $E' = E \cup \{(i^*, j) : j \in V\}$. Given a tour of G', we can easily produce a tour of G of no greater cost by shortcutting i^*, so that $OPT(G) \leq OPT(G')$. Let x be an optimal solution to the subtour LP for the graph

G. We now define a solution x' for G', where $x'_{ij} = x_{ij}$ if i and j are in the same connected component of G, while if i and j are in different connected components of G, then we set $x'_{ij} = 0$, $x'_{i^*i} = x_{ij}$, and $x'_{i^*j} = x_{ij}$. It is easy to see that the cost of x' is the same as that of x. We now argue that there is some solution x'' feasible for the subtour LP on G' such that its cost is no greater, so that $SUBT(G') \leq SUBT(G)$. It is clear that the bounds constraints (4.3) are satisfied for x' and the degree constraints (4.1) are satisfied for x' for all $i \in V$; however, the degree constraint for i^* may not be satisfied. Since for any component $C \subseteq V$ of G, $x(\delta(C)) \geq 2$, it is clear that $x'(\delta(i^*)) \geq 2$, but it may be the case that $x'(\delta(i^*)) > 2$. For the subtour constraints (4.2), consider any $S \subset V'$, $S \neq \emptyset$, such that $i^* \notin S$. Then $x'(\delta(S)) \geq x(\delta(S)) \geq 2$, and for any $S \subseteq V'$ with $i^* \in S$, $S \neq \{i^*\}$, $x'(\delta(S)) = x'(\delta(V' - S)) \geq 2$ by the previous argument. Finally, Goemans and Bertsimas [16] have shown (see also Williamson [39]) that if edge costs obey the triangle inequality, and there is some solution x' to the subtour LP in which degree constraints are exceeded but all other constraints are met, then there is another feasible solution x'' of no greater cost in which all constraints are satisfied. Hence we have that $SUBT(G') \leq SUBT(G)$. Thus we have that $OPT(G)/SUBT(G) \leq OPT(G')/SUBT(G')$. □

Lemma 4.4.2 *Let G be the graph of cost 1 edges in a 1,2-TSP instance. Then if G is not biconnected, there exists a biconnected G' such that $OPT(G)/SUBT(G) \leq OPT(G')/SUBT(G')$.*

Proof: By Lemma 4.4.1, we can assume that the graph $G = (V, E)$ is connected. Let $i_1, \ldots, i_k$ be all the cut vertices of G, and let $C_1, \ldots, C_\ell$ be all the connected components formed when these vertices are removed, so that $C_1, \ldots, C_\ell, \{i_1\}, \ldots, \{i_k\}$ form a partition of V. We create a new graph $G' = (V', E')$

by adding a new vertex i^*, and adding edges from i^* to each vertex in $C_1 \cup \cdots \cup C_\ell$, so that $V' = V \cup \{i^*\}$ and $E' = E \cup \{(i^*, j) : j \in C_p$ for some $p\}$. We note that G' is biconnected. As before, we have $OPT(G) \le OPT(G')$ since given a tour of G' we can shortcut i^* to get a tour of G. Let x be an optimal subtour LP solution for graph G. We now argue, as we did in the proof of Lemma 4.4.1, that we can create an x' that costs no more than x such that all the subtour and bounds constraints are obeyed, and all degree constraints are either met or exceeded; this will imply that $SUBT(G') \le SUBT(G)$, and complete the proof. Suppose without loss of generality that removing cut vertex i_1 creates components C_1 and $C = C_2 \cup \cdots \cup C_\ell \cup \{i_2\} \cup \cdots \cup \{i_k\}$, so that C_1, $\{i_1\}$, and C partition V. We set $x'_{ij} = 0$ and $x'_{i^*i} = x'_{i^*j} = x_{ij}$ if $i \in C_1$ and $j \in C$; $x'_{ij} = x_{ij}$ otherwise. If $i \in C_1$ and $j \in C$, then $(i, j) \notin E$ since i_1 is a cut vertex, so the cost of x' is no more than that of x. The arguments that all constraints are satisfied except for the degree constraint on i^* follow as in the proof of Lemma 4.4.1. We now must argue that $x'(\delta(i^*)) \ge 2$. To do this, we show that $\sum_{i \in C_1, j \in C} x_{ij} \ge 1$. Since $x(\delta(i_1)) = 2$, it must be the case that either $\sum_{j \in C} x_{i_1 j} \le 1$ or $\sum_{j \in C_1} x_{i_1 j} \le 1$; without loss of generality we assume the former is true. Then since $x(\delta(C_1 \cup \{i_1\})) \ge 2$, and $x(\delta(C_1 \cup \{i_1\})) = \sum_{j \in C} x_{i_1 j} + \sum_{i \in C_1, j \in C} x_{ij}$, it follows that $\sum_{i \in C_1, j \in C} x_{ij} \ge 1$, and the proof is complete. □

Notice that the proof of Lemma 4.4.1 shows that the worst-case gap of disconnected cost-1 graphs with n nodes can not be larger than the worst-case gap of connected cost-1 graphs with $n + 1$ nodes. Similarly, the proof of Lemma 4.4.2 shows that the worst-case gap of connected cost-1 graphs with $n + 1$ nodes can not be larger than the worst-case gap of biconnected cost-1 graphs with $n + 2$ nodes. Therefore, by finding worst-case gap on biconnected graphs with $6 \le n \le 10$ and worst-case gap for all graphs with $n = 11$ and $n = 12$, we show that the worst-case gap for $n \le 12$ is $\frac{10}{9}$.

CHAPTER 5

CONCLUSIONS AND OPEN QUESTIONS

In this chapter, we summarize our results and conclude with some open questions.

For the prize-collecting Steiner forest problem, our result answers negatively the open question by authors in [37] that one might be able to obtain an approximation ratio better than 2.54 for this problem using natural extension of Jain [24]'s iterative rounding. Our algorithm has an approximation ratio of 3 and is tight. Independently, Konemann et al. [27] and Hajiaghayi and Nasri [22] both gave iterative rounding 3-approximation algorithms for this problem with similar structures that every basic feasible solution to (LP) must have a variable with value greater or equal to $\frac{1}{3}$. They also gave similar tight examples as ours. Hajiaghayi and Nasri further extended their algorithm to solve the prize-collecting generalized Steiner network problem.

For the online Steiner forest problem, we give a primal-dual $O(\log n)$ approximation algorithm, and compare the solutions of our algorithm to the optimal solution in the offline case by conducting computational experiments on a set of randomly generated large-scale inputs similar to real-world instances. An interesting open question is whether primal-dual algorithms for the offline generalized Steiner network problem with r_i edge-disjoint paths for each terminal pair i (such as those in [40, 18]) can be adapted to the online case as we did here. It is known that if R_i is the set of terminals that have arrived by the ith time step, then there is a lower bound of $\Omega(|R_i|)$ on the competitive ratio. If $r_{\max} = \max_i r_i$, Gupta, Krishnaswamy, and Ravi [20] have given an $O(r_{\max} \log^3 n)$-competitive algorithm for the online generalized Steiner network problem, so such an adaptation might be possible.

For the worst-case integrality gap of 1,2-symmetric TSP, our result shows this gap is at most $\frac{106}{81} \approx 1.31 < \frac{4}{3}$. Our computational experiments shows the worst-case gap for $n \leq 12$ is $\frac{10}{9}$ and we conjecture the following.

Conjecture 5.0.3 *The integrality gap of the subtour LP for the 1,2-TSP is $\frac{10}{9}$.*

Schalekamp, Williamson, and van Zuylen [36] have conjectured that the integrality gap (or worst-case ratio) of the subtour LP occurs when the solution to the subtour LP is a fractional 2-matching. That is, the worst-case ratio for the subtour LP occurs for costs c such that an optimal subtour LP solution for c is the same as an optimal fractional 2-matching for c. Schalekamp et al. call such costs *c fractional 2-matching costs* for the subtour LP.

Conjecture 5.0.4 (Schalekamp et al. [36]) *The integrality gap for the subtour LP is attained for a fractional 2-matching cost for the subtour LP.*

We have shown in Theorem 4.3.2 that if an analogous conjecture is true for 1,2-TSP, then the integrality gap for 1,2-TSP is at most $\frac{7}{6}$; it would be nice to show that if the analogous conjecture is true for 1,2-TSP then the integrality gap is at most $\frac{10}{9}$.

BIBLIOGRAPHY

[1] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24:440–456, 1995.

[2] Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized Steiner problem. *Theoretical Computer Science*, 324:313–324, 2004.

[3] M. L. Balinski. Integer programming: Methods, uses, computation. *Management Science*, 12:253–313, 1965.

[4] Piotr Berman and Chris Coulston. On-line algorithms for Steiner tree problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 344–353, 1997.

[5] Piotr Berman and Marek Karpinski. 8/7-approximation algorithm for (1,2)-TSP. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 641–648, 2006.

[6] M. Bläser and L. Shankar Ram. An improved approximation algorithm for TSP with distances one and two. In Maciej Liskiewicz and Rüdiger Reischuk, editors, *Fundamentals of Computation Theory, 15th International Symposium, FCT 2005*, volume 3623 of *Lecture Notes in Computer Science*, pages 504–515. Springer, 2005.

[7] Sylvia Boyd and Robert Carr. Finding low cost TSP and 2-matching solutions using certain half-integer subtour vertices. *Discrete Optimization*, 8(4):525 – 539, 2011.

[8] Sylvia Boyd, René Sitters, Suzanne van der Ster, and Leen Stougie. TSP on cubic and subcubic graphs. In Oktay Günlük and Gerhard J. Woeginger, editors, *Integer Programming and Combinatorial Optimization, 15th International Conference, IPCO 2011*, number 6655 in Lecture Notes in Computer Science, pages 65–77. Springer, Berlin, Germany, 2011.

[9] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanitá. An improved LP-based approximation for Steiner tree. In *Proceedings of the 42st Annual ACM Symposium on Theory of Computing*, pages 583–592, 2010.

[10] Miroslav Chlebík and Janka Chlebíková. Approximation hardness of the

Steiner tree problem on graphs. In *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, pages 170–179. Springer, 2002.

[11] N. Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. In *Report 388*. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1976.

[12] IBM ILOG CPLEX 12.1. 2009.

[13] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410, 1954.

[14] Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms*, pages 942–951, 2008.

[15] Michel X. Goemans. Worst-case comparison of valid inequalities for the TSP. *Mathematical Programming*, 69:335–349, 1995.

[16] Michel X. Goemans and Dimitris J. Bertsimas. Survivable networks, linear programming relaxations, and the parsimonious property. *Mathematical Programming*, 60:145–166, 1990.

[17] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

[18] M.X. Goemans, A.V. Goldberg, S. Plotkin, D.B. Shmoys, É. Tardos, and D.P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 1994.

[19] Martin Grőtschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Springer, Berlin, Germany, 1993.

[20] Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Online and stochastic survivable network design. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 685–694, 2009.

[21] MohammadTaghi Hajiaghayi and Kamal Jain. Prize-collecting generalized

Steiner tree problem via a new approach of primal-dual schema. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 631–640, 2006.

[22] Mohammadtaghi Hajiaghayi and Arefeh A. Nasri. Prize-collecting Steiner networks via iterative rounding. In *Proceedings of The 9th Latin American Theoretical Informatics Symposium (LATIN)*, volume 6034, pages 515–526, 2010.

[23] Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4:369–384, 1991.

[24] Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21:39–60, 2001.

[25] David S. Johnson, Maria Minkoff, and Steven Phillips. The prize-collecting Steiner tree problem: Theory and practice. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000.

[26] Richard M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, NY, 1972.

[27] J. Konemann, F. Grandoni, T. Rothvoss, G. Schaefer, and C. Swamy. An iterated rounding 3-approximation algorithm for price-collecting Steiner forest. *Unpublished*, 2009.

[28] I. Ljubic, R. Weiskircher, U. Pferschy, G. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming*, 105:427–449, 2006.

[29] MathWorks MATLAB R2010b. 2010.

[30] Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–97, 1981.

[31] Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *Proceedings of the 52th Annual Symposium on Foundations of Computer Science*, pages 560–569, 2011.

[32] Marcin Mucha. 13/9-approximation for graphic TSP. Appears at `http://arxiv.org/abs/1108.1130.`, 2011.

[33] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *Proceedings of the 52th Annual Symposium on Foundations of Computer Science*, 2011.

[34] Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18:1–11, 1993.

[35] Sartaj Sahni and Trofilo Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23, 1976.

[36] Frans Schalekamp, David P. Williamson, and Anke van Zuylen. A proof of the Boyd-Carr conjecture. In *Proceedings of the 23nd ACM-SIAM Symposium on Discrete Algorithms*, 2012. To appear.

[37] Yogeshwer Sharma, Chaitanya Swamy, and David P. Williamson. Approximation algorithms for prize collecting forest problems with submodular penalty functions. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, pages 1275–1284, 2007.

[38] David B. Shmoys and David P. Williamson. Analyzing the Held-Karp TSP bound: A monotonicity property with application. *Information Processing Letters*, 35:281–285, 1990.

[39] David P. Williamson. Analysis of the Held-Karp heuristic for the traveling salesman problem. Master's thesis, MIT, Cambridge, MA, June 1990. Also appears as Tech Report MIT/LCS/TR-479.

[40] David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15:435–454, 1995.

[41] Laurence A. Wolsey. Heuristic analysis, linear programming and branch and bound. In *Combinatorial Optimization II*, volume 13 of *Mathematical Programming Studies*, pages 121–134. Springer, 1980.