A Primal-Dual Clustering Technique with Applications in Network Design

MohammadHossein Bateni

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Moses S. Charikar

September 2011

Abstract

Network design problems deal with settings where the goal is to design a network (i.e., find a subgraph of a given graph) that satisfies certain connectivity requirements. Each requirement is in the form of connecting (or, more generally, providing large connectivity between) a pair of vertices of the graph. The goal is to find a network of minimum length, and in some cases requirements can be compromised after paying their "penalties." These are usually called *prize-collecting Steiner network* problems.

In practical scenarios of physical networking, with cable or fiber embedded in the ground, crossings are rare or nonexistent. Hence, planar instances of network design problems are a natural subclass of interest. We can usually take advantage of this structure to find better performance guarantees.

In this thesis, we develop a primal-dual clustering technique called "prize-collecting clustering," which is used to give improved approximation algorithms for several planar and nonplanar network design problems. The technique is based on a famous moat growing procedure due to Agrawal, Klein, Ravi [AKR95] and Goemans and Williamson [GW95]. It provides a paradigm for clustering the vertices of a graph according to their "budgets" such that vertices of the same cluster are close compared to their budgets, whereas distinct clusters are far compared to their budgets.

We first improve the approximation ratio of (nonplanar) Prize-collecting Steiner Tree, Prize-collecting TSP, and Prize-collecting Stroll. For 17 years, the best known results for these problems were 2-approximation algorithms of Goeman and Williamson [AKR95]. We show how to get around the integrality gap of the natural linear-programming relaxation, and achieve an approximation ratio of $2 - c$ (for a fixed, yet small c).

Next we give a thorough complexity study of Steiner Forest for graphs of treewidth two, three, and $O(1)$ as well as planar and bounded-genus graphs. In particular, we provide a polynomial-time approximation scheme (PTAS) for Steiner

Forest on planar graphs. Prize-collecting clustering paradigm allows us to generalize PTASes for Euclidean Steiner Tree [Aro98, Mit99], Euclidean Steiner Forest [BKM08], and planar Steiner Tree [BKM09]. Our algorithm builds upon the *brick decomposition* technique of Borradaile et al. [BKM09], in addition to a nontrivial PTAS for bounded-treewidth Steiner Forest.

Finally, we look at several prize-collecting Steiner network problems on planar (and bounded-genus) graphs. We present a reduction from these instances to the bounded-treewidth special cases of those problems implying, in particular, that a PTAS carries over. For Prize-collecting Steiner Tree (Prize-collecting TSP, and Prize-collecting Stroll) as well as Multiplicative Prize-collecting Steiner Forest, we show that this leads to PTASes. However, we show that several seemingly simple problems in this area are APX-hard. As a result, we give the first provable separation between the complexity of a natural network design problem and its prize-collecting variant: a PTAS for planar Steiner Forest and APX-hardness for planar Prize-collecting Steiner Forest.

We hope that the prize-collecting clustering paradigm can be used to give PTASes and improved approximation guarantees for several other network design problems.

Acknowledgements

First and foremost, I feel indebted to my adviser, Moses Charikar, whose support has been invaluable during my years in Princeton. Only now, preparing to move on to the next stage of my academic life and getting out of the Princeton bubble, am I beginning to fully appreciate the fruits of his guidance. His patience and understanding my situation as well as assisting me with writing papers and preparing my presentations have been crucial. I should thank him once more since he was available for advice or academic help whenever I needed it most, no matter how late or inconvenient the time. I will always look up to him in my life, and aspire to attain one day his insight into theoretical computer science.

I have also learned a lot from my main collaborator, MohammadTaghi Hajiaghayi, who has been for me not only a collaborator, but a friend and a mentor, too. I have sought his advice on many issues, academic or otherwise, and I have never regretted the decisions I have made as a result. He is my coauthor on all the papers comprising the body of the current thesis. In fact, he introduced me to the field of network design, and the project finally leading to this thesis started when I was working with him as an intern in AT&T Labs–Research. This thesis, therefore, would not have been possible without his guidance and support.

I want to take on this opportunity to thank my thesis committee members—Moses Charikar, MohammadTaghi Hajiaghayi, Sanjeev Arora, Bernard Chazelle, and Robert Schapire—who have accommodated my timing constraints despite their full schedules, and provided me with precious feedback for the presentation of the results, in both written and oral form.

During my Ph.D. studies, I had the pleasure of collaborating with many researchers from each and every one of which I had things to learn. The list includes Aaron Archer, Moses Charikar, Julia Chuzhoy, Alexandre Gerber, Lukasz Golab, Venkatesan Guruswami, MohammadTaghi Hajiaghayi, Nicole Immorlica, Sina Jafar-

pour, Howard Karloff, Philip Klein, Hamid Mahini, Dániel Marx, Claire Mathieu, Dan Pei, Subhabrata Sen, and Morteza Zadimoghaddam. I wish to thank them all.

Living in Princeton without my good friends would not have been easy. I want to thank all my friends in the department; in particular, Aditya, Aravindan, Sina, Sushant, and Mohammad deserve special thanks, for they provided me with moments of joy and supported me in their own capacity as I needed. Outside the department, I wish to thank Abbas, Ahmad, Amin, Diya, Fethi, Hadee, Hadi, Hamid, Hani, Ilhan, Iman, Mohammad, Mohammad, Mohammad, Morteza, Noori, Parween, Reza, Reza, Reza, Shayan, Taher, Tiffany, Vahab, Wasim, and Yusuf, in particular, though I am sure I have missed at least a handful of others.

I appreciate the generous fellowships and grants supporting my study and research in Princeton University. I was honored to receive the Gordon Wu Fellowship for 2006-2010, and the Elizabeth Charlotte Procter Fellowship for 2010-2011. In addition, my research was supported by NSF ITR grants CCF-0205594, CCF-0426582 and NSF CCF 0832797, NSF CAREER award CCF-0237113, MSPA-MCS award 0528414, NSF expeditions award 0832797. I also spent three summers as interns in David Johnson's group at AT&T Labs–Research, working with Julia Chuzhoy at Toyota Technological Institute, and working with Niv Buchbinder at Microsoft Research–New England. I am grateful to them for the opportunities they provided for me.

Last but definitely not least, I want to express my deepest gratitude to my beloved parents, Mahmoud and Fatemeh, and my dearest siblings, Farzaneh and MohammadAmin. Their unflagging love and unwavering support have been crucial to my success, and a constant source of comfort and counsel. I can never do justice to explaining how much I owe them. During my studies at Princeton University, I sorely missed them. Though they were not here in person, I have always felt their presence in my heart. I dedicate this thesis to my parents.

به نام یگانه درخور عشق،

و چشم به راه یادگارش در زمین؛

تقدیم به پدر دانا و

مادر دلسوز و مهربانم.

Contents

List of Tables

List of Figures

List of Algorithms

Part I

Preliminaries

Chapter 1

Introduction

Network design problems deal with settings where the goal is to design a network (i.e., find a subgraph of a given graph) that satisfies certain connectivity requirements; see, e.g., [Win87, GK11]. Edges of the given graph describe the cost of the possible links the network may have, and each requirement is in the form of connecting (or, more generally, providing large connectivity between) a pair of vertices of the graph. The goal is to find the network of minimum cost (i.e., the subgraph of minimum length), where connectivity requirements can sometimes be compromised after paying their associated penalties. These are usually called *prize-collecting Steiner network* problems.

One of the most fundamental problems in network design is Steiner Tree (a generalization of Minimum Spanning Tree) in which, given a subset of vertices (called *terminals*), the goal is to find the minimum-length subgraph (a tree, without loss of generality) that spans all the terminals. A greedy 2-approximation algorithm was long known [GP68] before improvements came in the form of a series of more and more complicated greedy algorithms [Zel92, Zel93, BR92, Zel96, PS97, KZ97, HP99, RZ00, RZ05], culminating in an approximation ratio of $1 + \frac{1}{2}\ln 3 + \epsilon \approx 1.55$. A natural primal-dual algorithm achieving the approximation guarantee as well as

integrality gap of 2—the same as the first greedy algorithm—was recently enhanced to give an approximation guarantee of 1.55 [CKP10b, CKP10a, KPT11] and finally $\ln 4 + \epsilon \approx 1.387$ [BGRS10].

Steiner Tree has been generalized in multiple directions. If each demand requires two vertices to be connected to each other, we have Steiner Forest. Surprisingly, after two decades, a 2-approximation algorithm of Agrawal, Klein, and Ravi [AKR95] is still the best known for Steiner Forest.

In some applications, it is not necessary to satisfy all the demands. When serving at least k demands suffices, we obtain k-MST or k-Steiner Forest. While k-Steiner Forest is believed to be hard to approximate (since it can be used to solve the notorious k-Densest Subgraph [HJ06, BCC$^+$10]), k-MST has been studied extensively [RSM$^+$94, AABV95, RV95, BRV96, Gar96, AR98, AK06, Gar05]: the most recent algorithms use the fact that k-MST is the *Lagrangian relaxations* of Prize-collecting Steiner Tree (to be introduced below). The best approximation ratio for k-MST is a factor 2.

In order to model the tradeoff between the number of demands satisfied and the length of the solution, "prizes" (sometimes called "penalties") are assigned to demands: the cost of a solution is then the sum of the length of the subgraph we construct and the penalties of demands we do not serve. See, e.g., [Bal89, GW95, HJ06, NSW08].

The prize-collecting framework, in particular Prize-collecting Steiner Forest, can be used to address many applications. The standard application is that of designing telecommunications local access networks, where the goal is to create or expand a local access network to offer service to new customers [CRR01, dCLMR03]. Prizes (in the form of potential revenue) are assigned to new customers, whereas serving them imposes a cost on the network. Another application is in the area of energy distribution, in particular planning and extending district heating networks, proposed

by [LWP+06]. Lee et al. [LLP96] use Prize-collecting Steiner Tree to address a hub selection scenario in the design of digital data service networks. In addition to the above applications, Prize-collecting Steiner Tree has been used as a subroutine in solving several other problems such as cost allocation [EGLV98], multicasting games [CKR+03], and dynamic pricing [FP03] scenarios.

The prize-collecting framework is a broad setting that can be applied to different problems in which a penalty is associated with not satisfying certain demands. This has been studied, for instance, in the context of Traveling Salesman, Stroll, and Rural Postman Problem [Bal89, AFZ06]. Generalized versions of the penalty function have been considered, e.g., by [HJ06, SSW07, HKKN10], and extensions to higher connectivity requirements have been studied by [NSW08, HKKN10].

We note that, in network design, planarity is a natural restriction since in practical scenarios of physical networking, with cable or fiber embedded in the ground, crossings are rare or nonexistent. The setting of road networks, even with bridges an underpasses, is believed to have an embedding with a small genus [BDT09, OGS11]. Since Euclidean, planar, and bounded-genus instances are ubiquitous in practice and seem to be the more interesting cases, it makes sense to study these special cases further. It turns out that we can usually take advantage of this additional structure to find better performance guarantees.

There is a wealth of literature on obtaining improved approximation algorithms for planar graphs. Here we focus on PTASes. The seminal work of Baker [Bak94] obtained PTASes for several optimization problems on planar graphs (such as Minimum Vertex Cover and Maximum Independent Set) although the corresponding problems on general graphs are considerably harder to approximate. The main idea in her work is a decomposition approach that reduces the problem on a planar graph to the problem on graphs of bounded treewidth. This approach has been subsequently applied in a variety of contexts. (The algorithmic and graph-theoretic properties of

treewidth are extensively studied and a well-understood dynamic programming technique can solve several NP-hard problems on bounded-treewidth graphs.) The broad outline of the PTAS approach for planar graphs had to be augmented with a variety of nontrivial ideas and extensions.

A parallel framework was developed at around the same time to tackle many optimization problems on Euclidean metrics (see [Aro98, Mit99]). In particular, PTASes are known for Steiner Tree, Steiner Forest, Facility Location, and good approximation algorithms for Group Steiner Tree.

In this thesis, we develop a primal-dual clustering technique called "prize-collecting clustering," which is used to give improved approximation algorithms for several planar and nonplanar network design problems. The technique is based on a famous moat growing procedure due to Agrawal, Klein, Ravi [AKR95] and Goemans and Williamson [GW95]. It provides a paradigm for clustering the vertices of a graph according to their "budgets" such that vertices of the same cluster are close compared to their budgets, whereas distinct clusters are far compared to their budgets.

We first improve the approximation ratio of (nonplanar) Prize-collecting Steiner Tree, Prize-collecting TSP, and Prize-collecting Stroll. For 17 years, the best known results for these problems were 2-approximation algorithms of Goeman and Williamson [AKR95]. We show how to get around the integrality gap of the natural linear-programming relaxation, and achieve an approximation ratio of $2 - c$ (for a fixed, yet small c).

Next we give a thorough complexity study of Steiner Forest for graphs of treewidth two, three, and $O(1)$ as well as planar and bounded-genus graphs. In particular, we provide a polynomial-time algorithm for the case of treewidth two, an NP-hardness proof for the case of treewidth three, and polynomial-time approximation scheme (PTAS) for Steiner Forest on bounded-genus (as well as planar and bounded-treewidth) graphs. Prize-collecting clustering paradigm allows us to

generalize PTASes for Euclidean Steiner Tree [Aro98, Mit99], Euclidean Steiner Forest [BKM08], and planar Steiner Tree [BKM09]. Our algorithm builds upon the *brick decomposition* technique of Borradaile et al. [BKM09], in addition to a nontrivial PTAS for bounded-treewidth Steiner Forest.

Finally, we look at several prize-collecting Steiner network problems on planar (and bounded-genus) graphs. We present a reduction from these instances to the bounded-treewidth special cases of those problems implying, in particular, that a PTAS carries over. For Prize-collecting Steiner Tree (Prize-collecting TSP, and Prize-collecting Stroll) as well as Multiplicative Prize-collecting Steiner Forest, we show that this leads to PTASes. However, we show that several seemingly simple problems in this area are APX-hard. As a result, we give the first provable separation between the complexity of a natural network design problem and its prize-collecting variant: a PTAS for planar Steiner Forest and APX-hardness for planar Prize-collecting Steiner Forest.

We hope that the prize-collecting clustering paradigm can be used to give PTASes and improved approximation guarantees for several other network design problems.

Chapter 2

Definitions

2.1 Essential definitions

The ordering of items in a set does not matter, i.e., $\{a_1, a_2\}$ is the same set as $\{a_2, a_1\}$. When the ordering is important, we use $\langle a_1, a_2, \ldots, a_k \rangle$ to denote the sequence, or $(a_1, a_2, \ldots, a_k)$ to refer to a multiple.

We say a set A is the *disjoint union* of B and C if $A = B \cup C$ and $B \cap C = \emptyset$.

As is customary, we use A^T to denote the transpose of a matrix A. We think of a vector of size n as a matrix with dimensions $n \times 1$, hence we use the same notation for transposing vectors. The dot product of two vectors a and b is, then, simply the single entry in the result of their matrix products, i.e., $a^\mathsf{T} \cdot b$. To make the formulas more concise, we use $A \leq B$, for a pair of matrices A, B with the same dimensions, to denote that all entries of A are smaller than their corresponding entries in B.

2.1.1 Functions

For a function (or, equivalently, mapping) $f : A \mapsto B$, we say A is the *domain*, and B is the *range*. A function is *injective* if it does not map two elements of its domain to the same element in its range. A *surjective* function is one that maps at least one

element of the domain to each element in the range. A function that is both injective and surjective is called a *bijection* or a *one-to-one correspondence*. The inverse of a unction $f : A \mapsto B$ is denoted by f^{-1}. The inverse is a function itself if f is a bijection, but in general we use the same notation even if there is no inverse function; in this case $f^{-1}(x) = \{y : f(y) = x\}$ may be empty or contain more than one element.

For the ease of notation, we use $f(A') = \sum_{a \in A'} f(a)$ if $f : A \mapsto \mathbb{R}$ and $A' \subseteq A$. If x is a vector whose components are indexed by elements of A, we use a similar notation $x_{A'} = \sum_{a \in A'} x_a$ if $A' \subseteq A$. As we usually work with nonnegative functions, we use $\mathbb{R}^+, \mathbb{Z}^+$ to denote the set of nonnegative real and integers numbers, respectively.

The following standard asymptotic order notation are used throughout this thesis to describe the running time, space complexity, etc. Consider a function $f : \mathbb{R}^+ \mapsto \mathbb{R}^+$. Then, $f(n) = O(g(n))$ if and only if there exists $n_0, c > 0$ such that $f(n) \leq cg(n)$ for $n \geq n_0$. We then have $g(n) = \Omega(f(n))$. If $f(n) = O(g(n))$ and $g(n) = O(f(n))$, we say $f(n) = \Theta(g(n))$. In addition, $f(n) = o(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, and then we say $g(n) = \omega(f(n))$.

2.1.2 Partitions

A collection $\mathcal{S}$ is said to be *laminar* if and only if for any two sets $C_1, C_2 \in \mathcal{S}$, we have $C_1 \subseteq C_2$, $C_2 \subseteq C_1$, or $C_1 \cap C_2 = \emptyset$. Suppose $\mathcal{C}$ is a partition of a ground set V. Then, $\mathcal{C}(v)$ denotes for each $v \in V$ the class $C \in \mathcal{C}$ that contains v. Classes of a partition are sometimes called *sets*, *parts*, or *components* of the partition. A partition $\mathcal{C}$ of a ground set V can be considered as an equivalence relation on V. Hence, we use notation $(x, y) \in \mathcal{C}$ to say that x and y are in the same class of $\mathcal{C}$. We say that partition α is *finer* than partition β if $(x, y) \in \alpha$ implies $(x, y) \in \beta$; in this case, β is *coarser* than α. If $\alpha = \beta$, then α is both finer and coarser than β. We denote by $\alpha_1 \vee \alpha_2$ the unique finest partition α coarser than both α_1 and α_2.

2.1.3 Submodularity

A function $f : 2^U \mapsto \mathbb{R}$ is *submodular* if $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$ holds for every $X, Y \subseteq U$. Submodularity on set functions can be thought of as the analog of convexity for real functions. Submodular functions have found many applications in combinatorial optimization [Edm03, Fra93, Von07] especially because they exhibit the *diminishing returns* property (and, hence, can model many utility functions): i.e., $f(A \cup \{a\}) - f(A) \leq f(A' \cup \{a\}) - f(A')$ if $A' \subseteq A$ and $a \in U \setminus A$. In fact, this is an equivalent definition. There has been considerable work on optimizing submodular functions; see, e.g., [CFN77, NW78, NWF78, FNW78, FMV07] for maximization and [Fle00, Mcc05] for minimization. (These are mostly for the unconstrained setting, while there is a flurry of results in the recent years for optimizing submodular functions given extra restrictions [LSV09, LMNS10, GRST10, BHZ10].)

Since the description of submodular functions usually requires exponential space, we tend to have "oracle access" to these functions: for any subset $S \subseteq U$ of the ground set of the submodular function f, we can call a polynomial-time subroute (whose internal working is not transparent) to find $f(S)$. Among the many result for submodular function optimization, we use the following. (We only give credit to the first such result, although numerous consequent work has improved the running time.)

Theorem 2.1.1 (Grötschel, Lovász, and Schrijver [GLS88])**.** *Given oracle access to a submodular function, there is a (strongly) polynomial-time algorithm that computes a set minimizing the function.*

Examples of submodular functions are rank functions of matroids, size of set unions, and cut functions of directed graphs; see [Sch03, Chapter 44]. The first two examples are monotone (i.e., increasing) functions, whereas the last one is a nonmonotone submodular function.

2.1.4 Metrics

A *metric* on a set V is function $\mu : V \times V \mapsto \mathbb{R}^+$ such that $\mu(v, v) = 0$, $\mu(u, v) = \mu(v, u)$, and $\mu(u, v) + \mu(v, w) \geq \mu(u, w)$ for all $u, v, w \in V$. Metrics are abstractions of distance functions.

The two-dimensional Euclidean metric is a metric on (a subset of) the points in the two-dimensional plane that denotes the natural straight-line distances between points. This is a special case of ℓ_2 metric that can have arbitrary dimension. More generally, the d-dimensional ℓ_p metric assumes distance

$$\left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

between two points $x = (x_1, x_2, \ldots, x_d), y = (y_1, y_2, \ldots, y_d)$ in the d-dimensional space.

Another very important metric is the *graph metric*: i.e., the distance function obtained from the length of shortest paths in a given graph. If the given graph has additional structure, it may lead to more special metrics such as *planar graph metric*, *bounded-treewidth graph metric*, etc. Often times, when we mention a graph as the input to a problem, we merely mean the metric induced by that graph.

2.2 Computational complexity

2.2.1 Efficient algorithms

The running time and space usage of algorithms are usually measured via the order notations described above. An algorithm with a main parameter n can have a running time which is, for instance, polynomial, quasipolynomial, or exponential in n. Exponential running times can be divided into singly exponential, i.e., $2^{O(n)}$, doubly exponential, i.e., $2^{2^{O(n)}}$, etc. Edmonds [Edm65] was the first to suggest the notion

of polynomial running time—i.e., $O(n^c)$ for any fixed constant c—as a measure for *efficiency* of an algorithm. In his seminal paper, he gave the first efficient algorithm for finding a maximum matching on nonbipartite graphs.

Perhaps, one of the main techniques for deriving efficient algorithms is that of *dynamic programming*, or DP for short. Whereas a naïve algorithm attempting to solve the problem via recursion may resolve certain subproblems many times, DP stores sufficient information about solved subproblems to avoid resolving them. Hence, there is usually a table of values associated with all the possible subproblems of the original instance, and they are systematically solved based on information from previous table entries. The solution to the original instance then appears somewhere in the table; see [CLRS09, Chapter 15] for more details and several examples.

2.2.2 Linear programming

In *mathematical programming*, one tries to optimize (either maximize or minimize) an *objective function* subject to several *constraints*. A special case that has received particular attention in computer science is *linear programming*, in which a vector $x \in \mathbb{R}^n$ is sought to optimize a linear objective function subject to linear constraints. The following is a linear program in its standard form.

$$\begin{aligned} \text{minimize} \quad & c^{\mathsf{T}} x \\ \text{subject to} \quad & Ax \geq b \\ & x \geq 0, \end{aligned} \tag{2.1}$$

where b, c are vectors and A is a matrix. It is simple to observe that (2.1) captures an arbitrary linear program defined above: in particular, equality constraints ($A'x = b'$), constraints of the opposite direction ($A''x \leq b''$), unconstrained variables (i.e., without the nonnegativity constraints), and maximization objective functions can all be modeled by (2.1) via simple modifications of the parameters or adding new

constraints and variables; see [Sch86] for a thorough overview of linear programming.

Any linear program (LP) defined above can be solved in polynomial time [Kha79, Kar84]. Despite having a poorer running-time bound, the *ellipsoid method* [Kha79] has the advantage of not requiring all the constraints at once. In fact, it can find an optimal solution—more precisely, a solution with arbitrary inverse polynomial precision—in polynomial time given a polynomial-time *separation oracle*. A separation oracle is a procedure that takes as input a vector x and determines whether x satisfies all the constraints of the linear program or not; in the latter case, the separation oracle identifies and reports at least one violated constraint. This can be helpful when a linear program has exponentially many constraints, however, the combinatorial structure of the constraints make it possible to verify satisfiability of a candidate solution.

For any linear program, a *dual* program can be written; the original program is then called the *primal* program. The dual of LP (2.1) is

$$\begin{aligned} \text{maximize} \quad & b^{\mathsf{T}} y \\ \text{subject to} \quad & A^{\mathsf{T}} y \leq c \\ & y \geq 0. \end{aligned} \tag{2.2}$$

The primal and dual programs are related by the following duality theorems.

Theorem 2.2.1 (LP weak duality)**.** *Consider a primal minimization program with objective function $c^{\mathsf{T}} x$, and its corresponding dual program with objective function $b^{\mathsf{T}} y$. For any two feasible solutions x, y for primal and dual programs, respectively, we have $b^{\mathsf{T}} y \leq c^{\mathsf{T}} x$.*

The above theorem is easy to verify since $b^{\mathsf{T}} y \leq (Ax)^{\mathsf{T}} y = x^{\mathsf{T}} A^{\mathsf{T}} y \leq x^{\mathsf{T}} c = c^{\mathsf{T}} x$, where the inequalities follow from the constraints in the primal and dual programs, respectively. Von Neumann [N47] and Gale, Kuhn, and Tucker [GKT51] establish the

strong LP duality as follows.

Theorem 2.2.2 (LP strong duality [N47, GKT51])**.** *Consider a primal minimization program with objective function $c^\mathsf{T}x$, and its corresponding dual program with objective function $b^\mathsf{T}y$. If either of primal or dual has unbounded optimum, the other is infeasible. Otherwise, the optimal values of the objective functions of the two programs are identical, i.e., there exist solutions x^*, y^* for primal and dual programs, respectively, such that $c^\mathsf{T}x^* = b^\mathsf{T}y^*$.*

If variables are allowed to be constrained to take only integer values, for instance $x \in \{0,1\}^n$, we obtain an *integer (linear) program* or IP. It turns out—see NP-hardness in Section 2.2.3—that integer programs are hard problems to optimize. Relaxing the integrality constraints yields a *linear-programming relaxation*. These relaxations have been the subject of study for many years. The best algorithms for several problems are derived via *rounding* the fractional solutions to integral ones, or are inspired by the properties of the linear-programming relaxation (e.g., in the primal-dual method).

When we use LP relaxations for solving or approximating a problem, the *integrality gap* of the LP usually proves to be a barrier on how good our solution can be. For a minimization problem, the integrality gap is defined as

$$\sup_{\text{instance } \mathcal{I} \text{ of the problem}} \frac{\text{optimal solution of } \mathcal{I}}{\text{value of LP relaxation of } \mathcal{I}},$$

where the optimal solution is equal to the value of the integer program. In other words, the integrality gap shows how optimistic the relaxation may be compared to the actual solution. The integrality gap is defined similarly for a minimization problem:

$$\sup_{\text{instance } \mathcal{I} \text{ of the problem}} \frac{\text{value of LP relaxation of } \mathcal{I}}{\text{optimal solution of } \mathcal{I}}.$$

Sometimes it makes sense to relax a linear program by moving some of the hard constraints into the objective function and imposing a penalty on the objective func-

tion if those constraints are violated. This method is called the "Lagrangian relaxation" method. For example, $\min_{x,\lambda \geq 0} \left\{ cx - \lambda^{\mathsf{T}}(Ax - b) \right\}$ is a Lagrangian relaxation of LP (2.1)—it is possible to move only some of the constraints although we moved all of them in this example. Since the vector λ, called the *Lagrange multipliers*, is nonnegative, we get penalized if some of the LP constraints are violated, and get awarded if some are satisfied strictly. It is easy to see (since $\lambda = 0$ is a feasible solution) that the value of the Lagrangian relaxation is no worse than that of the original program. In some applications, either because rounding the original problem is difficult or since its running time is worse, we may employ this method, and work on the Lagrangian relaxation. By changing the range of Lagrangian multipliers and using the lower bound as a guide, one can optimize the original problem in an iterative fashion. This method was first developed by Held and Karp [HK70, HK71].

2.2.3 NP-completeness

As mentioned previously, algorithms with polynomial running times are most desirable. In fact, we call a problem admitting such an algorithm *tractable*, and others are called *intractable*. In complexity theory, problems are grouped into several classes according to how easy or hard they are. The class P contains all those problems with a polynomial-time algorithm. More technically, these classes usually apply to *decision problems* only, where the solution of a problem is either "yes" or "no." A bigger class, clearly containing P, is NP which contains any problem whose solution can be verified in polynomial time.

For a few decades, computer scientists have tried to answer the question $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$: i.e., whether all problem in NP admit polynomial-time algorithms. A problem L is $\mathsf{NP\text{-}hard}$ if it is "harder" than all problems in NP: more specifically, any (instance of any) problem in NP can be *reduced* to (an instance of) L in polynomial time in the sense that solving the resulting instance of L gives the solution to the original

problem; this concept was proposed by Cook [Coo71]. An NP-hard problem in NP is said to be NP-complete. In particular, giving a polynomial-time algorithm for any NP-complete problem settles the "P versus NP" question by proving P = NP. The celebrated Cook-Levin theorem [Coo71, Lev73] provided the first NP-complete problem: Circuit-SAT. Later, Karp [Kar72] proved NP-completeness for several natural problems through reductions from Circuit-SAT. Gary and Johnson collected many NP-complete problems in their book [GJ79].

We introduce a simple NP-complete problem here. An instance of 3SAT consists of m disjunctive clauses of exactly three boolean literals each, where each literal is a variable x_i or its negation $\overline{x_i}$. For instance, $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee x_4)$ is an instance of 3SAT with two clauses and four variables. The goal is to determine whether there exists an assignment of $\{0, 1\}$ values (i.e., true or false) to the variables that satisfies all the clauses simultaneously.

Theorem 2.2.3 ([Kar72])**.** 3SAT *is NP-complete.*

Since many natural problems are NP-complete, the focus has shifted to finding *approximation algorithms* for them. An algorithm $\mathcal{A}$ is an α-approximation algorithm for a minimization problem if, for any instance $\mathcal{I}$ of the problem, $\mathcal{A}$ produces in polynomial time a feasible solution whose "cost" is no more than $\alpha \cdot \mathsf{opt}(\mathcal{I})$. Then, α is called the *approximation ratio* or the *approximation guarantee*. If the approximation ratio is one, we have an *exact algorithm*. For maximization problems, the approximation ratio is defined similarly as

$$\sup_{\text{instance } \mathcal{I} \text{ of the problem}} \frac{\mathsf{opt}\ (\mathcal{I})}{\text{value of } \mathcal{A}\text{'s solution for } \mathcal{I}}.$$

If a problem admits an α-approximation algorithm, we say it belongs to $\mathsf{aprx}(\alpha)$.

For some NP-hard problems, it is still possible to obtain very good approximation ratios. A *polynomial-time approximation scheme*, or a PTAS for short, is a series of

$(1+\epsilon)$-approximation algorithms $\mathcal{A}_\epsilon$ that run in polynomial time as long as $\epsilon > 0$ is a constant. If the running time is $O(f(\epsilon)n^c)$ for a fixed constant c independent of ϵ, the approximation scheme is *efficient*, hence we have an EPTAS. If in addition, $f(\epsilon)$ is a polynomial function of $1/\epsilon$, the algorithm is a fully polynomial-time approximation scheme (FPTAS).

2.2.4 Hardness of approximation

Certain NP-hard problems can be shown to be hard even to approximate. The seminal work in this area is the PCP theorem [ALM+98, AS98, FGL+96] that among other things leads to inapproximability results for optimization problems such as Max 3SAT, Maximum Independent Set, etc.

A problem is said to be APX-hard if it does not admit a PTAS unless P = NP. More specifically, there exists a constant $\alpha > 1$ such that existence of an α-approximation algorithm for the problem proves P = NP.

Given a graph $G(V, E)$[1], Minimum Vertex Cover asks to find the minimum number of edges that cover all vertices as their endpoints. In this thesis, we use the following hardness result.

Theorem 2.2.4 ([AK00])**.** Minimum Vertex Cover *is APX-hard even when restricted to* 3*-regular graphs.*

Some optimization problems have a hard constraint on, e.g., an upper bound on the weight of certain objects in the solution (capacity), or a lower bound on the profit of items collected. Relaxing these hard constraints may lead to better approximation factors on the main objective. This is formalized in the form of *bicriteria* approximation guarantee. Suppose, for instance, that the goal is to minimize the "cost" while respecting a capacity constraint. An algorithm that always produces in polynomial

[1]Graphs are formally introduced in the next section.

time a solution of cost at most α times the cost of the optimal solution, but may violate the capacity constraint with a factor no more than β, is said to have a bicriteria (α, β) approximation guarantee.

2.3 Graph terminology

A *graph* $G(V, E)$ is a collection of *vertices*, V, and a collection of *edges*, E, which is a multiset of (unordered or ordered) vertex pairs. We may remove the reference to $V = V(G), E = E(G)$ if they are clear from the context and/or they are not particularly important for the sake of the argument. In this thesis, unless otherwise specified, we consider undirected graphs where edges correspond to unordered vertex pairs, whereas in directed edges consists of ordered vertex pairs. We call u, v the *endpoints* of an edge $e = (u, v)$. An edge whose endpoints are identical is called a *loop*. Two or more edges corresponding to one pair of vertices are called *parallel edges*. A graph without loops or parallel edges is called a *simple graph*. We mostly deal with simple graphs, using the term *multigraph* to emphasize the possibility of loops and parallel edges.

An edge $e = (u, v)$ is *incident on* vertices u and v. Existence of this edge makes u and v *adjacent*, and each of u or v is called a *neighbor* of the other. The *degree* of vertex v, denoted $\mathsf{deg}_G(v)$, is the number of edges incident on v, i.e., $\mathsf{deg}_G(v) = |\{e \in E : e = (u, v)\}|$. For a set $S \subseteq V$ of vertices, $\overline{V} = V \setminus S$ is the *complement* of S. A set of vertices and its complement define a *cut* consisting of all the edges having one endpoint in S and one in $\overline{S}$; we refer to this cut by $G[S, \overline{S}]$, $[S, \overline{S}]$, or $\delta_G(S) = \delta_G(\overline{S})$. A cut $\delta_G(S)$ is an *s-t* cut if it *separates* s and t: i.e., $|S \cap \{s, t\}| = 1$. A cut is called *simple* if its removal from the graph leaves exactly two connected components, i.e., one on each side. The *neighbor set* of S, denoted $\Gamma_G(S)$, is defined as the set of vertices outside S that have an edge to a vertex in S, i.e., $\left\{v \in \overline{S} : \exists u \in S, (u, v) \in E\right\}$. In

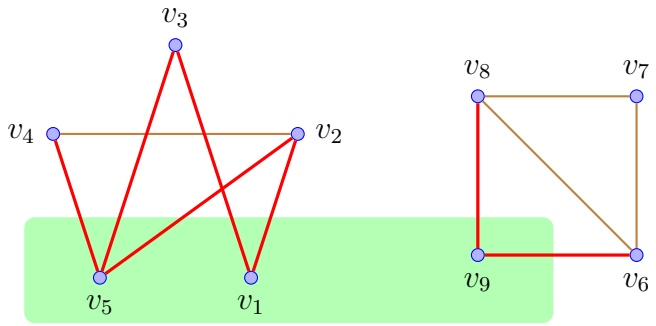


Figure 2.1: Illustration of basic graph definitions. The depicted graph has 9 vertices, 11 edges, and two connected components: $\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_6, v_7, v_8, v_9\}$. Notice that vertices $\{v_1, v_3, v_4\}$ have degree two, and the other vertices have degree three. Thicker edges show the cut $\delta(\{v_1, v_5, v_9\})$.

our notation, we usually drop the reference to G when it is clear from the context.

Proposition 2.3.1. *Sum of the degrees of vertices of a graph G is equal to twice the number of edges in G.*

The *complete* graph on n vertices, denoted K_n, has an edge between every pair of vertices. A graph is *bipartite* if it does not have any odd cycles. The vertices of a bipartite graph can be partitioned into two groups (or sides) such that no edge connects vertices of the same group. A *complete bipartite* graph $K_{m,n}$ is a bipartite graph with m vertices on one side and n on the other, with an edge between any pair of vertices from different sides. A graph is said to be d-regular if all its vertices have degree exactly d.

For visual illustration, graphs are normally represented by a diagram with vertices mapped to points and edges depicted by curves joining their endpoints. See Figure 2.1. Although not used for the most part of this thesis, we illustrate edges of directed graphs by arrows.

A *path* p between u and v consists of a sequence of vertices $\langle u = u_0, u_1, u_2, \ldots, u_{k-1}, u_k = v \rangle$ where any two consecutive vertices are connected by an edge, i.e., $(u_i, u_{i+1}) \in E$ for $0 \leq i < k$. The path is said to have size $|p| = k$, and its endpoints are u and v. (For weighted graphs, the length of the path, $\ell(p)$, denotes the

total length of all the edges on the path p.) A vertex v is said to be *reachable* from u if there exists a path between them. Reachability leads to an equivalency class on the vertices of the graph: a maximal set of vertices that are pairwise reachable is a *connected component* of the graph.

A graph $H(V', E')$ is a *subgraph* of $G(V, E)$ if $V \subseteq V'$ and $E' \subseteq E$. Then, G is said to be a *supergraph* of H. The graph $G(V, E)$ is *isomorphic* to $H(V', E')$ if there is a bijection $f : V \mapsto V'$ such that $(u, v) \in E \Leftrightarrow (f(u), f(v)) \in E'$. We say G contains H if G has a subgraph that is isomorphic to H. A subgraph $H(V', E')$ of $G(V, E)$ is the *(vertex-) induced* subgraph on V' if E' is the set of all edges in E both whose endpoints are in V'. We say $H(V', E')$ is an *(edge-) induced* subgraph by E' if V' is the set of endpoints of edges in E'. The notation $G[V']$ refers to the vertex-induced subgraph of G on V', i.e., H as defined above. Whenever it is clear from the context, we may use the edge set E' to refer to the induced subgraph $H(V', E')$.

A subgraph $H(V, E')$ is a *spanning subgraph* of $G(V, E)$ if H is connected. A connected graph $G(V, E)$ is span the set V of vertices. A cycle on n vertices is formed by adding an edge (v_n, v_1) to the path $\langle v_1, v_2, \ldots, v_n \rangle$. A graph $T(V, E)$ is a *forest* if it does not contain any cycles. A forest is a *tree* if it is connected. A vertex of degree one is called a *leaf*.

Proposition 2.3.2. *The number of edges in a forest on n vertices with k components is $n - k$. In particular, a tree on n vertices has exactly $n - 1$ edges.*

Proposition 2.3.3. *The average degree of a tree is less than two, and any tree with more one vertex has at least two leaves.*

A cycle in G is called *Hamiltonian* if it passes each vertex exactly one. A Hamiltonian path is defined similarly. An *Eulerian cycle* for a graph G is one that has exactly one copy of each edge of G. An Eulerian path is defined similarly. The following is the characterization of "Eulerian" graphs [Eul41, Hie73].

Theorem 2.3.4. *A graph G has an Eulerian cycle if and only if G is connected, and all its vertices have even degree. If G is connected, and has at most two vertices of odd degree, it has an Eulerian path.*

We focus on *weighted graphs* where a nonnegative length function $\ell : E \mapsto \mathbb{R}^+$ is associated with the edges of the graph. As per our general convention, we use $\ell(E')$ to denote the total length of all edges in E'.

We refer to the length of a path p connecting u and v by the distance from u to v along p. Then, the *distance* from u to v, denoted $\mathsf{dist}_G(u, v)$, is the minimum distance along all paths in G between u and v. In the case when no such path exists, the distance is said to be infinite. Otherwise, the minimum exists since lack of negative-length edges renders repeated use of an edge useless for getting shorter paths. Each path of length $\mathsf{dist}_G(u, v)$ from u to v is called a *shortest path* between u and v. We sometimes use the notation $\mathsf{dist}_G(u, S)$, for $S \subset V(G)$, to denote the length of a shortest path of u to the set S; i.e., $\mathsf{dist}_G(u, S) = \min_{v \in S} \mathsf{dist}_G(u, v)$.

Contracting an edge $e = (u, v)$ of a graph $G(V, E)$ gives a graph $H(V', E')$ in which u and v are identified, and given a new identity. Then, we remove the loops and parallel edges formed.[2] More formally, $V' = V \setminus \{u, v\} \cup \{z\}$ where z is a new vertex, and $e = (x, y) \in E'$ if $x, y \in V'$ and

1. $\{x, y\} \cap \{z\} = \emptyset$ and $e \in E$; or
2. $x = z$ and $y \neq z$ and $\{(u, y), (v, y)\} \cap E \neq \emptyset$.

We denote the result of contracting G along e by G/e. In addition, G/E' refers to $G/e_1/e_2/\ldots/e_k$ for an edge set $E' = \{e_1, e_2, \ldots, e_k\} \subseteq E$. The graph H is a *minor* of G if H is isomorphic to the result of contracting zero or more edges of a subgraph of G. Figure 2.2 illustrates the contraction operation.

[2] In the case of multigraphs, we do not need to remove the resulting loops and parallel edges. For weighted graphs, the removal of parallel edges is not arbitrary; i.e., we are required to keep the minimum-length one among a set of parallel edges, so as to guarantee certain desired shortest-path properties in the contracted graph.

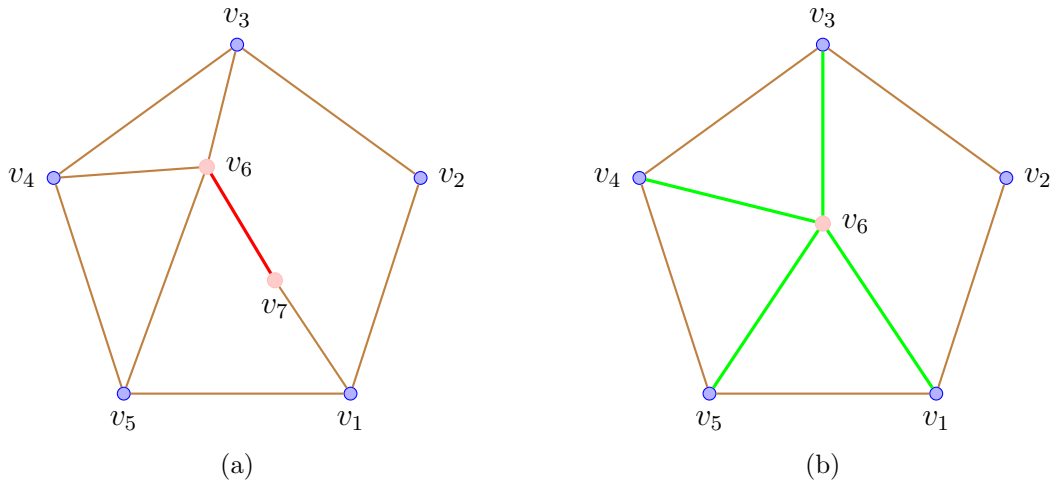


Figure 2.2: Illustration of edge contraction operation. (a) shows the original graph; (b) shows the result of contracting edge (v_6, v_7).

For details of these definitions, or a more thorough introduction to graph theory, the reader can refer to a standard textbook on the subject, e.g., [Wes00].

2.3.1 Graph classes

Here we define several classes of graphs. You can take a look at their relationship in Figure 2.6, where an arrow indicates any instance of one class is also an instance of the other.

Graphs of bounded treewidth

We start with the definition of *treewidth*, as introduced by Robertson and Seymour [RS86], which is a measure of how close a graph is to a tree. Notice that, since trees have a very simple structure, one expects that many graph optimization problems are tractable (or, at least, admit good approximation ratios) when the input instance is a graph with small treewidth—in fact, dynamic programming can be used to give exact algorithms for many of these scenarios.

To define *treewidth*, we consider representing a graph by a tree structure, called

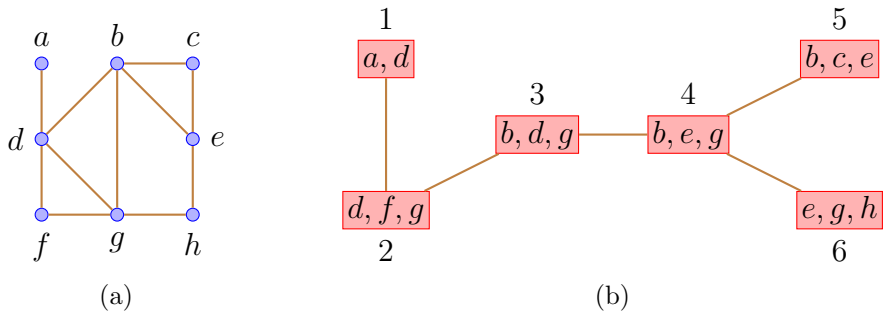


Figure 2.3: Illustration of tree decomposition. (a) shows a graph G of treewidth two (it is indeed a series-parallel graph. The vertices of G are $\{a, b, c, d, e, f, g, h\}$. (b) presents a tree decomposition of width two for the graph. The contents of the bag of each node of the tree decomposition are shown inside it. Four vertices a, c, f, h, each appears in only one node of the tree decomposition. Vertex b appears in bags for $3, 4, 5$; vertex d appears in bags of $1, 2, 3$; and vertex e belongs to three bags, those of $4, 5, 6$. There is only one vertex, g, that belongs to four nodes. Notice that the bags containing each vertex form a connected subgraph.

a *tree decomposition*; see Figure 2.5. More precisely, a tree decomposition of a graph $G(V, E)$ is a pair $(T, \mathcal{B})$ in which $T(I, F)$ is a tree and $\mathcal{B} = \{B_i \mid i \in I\}$ is a family of subsets of $V(G)$ such that

1. $\bigcup_{i \in I} B_i = V$;
2. for each edge $e = (u, v) \in E$, there exists an $i \in I$ such that both u and v belong to B_i; and
3. the set of nodes $\{i \in I \mid v \in B_i\}$ forms a connected subtree of T for every $v \in V$.

To distinguish between vertices of the original graph G and vertices of T in the tree decomposition, we call vertices of T *nodes* and their corresponding B_i's *bags*. The *width* of the tree decomposition is one less than the maximum size of a bag in $\mathcal{B}$. The *treewidth* of a graph G, denoted $\mathsf{treewidth}(G)$, is the minimum width over all possible tree decompositions of G.

We say a graph has bounded treewidth if its treewidth is a constant. A simple observation is that the treewidth is equal to one if and only if the graph is a tree.

As useful a parameter as it is (see, e.g., [Bod98]), computing the treewidth is NP-hard even for restricted classes of graphs such as graphs of bounded degree [BT97], bipartite graphs [Klo96], and their complements [ACP87]. As long as the treewidth is a constant, though, there is a linear-time algorithm to compute the treewidth and a corresponding tree decomposition [Bod96]. From an algorithmic point of view, we usually work with *bounded-treewidth graphs* on which many problems are solvable in polynomial time [Bod93]. In this case, a treewidth decomposition can be constructed in linear time.

In addition, there has been a significant amount of work to approximate the treewidth for different classes of graphs. Constant approximation factors exist for planar graphs [KR91, AKR03], and for H-minor-free graphs [Haj05]. For general graphs of a specific treewidth, the best approximation ratio is $O(\sqrt{\log(\mathsf{treewidth})})$ [Haj05, FHL08].

For algorithmic purposes, it is convenient to define a restricted form of tree decomposition. We say that a tree decomposition $(T, \mathcal{B})$ is *nice* if the tree T is a rooted tree such that for every $i \in I$ either

1. i has no children (i is a *leaf node*),
2. i has exactly two children i_1, i_2 and $B_i = B_{i_1} = B_{i_2}$ holds (i is a *join node*),
3. i has a single child i' and $B_i = B_{i'} \cup \{v\}$ for some $v \in V$ (i is an *introduce node*), or
4. i has a single child i' and $B_i = B_{i'} \setminus \{v\}$ for some $v \in V$ (i is a *forget node*).

It is well-known that every tree decomposition can be transformed into a nice tree decomposition of the same width in polynomial time. Furthermore, we can assume that the root bag contains only a single vertex.

We will use the following lemma to obtain a nice tree decomposition with some further properties; a similar trick was used in [Mar07].

Lemma 2.3.5. *Let G be a graph having no adjacent degree 1 vertices. G has a nice tree decomposition of polynomial size having the following two additional properties:*

1. *No introduce node introduces a degree 1 vertex.*

2. *The vertices in a join node have degree greater than 1.*

Proof. Consider a nice tree decomposition of graph G. First, if v is a vertex of degree 1, then we can assume that v appears only in one bag: if w is the unique neighbor of v, then it is sufficient that v appears in any one of the bags that contain w. Let $B_v = \{v, x_1, \ldots, x_t\}$ be this bag where $x_1 = w$. We modify the tree decomposition as follows. We replace B_v with $B'_v = B_v \setminus \{v\}$, insert a bag $B''_v = B_v \setminus \{v\}$ between B'_v and its parent, and create a new bag $B^t = B_v \setminus \{v\}$ that is the other child of B''_v (thus B''_v is a join node). For $i = 1, \ldots, t-1$, let $B^i = \{x_1, \ldots, x_i\}$, and let B^i be the child of B^{i+1}. Finally, let $B_w = \{w, v\}$ be the child of B^1 and let $B = \{v\}$ be the child of B_w. Observe that B^i $(2 \leq i \leq t)$, B_w are introduce nodes, B^1 is a forget node, and B is a leaf node. This operation ensures that vertex v appears only in a leaf node. It is clear that, after repeating this operation for every vertex of degree 1, the two required properties will hold. □

Series-parallel graphs

A series-parallel graph is a graph that can be built using series and parallel composition; see Figure 2.4. Formally, a *series-parallel graph* $G(x, y)$ with distinguished vertices x, y is an undirected graph that can be constructed using the following rules.

1. A graph with only two vertices x, y and an edge (x, y) between them is a series-parallel graph.

2. If $G_1(x_1, y_1)$ and $G_2(x_2, y_2)$ are series-parallel graphs, then the graph $G(x, y)$ obtained by identifying x_1 with x_2 and y_1 with y_2 is a series-parallel graph with distinguished vertices $x := x_1 = x_2$ and $y := y_1 = y_2$—a parallel connection.

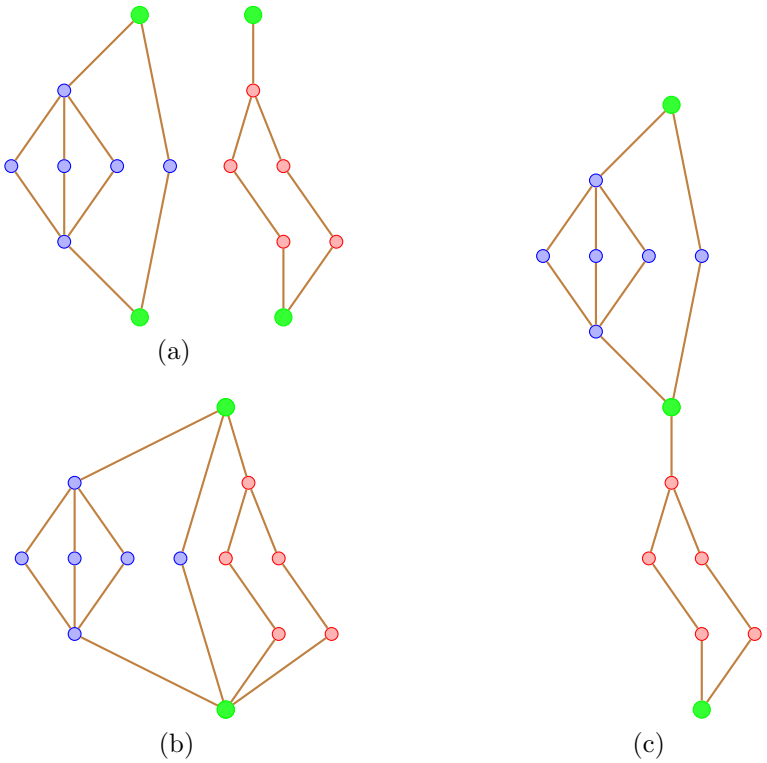
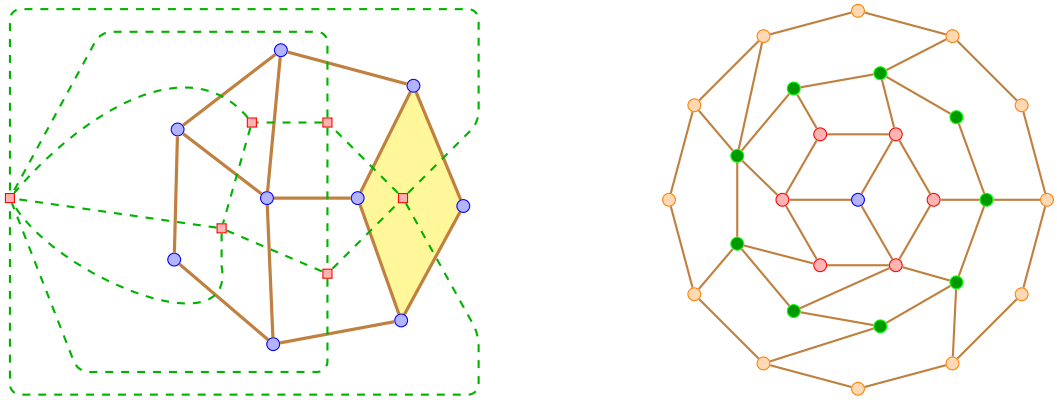


Figure 2.4: Illustration of series and parallel connections. (a) shows two graphs side by side; (b) illustrated those two graphs connected via a parallel connection; and (c) shows the combination of the two graphs in a series connection.

3. If $G_1(x_1, y_1)$ and $G_2(x_2, y_2)$ are series-parallel graphs, then the graph $G(x, y)$ obtained by identifying y_1 with x_2 is a series-parallel graph with distinguished vertices $x := x_1$ and $y := y_2$—a series connection.

We sometimes call x the left exit point, and y the right exit point of the graph since the two distinguished vertices are the only vertices that may interact with vertices outside G if G is used in constructing a more complicated series-parallel graph.

It is well-known that the treewidth of a graph is at most 2 if and only if it is a subgraph of a series-parallel graph [Bod98]. Since setting the length of an edge to ∞ in a connectivity problem is essentially the same as deleting the edge, solving such problems on graphs with treewidth at most 2 is equivalent to solving them for series-parallel graphs.



(a) (b)

Figure 2.5: (a) depicts a planar graph with nine vertices, and six faces. There are five finite faces, and one infinite face. One face is shaded. The edges of this graph are shown by solid lines. The edges of the dual graph are drawn via dashed lines/curves. (b) shows a 4-outerplanar graph. The vertices of the same color (and shape) form different layers than can be peeled off one by one.

Duffin [Duf65] proved that a graph is series-parallel if and only if it has no K_4 minor.

It can be easily observed that graphs with treewidth at most 2 have planar embeddings (see below for definition of planarity), but there are graphs of treewidth three—e.g., $K_{3,3}$—that are not planar.

Planar graphs

A graph if *planar* if it can be embedded into $\mathbb{R}^2$ in such a way that two edges may only intersect at their endpoints. When working with planar graphs, we usually assume such an embedding exists. (There are equivalent combinatorial definitions though.) Each component of $\mathbb{R}^2 \setminus G$ is a face of G. A vertex or edge of G is said to be incident on a face f if the boundary of f, usually denoted ∂f, contains it. Two faces are adjacent if one edge is incident on both. There is a unique unbounded face, called the *infinite face*; all other faces are bounded.

Euler's formula relates the number of vertices, edges, and faces of a planar graph

as follows.

Theorem 2.3.6. *Let n, e, f, c denote the number of vertices, edges, faces, and connected components of a planar graph. Then, we have $n + f = e + c + 1$.*

An simple corollary bounds the number of edges of a planar graph in terms of its number of vertices, hence it guarantees at least one vertex of small degree.

Corollary 2.3.7. *No connected simple planar graph with at least three vertices has more than $3n - 6$ edges. Every simple planar graph has a vertex of degree no more than 5.*

The first combinatorial characterization of planar graphs is due to Kuratowski [Kur30]. We state an extension of Kuratowski's theorem by Wagner [Wag37] which gives a forbidden minor characterization for planar graphs.

Theorem 2.3.8 (Wagner [Wag37])**.** *A graph is planar if and only if it does not have $K_{3,3}$ or K_5 as a minor.*

The dual G^* of an embedded planar graph $G = (V, E)$ is a graph, whose vertex set is the set of faces of G, that has for each edge e an edge e^* connecting the two faces e is incident on. The dual G^* is planar, and $(G^*)^*$ is isomorphic to G if G is connected.

Outerplanar graphs

A graph G is *outerplanar* if it has a planar embedding with all vertices incident on the infinite face. A simple corollary of Kuratowski's theorem shows that a graph is outerplanar if and only if it has no K_4 or $K_{2,3}$ minor.

For $k \geq 1$, we say a graph G is *k-outerplanar* if removing all vertices incident on the infinite face from G (in addition to all edges incident on them) yields a $(k-1)$-outerplanar graph. In other words, the vertices of G can be peeled off in k levels. Bodlaender [Bod96] observed that these graphs have treewidth at most $3k - 1$.

Bounded-genus graphs

We also need a basic notion of embedding; see, e.g., [RS94, CM05]. In this thesis, an *embedding* refers to a 2-*cell embedding*, i.e., a drawing of the vertices and edges of the graph as points and arcs in a surface such that every face (connected component obtained after removing edges and vertices of the embedded graph) is homeomorphic to an open disk. We use basic terminology and notions about embeddings as introduced in [MT01]. We only consider compact surfaces without boundary. Occasionally, we refer to embeddings in the plane, when we actually mean embeddings in the 2-sphere. If S is a surface, then for a graph G that is (2-cell) embedded in S with f facial walks, the number $g = 2 - |V(G)| + |E(G)| - f$ is independent of G and is called the *Euler genus* of S. The Euler genus coincides with the crosscap number if S is nonorientable, and equals twice the usual genus if the surface S is orientable. A graph is planar if and only if it has Euler genus zero.

We note that many natural networks (e.g., caber of fiber connections in telecommunications networks, or road networks) are almost planar. The setting of road networks, even with bridges an underpasses, is believed to have an embedding with a small genus [BDT09, OGS11].

H-minor-free graphs

A (possibly infinite) family of (finite) graphs is *minor-closed* if all minors of any graph in the family belongs to the family. The celebrated Graph Minor Theorem [RS94] implies that there is a forbidden minor characterization for any minor-closed family of finite graphs; i.e., there exists a finite set of "forbidden" graphs that the given minor-closed family is equal to the set of graphs excluding those forbidden graphs as minors. We have mentioned such characterizations for planar graphs (Theorem 2.3.8), series-parallel, and outerplanar graphs.

Robertson and Seymour also give a characterization of the set of graphs excluding

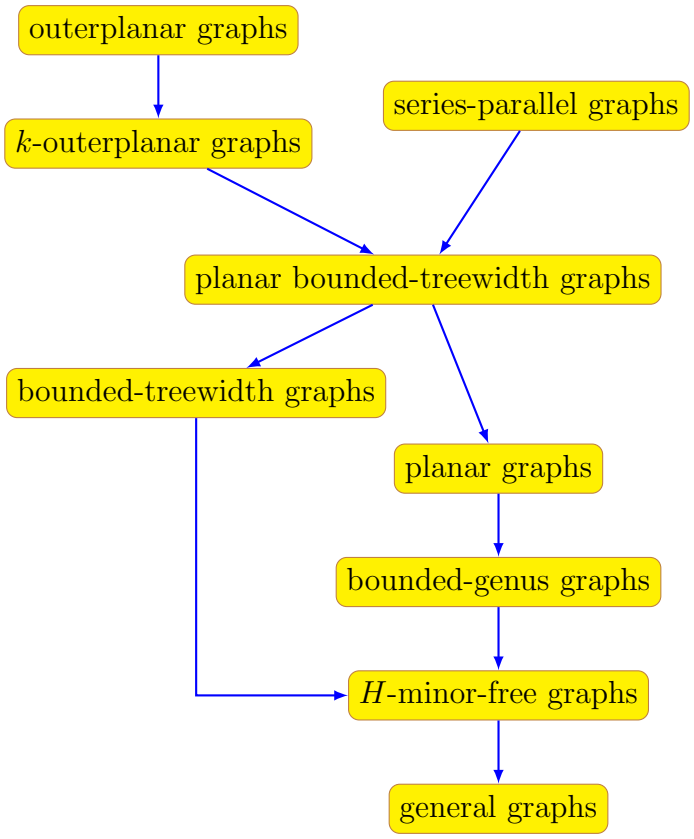


Figure 2.6: The relationship between different graph classes introduced. An arrow from class C_1 to C_2 denotes that any graph in class C_1 also belongs to class C_2.

a fixed minor H. All these graphs can be constructed by the "clique-sum" operation on smaller graphs embedded on a surface of small genus.

An analog of Corollary 2.3.7 exists for H-minor-free graphs. If $h = |V(H)|$, the number of edges in a simple H-minor-free graph is at most $O(nh\sqrt{\log h})$. As a result, H-minor-free graphs are guaranteed to have constant-degree vertices.

2.4 Problem definitions

2.4.1 Connectivity problems

Minimum Spanning Tree is perhaps one of the oldest network design problems. Given a weighted graph G, the goal is to find the minimum-length subgraph that spans all the vertices. Since cycles do not help in achieving more connectivity or reducing the length, we can assume that the solution is a tree. This problem is sometimes called MST for short.

Since all the problems discussed in this thesis involve finding a subgraph of minimum length, we usually choose to drop the identifier Minimum in the problem names for the brevity of notation.

In the Traveling Salesman Problem, or TSP for short, the goal is find a tour[3] of minimum length in the input graph that visits all the vertices, i.e., each vertex has to appear at least once in the tour. A similar problem is Stroll[4], which is sometimes referred to as Traveling Salesman Path Problem, where we seek to find a walk rather than a tour.

A generalization of spanning tree is that of the Steiner tree, where only some of the vertices in the graph are *terminals* while others, called *nonterminals* or *Steiner*

[3] *Tour* is the same as cycle, although it is usually used to emphasize that the cycle need not be simple.

[4] *Stroll* or *walk* is the same as path, although it is usually used to emphasize that the path need not be simple.

vertices, are not required to appear in the solution. However, their inclusion may help reduce the length. More formally, the input to Steiner Tree is a weighted graph $G(V, E)$ with a subset $R \subseteq V$ of required vertices, i.e., the terminals, and one is asked to find the minimum-length subgraph—subtree as per the discussion above—to span all the terminals. We sometimes think of a rooted variant of the problem where one particular vertex is called the *root*, to which all terminals should be connected. The rooted variant can be reduced to the nonrooted version since the root can be added to the set of terminals R. To solve the nonrooted version using the rooted variant, it suffices to pick one terminal as the root.

If each demand[5] consists of a pair of vertices that should be connected to each other, we have Steiner Forest. An instance with graph G and demand set $\mathcal{D} = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$ looks for a minimum-length subgraph in which every s_i, t_i are connected to each other, however, it does not matter whether s_i, t_j, for $i \neq j$, are connected. An equivalent definition is that we have different sets of terminals, and the terminals in each set should be connected (as in Steiner Tree). The main difference with Steiner Tree—that has in fact made the problem harder to tackle—is that the solution need not be connected; the solution is without loss of generality a forest, hence the name of the problem. Notice that, if we knew which demand pairs are connected to each other (in addition to the instance requirements), the instance could be reduced to Steiner Tree.

Group Steiner Tree generalizes both Steiner Tree and Set Cover. The input to this problem is a weighted graph G, a root vertex r, and a family $\{g_1, g_2, \ldots, g_k\}$ of sets of vertices called *groups*. The goal is to find a subgraph of minimum length that connected at least one element of each group g_i to the root r. While the case of singleton groups is equivalent to Steiner Tree, the setting when G is a star can model Set Cover (the leaves of the star are the sets of the Set

[5]Demands are sometimes called *requests* or *requirements*.

Cover instance).

In Facility Location[6], we are given a weighted graph G, a subset F of vertices called *facilities*, a subset C of vertices called *demands*, and an opening cost function $\phi : F \mapsto \mathbb{R}^+$. A solution specifies a subset F' of facilities that ideally minimizes the cost, defined as $\phi(F') + \sum_{j \in C} \mathsf{dist}_G(j, F')$; i.e., we "open" the subset F' of facilities, and then connect each client to its closest open facility. The cost has two portions: the facility cost and the connection cost.

Hard constraint on the number of demands served

In the above problems, a set of demands were given and had to be satisfied. If we "relax" this condition and require that at least k demands has to be met[7], we obtain a host of new problems: k-MST, k-Steiner Forest, and k-Median. The first two are simple extensions of Minimum Spanning Tree and Steiner Forest. The latter, k-Median, is almost an extension of Facility Location as explained below. Given a graph G and a set of clients C, k-Median seeks to find a set of at most k centers O such that $\sum_{j \in C} \mathsf{dist}_G(j, O)$ is minimized. It is similar to Facility Location except that, instead of the facility cost term, we have a hard constraint on the number of facilities we open.

Later in this section, we introduce prize-collecting versions of the above connectivity problems, and show that the hard-constrained variants just introduced are harder than the prize-collecting ones.

[6]In this thesis, all references to Facility Location, unless otherwise specified, are indeed to the problem that is more precisely called Uncapacitated Facility Location. In the Capacitated Facility Location , each facility has a capacity and each client has a demand; then, the total demand of the clients served by one facility cannot exceed the capacity of that server.

[7]Notice this is not really a relaxation since the problem becomes more general: letting k be the number of demands gives the original problem.

2.4.2 Prize-collecting framework

Prize-collecting problems involve situations, as described above, in which there are various demand points that desire to be included in some structure and we must find the structure of lowest cost to accomplish this. However, if some of the demand points are too expensive to include, then we can refuse to include them and instead pay a penalty. The most famous such problems, perhaps, are the Prize-collecting Steiner Tree (PCST), Prize-collecting Steiner Forest (PCSF), and Prize-collecting TSP (PCTSP) problems.

In particular, in the prize-collecting variant of a problem, the input is the same except that there is an additional "penalty" (or prize) function $\pi : \mathcal{D} \mapsto \mathbb{R}^+$. The cost of a structure (i.e., tree, path, cycle, etc. depending on the problem) is the sum of the length of the structure (i.e., the cost with respect to the original problem) and the penalty of demands not served by the structure. For instance, the solution to Prize-collecting Steiner Tree is a tree whose cost is the sum of the length of the tree and the penalty of the terminals it does not span. Prize-collecting TSP is defined similarly. For Prize-collecting Steiner Forest, the cost of the output forest is the sum of the length of the forest and the penalty of demands whose endpoints are not connected in the forest.

These problems are interesting for several reasons. In particular, prize-collecting Steiner network design problems are well-known network design problems with several applications in expanding telecommunications networks (see, e.g., [JMP00, SCRS00]), district heating networks (see, e.g., [LWP+06]), cost sharing, Lagrangian relaxation techniques (see, e.g., [JV01, CRW01]), and pricing scenarios (see, e.g., [CKR+03, FP03]).

When $\pi(v) = \infty$ for all $v \in V$, Prize-collecting TSP simplifies to the ordinary Traveling Salesman because we have no choice but to visit all nodes, and Prize-collecting Path (PC-Path) simplifies to Path-TSP, a.k.a. Min-Cost Hamil-

TONIAN PATH. Similarly, if penalty of each vertex is 0 or ∞, PRIZE-COLLECTING STEINER TREE simplifies to an instance of the ordinary STEINER TREE problem. In this case, the vertices $\{v \in V : \pi(v) = \infty\}$ are called *terminals*, which the Steiner tree *must* connect, and the vertices $\{v \in V : \pi_v = 0\}$ are called *Steiner* vertices, which the tree may use if it helps. These are all famous NP-hard problems [Kar72], so PRIZE-COLLECTING STEINER TREE, PRIZE-COLLECTING TSP, and PRIZE-COLLECTING PATH are NP-hard as well. Thus, we are interested in obtaining approximation algorithms for them.

When in addition all penalties are ∞ in these prize-collecting problems, we have the classic APX-hard problems STEINER TREE, TRAVELING SALESMAN, and STROLL for which the best approximation factors in order are 1.39 [BGRS10], $\frac{3}{2}$ [Chr76], and $\frac{3}{2}$ [Hoo91].

If PCTSP and PC-PATH look for *simple* cycles and paths, one typically assumes that the graph is complete and the edge costs satisfy the triangle inequality. Otherwise, the problem is inapproximable since determining whether the edges of cost zero include a Hamiltonian cycle or path is NP-hard. These assumptions open the door to algorithms that use shortcutting to convert an Eulerian tour (respectively, path) to a Hamiltonian cycle (respectively, path). In PCST, these extra assumptions have no effect. One alternate way to look at this extra assumption is to relax the problem, so that the tour may visit vertices multiple times. This is the assumption we take in this work since it has the advantage of making the definitions simpler for planar (or bounded-treewidth, etc.) versions of the problem where the graph is not complete.

In the *rooted* version of PCST and PCTSP, there is a specified root vertex r that must be spanned. The rooted and nonrooted versions are reducible to each other, while preserving approximation ratios. To use an algorithm for the nonrooted case to solve the rooted case, just set $\pi(r) = \infty$. To go the other direction, just try all possible roots. Thus, we consider only the rooted version for the rest of the paper. In

the case of PC-Path, there are actually three versions of the problem, PC-Path-0, PC-Path-1, and PC-Path-2, depending on whether we specify neither, one, or both endpoints of the path. By guessing an endpoint, one can use an algorithm for PC-Path-2 to solve PC-Path-1, or PC-Path-1 to solve PC-Path-0, but not the other way around.

We next mention two natural special and general forms of PCSF. In both cases, the form of the penalty function is changed or restricted. In the Submodular Prize-collecting Steiner Forest, the input is the same as PCSF except that, instead of a penalty function $\pi : \mathcal{D} \mapsto \mathbb{R}^+$, we are given a monotone submodular penalty function $\pi : 2^{\mathcal{D}} \mapsto \mathbb{R}^+$ that determines the penalty we need to pay if a subset of demands are not satisfied. This clearly generalizes PCSF since additive functions are submodular, however, it can model more realistic penalty functions, as submodular functions are natural models for utility functions.

An important special case of PCSF is Multiplicative Prize-collecting Steiner Forest in which the demand set consists of all vertex pairs, and the penalty function is $\pi(\{u, v\}) = \phi(u)\phi(v)$ for some fixed "potential" (or mass) $\phi(u)$ on vertices. This is inspired by Product Multi-Commodity Flow in [LR99, Bon04, KS02], and its applications in wireless networks [MSL08] or routing [CKS04, CKS05]. However, the main motivation is the setting when the weights (after some normalization) denote the probability of a vertex appearing in a set of active vertices, and a demand will be realized between each pair of active vertices.

The above definition is for the more interesting *symmetric* Multiplicative Prize-collecting Steiner Forest, however, a more general *asymmetric* is defined via the penalty function $\pi(\{u, v\}) = \phi_s(v)\phi_t(v)$ where we are given two (possibly) different mass functions ϕ_s, ϕ_t on vertices. This, for instance, models the scenario where the only valid demands are those between two disjoint sets A, B; i.e., there is a demand for a pair (u, v) if $u \in A, v \in B$.

Similarly to the hard-constraint settings for Steiner Tree, Steiner Forest, etc., we can define a fixed-prize version of Multiplicative Prize-collecting Steiner Forest. In Π-Multiplicative Prize-collecting Steiner Forest (Π-MPCSF), an additional parameter Π is provided such that any feasible solution has to collect a prize of at least Π, i.e., pay no more than $\pi(\mathcal{D}) - \Pi$ in penalty. We show implicitly in Section 10.5.4 that MPCSF is not easier than Π-MPCSF, however, the proof is not as simple as that of Lemma 2.4.1 because penalties for individual demands cannot be easily manipulated as required by the proof.

We study several prize-collecting and non-prize-collecting network design problems in planar graphs. Since these are natural instances in practice, obtaining better approximation ratios for them is highly valued. In particular, we look at Steiner Forest, PCST, sppctsp, PCSF as well as Submodular Prize-collecting Steiner Forest and Multiplicative Prize-collecting Steiner Forest in this setting.

Connection between connectivity problems

Here we discuss how the problems defined above are related. See Figure 2.7 for a diagram showing reductions between some of the problems. Most of the reductions take one instance of a problem and map it to an instance of the other; however, a few reductions take an instance of one problem, and solve it by invoking polynomially many instances of the harder problem. Below we explain those reductions that do not follow immediately from definitions, and have not been already discussed.

We already mentioned that PCSF and PCST generalize Steiner Forest and Steiner Tree, respectively. We now show that they are not harder than k-Steiner Forest and k-MST, respectively. In fact, we only give the formal proof for the case of PCSF and k-Steiner Forest, as the other proof is essentially the same.

Lemma 2.4.1. *An α-approximation algorithm for k-*Steiner Forest *gives an $\alpha(1+\epsilon)$-approximation algorithm for the* Prize-collecting Steiner Forest *problem,*

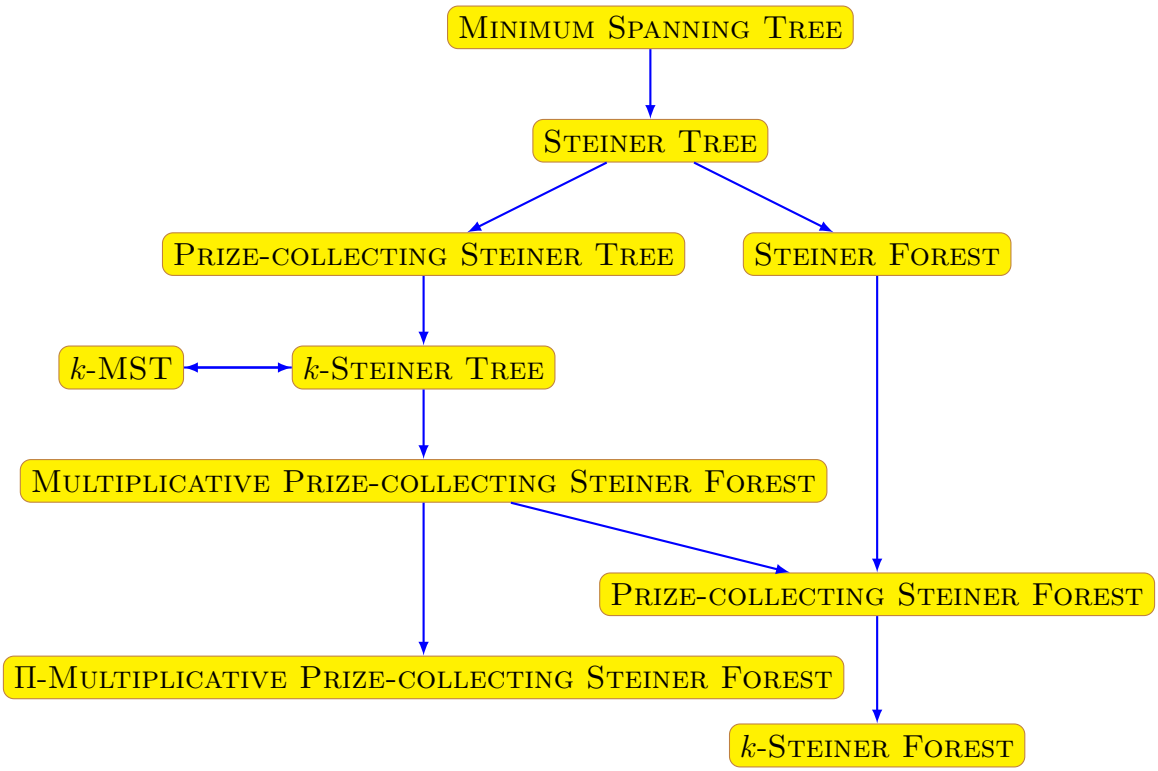


Figure 2.7: The relationship between some connectivity problems we discuss. An arrow from problem P_1 to P_2 denotes a reduction from P_1 to P_2.

for any constant $\epsilon > 0$.

Proof. We show how to approximate a PCSF instance $\mathcal{I}$ by invoking several (polynomially many) instances $\mathcal{I}'$ of k-Steiner Forest. Obtain an estimate ω for $\mathcal{I}$ such that $\frac{\omega}{3} \leq \mathsf{opt} \leq \omega$, using a 3-approximation algorithm for PCSF. Let $\pi(i)$ be the penalty of the pair i in $\mathcal{I}$, and let m denote the number of demands in $\mathcal{I}$. Let us assume, without loss of generality, that $\alpha \leq 3$ and $\pi(i) \leq 2\omega$ for any pair i. Let $\theta = \epsilon\omega/3m$. Place $p_i = \lfloor \frac{\pi(i)}{\theta} \rfloor$ copies of the pair i in $\mathcal{I}'$. Find an α-approximate solution to the resulting k-forest instance for every value of $0 \leq k \leq m'$, where m' is the number of pairs in $\mathcal{I}'$. Compute the cost for each of these solutions with respect to the PCSF objective, and report the best one.

We show that at least one of these candidate solutions is good. To do so, we first argue that for a careful choice of k, the optimum of the k-Steiner Forest instance is small. Next we show that for any α-approximate solution of that instance, the incurred penalty is bounded conveniently.

Let $\ell(\mathsf{opt})$ and $\pi(\mathsf{opt})$, respectively, be the length of the forest of and the penalty paid by the optimal solution opt. Suppose opt connects a subset of terminal pairs Q. Then, $\pi(\mathsf{opt}) = \sum_{i \notin Q} \pi(i)$. Focus on the candidate solution with $k = \sum_{i \in Q} \lfloor \pi(i)/\theta \rfloor$. The optimum of the corresponding k-Steiner Forest instance is at most $\ell(\mathsf{opt})$ because one possible solution is that of connecting the copies of Q.

Now we have an α-approximate solution to the k-Steiner Forest instance with k as defined above. To compute the PCSF cost corresponding to this solution, we add the penalty of pairs that are not connected in this forest. We can assume, for each demand pair in the PCSF instance, either all or none of its copies are connected. The number of (k-Steiner Forest) pairs not connected is at most $m' - k$, and their

penalties sum to no more than

$$\begin{aligned}
\sum_{i \text{ not connected}} \pi(i) &\leq \sum_{i \text{ not connected}} (p_i + 1)\theta \\
&\leq \left(\sum_{i \text{ not connected}} p_i\theta \right) + m\theta \\
&\leq (m' - k)\theta + m\theta \\
&\leq \pi(\mathsf{opt}) + m\theta \\
&= \pi(\mathsf{opt}) + \epsilon\omega/3 \\
&\leq \pi(\mathsf{opt}) + \epsilon\,\mathsf{opt}.
\end{aligned}$$

Thus, the PCSF cost of the best candidate solution is at most $\alpha\ell(\mathsf{opt}) + \pi(\mathsf{opt}) + \epsilon\mathsf{opt} \leq \alpha(1+\epsilon)\mathsf{opt}$.

It remains to show that the instances $\mathcal{I}'$ have polynomial size. Since $\pi(i) \leq 2\omega$, each pair i will have $p_i \leq 6m\epsilon^{-1}$ copies. Hence, $\mathcal{I}'$ has polynomial size and we can use the approximation algorithm for the k-Steiner Forest. □

The above proof, mutatis mutandis, shows that k-Steiner Tree is harder than Prize-collecting Steiner Tree. A similar trick shows that k-MST and k-Steiner Tree are equivalent. One direction is trivial by definition, hence we only need to argue that k-Steiner Tree can be reduced to k-MST. Suppose we have an instance of k-Steiner Tree with n vertices. Replace each terminal with n^2 vertices at distance zero from each other. Treat this as a k'-MST instance with $k' = kn^2$. Clearly the original solution for k-Steiner Tree carries over with the same cost. Notice that any solution of the new instance has to span at least k terminals from the original instance since $(k-1)n^2 + n < kn^2$.

Next we show that (even the symmetric) Π-MPCSF is a generalization of the rooted k-MST problem (for which the best known approximation guarantee is 2). Suppose we are given an instance $\mathcal{I}$ of rooted k-MST. It consists of a weighted graph

$G(V, E)$, a root vertex r and a number k. Suppose r is not to be counted among the k vertices. We build the new instance $\mathcal{I}'$ of Π-MPCSF as follows. The graph G' is the same as G. The weights of all vertices are one, except for r whose weight is n^2. Then, the goal will be to find the minimum-length forest that gathers a prize of at least $\Pi = (n^2 + k)^2 = n^4 + 2n^2k + k^2$.

Theorem 2.4.2. *The instance $\mathcal{I}$ of rooted k-*MST *is equivalent to the instance $\mathcal{I}'$ of* Π-MPCSF*.*

Proof. Obviously, any tree connecting k vertices to the root is translated to a forest that collects a prize of at least Π: let each vertex not spanned by the tree be a singleton component in the forest.

Finally, we claim that any solution of prize Π or higher translates to a tree spanning at least k vertices in the original instance. The resulting tree is just the component of the forest containing the root vertex. Suppose for the sake of reaching a contradiction that the component spans $k' < k$ nonroot vertices. The total prize collected is at most

$$\begin{aligned}(n^2 + k')^2 + (n - k' - 1)^2 &< n^4 + 2n^2k' + k'^2 + n^2 \\ &= \Pi + 2n^2(k' - k) + k'^2 + n^2 - k^2 \\ &< \Pi + 2n^2(k' - k + 1) \\ &\leq \Pi,\end{aligned}$$

yielding a contradiction, and proving the supposition is false. □

Chapter 3

Thesis Organization and Contributions

Chapter 2 goes over the terminology and the definition of the problems. Part II explains the techniques we use to derive the algorithms. Part III discusses in detail several applications of the techniques to planar and nonplanar prize-collecting Steiner network problems. Below we give a high-level summary of the techniques, and a summary of the major results. The secondary results as well as more details can be found in the appropriate chapters.

3.1 Techniques

We give a short of summary of each chapter of Part II.

3.1.1 Prize-collecting clustering

Chapter 4 introduces the main technique developed in this thesis. The technique is used in all the applications later on. It essentially defines a paradigm for clustering vertices of a graph such that the connections required inside each cluster is within

the total budget of the vertices in there. On the other hand, if some vertices fall into different clusters, then their distance should be large compared to their budgets.

We start the chapter by introducing `PC-MoatGrowing`: a primal-dual procedure that is the crux of Steiner Forest and Prize-collecting Steiner Tree algorithm [AKR95, GW95]. We then state and prove two concrete theorems in the *prize-collecting clustering* paradigm: the "classification theorem" (Theorem 4.2.1) and the "superclustering theorem" (Theorem 4.3.1). The chapter concludes with an extension of `PC-MoatGrowing` to submodular budgets, called `SubmodPC-Cluster`.

3.1.2 Spanner framework

In Chapter 5 we summarize some of the previous work (e.g., [Bak94, AGK+98, Epp00, Kle06, DHM07, Kle08, BKM09, DHK11]) that provides a framework for obtaining good approximation algorithms on special classes of graphs. In particular, this framework allows us, provided that we can carry out certain tasks, to reduce a planar instance of the problem to one that has small treewidth. Assuming that we can tackle the bounded-treewidth case easily (say, it is in P or admits a PTAS), we obtain a PTAS for the planar case. We define the notion of "spanner" as a subgraph of small length (i.e., $O(\mathsf{opt})$) that contains a near-optimal solution. Then, depending on the problem, we delete or contract a small subset of edges in the spanner in order to find the simpler instance.

3.1.3 Spanner construction

Chapter 6 reviews a spanner construction procedure due to Borradaile, Klein, and Mathieu [BKM09] that is based on a novel *brick decomposition* procedure. A subtree of a planar graph can be expanded to a subgraph, called the *mortar graph*, that partitions the graph into a set of "bricks." The structure of each brick allows us to break the problem into that of a subproblem (slightly different than the original)

inside the bricks.

3.1.4 Granularization

Next in Chapter 7, we formalize and extend a standard technique to deal with large integer or real numbers in order to obtain a PTAS from a pseudopolynomial-time algorithm. This is done by rounding up or down the real (or large integer) values to the next multipel of a sufficiently small precision unit. Then, the range of values falls into a small range, and a pseudopolynomial-time algorithm indeed runs in polynomial time. The analysis should bound the amount of error introduced due to rounding, and show that it is small.

3.2 Results

3.2.1 Prize-collecting problems in general graphs

Chapter 8 studies Prize-collecting Steiner Tree, Prize-collecting TSP, and Prize-collecting Stroll. We prove the following theorem, improving the approximation ratios for these problems after 17 years.

Theorem 3.2.1. Prize-collecting Steiner Tree*,* Prize-collecting TSP*, and* Prize-collecting Stroll *admit approximation algorithms with guarantees* $2 - c$ *for some fixed positive constant* c*.*

In fact, we show how to use Theorem 4.3.1 to bypass the integrality gap of a natural linear-programming relaxation for these problem—which is 2—that had proved to be a major barrier for this task.

3.2.2 Steiner Forest on planar graphs

In Chapter 9, we look at planar Steiner Forest, and resolve the complexity of the problem for a large range of graphs. For graphs of treewidth two, we show that the algorithm is in P.

Theorem 3.2.2. Steiner Forest *can be solved in polynomial time on series-parallel graphs.*

The above result is complemented by a proof that the treewidth of two is a limit.

Theorem 3.2.3. Steiner Forest *is NP-hard for planar graphs of treewidth at most three.*

Although the problem is NP-hard, we show how to obtain a PTAS when the treewidth is a constant. The algorithm is quite nontrivial compared to usual dynamic-programming algorithms for bounded-treewidth problems. We need a structural property to simplify the solution set we want to search in, so that we can find the optimal one using a DP approach, but is, on the other hand, so rich to contain a near-optimal solution.

Theorem 3.2.4. Steiner Forest *admits a PTAS on graphs of bounded treewidth.*

This is a rare instance of a problem whose complexity changes when the treewidth changes from two to three.

Finally, we use Theorem 4.2.1 in conjunction with the spanner framework of Chapter 5 and the brick-decomposition-based spanner construction of Chapter 6 to present a PTAS for planar (and, more generally, bounded-genus) case of Steiner Forest.

Theorem 3.2.5. Steiner Forest *admits a PTAS on bounded-genus graphs.*

This result generalizes PTASes for Euclidean Steiner Tree [Aro98, Mit99], planar Steiner Tree [BKM09], and Euclidean Steiner Forest [BKM08].

3.2.3 Planar Prize-collecting Steiner Forest

Chapter 10 consider several prize-collecting Steiner network problems in planar graphs. We first show that, surprisingly, these problems tend to be hard on very simple metrics.

Theorem 3.2.6. PCSF *is APX-hard on (1) planar graphs of treewidth two, and on (2) the two-dimensional Euclidean metric.*

The reduced instance in the proof has a very simple structure. The reader is referred to the discussion in Section 10.3 to find about the instance and its implications for similar problems.

The hardness for Euclidean metric answers an open question raised in [BH10]. It also provides the first provable separation between the complexity of a natural optimization problem and that of its prize-collecting version: Euclidean Steiner Forest admits a PTAS [BKM08], but Euclidean PCSF is APX-hard.

On the positive side, we present a reduction from bounded-genus to bounded-treewidth instances of large classes of prize-collecting problems.

Theorem 3.2.7. *For any given constant $\epsilon > 0$, an α-approximation algorithm for* SPCSF *on graphs of bounded treewidth implies an $(\alpha + \epsilon)$-approximation algorithm for* SPCSF *on bounded-genus graphs. The α-approximation algorithm is run on a graph with the same vertex set and the same penalty function. The result holds for* SPCTSP *and* SPCS*, too.*

Although we do not know of an approximation guarantee better than 2.54 for bounded-treewidth SPCSF, we can obtain a PTAS for the case of multiplicative prizes.

Theorem 3.2.8. MPCSF *admits a PTAS on bounded-treewidth graphs.*

The above result immediately implies a PTAS for the bounded-genus version using Theorem 3.2.7.

3.3 Credits

Chapter 4 is an abstraction of the prize-collecting clustering technique that I gradually developed with my coauthors [ABHK11, BHM10a, BHM10b]. It goes over the moat-growing procedure due to [AKR95, GW95] before explaining the prize-collecting clustering theorems. Chapters 5, 6 overview the relevant literature. Chapter 7 is an abstraction of the ideas I developed in joint works with Hajiaghayi [BH09a, BH10].

Chapter 8 is based on a joint work with Archer, Hajiaghayi, and Karloff [ABHK09, ABHK11]. Chapter 9 is based on a joint work with Hajiaghayi and Marx [BHM10a]. Chapter 10 is based on a joint work with Hajiaghayi and Marx [BHM10b] and a joint work with Hajiaghayi [BH10]. The results of Section 10.5 borrow ideas from [BH10], however, the results in this form appear in this thesis for the first time.

During my Ph.D. studies, I have also worked on other problems that are not closely related to the topic of this thesis. These include network optimization [BH09a, BGHS09, BGHS10, BH09b, BC10, BHKM11], fair allocation [BCG09a, BCG09b], bargaining games [BHIM10], submodular secretary problem [BHZ10], database scheduling [BGHK09], and pricing mechanisms [BHJP11].

Part II

Techniques

Chapter 4

Prize-collecting Clustering

Clustering is a technique in unsupervised learning or statistical data analysis to classify a set of items into different groups, called *clusters* [JMF99]. The items that are placed in one cluster are supposed to be similar to each other. Clustering has been used in a wide range of fields such as social network analysis [SC08, MSST07], machine learning [LCFX07, YYH03, YLZ06, BC04], image analysis [JF96], pattern recognition [And73], information retrieval [Ras92, Sal91], and other fields [JD88].

We may have a tendency to minimize the number of clusters, which is particularly helpful in terms of efficiency if some operation is to be performed on each cluster. There may be an upper bound, often called the *diameter*, on how different two items in one cluster can be. The diameter may be dynamic in the sense that it is not the same for different clusters.

The notion of similarity of items in a cluster is a problem-specific notion. The simplest case is when items are already associated with "types," and items of the same type should be grouped together. There may be "features" that the items share, and two items with significant number of common features can belong to the same cluster. However, more general cases are those in which a metric function describes the distance—i.e., dissimilarity—between any two items. The metric can

be the Euclidean metric, the ℓ_p distance function, or the shortest-path metric defined on a weighted graph.

In each case, the items put in one cluster enjoy some structure that can later be exploited. For instance, there can be a geometric structure, say, for separating items of different clusters. We usually think of clustering as a tool that initiates a divide-and-conquer strategy. From this perspective, we desire a small number of clusters, and certain structural properties associated with each cluster.

Next we give a flavor of the clustering paradigm we are interested in, followed by two concrete theorems. Before stating and proving Theorems 4.2.1 and 4.3.1, we give in Section 4.1 an overview of a classic moat-growing procedure that is crucial in the proof of both theorems. This procedure is later extend (to submodular budgets) in Section 4.4.

Think of a scenario in which several radio transmitters are to be clustered according to their proximity. Were all the transmitters of the same type, they would have the same power, and could serve the same radius. However, if they have different powers, each can reach to a different distance. This introduces an element of asymmetry in the clustering.

Now consider a setting where different towns should join each other in some regional divisions. Each division should build a road network connecting all its towns to each other. Since each division will form an autonomous entity, the funding for the project in a division has to come from the towns in it; each town has a particular budget. Then, the goal is to find the divisions and decide what roads to construct in each. Moreover, certain maximality condition needs to be met so that, roughly speaking, the number of divisions is minimized.

4.1 Moat-growing procedure

The moat-growing algorithm was first proposed by Agrawal, Klein, and Ravi [AKR95] to give a 2-approximation algorithm for Steiner Forest. Goemans and Williamson [GW95] simplified the exposition and the analysis, although using essentially the same procedure, to obtain 2-approximation algorithms for *constrained forest problems* as well as for Prize-collecting Steiner Tree and Prize-collecting TSP. One major difference is that the original algorithm enjoys a *lazy buying* scheme, whereas the latter buys many edges in the "growth" phase and later on "prunes" the unnecessary ones.

We give a brief overview of the algorithm for Prize-collecting Steiner Tree and its concepts since they serve as the core of the algorithms used to prove the prize-collecting clustering theorems. More details can be found in [GW95].

Consider an instance of Prize-collecting Steiner Tree with the graph $G(V, E)$, edge-length function $\ell : E \mapsto \mathbb{R}^+$, and penalty function $\pi : V \mapsto \mathbb{R}^+$. The natural linear-programming relaxation—see LP (4.1)—is written for the rooted version of the problem, where a special vertex $r \in V$ has to be included in the final solution. Recall that rooted and nonrooted variants are equivalent. For simplicity of notation, we assume $\pi(r) = \infty$ even in the rooted version.

$$\text{minimize} \quad \sum_{e \in E} \ell(e) x_e + \sum_{v \in V} \pi_v z_v \tag{4.1}$$

$$\text{subject to} \quad \sum_{e \in \delta(S)} x_e \geq 1 - z_v \qquad \forall v \in S \subseteq V \setminus \{r\} \tag{4.2}$$

$$x_e \geq 0 \qquad \forall e \in E \tag{4.3}$$

$$z_v \geq 0 \qquad \forall v \in V, \tag{4.4}$$

where x_e denotes whether an edge e is included in the solution, and z_v signifies whether

a vertex v is left out. Hence, the solution needs to include at least one edge leaving any set $S \subseteq V \setminus \{r\}$ containing a vertex v with $z_v = 0$.

Had integral constraints $x_e, z_v \in \{0, 1\}$ been substituted in place of (4.3) and (4.4), the above formulation would have exactly matched the optimum of Prize-collecting Steiner Tree. However, the relaxation has an integrality gap approaching to 2, even for the special case of Steiner Tree, i.e., when $\pi_v \in \{0, \infty\}$ [GW95]; we discuss the integrality gap and some of its implications in Section 8.3. The following is essentially the dual of the above program.

$$\text{maximize} \quad \sum_{v \in S \subseteq V \setminus \{r\}} y_{S,v} \tag{4.5}$$

$$\text{subject to} \quad \sum_{\substack{S \subseteq V \\ e \in \delta(S)}} \sum_{v \in S} \leq \ell(e) \qquad \forall e \in E \tag{4.6}$$

$$\sum_{v \in S \subseteq V} y_{S,v} \leq \pi(v) \qquad \forall v \in V \tag{4.7}$$

$$y_{S,v} \geq 0 \qquad \forall v \in S \subseteq V. \tag{4.8}$$

Notice that we have included additional variables $y_{S,v}$ for sets S containing the root. These variables do not contribute to the objective function, though, and, as a result, can be set to zero in any optimal solution. However, the primal-dual algorithm may assign nonzero values to some of them. To simplify the presentation, we often use the shorthand $y_S := \sum_{v \in S} y_{S,v}$. We sometimes refer to constraints (4.6), (4.6) as edge packing and penalty packing constraints, respectively.

The algorithm `GW`, as well as the procedures used to prove Theorems 4.2.1 and 4.3.1, has two phases. In the first phase, *moats* grow around vertices—resulting in increase of dual variables—and consume their budgets. As moats grow, some edges—i.e., those whose dual constraints become tight—are bought and the connectivity in the solution increases. Roughly speaking, this continues until the budgets run out,

or no more connectivity is possible. A pruning phase follows that is different for Steiner Forest and Prize-collecting Steiner Tree. (We introduce our own pruning rules in the proofs to come.) In general, edges that are not "necessary" will be removed.

The analysis then seeks to charge the cost of the final solution to twice the sum of the dual variables (only those thereof that appear in the objective function), which by weak duality—see Theorem 2.2.1—is at most the optimum of LP (4.1), which is in turn no more than `opt`. The charging is done by apportioning the cost of the solution over "time," and then showing the rate of increase in solution cost at each point in time is at most twice the rate of increase in the sum of dual variables. The argument, after much preparatory work, boils down to the fact that the average degree of a forest is at most two.

In what follows, we discuss the growth phase in more detail (`PC-MoatGrowing`); however, we leave the discussion of the pruning phase to where it is used since the very pruning procedure employed depends on the theorem we aim to prove, and, although similar in nature, is different from that of Prize-collecting Steiner Tree.

`PC-MoatGrowing` takes as input a graph $G(V, E)$ and a penalty function π (sometimes called the "budget" or "potential"). It produces a feasible solution y to dual LP (4.5), and a forest F. The forest is empty at the beginning, and y is initialized to a zero vector. We maintain a partition $\mathcal{C}$ of vertices V into clusters; it initially consists of singleton sets. Each cluster is either *active* or *inactive*; the cluster $C \in \mathcal{C}$ is *active* if and only if $\sum_{C' \subseteq C} \sum_{v \in C'} y_{C',v} < \sum_{v \in C} \pi_v$. A vertex v is *alive* if and only if $\sum_{C \ni v} y_{C,v} < \pi_v$. A vertex that is not alive is called *dead*. Equivalently, a cluster $C \in \mathcal{C}$ is active if and only if there is a live vertex $v \in C$. We simultaneously *grow* all the active clusters by η. In particular, if there are $\kappa(C) > 0$ live vertices in an active cluster C, we increase $y_{C,v}$ by $\eta/\kappa(C)$ for each live vertex $v \in C$. Hence, y_C defined as $\sum_{v \in C} y_{C,v}$ increases by η for an active cluster C. We pick the largest value for η that

does not violate any of the constraints in (4.6) or (4.7). Except possibly for the last step of `PC-MoatGrowing`, η will be finite. (In the last step, a cluster consisting of all the vertices may experience an infinite growth.) Hence, at least one such constraint goes tight after each growth step (except possibly for the last one). If this happens for an edge constraint for $e = (u, v)$, then there are two clusters $C_u \ni u$ and $C_v \ni v$ in $\mathcal{C}$; at least one of the two is growing. (In this case, we may say that the edge e is now tight.) We merge the two clusters into $C = C_u \cup C_v$ by adding the edge e to F, remove the old clusters, and add the new one to $\mathcal{C}$. Nothing needs to be done if a constraint (4.7) becomes tight. The number of iterations is at most $2|V|$ because at each event point either a vertex dies, or the size of $\mathcal{C}$ decreases.

We can think of the growth stage as a process that paints portions of the edges of the graph. This gives a better intuition into the algorithm, and makes several lemmas in the proof intuitively simple. Consider a topological structure in which vertices of the graph are represented by points, and each edge is a curve connecting its endpoints whose length is equal to the weight of the edge. Suppose a cluster C is growing by an amount η. Recall that this is distributed among all the live vertices $v \in C$ where $y_{C,v}$ is increased by $\eta' := \eta/\kappa(C)$. As a result, we paint by color v a connected portion with length η' of all the edges in $\delta(C)$. Finally, each edge e gets exactly $\sum_{C:e\in\delta(C)} y_{C,v}$ units of color v. Although it is not necessary for any of the proofs, we can perform a clean-up procedure such that all the portions of color v are consecutive on an edge.[1] Hence, as a cluster expands, it paints its boundary by the amount of growth. At the time when two clusters merge, their colors barely touch each other. At each point in time, the colors associated with the vertices of a cluster form a connected region. Algorithm 4.1 outlines `PC-MoatGrowing`.

[1] This can also be achieved without a clean-up procedure if we perform the coloring in a lazy manner. That is, we do not do the actual color assignment until the edge goes tight or the algorithm terminates. At this point, we go about putting colors on the edges, and we make sure the color corresponding to any pair (S, v) forms a consecutive portion of the edge. This property is not required as part of our algorithm, though, and is merely for the sake of having a nice coloring which is of independent interest.

Algorithm 4.1 `PC-MoatGrowing` (G, π)

Input: graph $G(V, E)$, and budgets $\phi_v \geq 0$
Output: forest F, dual LP solution y, and laminar family of clusters $\mathcal{S}$

$F \leftarrow \emptyset$
$y_{S,v} \leftarrow 0$ for any $v \in S \subseteq V$
$\mathcal{S} \leftarrow \mathcal{C} \leftarrow \{\{v\} : v \in V\}$
while there is a live vertex **and** $|\mathcal{C}| > 1$ **do**
  Let η be the largest possible value such that simultaneously increasing y_C by η for all active clusters C does not violate Constraints (4.6)-(4.8)
  $y_{\mathcal{C}(v),v} \leftarrow y_{\mathcal{C}(v),v} + \frac{\eta}{\kappa(\mathcal{C}(v))}$ for all live vertices v
  if $\exists e \in E$ that is tight and connects two clusters **then**
    Pick one such edge $e = (u, v)$
    $F \leftarrow F \cup \{e\}$
    $C \leftarrow \mathcal{C}(u) \cup \mathcal{C}(v)$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\} \setminus \{\mathcal{C}(u), \mathcal{C}(v)\}$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$
$y_{V,u} \leftarrow \phi(u) - \sum_{S \subset V} y_{S,u}$ # no new connectivity
return $(F, y, \mathcal{S})$

The last step of the procedure ensures the budget of every vertex is fully used. If there is any vertex with residual budget, F spans all vertices, and $y_V = 0$. We assign the residual budget of every vertex to this cluster, whose growing does not have any effect on the connectivity of F. Since there is no edge in $\delta(V) = \emptyset$, we are never going to charge any cost to these variables. Although this last step ensures that V does not have any remaining budget, we do not consider V to be inactive when defining $\mathcal{B}$.

4.1.1 Implementation and further discussion

Although there are exponentially many variables in the LP, only a polynomial number of them will take nonzero values during the algorithm, and the algorithm runs in polynomial time.

To implement `PC-MoatGrowing`, all we need to do is compute the time at which each active cluster would die and the time at which each edge would go tight, assuming no other event occurs first. We then process the earliest of these events, update the appropriate time forecasts in our priority queue, and repeat. If two or more distinct

events are set to occur at the same time, then we can arbitrarily choose which one to go first. Goemans and Williamson showed how to implement `PC-MoatGrowing` in $O(n^2 \log n)$ time [GW95], which Gabow and Pettie later improved to $O(n^2)$ [GP02].

The purpose of the two types of events is to prevent the penalty and edge packing constraints from being violated. Thus, by construction, the vector of dual variables forms a feasible solution to LP (4.5) at every moment during `PC-MoatGrowing`.

Although the tie-breaking is arbitrary when two events are set to occur at the same time, notice that processing one event may cause one of the other tied events to be deleted from the queue, thereby preventing it from occurring at that time (or perhaps ever). For instance, if the edge packing constraints for two distinct edges between clusters C_1 and C_2 become tight simultaneously, then processing one of them will merge the components, thus preventing the other event from occurring. Therefore, F remains a forest. Similarly, recall that we trigger a tight edge event on e only if e connects two distinct clusters C_1 and C_2 and at least one of them is active. The reason is that otherwise there is no immediate danger of violating the edge packing constraint. Thus, if C_1 is already dead and C_2 is set to die at the same moment that e goes tight, then choosing to first process the death of C_2 prevents the tight edge event on e from occurring at this time, although it could occur later if an active component merges with C_1 or C_2. Similarly, processing the tight edge event first would prevent C_2 from dying because it would already have merged with C_1 (although in this case, $C_1 \cup C_2$ would be about to die because $P_{C_1 \cup C_2} = 0$).

It is helpful to visualize `PC-MoatGrowing` as a geometric process where moats of water encircling clusters of vertices expand over time. The dual variable y_C represents the width of the moat encircling cluster C. As y_C increases, the moat expands across the edges $\delta(C)$ that cross the moat. As soon as the moats have collectively expanded across an edge (possibly from both directions, meeting somewhere in the middle), the edge packing constraint becomes tight and it triggers a tight edge event, putting that

edge into F. This geometric view of the dual variables as moats was suggested by Jünger and Pulleyblank [JP95], predating `GW`. The potential $P_C = \phi(C) - \sum_{S \subseteq C} y_S$ corresponds to the height of a reservoir that shrinks at the same rate that the width of the surrounding moat expands.

4.1.2 Goemans-Williamson's algorithm for PCST

The algorithm `GW` consists of two phases. The first phase is `PC-MoatGrowing` described above. We now explain its pruning phase. The set of edges that `PC-MoatGrowing` includes in F may be very expensive, so the pruning phase throws some of them away. First, it discards all trees in the forest F aside from the one containing r. These are precisely the dead clusters that still existed when `PC-MoatGrowing` terminated. Next, we consider all clusters that existed at any point during `PC-MoatGrowing`, and we color black any cluster that died (whether or not it later merged with an active cluster). Now, we iteratively perform the following pruning step. While there exists a black cluster C such that $|F \cap \delta(C)| = 1$, delete all edges from F that are incident to any vertex of C (i.e., the edges in $\delta(C)$, plus all those internal to C). Once there are no more such black clusters left to prune, `GW` returns F. This concludes the description of `GW`.

Algorithm 4.2 `GW` (G, π, r)

Input: graph $G(V, E)$, root $r \in V$, and penalties $\pi(v) \geq 0$ such that $\pi(r) = \infty$
Output: tree T

1: $(F, y, \mathcal{S}) \leftarrow$ `PC-MoatGrowing`(G, π)
2: $\mathcal{B} \leftarrow \left\{ S \in \mathcal{S} \setminus \{V\} \mid \sum_{S' \subseteq S} \sum_{v \in S'} y_{S',v} = \sum_{v \in S} \phi_v \right\}$
3: **while** $\exists S \in \mathcal{B}$ such that $|F \cap \delta(S)| = 1$ **do**
4: $\quad F \leftarrow F \setminus \{\text{all edges incident on } S\}$
5: $T \leftarrow$ the component of F containing r
6: **return** T

Clearly, none of these pruning operations disconnects F since that would require

$|F \cap \delta(C)| \geq 2$. Moreover, none prunes away r since the root component never dies in `PC-MoatGrowing`. Hence, the final returned set F is a tree that includes r. Refer to [GW95] for a proof of why this is a 2-approximate solution for PCST.

As a historical note, the moat-growing procedure originally given by Goemans and Williamson [GW95] treats the root vertex differently from others, in that any component containing the root does not grow at all. The nonrooted growth explained above, that avoids guessing the root vertex, is due to Johnson, Minkoff, and Phillips [JMP00]; however, a minor error in their analysis was fixed later by Feofiloff et al. [FFFdP10]. Our exposition (of the moat-growing and pruning phases) matches that of the latter work. Our proofs heavily rely on the fact that all vertices are treated the same, i.e., there is no special root vertex.

4.2 Classification theorem

We start with the first prize-collecting clustering theorem. Given a weighted graph with budgets on vertices, this theorem classifies the vertices of a graph into clusters such that vertices of each cluster can independently of others build a tree among themselves, using only twice their own budget. In addition, for any two (or more) vertices classified into different clusters (barring some technical restrictions), we can argue that placing them in the same cluster, while satisfying the above condition, is impossible.

We first introduced this theorem in [BHM10a] to tackle Steiner Forest on planar and bounded-genus graphs; see Chapter 9. Subsequently, it was invoked in [BCE+11] to study Prize-collecting Steiner Forest on planar graphs—see Chapter 10—and in [BHKM11] to present a PTAS for Planar Multiway Cut. In all these instances, the theorem serves as a tool to initiate a divide-and-conquer strategy.

Theorem 4.2.1 (Classification)**.** *Let $G(V, E)$ be a graph with nonnegative lengths $\ell : E \mapsto \mathbb{R}^+$ on edges, and a budget $\phi : V \mapsto \mathbb{R}^+$ for each vertex. In polynomial time, we can find a subgraph Z such that the following hold.*

1. *The length of Z is at most $2\phi(V)$; in particular, $\ell(T) \leq 2\phi(V(T))$ for any connected component T of Z.*

2. *For any subgraph L of G, there is a set Q of vertices such that*

   (a) *$\phi(Q)$ is at most the length of L, and*

   (b) *if two vertices $v_1, v_2 \notin Q$ are connected by L, then they are in the same component of Z.*

We shortly describe an algorithm—see `PC-Classify` below—that is the underlying procedure in the proof of Theorem 4.2.1. The algorithm consists of the nonrooted growth procedure `PC-MoatGrowing` followed by a special pruning phase. The analysis bears similarities to the primal-dual method due to [AKR95, GW95], yet we strengthen the previous approaches by proving local guarantees (instead of the global guarantee provided in previous algorithms).

Algorithm 4.3 `PC-Classify` (G, ϕ)

Input: graph $G(V, E)$, and budgets $\phi(v) \geq 0$
Output: forest F

1: $(F, y, \mathcal{S}) \leftarrow$ `PC-MoatGrowing`(G, ϕ)
2: $F' \leftarrow F$ # only used in the analysis
3: $\mathcal{B} \leftarrow \left\{ S \in \mathcal{S} \setminus \{V\} \mid \sum_{S' \subseteq S} \sum_{v \in S'} y_{S',v} = \sum_{v \in S} \phi_v \right\}$
4: **while** $\exists S \in \mathcal{B}$ such that $F \cap \delta(S) = \{e\}$ for an edge e **do**
5: $\quad F \leftarrow F \setminus \{e\}$
6: **return** F

Pruning rule for `PC-Classify` Let $\mathcal{S}$ contain every set that is a cluster at some point during the execution of `PC-MoatGrowing`. It can be easily observed that the

clusters $\mathcal{S}$ are laminar and the maximal clusters are those in $\mathcal{C}$. In addition, notice that $F[C]$ is connected for each $C \in \mathcal{S}$.

Let $\mathcal{B} \subseteq \mathcal{S} \setminus \{V\}$ be the set of all such clusters that are tight, namely, for each $C \in \mathcal{B}$, we have $\sum_{S \subseteq C} \sum_{v \in S} y_{S,v} = \sum_{v \in C} \phi_v$. In the pruning phase, we iteratively remove some edges from F. More specifically, as long as there is a cluster $C \in \mathcal{B}$ such that $F \cap \delta(C) = \{e\}$, we remove the edge e from F. (In contrast, the pruning rule for `GW` would remove all edges incident on $V(C)$, not only the edge e leaving C.)

A cluster C is called a *pruned cluster* if it is pruned in the second phase in which case we have $\delta(C) \cap F = \emptyset$. We argue that a pruned cluster cannot have nonempty and proper intersection with a connected component of F. Notice that, at the time the cluster C is pruned, the single remaining edge of $F \cap \delta(C)$ is removed from F, thus the final F cannot have any edge in $\delta(C)$. Therefore, no connected component of F can have two vertices such that one is inside C and the other is not.

We first bound the length of the final forest F output by `PC-Classify`. The following lemma is similar to the analysis of the algorithm in [GW95]. However, we do not have a primal LP to give a bound on the dual. Rather, the upper bound for the length is the sum of all the budget $\phi(V)$. In addition, we bound the length of a forest F that may have more than one connected component, whereas the pruning rule for Prize-collecting Steiner Tree algorithm of [GW95] guarantees a connected graph at the end.

Lemma 4.2.2. *The length of F is at most $2\phi(V)$. In particular, the length of each connected component of F can be charged to twice the budget of vertices inside it.*

Proof. The strategy for establishing the first part of the lemma is to prove that the length of the forest F is at most $2 \sum_{v \in S \subseteq V} y_{S,v} \leq 2 \sum_{v \in V} \phi(v)$, where the inequality follows from Equation (4.7). To obtain the stronger claim in the statement, we argue that the variables $y_{S,v}$ with $v \in V(T)$ suffice to account for the length of a connected component T.

Recall that the growth phase has several events corresponding to an edge or set constraint going tight. Conceptually, it is helpful to think of `PC-MoatGrowing` as progressing continuously over time, although the actual computation is confined to a sequence of discrete *event points*. Time begins at 0 and unfolds in *epochs*, which are the intervals of time between two consecutive event points (possibly an empty interval if two event points occur simultaneously). Each active cluster C increases its y_C value at rate 1 for the duration of the epoch, while all other dual variables remain unchanged. We first break apart y variables by epoch. Let t_j be the time at which the j^{th} event point occurs in `PC-MoatGrowing` ($0 = t_0 \leq t_1 \leq t_2 \leq \cdots$), so the j^{th} epoch is the interval of time from t_{j-1} to t_j. For each cluster C, let $y_C^{(j)}$ be the amount by which $y_C := \sum_{v \in C} y_{C,v}$ grew during epoch j, which is $t_j - t_{j-1}$ if C was active during this epoch, and zero otherwise. We have $y_C = \sum_j y_C^{(j)}$. Because each edge e of F was added at some point by `PC-MoatGrowing` when its edge packing constraint (4.6) became tight, we can exactly apportion the length $\ell(e)$ amongst the collection of clusters $\{C : e \in \delta(C)\}$ whose variables "pay for" the edge, and can divide this up further by epoch. In other words, $\ell(e) = \sum_j \sum_{C : e \in \delta(C)} y_C^{(j)}$. We prove that the total edge length from F that is apportioned to epoch j is at most $2 \sum_C y_C^{(j)}$. In other words, during each epoch, the total rate at which edges of F are paid for by all active clusters is at most twice the number of active clusters. Summing over the epochs yields the desired conclusion.

We now analyze an arbitrary epoch j. Let $\mathcal{C}_j$ denote the set of clusters that existed during epoch j. Consider the graph F, and then collapse each cluster $C \in \mathcal{C}_j$ into a supervertex. Call the resulting graph H. Although the vertices of H are identified with clusters in $\mathcal{C}_j$, we will continue to refer to them as clusters, in order to avoid confusion with the vertices of the original graph. Some of the clusters are active and some may be inactive. Let us denote the active and inactive clusters in $\mathcal{C}_j$ by $\mathcal{C}_{act}$ and $\mathcal{C}_{dead}$, respectively. The edges of F that are being partially paid for during epoch

j are exactly those edges of H that are incident to an active cluster, and the total amount of these edges that is paid off during epoch j is $(t_j - t_{j-1}) \sum_{C \in \mathcal{C}_{act}} \mathsf{deg}_H(C)$. Since every active cluster grows by exactly $t_j - t_{j-1}$ in epoch j, we have

$$\sum_C y_C^{(j)} \geq \sum_{C \in \mathcal{C}_j} y_C^{(j)} = (t_j - t_{j-1})|\mathcal{C}_{act}|.$$

Thus, it suffices to show that $\sum_{C \in \mathcal{C}_{act}} \mathsf{deg}_H(C) \leq 2|\mathcal{C}_{act}|$.

First we must make some simple observations about H. Since the final solution F is a subset of the edges F' output by `PC-MoatGrowing` where each cluster represents a disjoint induced connected subtree, the contraction to H introduces no cycles. Thus, H is a forest. All the leaves of H must be alive because otherwise the corresponding cluster C would be in $\mathcal{B}$ and hence would have been pruned away.

With this information about H, it is easy to bound $\sum_{C \in \mathcal{C}_{act}} \mathsf{deg}_H(C)$. The total degree in H is at most $2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|)$ since H is a forest. Noticing that the degree of dead clusters is at least two, we get $\sum_{C \in \mathcal{C}_{act}} \mathsf{deg}_H(C) \leq 2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|) - 2|\mathcal{C}_{dead}| = 2|\mathcal{C}_{act}|$ as desired.

Finally, we argue that the length of an edge $e \in F$ is only charged to $y_{S,v}$ where v is connected to endpoints of e in F. If we do not perform any pruning, this is simple because $F'[S]$ is connected for all clusters $S \in \mathcal{S}$. It suffices to show, thus, that the pruning does not disconnect v from endpoints of e. Let p be a path in F' connecting v to the closer endpoint of e; i.e., e is not on p, but exactly one endpoint of e is in p. All the vertices of p belong to S, and $e \in \delta(S)$ is never pruned. To reach a contradiction, suppose e' is the first edge on p that is pruned. For the cluster $S' \in \mathcal{S}$ that causes this, we have $\delta(S') \cap p = \{e'\}$. Laminarity of $\mathcal{S}$ gives $S' \subseteq S$. This cannot happen with $v \in S'$ since $y_{S,v} > 0$ contradicts $S' \in \mathcal{B}$. In the other case, $e \in \delta(S')$, hence the intersection of $\delta(S')$ with F (at the time of pruning e') has at least two edges, $\{e, e'\}$, which is a contradiction. Therefore, no edge of p is pruned, and the claim follows. $\square$

The following lemma gives a sufficient condition for two vertices that end up in the same component of F. This is a corollary of our pruning rule which is different from previous ones. Recall that, unlike previous work, we do not prune the entire subgraph; rather, we only remove some edges, increasing the number of connected components.

Lemma 4.2.3. *Two vertices u and v of V are connected via F if there exist sets S, S' both containing u, v such that $y_{S,v} > 0$ and $y_{S',u} > 0$.*

Proof. `PC-MoatGrowing` connects u and v since $y_{S,v} > 0$ and $u, v \in S$. Consider the path p connecting u and v in F'. All the vertices of p are in S and S'. For the sake of reaching a contradiction, suppose some edges of p are pruned. Let e be the first edge being pruned on the path p. Thus, there must be a cluster $C \in \mathcal{B}$ cutting e; furthermore, $\delta(C) \cap p = \{e\}$ since e is the first edge pruned from p. As C cuts e, it only has one endpoint of the edge. Then, the laminarity of the clusters $\mathcal{S}$ gives $C \subset S, S'$. In addition, we show that C contains exactly one endpoint of the path p (as opposed to exactly one endpoint of the edge e). This holds because, if C contained neither or both endpoints of p, the cluster C could not cut p at exactly one edge. Call the endpoint of p in this cluster v. We then have $\sum_{C' \subseteq C} y_{C',v} = \phi(v)$ because C is tight. However, as C is a *proper* subset of S, this contradicts with $y_{S,v} > 0$, proving the supposition false. The case when C contains u is symmetric. □

Consider a pair (v, S) with $y_{S,v} > 0$. If subgraph G' of G has an edge that goes through the cut $(S, \overline{S})$, at least a portion of length $y_{S,v}$ of G' is painted with the color v due to the set S. Thus, if G' cuts all the sets S for which $y_{S,v} > 0$, we can charge part of the length of G' to the budget of v. Later in Lemma 4.2.5, we are going to use budgets as a lower bound on the length of a graph. (When it is invoked in Theorems 9.5.2 and 10.2.2, this serves as a lower bound on the optimum.) More formally, we say a graph $G'(V, E')$ *exhausts a color* u if and only if $E' \cap \delta(S) \neq \emptyset$ for

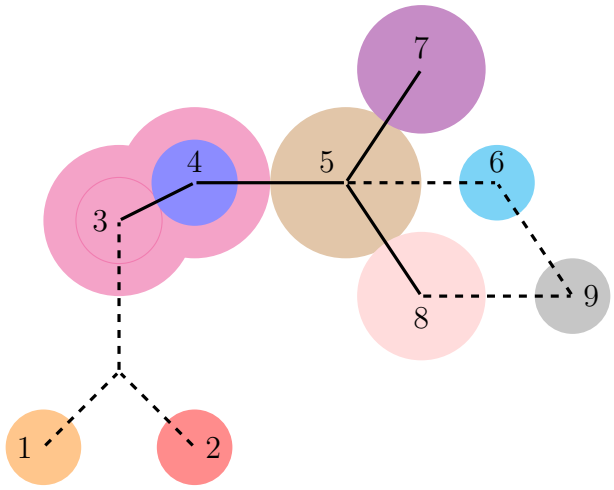


Figure 4.1: `PC-MoatGrowing` paints the graph with different colors (corresponding to budgets of vertices), and each segment of an edge may receive a different color. Solid edges represent F, the forest output by `PC-MoatGrowing`. There are five connected components, namely $K_1 = \{1\}, K_2 = \{2\}, K_3 = \{3, 4, 5, 7, 8\}, K_4 = \{6\}$ and $K_5 = \{9\}$. (The final pruning step of `PC-Classify` may remove some edges, and break up certain components into smaller ones, but for simplicity of presentation in this figure, we assume these are the actual components.) The dashed edges correspond to another forest F', say the optimum. Roughly speaking, the budget of vertices connected by F' but not by F gives a lower bound on the length of F' since this charging can be done for one pair at a time. For instance, the path from 1 to 3 passes through the colors corresponding to 1 and 3, and, since both vertices are exhausted, the length of the path is at least the sum of the budgets of the two vertices—in general, at least one of the involved vertices is exhausted by F' whenever it connects two separate components of `PC-Classify`.

any $S : y_{S,u} > 0$.

Lemma 4.2.4. *If a subgraph $G'(V, E')$ of G connects two vertices u, v from different components of F, then G' exhausts the color corresponding to at least one of u and v.*

Proof. Suppose that G exhausts neither u nor v. Then, there are a set S containing u and a set S' containing v such that $y_{S,v}, y_{S',u} > 0$ and $E' \cap \delta(S) = E' \cap \delta(S') = \emptyset$. Since G' connects u and v, this is only possible if u and v are both in S and S'. By Lemma 4.2.3, this implies that F connects u and v, which is a contradiction. □

We can relate the length of a subgraph to the budget of the colors it exhausts.

Lemma 4.2.5. *Let Q be the set of colors exhausted by a subgraph G' of G. The length of $G'(V, E')$ is at least $\phi(Q)$.*

This is quite intuitive. Recall that the y variables color the edges of the graph. Consider a segment on edges corresponding to a cluster S with color v. At least one edge of G' *passes through* the cut $(S, \overline{S})$. Thus, a portion of the length of G' can be charged to $y_{S,v}$. Hence, the total length of the graph G' is at least as large as the total amount of colors paid for by Q.

We now provide a formal proof.

Proof. The length of $G'(V, E)$ is

$$\begin{aligned}
\sum_{e \in E'} \ell(e) &\geq \sum_{e \in E'} \sum_{S : e \in \delta(S)} y_S && \text{by (4.6)} \\
&= \sum_{S} |E' \cap \delta(S)| y_S \\
&\geq \sum_{S : E' \cap \delta(S) \neq \emptyset} y_S \\
&= \sum_{S : E' \cap \delta(S) \neq \emptyset} \sum_{v \in S} y_{S,v} \\
&= \sum_{v} \sum_{S \ni v : E' \cap \delta(S) \neq \emptyset} y_{S,v}
\end{aligned}$$

$$\geq \sum_{v \in Q} \sum_{S \ni v : E' \cap \delta(S) \neq \emptyset} y_{S,v}$$
$$= \sum_{v \in Q} \sum_{S \ni v} y_{S,v},$$

because, for $v \in Q$, we have $y_{S,v} = 0$ if $E' \cap \delta(S) = \emptyset$,

$$= \sum_{v \in Q} \phi_v,$$

since (4.7) holds with equality for all vertices. □

Now we are ready to prove Theorem 4.2.1.

Proof of Theorem 4.2.1. Use `PC-Classify` on G and ϕ. The subgraph Z is the output forest F. Condition 1 is given by Lemma 4.2.2. For condition 2, let Q be the set of vertices exhausted by L. Condition 2(a) follows Lemma 4.2.5. For condition 2(b), notice that $v_1, v_2 \notin Q$ implies that L does not exhaust the budget of neither vertex. Let S_1, S_2 be the largest sets such that $y_{S_1,v_1} > 0, y_{S_2,v_2} > 0$. Since `PC-MoatGrowing` exhausts the budgets of these vertices, but L that connects v_1, v_2 does not, it has to be the case that $u_1, v_2 \in S_1$ and $u_1, v_2 \in S_2$. Then, Lemma 4.2.4 gives condition 2(b). □

4.3 Superclustering theorem

The previous clustering theorem allows us to classify graph vertices into clusters such that each cluster forms an autonomous entity. We discuss another scenario in this section when all the information is not available at the beginning, hence the clustering is achieved in two stages. Consider the setting in which each vertex of the graph is either a client or a server. Each client needs to be connected to exactly one server. Thus, the network we seek looks like a forest each of whose connected components

has exactly one server, but it can have any number of clients (possibly none). Again, each connected component forms an autonomous entity, and the funding for building the connection therein has to come from its vertices (roughly speaking). One point of divergence from the previous theorem is that, since the server does not benefit from its connection to clients, it is not willing to fund those connections. As a result, the cost of each connected component should be charged to its clients only. The second difference is the lack of full information at the beginning: we do not know which vertices are the servers; more precisely, we are only aware of one of them.

The desired procedure `PC-Superclustering`, given the graph G, budgets ϕ, and a known server $r \in V$, commits to a subset of exempt vertices $Q \not\ni r$—see below for technical restrictions on Q—as well as a tree T spanning $V \setminus Q$. After the announcement of Q and T, the set of servers $I \ni r$ is revealed, perhaps adversarially, at which point T should be pruned into a forest F such that each connected component of F contains exactly one server, and the length of the component can be charged to its clients. For technical reasons, $\phi(Q)$ serves as a shared resource that all components can use to build their trees.

Notice that for the task to be doable, having a flexibility in exempting a subset Q of vertices is necessary. Consider an instance where one vertex v has budget zero, however, the distance of v to the closest vertex is considerably larger than all the budgets combined. No matter what tree T we commit to, we cannot pay for the component containing v in the final step in case $v \notin I$. Therefore, we need the flexibility to exempt v at the beginning. On the other hand, there has to be a restriction preventing us from declaring all vertices exempt, making the construction trivial, and the theorem useless. The condition we propose is a technical one, and is exactly what we require in invoking the theorem in Chapter 8 to improve the approximation ratio for Prize-collecting Steiner Tree and Prize-collecting TSP. We ensure that the total budget of exempt vertices is no more than the optimum of the PCST

instance defined on $G, \pi = \phi$. Let $\mathsf{opt}_{\mathrm{PCST}}(G, \phi)$ denote the latter term.

Theorem 4.3.1 (Superclustering)**.** *Let $G(V, E)$ be a graph with nonnegative lengths $\ell : E \mapsto \mathbb{R}^+$ on edges, and budgets $\phi : V \mapsto \mathbb{R}^+$ for vertices $v \in V$. Given a root vertex $r \in V$, we can find in polynomial time a set Q of exempt vertices, and a tree T spanning $V \setminus Q$ such that*

1. *$\phi(Q) \leq \mathsf{opt}_{\mathrm{PCST}}(G, \phi)$, and*
2. *given any set $I \ni r$ of servers, we can prune down (i.e., remove some edges of) T to obtain a forest F with the following properties.*
   (a) *Each connected component of F contains exactly one server.*
   (b) *The length of F is at most $2\phi(V \setminus I)$; in particular, after distributing $\phi(Q)$ among the connected components of F, the length of each component is at most twice the budget of clients in that component plus its share of $\phi(Q)$.*

Superclustering is conceptually similar to hierarchical clustering (see, e.g., [LW67]). In hierarchical clustering, one produces a tree on the items to be clustered. Then, any cross section of the tree yields one clustering of the items. In some sense, the produced tree is an object that contains within itself many different clusterings, and the actual clustering is realized only after more information is provided. In a similar manner, the Superclustering Theorem produces and commits to a tree, in addition to the set of exempted vertices, and depending on what vertices are revealed as servers later, the tree turns into a clustering with certain guarantees. Thus, superclustering, too, produces an object that holds several different clusterings hidden in itself.

Furthermore, notice that the theorem does not ask the pruning procedure to run in polynomial time. When we use this theorem in Chapter 8, we do not even need a polynomial-time algorithm for constructing T. We merely need an efficient procedure

to identify Q, and guarantee the other conditions of the theorem existentially. In fact, the proof is very similar to the algorithm `GW`; however, interestingly this is the only instance of `GW`, that we are aware of, to be used nonconstructively.

Procedure `PC-Superclustering` is the polynomial-time algorithm promised in the theorem. The pruning procedure required after the servers are revealed is only discussed in the proof of Theorem 4.3.1.

Algorithm 4.4 `PC-Superclustering` (G, ϕ, r)

Input: graph $G(V, E)$, budgets $\phi(v) \geq 0$, and root r
Output: exempt vertices Q, and forest F

1: Define $\phi'(v) = \begin{cases} \phi(v) & \text{if } v \neq r \\ \infty & \text{if } v = r \end{cases}$
2: $(F, y, \mathcal{S}) \leftarrow$ `PC-MoatGrowing`(G, ϕ')
3: $\mathcal{B} \leftarrow \left\{ S \in \mathcal{S} \setminus \{V\} \mid \sum_{S' \subseteq S} \sum_{v \in S'} y_{S',v} = \sum_{v \in S} \phi_v \right\}$ # only for analysis
4: $T \leftarrow$ the connected component of F containing r
5: $Q \leftarrow \bigcup_{S \in \mathcal{B}} S$
6: **return** (Q, T)

The following two lemmas prove the above theorem. The first one giving the first condition of the theorem is fairly simple. The proof of the other lemma, that achieves the second condition involving the pruning stage, is more complicated.

Lemma 4.3.2. *For the set Q of exempt vertices returned by `PC-Superclustering`, we have $\phi(Q) \leq \mathsf{opt}_{\mathrm{PCST}}(G, \phi)$.*

Proof. For every vertex $v \in Q$, we have some cluster $S \ni v$ that became inactive during `PC-MoatGrowing`. This implies that

$$\sum_{S' \ni v} y_{S',v} \geq \sum_{S' : v \in S' \subseteq S} y_{S',v} = \phi(v).$$

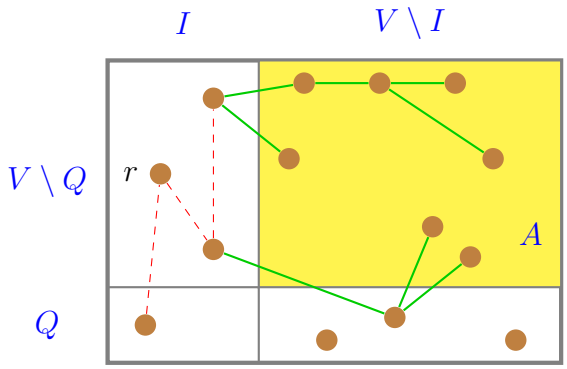


Figure 4.2: Illustration of Lemma 4.3.3 and Lemma 8.4.2. The Venn diagram shows sets $I, Q \subseteq V$. The shaded region is A, and the solid edges represent the forest F. In the proof of Lemma 8.4.2, we take $I = V(O)$ and $Q = D_{\mathcal{I}_\beta}$, then augment the optimal tree of dashed edges with the forest F of solid edges to obtain a tree spanning at least $V - Q$.

Summing up for all $v \in Q$, we obtain

$$\begin{aligned}\phi(Q) &= \sum_{v \in Q} \sum_{S \ni v} y_{S,v} \\ &\leq \sum_{v \in V} \sum_{S \ni v} y_{S,v}.\end{aligned}$$

`PC-MoatGrowing` ensures that the dual solution y is feasible for LP (4.5), and observe that $y_{V,v} = 0$ for $v \in Q$. Hence, weak duality yields the statement of the lemma. □

Lemma 4.3.3. *Let Q and T be the output of* `PC-Superclustering` *(G, ϕ, r), and let y be the vector of dual variables generated by the internal call to* `PC-MoatGrowing`*. Let $I \subseteq V \setminus Q$ be any set of vertices containing r, and let $A = V \setminus Q \setminus I$. Then, we can prune T down to a forest F, possessing the following three properties.*

Property 1. *$V(F)$ contains every vertex in A.*

Property 2. *Each tree in the forest F includes exactly one vertex from I.*

Property 3. *$\ell(F) \leq 2 \sum_{v \in \overline{I}} \sum_S y_{S,v} \leq 2\phi(\overline{I})$. More specifically, the length of each tree T' in F can be charged to twice the budget of $V(T') \setminus I$, possibly using a portion of $\phi(Q)$ which is shared among all trees.*

Before giving the proof, we introduce some definitions. Recall that $\mathcal{S}$ denotes the set of all clusters formed during `PC-MoatGrowing`. Since we have $y_S = 0$ for any set $S \notin \mathcal{S}$, it suffices to consider only the sets in $\mathcal{S}$. The following set of clusters of $\mathcal{S}$ plays an important role in the proof of the lemma.

Definition 4.3.4. *Let* $\mathcal{C}_{\overline{I}} = \left\{C \in \mathcal{S} : C \subseteq \overline{I}\right\}$.

We will refer to the clusters in $\mathcal{C}_{\overline{I}}$ as $\overline{I}$*-clusters*. All other clusters contain at least one element of I, so we term them I*-intersecting* clusters. Thus, the main technical point we need to prove is $c(F) \leq 2\sum_{C \in \mathcal{C}_{\overline{I}}} y_C$, to yield the first inequality in Property 3 from Lemma 4.3.3. To obtain the stronger claim in Property 3, it suffices to argue that the length of each tree T' is only charged to variables $y_{S,v}$ such that $v \in V(T') \cup Q$.

Proof of Lemma 4.3.3. The forest we derive must connect each vertex in A to a vertex in I. We will derive F from T by deleting some edges in two phases, illustrated by Figure 4.3. Since $\overline{V(T)}$ consists of the union of the dead clusters that still existed at the end of `PC-MoatGrowing`, $\overline{V(T)} \subseteq Q$, by definition of Q. Thus, T already satisfies Property 1, and both phases of edge deletion will preserve it as an invariant. The first phase will accomplish Property 2, and the second will preserve it as an invariant. Both phases of edge deletion taken together will ensure that

$$\ell(F) \leq 2 \sum_{C \in \mathcal{C}_{\overline{I}}} y_C, \tag{4.9}$$

which is the most complicated part of the argument. By the penalty packing constraint (4.7), the summation is at most $\phi(\overline{I})$ since y is a feasible solution to LP (4.5). Property 3 then follows.

We now show how to construct F by deleting edges from T. To help us prove (4.9), we will also ensure that F satisfies the following additional property. For each pair

of distinct vertices $u, v \in I$, let P_{uv} denote the path from u to v in T, and let e_{uv} be the last edge on P_{uv} that was added to T during `PC-MoatGrowing`.

Property 4. *For each pair of vertices $u, v \in I$, we have $e_{uv} \notin F$.*

Start with $F = T$. As we observed above, $\overline{V(T)} \subseteq Q$, and hence T spans at least $V \setminus Q$. Consider the edges of T in the reverse of the order in which they were added during `PC-MoatGrowing`. When we arrive at an edge e, if there exist two vertices in I that are still in the same tree of F but would be separated by deleting e, then do so. These deletions preserve the invariant that each tree in the forest F contains at least one vertex from I (this holds initially because T and I both contain r). Thus, none of the edge deletions isolate any vertices in A, so Property 1 holds throughout this pruning phase. We will satisfy Property 4 by the end because, at the time we consider edge e_{uv}, vertices u and v are still connected by the path P_{uv}, and so we delete e_{uv}. Thus, F will end up with exactly one vertex of I per tree, satisfying Property 2. Another way to view this first pruning phase is that it simply deletes e_{uv}, for all distinct vertices $u, v \in I$. In order to achieve (4.9), we must do another pruning phase.

In the second phase, we prune each tree in the forest F in precisely the same way that `GW` would do; the result is different, though, because of the pruning phase already performed. We color a cluster C black if C died at any point during `PC-MoatGrowing`. (Notice that in this case $C \in \mathcal{C}_{\overline{I}}$ since $C \subseteq Q \subseteq V \setminus I$.) Then, as in `GW`, while there exists a black cluster C such that $|F \cap \delta(C)| = 1$, we prune away all edges of F that are incident to one or two vertices of C. In other words, we prune away the one edge in $F \cap \delta(C)$, plus all edges of F internal to C. Since the black clusters contain only vertices from $Q \subseteq \overline{I}$, this pruning does not throw away any vertex from A or I. Therefore, C does not contain any entire trees of F (because that would include a vertex from I, by Property 2). This, along with $|F \cap \delta(C)| = 1$, implies that C must contain vertices from only one of the trees of F. Thus, each pruning step throws away

an entire limb from one of the trees of F, and the remaining part of the pruned tree is still a single component. Thus, Properties 1 and 2 are maintained. The remaining forest is our final F.

We now prove that this F satisfies (4.9). Our proof is very similar to the famous proof of Theorem 8.3.1 [GW95, FFFdP10]. After much preparatory work, this proof boils down to the fact that the average degree of the vertices in a tree is just under 2, and excluding some nonleaf vertices from the average preserves this property since their degrees are all at least 2. This is what causes the factor of 2 to appear in (4.9).

We first break apart the dual variables by epoch. Let t_j be the time at which the j^{th} event point occurs in `PC-MoatGrowing` ($0 = t_0 \leq t_1 \leq t_2 \leq \cdots$), so that the j^{th} epoch is the interval of time from t_{j-1} to t_j. For each cluster C, let $y_C^{(j)}$ be the amount by which y_C grew during epoch j, which is $t_j - t_{j-1}$ if C was active during this epoch, and 0 otherwise. Thus, $y_C = \sum_j y_C^{(j)}$. Because each edge $e \in F$ was added by `PC-MoatGrowing` when its edge packing constraint (4.6) became tight, we can exactly apportion the cost $\ell(e)$ amongst the dual variables $\{y_C : C \in \mathcal{S}, e \in \delta(C)\}$ that appear in the sum on the left side of the tight edge packing constraint for e. We can then further divide this up by epoch. In other words, $\ell(e) = \sum_{C \in \mathcal{S}: e \in \delta(C)} y_C = \sum_{C \in \mathcal{S}: e \in \delta(C)} \sum_j y_C^{(j)}$. Hence, to prove (4.9), it suffices to prove that the total edge cost from F that is apportioned to epoch j is at most twice the amount paid by the clusters in $\mathcal{C}_{\overline{I}}$, i.e.,

$$\sum_{e \in F} \sum_{C \in \mathcal{S}: e \in \delta(C)} y_C^{(j)} \leq 2 \sum_{C \in \mathcal{C}_{\overline{I}}} y_C^{(j)}. \tag{4.10}$$

In other words, since $y_C^{(j)}$ is 0 for inactive clusters and the same for all active clusters, we wish to show that during each epoch, the total rate at which edges of F are paid for by all active clusters is at most twice the number of active clusters in $\mathcal{C}_{\overline{I}}$. Summing

over the epochs yields (4.9). More formally,

$$\begin{aligned}\ell(F) &= \sum_{e \in F} \ell(e) = \sum_{e \in F} \sum_{C \in \mathcal{S}: e \in \delta(C)} \sum_{j} y_C^{(j)} \\ &= \sum_{j} \left[\sum_{e \in F} \sum_{C \in \mathcal{S}: e \in \delta(C)} y_C^{(j)} \right] \leq \sum_{j} \left[2 \sum_{C \in \mathcal{C}_{\overline{I}}} y_C^{(j)} \right] \\ &= 2 \sum_{C \in \mathcal{C}_{\overline{I}}} \sum_{j} y_C^{(j)} = 2 \sum_{C \in \mathcal{C}_{\overline{I}}} y_C,\end{aligned}$$

where the inequality follows from (4.10).

We now fix an arbitrary epoch j and analyze it to prove (4.10), at which point we will be done. Let $\mathcal{C}_j$ denote the set of clusters that existed during epoch j and contain a vertex from $V(F)$. The entire set of clusters that existed during epoch j partitions V, but we are concerned only with the partition of $V(F)$ induced by intersecting each cluster with $V(F)$, and hence we care about only the clusters in $\mathcal{C}_j$.

Consider the graph $(V(F), F)$, and then contract each cluster $C \in \mathcal{C}_j$ into a supervertex. Call the resulting graph H. Notice that there is an alternate equivalent way to construct H: starting with T, first contract all clusters that existed during epoch j, then delete the edges corresponding to $T \setminus F$, then clean up by deleting any isolated clusters that are not in $\mathcal{C}_j$ (i.e., contain no vertices from $V(F)$). Figure 4.3 illustrates this. From this alternate description, it is obvious that each edge of H corresponds to a unique edge in F since T is a tree and we contract clusters which correspond to disjoint connected subtrees of T. It may be the case that some cluster $C \in \mathcal{C}_j$ contains vertices from distinct trees in the forest F, in which case H will have fewer distinct connected components than F does. Although the clusters in $\mathcal{C}_j$ are identified with vertices of H, we will continue to refer to them as clusters rather than vertices, in order to to avoid confusion with the vertices of the original graph $G = (V, E)$.

We will use the graph H to analyze epoch j and prove (4.10). Let us begin with

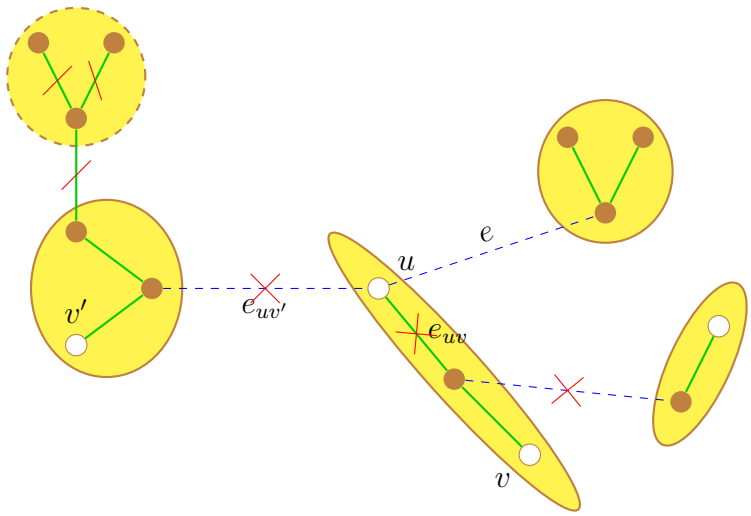


Figure 4.3: This figure illustrates the analysis of an epoch j. The hollow vertices are in I, and the solid vertices are in $\overline{I}$. The tree T consists of all 13 edges. The solid edges form the forest F_j of edges already added by `PC-MoatGrowing` prior to epoch j, and the shaded ovals denote the clusters that existed during epoch j. The dashed edges were added to T later on. The 3 edges that are crossed out were deleted in the first pruning phase, in order to separate the hollow vertices (in I). The cluster with the dashed border had already died prior to epoch j, so the second pruning phase colored it black. Since it also has only one external edge in T, this cluster was pruned in the second phase, denoted by the slashes across the 3 edges incident on one or two of its vertices. The 7 edges with no slashes or crosses make up the final forest F. The 4 clusters with solid borders are the vertices of the forest H, which contains 3 trees: 2 isolated vertices and one pair of vertices connected by edge e. The edge e is the only edge of F that is being paid for during this epoch.

three simple observations about H.

Observation 4.3.5. *H is a forest.*

Proof. Consider the alternate construction of H, in which we contract clusters first, then delete edges. Since each cluster is a contiguous subset of vertices from the tree T, contracting them introduces no cycles, so the contracted graph is still a tree. Deleting edges just breaks up this tree into a forest. □

Observation 4.3.6. *Every dead leaf of H is I-intersecting.*

Proof. First note that H has no black leaves. A cluster C being a leaf in H is equivalent to having $|F \cap \delta(C)| = 1$ since every edge of H corresponds to a unique edge of F. Thus, if $C \in \mathcal{C}_j$ were a black leaf, then we would have pruned it away in our second pruning phase. Therefore, every dead leaf of H is I-intersecting since otherwise it would be colored black. □

Observation 4.3.7. *Within each tree of the forest H, there is* exactly *one I-intersecting cluster.*

Proof. By Property 2, each tree in F includes exactly one vertex of I. Thus, each tree in H includes at least one I-intersecting cluster, so we just need to show that there is at most one. We show that for each pair u, v of distinct vertices in $I \cap V(F)$, either u and v are contracted into the same vertex of H, or they end up in different trees of H. Therefore, if C_u and C_v are the corresponding I-intersecting clusters, then either $C_u = C_v$, or they are in different trees of H.

Consider the edge e_{uv} and the path P_{uv} defined previously. Let $F_j \subseteq T$ denote the forest of edges that `PC-MoatGrowing` had built within T prior to epoch j; this is the forest whose connected components are the clusters that existed during epoch j. Suppose that $e_{uv} \in F_j$. Then $P_{uv} \subseteq F_j$ because e_{uv} was the last edge along path P_{uv} to be added (and hence the remaining edges of P_{uv} have already been added). Thus,

u and v are connected in F_j, hence in the same cluster in $\mathcal{C}_j$, so they are contracted into the same vertex of H. This situation corresponds to vertices u, v in Figure 4.3.

Now suppose instead that $e_{uv} \notin F_j$. Then the endpoints of e_{uv} are in different connected components of F_j, i.e., different clusters. Now consider the alternate construction of H, in which we first contract clusters, then delete the edges corresponding to $T \setminus F$. When we contract clusters, edge e_{uv} survives (because its endpoints are in different clusters), and u and v are contracted into clusters that are on opposite sides of the edge corresponding to e_{uv} in the contracted tree. Property 4 guarantees $e_{uv} \notin F$, so the subsequent deletion of the edges corresponding to $T \setminus F \ni e_{uv}$ ensures that these two clusters are separated into different trees in the forest H. This situation corresponds to vertices u, v' in Figure 4.3. □

We now use Observations 4.3.5, 4.3.6 and 4.3.7 to prove (4.10). During the j^{th} epoch, some of the clusters are active and the rest are dead. Let us denote the sets of active and dead clusters in $\mathcal{C}_j$ by $\mathcal{C}_{act}$ and $\mathcal{C}_{dead}$, respectively. The edges of F that are being partially paid for during epoch j correspond exactly to those edges of H that are incident to an active cluster, and the total portion of these edge costs that is paid off during epoch j is

$$(t_j - t_{j-1}) \sum_{C \in \mathcal{C}_{act}} \mathsf{deg}_H(C), \tag{4.11}$$

since $y_C^{(j)} = t_j - t_{j-1}$ for every active cluster C, and C pays this amount toward $\mathsf{deg}_H(C)$ different edges in F. Thus, expression (4.11) equals the left side of (4.10). Simplifying the right side of (4.10), we have $\sum_{C \in \mathcal{C}_{\bar{I}}} y_C^{(j)} \geq \sum_{C \in \mathcal{C}_{\bar{I}} \cap \mathcal{C}_j} y_C^{(j)} =$

$\sum_{C \in \mathcal{C}_{\overline{I}} \cap \mathcal{C}_{act}} y_C^{(j)} = (t_j - t_{j-1})|\mathcal{C}_{\overline{I}} \cap \mathcal{C}_{act}|$.[2] Thus, to prove (4.10), it suffices to show that

$$\sum_{C \in \mathcal{C}_{act}} \mathsf{deg}_H(C) \leq 2|\mathcal{C}_{\overline{I}} \cap \mathcal{C}_{act}|. \tag{4.12}$$

Let k be the number of trees in H, and l be the number of dead leaves in H. By Observation 4.3.7, we know that exactly k clusters are I-intersecting. By Observation 4.3.6, the l dead leaves must be among them, so $l \leq k$. Thus, the number $|\{C \in \mathcal{C}_{act} : C \cap I \neq \emptyset\}|$ of active I-intersecting clusters is at most $k - l$. Since every active cluster is either an $\overline{I}$-cluster or I-intersecting,

$$\begin{aligned} |\mathcal{C}_{act}| &= |\mathcal{C}_{\overline{I}} \cap \mathcal{C}_{act}| + |\{C \in \mathcal{C}_{act} : C \cap I \neq \emptyset\}| \\ &\leq |\mathcal{C}_{\overline{I}} \cap \mathcal{C}_{act}| + (k - l). \end{aligned}$$

Therefore

$$|\mathcal{C}_{\overline{I}} \cap \mathcal{C}_{act}| \geq |\mathcal{C}_{act}| - (k - l). \tag{4.13}$$

The number of clusters in H is $|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|$, so the number of edges is $(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|) - k$, there being k trees in H. Hence, $\sum_{C \in \mathcal{C}_{act} \cup \mathcal{C}_{dead}} \mathsf{deg}_H(C) = 2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}| - k)$. Since there are only l dead leaves, $\sum_{C \in \mathcal{C}_{dead}} \mathsf{deg}_H(C) \geq 2(|\mathcal{C}_{dead}| - l) + l =$

[2] The inequality may be strict if some $C \in \mathcal{C}_{\overline{I}}$ is growing in epoch j, and hence $y_C^{(j)} > 0$, but $C \cap V(F) = \emptyset$, so $C \notin \mathcal{C}_j$.

$2|\mathcal{C}_{dead}| - l$ since every nonleaf cluster has degree at least 2. Thus,

$$\begin{aligned}
\sum_{C \in \mathcal{C}_{act}} \deg_H(C) &= \sum_{C \in \mathcal{C}_{act} \cup \mathcal{C}_{dead}} \deg_H(C) - \sum_{C \in \mathcal{C}_{dead}} \deg_H(C) \\
&\leq 2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}| - k) - (2|\mathcal{C}_{dead}| - l) \\
&= 2|\mathcal{C}_{act}| - 2k + l \\
&= 2(|\mathcal{C}_{act}| - (k - l)) - l \\
&\leq 2(|\mathcal{C}_{act}| - (k - l)) \\
&\leq 2|\mathcal{C}_{\overline{I}} \cap \mathcal{C}_{act}|,
\end{aligned}$$

where the last inequality following from (4.13). Thus, we have established (4.12), as we wished. □

Finally, we are at a position to prove the Superclustering Theorem.

Proof of Theorem 4.3.1. We use `PC-Superclustering` (G, ϕ, r) to obtain T and Q. Lemmas 4.3.2 and 4.3.3 yield the first and second conditions of the theorem, respectively. □

4.4 Submodular clustering

This section introduces the procedure `SubmodPC-Cluster`, which can be seen as an extension of `PC-MoatGrowing` in two directions. First, it deals with budgets that are associated with source-sink pairs instead of being associated to single vertices—in this sense, this is a generalization of PCST to PCSF—and, secondly, the budget is a submodular function of the demands involved in contrast to the usual additive function.

`SubmodPC-Cluster` can be used to obtain a constant-factor approximation algorithm for Submodular Prize-collecting Steiner Forest, however, we do not

plan to do so. Rather, we are going to employ it in Section 10.2 to reduce planar SPCSF instances to bounded-treewidth instances of the same problem.

Consider an instance $(G(V,E),\mathcal{D},\pi)$ of SPCSF. A set $S \subseteq V$ is said to *cut* a demand $d = \{s,t\}$ if and only if $|S \cap d| = 1$. We denote this by the shorthand $d \odot S$, and say the demand d *crosses* the set S. In the linear program (4.14)–(4.16), there is a variable $y_{S,d}$ for any $S \subseteq V, d \in \mathcal{D}$ such that $d \odot S$. For convenience, we use the shorthands $y_S := \sum_{d\in\mathcal{D}} y_{S,d}$ and $y_d := \sum_{S\subseteq V} y_{S,d}$. Notice that this is merely a set of linear equations with no explicit objective function, however, it looks like the dual program of a natural linear-programming relaxation for SPCSF.

$$\sum_{S:e\in\delta(S)} y_S \leq c_e \qquad \forall e \in E \tag{4.14}$$

$$\sum_{d\in D} y_d \leq \pi(D) \qquad \forall D \subseteq \mathcal{D} \tag{4.15}$$

$$y_{S,d} \geq 0 \qquad \forall d \in \mathcal{D}, S \subseteq V, d \odot S. \tag{4.16}$$

The following theorem presents some guarantees of `SubmodPC-Cluster` which produces a solution to the above LP. For convenience, we use the notation $y(D) := \sum_{d\in D} y_d$ for any $D \subseteq \mathcal{D}$.

Theorem 4.4.1. *Given an instance $(G,\mathcal{D},\pi)$ of* SPCSF*, we produce in polynomial time a forest F and a subset $\mathcal{D}^{\mathsf{unsat}} \subseteq \mathcal{D}$ of demands, along with a feasible vector y for Equations (4.14)-(4.16) such that*

1. *$y(\mathcal{D}^{\mathsf{unsat}}) = \pi(\mathcal{D}^{\mathsf{unsat}})$;*
2. *F satisfies any demand in $\mathcal{D}^{\mathsf{sat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{unsat}}$; and*
3. *$\ell(F) \leq 2y(\mathcal{D})$.*

`SubmodPC-Cluster` is in essence very similar to `PC-MoatGrowing`, however, these are the major technical difficulties we face. One source of budget can be spent in different clusters simultaneously because each demand is associated with two vertices whose clusters grow independently until they reach each other or run out of budget. In addition, the algorithm will find tight sets (that stop budgets of some demands from being used any longer) that do not form connected subgraphs. Performing the growth step faces some challenge as finding the next event point is not trivial; in fact, it requires solving an auxiliary LP that can be dealt with only through calling a submodular minimization procedure as the separation oracle. Last but not least, the lower bound argument cannot charge the length to individual demands, and has to take several demands at once.

We now describe the two phases of `SubmodPC-Cluster`: the growth and pruning phases.

Growth We begin with a zero vector y, and an empty set F of edges. A demand $d \in \mathcal{D}$ is said to be *alive* if and only if $y(D) < \pi(D)$ for any $D \subseteq \mathcal{D}$ that $d \in D$—notice that, unlike in `PC-MoatGrowing`, we cannot define this by considering the demand in question alone. If a demand is not alive, it is called *dead*. During the execution of the algorithm, we maintain a partition $\mathcal{C}$ of vertices V into clusters; it initially consists of singleton sets. Each cluster is either *active* or *inactive*; the cluster $C \in \mathcal{C}$ is *active* if and only if there is a live demand $d : d \odot C$. We simultaneously *grow* all the active clusters by η. In particular, if there are $\kappa(C) > 0$ live demands crossing an active cluster C, we increase $y_{C,d}$ by $\eta/\kappa(C)$ for each live demand $d : d \odot C$. Hence, y_C is increased by η for every active cluster C. We pick the largest value for η that does not violate any of the constraints in (4.14) or (4.15). Obviously, η is finite in each iteration because the values of these variables cannot be larger than $\pi(\mathcal{D})$. Hence at least one such constraint goes tight after each growth step. If this happens for an

edge constraint for $e = (u, v)$, then there are two clusters $C_u \ni u$ and $C_v \ni v$ in $\mathcal{C}$, and at least one of which is growing. We merge the two clusters into $C = C_u \cup C_v$ by adding the edge e to F, remove the old clusters and add the new one to $\mathcal{C}$. Nothing needs to be done if a constraint (4.15) becomes tight. The number of iterations is at most $2|V|$ because at each event either a demand dies, or the size of $\mathcal{C}$ decreases.

As noted above, computing η is nontrivial here. In particular, we have to solve an auxiliary linear program to find its value. New variables $y^*_{S,d}$ denote the value of vector y after a growth of size η.

$$\text{maximize} \quad \eta \tag{4.17}$$

$$\text{subject to} \quad y^*_{S,d} = y_{S,d} + \frac{\eta}{\kappa(S)} \qquad \forall d \in \mathcal{D}, S \subseteq V, d \odot S, \kappa(S) > 0 \tag{4.18}$$

$$y^*_{S,d} = y_{S,d} \qquad \forall d \in \mathcal{D}, S \subseteq V, d \odot S, \kappa(S) = 0 \tag{4.19}$$

$$\sum_{S: e \in \delta(S)} y^*_S \leq c_e \qquad \forall e \in E \tag{4.20}$$

$$\sum_{d \in D} y^*_d \leq \pi(D) \qquad \forall D \subseteq \mathcal{D} \tag{4.21}$$

$$y^*_{S,d} \geq 0 \qquad \forall d \in \mathcal{D}, S \subseteq V, d \odot S. \tag{4.22}$$

All the constraints are written for the new variables. There are exponentially many constraints in this LP, however, it admits a separation oracle and thus can be optimized as described below. Notice that there are only a polynomial number of nonzero variables at each step since $y_{S,d}$ may be nonzero only for clusters S, and these clusters form a laminar family in our algorithm. Verifying constraints (4.18)-(4.20) and (4.22) is very simple. Verifying constraints (4.21) is equivalent to finding $\min_{D \subseteq \mathcal{D}}[\pi(D) - y^*(D)]$ and ensuring that it is nonnegative. The function to minimize is submodular (because $\pi(D)$ is submodular and $y^*(D)$ is additive), and thus can be minimized in polynomial time; see Theorem 2.1.1. A standard argument shows that the values of these variables have polynomial size.

Pruning Let $\mathcal{S}$ denote the set of all clusters formed during the execution of the growth step. It can be easily observed that the clusters $\mathcal{S}$ are laminar and the maximal clusters are the clusters of $\mathcal{C}$. In addition, notice that $F[C]$ is connected for each $C \in \mathcal{S}$.

Let $\mathcal{B} \subseteq \mathcal{S} \setminus \{V\}$ be the set of all clusters C that do not cut any live demands. Notice that a demand d may still be live at the end of the growth phase if it is satisfied; roughly speaking, the demand is satisfied before it exhausts its *budget*. In the pruning phase, we iteratively remove edges from F to obtain the final forest. More specifically, as long as there is a cluster $S \in \mathcal{B}$ such that $F \cap \delta(S) = \{e\}$, we remove the edge e from F.

A cluster C is called a *pruned cluster* if it is pruned in the second phase in which case, $\delta(C) \cap F = \emptyset$. Hence, a pruned cluster cannot have nonempty and proper intersection with a connected component of F.

Algorithm 4.5 `SubmodPC-Cluster` $(G, \mathcal{D}, \pi)$

Input: graph $G(V, E)$, set of demands $\mathcal{D}$, and monotone submodular penalty function $\pi : 2^{\mathcal{D}} \mapsto \mathbb{R}^+$

Output: forest F, subset of demands $\mathcal{D}^{\mathsf{unsat}}$ and fractional solution y

$F \leftarrow \emptyset$
$y_{S,d} \leftarrow 0$ for any $d \in \mathcal{D}, S \subseteq V, d \odot S$
$\mathcal{S} \leftarrow \mathcal{C} \leftarrow \{\{v\} : v \in V\}$
while there is a live demand **do**
  Compute η via LP (4.17)
  $y_{C,d} \leftarrow y_{C,d} + \frac{\eta}{\kappa(C)}$ for all live demands $d : d \odot C$
  if $\exists e \in E$ that is tight and connects two clusters C_1 and C_2 **then**
    $F \leftarrow F \cup \{e\}$
    $C \leftarrow C_1 \cup C_2$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\} \setminus \{C_1, C_2\}$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$
$F' \leftarrow F$ # only for the analysis
$\mathcal{B} \leftarrow$ set of all clusters $S \in \mathcal{S}$ that do not cut any live demands
while $\exists S \in \mathcal{B}$ such that $F \cap \delta(S) = \{e\}$ for an edge e **do**
  $F \leftarrow F \setminus \{e\}$
$\mathcal{D}^{\mathsf{unsat}} \leftarrow$ set of dead demands
return $(F, \mathcal{D}^{\mathsf{unsat}}, y)$

We first bound the length of the forest F. The following lemma is similar to the analysis of the algorithm in [GW95] as well as those already given in this chapter.

Lemma 4.4.2. $\ell(F) \leq 2y(\mathcal{D})$.

Proof. Recall that the growth phase has several events corresponding to an edge or set constraint going tight. We first break apart y variables by epoch. Let t_j be the time at which the j^{th} event point occurs in the growth phase ($0 = t_0 \leq t_1 \leq t_2 \leq \cdots$), so the j^{th} epoch is the interval of time from t_{j-1} to t_j. For each cluster C, let $y_C^{(j)}$ be the amount by which y_C grew during epoch j, which is $t_j - t_{j-1}$ if it was active during this epoch, and zero otherwise. We clearly have $y_C = \sum_j y_C^{(j)}$. Because each edge e of F was added at some point by the growth stage when its edge packing constraint (4.14) became tight, we can exactly apportion the length c_e amongst the collection of clusters $\{C : e \in \delta(C)\}$ whose variables "pay for" the edge, and can divide this up further by epoch. In other words, $c_e = \sum_j \sum_{C:e\in\delta(C)} y_C^{(j)}$. We will now prove that the total length from F that is apportioned to epoch j is at most $2\sum_C y_C^{(j)}$. In other words, during each epoch, the total rate at which edges of F are paid for by all active clusters is at most twice the number of active clusters. Summing over the epochs yields the desired conclusion.

We now analyze an arbitrary epoch j. Let $\mathcal{C}_j$ denote the set of clusters that existed during epoch j. Consider the graph F, and then collapse each cluster $C \in \mathcal{C}_j$ into a supervertex. Call the resulting graph H. Although the vertices of H are identified with clusters in $\mathcal{C}_j$, we will continue to refer to them as clusters, in order to to avoid confusion with the vertices of the original graph. Some of the clusters are active and some may be inactive. Let us denote the active and inactive clusters in $\mathcal{C}_j$ by $\mathcal{C}_{act}$ and $\mathcal{C}_{dead}$, respectively. The edges of F that are being partially paid for during epoch j are exactly those edges of H that are incident to an active cluster, and the total amount of these edges that is paid off during epoch j is $(t_j - t_{j-1}) \sum_{C\in\mathcal{C}_{act}} \mathsf{deg}_H(C)$. Since every active cluster grows by exactly $t_j - t_{j-1}$ in epoch j, we have $\sum_C y_C^{(j)} \geq$

$\sum_{C \in \mathcal{C}_j} y_C^{(j)} = (t_j - t_{j-1})|\mathcal{C}_{act}|$. Thus, it suffices to show that $\sum_{C \in \mathcal{C}_{act}} \deg_H(C) \leq 2|\mathcal{C}_{act}|$.

First we must make some simple observations about H. Since F is a subset of the edges in F', and each cluster represents a disjoint induced connected subtree of F', the contraction to H introduces no cycles. Thus, H is a forest. All the leaves of H must be live clusters because otherwise the corresponding cluster C would be in $\mathcal{B}$ and hence would have been pruned away.

With this information about H, it is easy to bound $\sum_{C \in \mathcal{C}_{act}} \deg_H(C)$. The total degree in H is at most $2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|)$. Noticing that the degree of dead clusters is at least two, we get $\sum_{C \in \mathcal{C}_{act}} \deg_H(C) \leq 2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|) - 2|\mathcal{C}_{dead}| = 2|\mathcal{C}_{act}|$ as desired. □

Finally we can prove Theorem 4.4.1 that characterizes the output of `Submod-PC-Cluster`.

Proof of Theorem 4.4.1. For every demand $d \in \mathcal{D}^{\mathsf{unsat}}$ we have a set $D \ni d$ such that $y(D) = \pi(D)$. The definition of $\mathcal{D}^{\mathsf{unsat}}$ guarantees $D \subseteq \mathcal{D}^{\mathsf{unsat}}$. Therefore, we have sets $D_1, D_2, \ldots, D_l$ that are all tight—i.e., $y(D_i) = \pi(D_i)$—and span $\mathcal{D}^{\mathsf{unsat}}$—i.e., $\mathcal{D}^{\mathsf{unsat}} = \cup_i D_i$. To prove $y(\mathcal{D}^{\mathsf{unsat}}) = \pi(\mathcal{D}^{\mathsf{unsat}})$, we use induction and combine D_i's two at a time. For any two tight sets $A, B \subseteq \mathcal{D}$ we have

$$\begin{aligned} y(A \cup B) &= y(A) + y(B) - y(A \cap B) \\ &= \pi(A) + \pi(B) - y(A \cap B) \\ &\geq \pi(A) + \pi(B) - \pi(A \cap B) \\ &\geq \pi(A \cup B), \end{aligned}$$

where the second equation follows from tightness of A and B, the third step is a result of Constraint (4.15), and the last step follows from submodularity. Constraint (4.15) has it that $\pi(A \cup B) \geq y(A \cup B)$, therefore, it has to hold with equality.

Clearly, at the end of execution of `SubmodPC-Cluster`, any live demand is already satisfied. Notice that such demands are not affected in the pruning stage. Hence, only dead demands may be unsatisfied. This guarantees the second condition. The third condition follows from Lemma 4.4.2. □

Chapter 5

Spanner Framework

In this chapter, we discuss approximation algorithms and, in particular, PTASes for problems on planar graphs. See [Bak94, Haj05, DH08a, DH08b] for more details. The two main tools for tackling such problems are finding small separators [LT79, LT80] and using Baker's idea [Bak94] or its extensions [Epp00, Kle06, DHM07, Kle08, BKM09, DHK11]. The first one applies a divide-and-conquer strategy, whereas the second reduces the problem into instances that are structurally simpler.

The first technique looks for small edge sets, called "separators," in the graph whose removal breaks the graph into pieces each of which is a constant factor small than the original. Continuing this until all the final pieces have a constant size yields a "recursion tree." The leaves of the tree, corresponding to the constant-size instances are handled via brute force, and all that remains is to glue the solutions together and move up in the recursion tree. That the separators are small makes this possible. Usually the resulting error is charged to the total size of all separators that is bounded by ϵn. If the optimum has size $\Omega(n)$, this approach leads to a PTAS. The limitation on the size of optimum can sometimes be avoided through the use of kernelization (see, e.g., [AKCF+04]).

Baker introduced a technique that presents PTASes for several problem including

the Maximum Independent Set: given a graph, find the largest set of vertices that do not cover both endpoints of any edge. The algorithm starts by embedding the graph into the plane, and taking outer layers one at a time, numbering the vertices in each layer with the number of the layer starting from zero for the outermost layer. Notice, for $0 \leq i < \zeta$, that removing all layers congruent to i modulo ζ produces a ζ-outerplanar graph, on which Maximum Independent Set can be solved using dynamic programming. The rest of the argument goes by observing that there is one layer that forms no more than a $1/\zeta$ fraction of the optimum. Therefore, trying all the possibilities leads to a $\frac{1}{1-1/\zeta}$-approximation algorithm that runs in polynomial time for constant ζ.

In a more abstract sense, what Baker's approach [Bak94]—or its generalization by Eppstein [Epp00]—does is partition the edges of the graph into ζ sets removal of each produces a bounded-treewidth graph. Such results have been proved not only for planar graphs [Bak94], but also for bounded-genus [Epp00] and even H-minor-free graphs [DDO+04, DHK05]. Though this approach seems to only work for *deletion-closed* problems, similar theorems have also been proved for *contraction-closed* problems [Kle06, DHM07, Kle08, DHK11]. In a deletion-closed (respectively, contraction-closed) problem, the solution may only improve if edges are deleted (respectively, contracted). Most of the problems discussed in this thesis (e.g., Steiner Tree, PCST, PCSF) are contraction-closed.

5.1 Spanners

All the above techniques have the inherent limitation that the optimum must have a value $\Omega(n)$. This is because the loss due to the set of edges we remove or contract can only be compared to the total number of edges. One workaround used to alleviate this issue, i.e., kernelization, only works for unweighted graphs.

A more successful approach to extend the applicability of the above decomposition theorems is through *spanners*. Roughly speaking, a spanner (defined with respect to an instance of a problem) is a subgraph of the original input graph whose total length is relatively small (comparable to opt), but still guarantees to contain a good (usually $(1+\epsilon)$-approximate) solution. After constructing a spanner, we apply the decomposition theorem (for deletion or contraction) on the edges of the spanner. Therefore, one edge set has a "negligible" cost/profit which we can ignore, and the problem turns into a simpler instance (say, on a bounded-treewidth graph).

Definition 5.1.1. *Given are an instance of a minimization problem on a graph G, and fixed constant $\epsilon > 0$. We say H is a* spanner *for G if*

1. *H is a subgraph of G,*
2. *$\ell(H) \leq O(\mathsf{opt})$, and*
3. *$\mathsf{opt}_H \leq (1+\epsilon)\mathsf{opt}_G$.*

Although the definition depends on the problem, the specific instance in question, and the parameter ϵ, we usually omit these references when the context is clear.

For some problems, this is equivalent to preserving all the distances approximately. This concept has enjoyed active research in the fields of computational geometry [Che86, KG92, GNS08, BS11] and algorithmic graph theory [PS89, Elk08, BS08, Kle06, Kle08].

Let us introduce the two main theorems here.

Theorem 5.1.2 ([Bak94, Epp00, DHK05])**.** *There is a polynomial-time algorithm that, for each $\zeta \geq 1$, decomposes the edges of a planar, bounded-genus, or H-minor-free graph into ζ color classes such that deleting the edges in each class results in a graph with treewidth at most $O(\zeta)$. The treewidth is $O(g\zeta)$ for an input graph of*

genus g, and $O(c_H\zeta)$ for an H-minor-free graph where c_H is a constant depending on the size of the minor H.

Theorem 5.1.3 ([Kle08, DHM07, DHK11])**.** *There is a polynomial-time algorithm that, for each $\zeta \geq 1$, decomposes the edges of a planar, bounded-genus, or H-minor-free graph into ζ color classes such that contracting the edges in each class results in a graph with treewidth at most $O(\zeta)$. The treewidth is $O(g\zeta)$ for an input graph of genus g, and $O(c_H\zeta)$ for an H-minor-free graph where c_H is a constant depending on the size of the minor H.*

Whereas the running time of the procedure for bounded-genus graphs given in [DHM07] is $O(n^{3/2}\log n)$, Borradaile et al. [BDT09] observe that the running time can be improved to $O(n \log n)$ using ideas from [CC07].

5.2 An example: exact algorithm for planar k-Cut

An instance of k-Cut is described by an undirected graph G and a parameter k, where the goal is to delete the minimum number of edges that break G into at least k connected components. The problem in NP-hard in general graphs, but Kawarabayashi and Thorup [KT11] recently found a polynomial-time algorithm for planar graphs when k is a fixed constant. Their approach is readily extended to H-minor-free graphs via the stronger part of Theorem 5.1.2.

The algorithm proceeds as follows. Since each planar graph has a vertex of degree no more than five (see Corollary 2.3.7), there is a solution to the k-Cut instance with at most $5k$ edges: repeatedly take out all edges incident on the minimum-degree vertex until the number of connected components becomes at least k. We invoke Theorem 5.1.2 with $\zeta = 5k + 1$. Then, at least one color class does not intersect the optimum. We can guess that color class (i.e., enumerate all possibilities), contract it, and solve the problem on the resulting bounded-treewidth graph—k-Cut is solvable

in polynomial time on graphs of bounded treewidth. Notice that, if a set of edges E' are guaranteed not to be part of the k-Cut solution, there is no harm in contracting all edges in E'—in fact, contracting an edge is tantamount to making it impossible to cut.

5.3 The general reduction

We restate (a slightly more general form of) Theorem 2 of [DHK11] here.

Consider a minimization problem P on weighted graphs that is closed under contraction (i.e., contracting edges may only decrease the optimum). Suppose P satisfies the following properties.

1. P admits an exact algorithm or a PTAS[1] on graphs of bounded treewidth.
2. There is a polynomial-time procedure that, given a weighted H-minor-free graph G and a fixed parameter $\delta > 0$, computes an H-minor-free graph G' such that
   (a) $\ell(G') \leq \alpha \mathsf{opt}_{G'}$ for some constant $\alpha > 0$, and
   (b) any ρ-approximate solution to G' can be converted to a $[(1+\delta)\rho]$-approximate solution for G in polynomial time.
3. There is a polynomial-time algorithm that, given a subset S of the edges of G and a solution L for the instance G/S, finds a solution L' for G of cost at most $\kappa(L) + \beta\ell(S)$, where $\kappa(L)$ denotes the cost of L and $\beta > 0$ is a fixed constant.

Then, there is a PTAS for problem P on H-minor-free instances.

[1] Prior to our work [BHM10a] on planar Steiner forest, all the algorithms using this framework, to the best of our knowledge, used the premise that the bounded-treewidth instance is solvable in polynomial time. Our work was the first that used the weaker premise—existence of a PTAS—to obtain a PTAS on the planar graphs. Notice that the problem is NP-hard even on bounded-treewidth graphs. Moreover, we emphasize that the stronger version of the theorem does not appear in [DHK11], although the proof is similar.

The PTAS starts by turning G into G' (which is the spanner) using Property 2a. Then, the edges of G' are partitioned into $\zeta = 1/\epsilon$ color classes using Theorem 5.1.3. Let S be the color class with the smallest total length—in particular, $\ell(S) \leq \alpha\mathsf{opt}/\zeta = \alpha\epsilon\mathsf{opt}$. Next, Property 1 allows us to find a solution of cost at most $(1+\epsilon)\mathsf{opt}_{G'/S} \leq (1+\epsilon)\mathsf{opt}_{G'}$ on G'/S, which by Property 3 can be turned into a solution of cost at most $[(1+\epsilon)+\beta\alpha\epsilon]\mathsf{opt}_{G'}$ for G'. Finally, we can turn this into a $(1+f(\alpha,\beta,\delta)\epsilon)$-approximate solution via Property 2b.

The general form of the reduction may look overly complicated, but in this thesis, when we invoke the theorem,

- $\beta = 1$ usually, and $\beta = 2$ for problems involving TSP's;
- $\kappa(L)$ simply denotes the length of the edges in L, i.e., $\ell(L)$;
- G' is always a subgraph of G; and
- any solution for G' is a solution for G, so Property 2b is trivial.

With the above discussion, most of the work in the design of PTASes in this thesis goes into constructing a spanner on which the decomposition theorems can be applied. In a couple of examples, the algorithm for bounded-treewidth instances is not straight-forward, and requires elegant ideas.

Chapter 6

Spanner Construction

Here we give an overview of a spanner construction technique originally proposed by Klein [Kle06, Kle08], and later improved and enhanced in [BKM09, BDT09, BHM10a, BHKM11]. It has been used, among other applications, to give PTASes for Traveling Salesman, Steiner Tree, Steiner Forest, Multiway Cut, etc. on planar instances. The construction is usually performed in four steps:

1. Preprocessing
2. Brick decomposition
3. Portal designation
4. Brick processing

6.1 Preprocessing

The first step is trivial for Traveling Salesman and Steiner Tree; it is a divide-and-conquer procedure based on `PC-Classify` for Steiner Forest, and is an involved procedure consisting of several different pieces, including `PC-Classify`, for Multiway Cut.

The goal of preprocessing is to prepare the instance for the rest of the construction. This is going to be revisited in Section 9.5.1 for Steiner Forest, and in Section 10.2 for Prize-collecting Steiner Tree and Prize-collecting Steiner Forest.

6.2 Brick decomposition

This step involves finding a grid-like subgraph, called the *mortar graph*, that spans the so-called *terminals*—we are not dealing with terminals in Multiway Cut, e.g.—and has length $O(\mathsf{opt})$. The mortar graph is in certain ways similar to the quadtree partitioning used for obtaining PTASes for several Euclidean network design problems. It provides a "grid" that guides the rest of the construction, that is going to happen inside each grid cell, called a *brick*.

Suppose g, denoting the genus of a graph G, is a constant. Let constant $\epsilon > 0$ be a given parameter, and recall that $\mathsf{span}_G(Q)$ denotes the minimum length of a Steiner tree in G that spans all vertices Q. The mortar graph[1] of G with respect to a subset Q of vertices, called *terminals*, and parameter ϵ is a subgraph G_M of G with the following properties, among others:

1. $\ell(G_M) \leq \gamma(\epsilon, g)\mathsf{span}_G(Q)$, where $\gamma(\epsilon, g) = 2(8g+2)(\epsilon^{-1}+1)^2$.
2. A face B of G_M with all edges and vertices of G embedded inside it is called a brick. Every brick is planar.
3. The boundary of each brick B, usually denoted ∂B, consists of four *sides* W, N, E, S in clockwise order; they are coded for the cardinal geographical directions. The total length of all W- and E-boundaries, called *supercolumns*, is at most $\epsilon^2 \cdot \mathsf{span}_G(Q)$.
4. All terminals Q fall on N- or S-boundaries.

[1] The reader can refer to Definition 3.1 of [BDT09] for a complete definition of the mortar graph.

Borradaile, Klein, and Mathieu [BKM09] show how to construct a brick decomposition for any subset Q of vertices of a planar graph G. The genus is one for such a graph, and the guarantee of their construction is slightly better than stated above. However, in this work, we do not intend to optimize the constants, hence, we state the stronger form for bounded-genus graphs due to Borradaile, Demaine, and Tazari [BDT09].

The reader is encouraged to refer to the above works for details of the procedures. We only mention that the running time is $O(n \log n)$ in both cases. We refer to the morar graph procedure by `Brick-Decomposition`(G, T, ϵ).

6.3 Portal designation

Guided by the brick decomposition, we seek to find inside each brick, and add to the spanner, any structure that the optimum may require. In order to do this, though, we first restrict our attention to a well-behaved near-optimum. In particular, we look for solutions that cross the boundary of each brick only a constant number of times. This limited interaction between different bricks allows us to look for simpler structures inside each brick. In fact, we require that a solution may only intersect the boundary of each brick at a small set of predesignated "portal vertices." The claim is that, if the number of portals on each brick is large enough compared to the maximum number of crossings per brick, then the additional cost due to making a solution *portal-respecting* is not too much.

The following theorem ensures that portal designation and focusing on portal-respecting solutions is possible since there exists a near-optimum that intersects the boundary of each brick a small number of times.

Lemma 6.3.1. *[BKM09, Theorem 10.7] Consider a forest F falling inside or on the boundary of a brick B of a brick decomposition. Then, there exists a forest F' with*

the following properties.

1. $\ell(F') \leq (1+\epsilon)\ell(F)$.
2. *F' crosses the boundary of B at most α times, for some constant α depending on ϵ.*
3. *Let X denote the union of N- and S-boundaries of B. Then, any two vertices of X connected by F are also connected in F'.*

For $\theta = 2\gamma\alpha/\epsilon$, we next designate $O(\theta)$ almost-equidistant portals on the boundary of each brick. More specifically, we find[2] a set of portals such that the distance between any vertex on the boundary of a brick B has distance at most $\ell(\partial B)/\theta$ to a portal.

Corollary 6.3.2. *For the same situation as in Lemma 6.3.1, there exists a forest F' of length at most $(1+\epsilon)\ell(F) + \ell(\partial B)\epsilon/2\gamma$ that intersects the boundary of each brick only at its portals.*

Proof. Take the forest F' from Lemma 6.3.1, and connect each crossing point to the closest portal. The additional length is bounded by $\alpha\ell(\partial B)/\theta$. □

6.4 Brick processing

The final step in the construction of the spanner is to include the very structures the optimum may need. Notice that, up to now, we have more or less built a grid and simplified the structure of the optimum. This last step concludes the spanner construction, in much the same way as the base cases of a mathematical induction do the argument there.

For some problems—e.g., Traveling Salesman, Steiner Tree, Steiner Forest—this is relatively simple since all a portal-respecting solution may do inside a brick is provide some connectivity between the portals. In particular, for any

[2] a greedy algorithm suffices.

subset of portals in a brick—notice there are only a constant number of them—we add to the spanner the optimal Steiner tree connecting those portals together. This guarantees the first property of the spanner, i.e., the existence of a good solution in it. For the second condition, i.e., the $O(\mathsf{opt})$ length bound, we note that the length of each Steiner tree added in the brick processing step can be charged to the boundary of the corresponding brick, and a constant number of Steiner trees will be charged to each brick. We will discuss this argument later in more detail.

As for Multiway Cut, the brick processing step is also more involved than what was outlined above. As hinted previously, the structures provisioned inside each brick need to separate certain vertices from each other, hence some digression from the approach that only guarantees connectivity between portals.

6.5 Overview of analysis

The analysis is slightly different for each specific problem. However, we outline the general approach here.

In order to show that the total length of the construction is $O(\mathsf{opt})$, we first of all need the fact that the preprocessing gives us "backbone" graphs whose total length is $O(\mathsf{opt})$. The preprocessing steps for PCST and PCTSP just produces a 2-approximate solution, hence this part is trivial. For planar Steiner Forest or Multiway Cut, this is more complicated. Next, we note that the length of the mortar graph is bounded in terms of the length of the backbone graphs. Finally, inside each brick, for a constant number of configurations (e.g., all subsets of portals), we throw in pieces in the brick processing step such that the length of each piece is no more than the length of the brick. Therefore, the total length of the construction is $O(\mathsf{opt})$.

To show that a near-optimal solution exists in the constructed spanner, we need

some guarantees from the preprocessing step, namely that the loss due to working on different backbone graphs separately is negligible. For each backbone, we include in the solution all supercolumns of the mortar graph; the total length of all supercolumns is small (i.e., $O(\epsilon\mathsf{opt})$). Then, the solution inside each brick is changed via Corollary 6.3.2 to obtain a portal-respecting solution. The loss due to this is bounded by $\epsilon\mathsf{opt} + \epsilon\mathsf{span}(Q)$ if Q is the set of vertices on the backbone. Preprocessing guarantees that the sum of $\mathsf{span}(Q)$ for all backbones is $O(\mathsf{opt})$, hence, the additional length to make the solution portal-respecting is $O(\epsilon\mathsf{opt})$. The part of the solution inside each brick is found optimally in the brick processing step. This guarantees a near-optimal solution in the spanner.

Chapter 7

Granularization

There are algorithms, for certain problems involving integer numbers, that run in *pseudopolynomial* time, rather than polynomial time; i.e., the running time is polynomial in the *numeric* value of the input data which is exponential in its *length* (i.e., its bit representation). For example, testing primality of an input number n can be easily done in pseudopolynomial time, however, the algorithm that achieves this in polynomial time is much more complicated [AKS04].

For other problems—e.g., Knapsack and Subset Sum—the only known algorithms run in pseudopolynomial time; no polynomial-time algorithm has been discovered (and is unlikely to exist). However, a relaxation of the problem admits a polynomial-time algorithm. Consider a Knapsack instance, with items having values and weights, that seeks to maximize the value of a set of items that fit into a knapsack of capacity W. The problem, as such, is NP-hard, so unlikely to have a polynomial-time exact algorithm. Yet, the problem can be relaxed to allow for sets of items whose total weight is no more than $(1+\epsilon)W$ for a fixed constant $\epsilon > 0$. The value of the solution, however, is compared to the optimum among actual feasible solutions with weights no more than W. To solve the relaxed problem, a technique is used that we call *granularization*: the weights of items are first rounded down to

the next multiple of a precision unit θ, and then divided by θ. With a careful choice of θ, one can make sure all weights are polynomial integers, and the error in the computations is conveniently small.

In this chapter, we try to formalize the "granularization" part of the above algorithm, that will be helpful in other applications as well. More specifically, we are interested in a concise representation of a number that is updated via add operations. In particular, the following three operations should be supported.

- $R \leftarrow \mathsf{init}(x)$ for a nonnegative real number x produces a representation R such that $\mathsf{val}(R) = x$ is the intended value.
- $R \leftarrow \mathsf{add}(R_1, R_2)$ for two representations R_1, R_2 outputs a representation R such that $\mathsf{val}(R) = \mathsf{val}(R_1) + \mathsf{val}(R_2)$.
- $x \leftarrow \mathsf{recover}(R)$ for a representation R outputs a real number x. Ideally, $x = \mathsf{val}(R)$, however, since the representations are not usually lossless, we can only guarantee that x is close to $\mathsf{val}(R)$ in some sense.

This is the description of the "value addition" scenario. Based on a standard technique—that we call "static granularization"—we can obtain additive guarantees for error if the number of operations are polynomially bounded; see Section 7.1. We call the granularization "static" because the same precision unit θ is used throughout the algorithm. There are two caveats in this representation: first, we need to set an upper bound on the value R represents—in practice, we can truncate the values since larger values are indistinguishable by the algorithm. Secondly, the additive error can be relatively large for representations of small values.

A special case of the above is the "set union" scenario that we describe below. (We are able to address the issues raised above here.) Let U be a ground set of elements of size $|U|$. Given is a weight function $w : U \mapsto \mathbb{R}^+$. Then, we are interested in a

concise representation of $w(S)$ for sets $S \subseteq U$ obtained via init and union operations below.

- $R \leftarrow \mathsf{init}()$ yields a representation R that has $\mathsf{val}(R) = \emptyset$.
- $R \leftarrow \mathsf{init}(u)$ for an element $u \in U$ produces a representation R such that $\mathsf{val}(R) = \{u\}$ is the intended meaning.
- $R \leftarrow \mathsf{union}(R_1, R_2)$ for representations R_1, R_2 of two disjoint sets, $\mathsf{val}(R_1)$ and $\mathsf{val}(R_2)$, outputs a representation R such that $\mathsf{val}(R) = \mathsf{val}(R_1) \cup \mathsf{val}(R_2)$.
- $x \leftarrow \mathsf{recover}(R)$ for a representation R outputs a real number x. Ideally, $x = w(\mathsf{val}(R))$, however, since the representations are not usually lossless, we can only guarantee that x is close to $w(\mathsf{val}(R))$ in some sense.

Using a technique that we dub "dynamic granularization," we can get a concise representation achieving small multiplicative error, provided that $|U|$ is polynomially small; see Section 7.2. Here we call the granularization "dynamic" because the precision unit changes during the course of the algorithm. It is small for smaller sets, and large for larger sets. The intuition is that, in order to guarantee a fixed multiplicative error, smaller sets need a finer additive representation, hence a smaller precision unit. We have used this technique in the study of Unsplittable Hard-Capacitated Facility Location and Euclidean MPCSF [BH09a, BH10].

7.1 Static granularization

This section uses a simple granularization trick that is widely used in Knapsack, Subset Sum, etc. to prove the following theorem. We give a concise representation for an instance of value addition scenario to approximate the values in the range $[0, \tau]$.

Theorem 7.1.1. *Given n, ϵ, τ, there is a representation for the value addition scenario such that*

1. *R requires $O(\log(n, \epsilon^{-1}))$ bits, and,*

2. *letting $f(x) = \min(x, \tau)$, we have $f(\mathsf{val}(R)) - \epsilon\tau \leq \mathsf{recover}(R) \leq f(\mathsf{val}(R))$ if R is formed by at most n init operations.*

Let $\theta = \frac{\tau\epsilon}{n}$ be the (static) precision unit, and let $\tau_R = \lfloor \tau/\theta \rfloor$. Roughly speaking, the representation R is an integer in the range $[0, \tau_R]$ that is close to $\mathsf{val}(R)/\theta$. More specifically, $R \leftarrow \mathsf{init}(x)$ produces an integer number $R = \min(\lfloor x/\theta \rfloor, \tau_R)$. Further, $R \leftarrow \mathsf{add}(R_1, R_2)$ outputs $R = \min(R_1 + R_2, \tau_R)$. Finally, $\mathsf{recover}(R)$ outputs the real number θR.

Proof of Theorem 7.1.1. In the above representation, R takes integer values in range $[0, \tau_R]$. Notice that $\tau_R \leq \tau/\theta = n/\epsilon$. Therefore, the first condition holds.

We use mathematical induction on the number of init operations used to build R to prove a stronger claim.

Claim 1. *Let R be formed by k init operations. Then, $f(\mathsf{val}(R)) - k\theta \leq \mathsf{recover}(R) \leq f(\mathsf{val}(R))$.*

Proof. We use induction on k. For $k = 1$, consider the corresponding operation $R \leftarrow \mathsf{init}(x)$. By definition, $x' = \mathsf{recover}(R) = \theta \min(\lfloor x/\theta \rfloor, \tau_R)$. Thus, $x' = \min(\theta\lfloor x/\theta \rfloor, \theta\lfloor \tau/\theta \rfloor) \leq \min(x, \tau) = f(x)$ gives the upper bound. For the lower bound, we have $x' = \min(\theta\lfloor x/\theta \rfloor, \theta\lfloor \tau/\theta \rfloor) > \min(x - \theta, \tau - \theta) = \min(x, \tau) - \theta = f(x) - \theta$ as desired.

Now suppose $k > 1$, and let $R \leftarrow \mathsf{add}(R_1, R_2)$ be the last operation. Let k_1, k_2 be the number of init operations used to produce R_1, R_2, respectively. We have $k = k_1 + k_2$ and $k_1, k_2 < k$. The recovered value is

$$\begin{aligned} x' &= \theta \min(R_1 + R_2, \tau_R) \\ &= \min(\theta(R_1 + R_2), \theta\tau_R) \end{aligned}$$

$$
\begin{aligned}
&\leq \min(\theta R_1 + \theta R_2, \tau) \\
&\leq \min(\mathsf{val}(R_1) + \mathsf{val}(R_2), \tau) && \text{by the inductive hypothesis} \\
&= \min(\mathsf{val}(R_1 + R_2), \tau) \\
&= f(R).
\end{aligned}
$$

For the lower bound we have

$$
\begin{aligned}
x' &= \theta \min(R_1 + R_2, \tau_R) \\
&= \min(\theta(R_1 + R_2), \theta\tau_R) \\
&\geq \min(\theta R_1 + \theta R_2, \tau - \theta) \\
&\geq \min(\mathsf{val}(R_1) - k_1\theta + \mathsf{val}(R_2) - k_2\theta, \tau - \theta) && \text{by the inductive hypothesis} \\
&= \min(\mathsf{val}(R_1 + R_2) - k\theta, \tau - \theta) \\
&\geq \min(\mathsf{val}(R_1 + R_2), \tau) - k\theta \\
&= f(R) - k\theta. && \square
\end{aligned}
$$

The Theorem follows by noting that $k \leq n$ bounds the error by $n\theta \leq \epsilon\tau$. $\square$

7.2 Dynamic granularization

Theorem 7.2.1. *Given U, ϵ, w, there is a representation for the value addition scenario for the weight function w on the ground set U such that*

1. *R requires $O(\log(|U|, \epsilon^{-1}))$ bits, and*

2. *$(1 - \epsilon)w(\mathsf{val}(R)) \leq \mathsf{recover}(R) \leq w(\mathsf{val}(R))$.*

In the dynamic granularization, $R = (x, y)$ consists of two parameters. The "base" of R is an element x of U with the largest weight, whereas $y \in \mathbb{Z}^+$, called

the "multiplier" of R, plays a role similar to that of R in the static granularization framework of the previous section. Let $\theta = \epsilon/2|U|$, and have in mind that θ is not used as a fixed precision unit. The precision unit depends on $w(x)$ as well as on θ. In particular, $R = (x, y)$ represents a set of weight close to $y\theta w(x)$.

Construction of the representation is as follows. Let us first assume that the first operation $R \leftarrow \mathsf{init}()$ is not used at all. The operation $R \leftarrow \mathsf{init}(x)$ produces a representation $R = (x, \lfloor 1/\theta \rfloor)$. Further, $R \leftarrow \mathsf{union}(R_1, R_2)$ with $R_1 = (x_1, y_1)$ and $R_2 = (x_2, y_2)$ outputs $R = \min(x, y)$ such that $x = \arg\max_{x' \in \{x_1, x_2\}} w(x')$ is the base with the larger weight, and

$$y = \left\lfloor \frac{y_1 \theta w(x_1) + y_2 \theta w(x_2)}{\theta w(x)} \right\rfloor . \tag{7.1}$$

Finally, $\mathsf{recover}(R)$ for $R = (x, y)$ outputs the real number $y\theta w(x)$.

For the first $R \leftarrow \mathsf{init}()$ operation, we have a special case $R = \emptyset$. The union operations work as follows: $\emptyset = \mathsf{union}(\emptyset, \emptyset)$, $R = \mathsf{union}(\emptyset, R)$, and $R = \mathsf{union}(R, \emptyset)$. We let $0 \leftarrow \mathsf{recover}(\emptyset)$.

Proof of Theorem 7.2.1. We observe that the operations regarding $R = \emptyset$ can be safely ignored. They do not add any precision errors, and only add one more representation, hence the bound on the size follows if we can prove it for the case these operations do not exist. Through the rest of the proof, we have this assumption.

We first prove the following claim about the proposed representation.

Claim 2. *For any representation $R = (x, y)$, the base x is the element of $\mathsf{val}(R)$ of largest weight, and we have*

$$w(\mathsf{val}(R)) - (2|\mathsf{val}(R)| - 1)\theta w(x) \leq y\theta w(x) \leq w(\mathsf{val}(R)).$$

Proof. We use induction on $k = |\mathsf{val}(R)|$. For $k = 1$, consider the corresponding

operation $R \leftarrow \mathsf{init}(x)$ with $R = (x, y)$. Clearly, x is the element of largest weight. The definition $y = \lfloor 1/\theta \rfloor$ implies $1 - \theta < y\theta \leq 1$, which proves the claim since $\mathsf{val}(R) = \{x\}$ in this case.

Now suppose $k > 1$, and let $R \leftarrow \mathsf{add}(R_1, R_2)$ be the last operation. Suppose we have $R = (x, y), R_1 = (x_1, y_1), R_2 = (x_2, y_2)$. The definition guarantees that x is the largest element of $\mathsf{val}(R) = \mathsf{val}(R_1) \cup \mathsf{val}(R_2)$. The inductive hypothesis gives

$$y_1\theta w(x_1) + y_2\theta w(x_2) \leq w(\mathsf{val}(R_1)) + w(\mathsf{val}(R_2)) = w(\mathsf{val}(R)),$$

where the equality follows from additivity of w since R is the disjoint union of R_1, R_2. Thus, $\mathsf{recover}(R) = y\theta w(x) \leq w(\mathsf{val}(R))$.

The inductive hypothesis gives

$$\begin{aligned} y_1\theta w(x_1) + y_2\theta w(x_2) &\geq w(\mathsf{val}(R_1)) - (2|\mathsf{val}(R_1)| - 1)\theta w(x_1) \\ &\quad + w(\mathsf{val}(R_2)) - (2|\mathsf{val}(R_2)| - 1)\theta w(x_2) \\ &\geq w(\mathsf{val}(R_1)) + w(\mathsf{val}(R_2)) - (2|\mathsf{val}(R)| - 1)\theta w(x) \end{aligned}$$

since R is the disjoint union of R_1 and R_2 and $w(x) = \max(w(x_1), w(x_2))$,

$$= w(\mathsf{val}(R)) - (2|\mathsf{val}(R)| - 1)\theta w(x). \qquad \square$$

The above claim immediately gives the desired accuracy in the theorem since, for $x \in \mathsf{val}(R)$ we have $(2|\mathsf{val}(R)| - 1)\theta w(x) \leq \epsilon w(\mathsf{val}(R))$ by definition of θ and noticing $w(x) \leq w(\mathsf{val}(R))$.

We finally study the size of the representation. The above claim guarantees that x is the largest-weight element of $\mathsf{val}(R)$ if $R = (x, y)$. Thus, $w(\mathsf{val}(R)) \leq |\mathsf{val}(R)|w(x)$. Then, the upper bound from the claim gives $y \leq w(\mathsf{val}(R))/\theta w(x) \leq |\mathsf{val}(R)|/\theta \leq |U|/\theta = |U|^2/\epsilon$. Therefore, the representation of R requires no more

than $O(\log(|U|^3/\epsilon))$ bits. $\square$

We emphasize once more that the dynamic granularization does not need the truncation at τ, and it provides a stronger guarantee whose error is multiplicative, but works only for the special case of set union scenarios. The multiplicative error is more desirable since two values represented as such can also be multiplied, whereas with additive errors multiplication can introduce an unbearable error.

We finally remark that the dynamic granularization is reminiscent of the standard floating-point representation of real numbers: there a number x is represented (possibly with some error) via two numbers, a "mantissa" y and an "exponent" z, with $x = y2^z$. In this representation, too, there is a normalization that ensures the mantissa is a decimal number in the range $[1, 2)$. The multiplicative precision error for x is then only dependent on the size of the mantissa, provided that x is within the range covered by the exponent size.

Part III

Applications

Chapter 8

Prize-collecting Steiner Tree and TSP

The material in this chapter is based on a joint work with Aaron Archer, MohammadTaghi Hajiaghayi, and Howard Karloff [ABHK09, ABHK11]. For the first time, after 17 years of failed attempts, we present approximation algorithms for Prize-collecting Steiner Tree and Prize-collecting TSP whose performance guarantees are constants smaller than two. This proves Theorem 3.2.1.

We structure the rest of the chapter as follows. Section 8.1 overviews the literature on PCST, PCTSP, and related problems. Then, the outline of the algorithm is presented in Section 8.2. Our proposed algorithm produces two candidate solutions and outputs the better of the two. Section 8.3 discusses the first candidate solution along with a bad instance that motivates the second candidate solution. In Section 8.4 we formally describe our algorithm for PCST, and prove the approximation ratio of $2 - \epsilon$. Section 8.5 applies the same techniques to derive a $(2 - \epsilon)$-approximation algorithm for PCTSP, and Section 8.6 does the same for PC-Path-1 and PC-Path-2.

8.1 Background

Recall that in all these problems we are given a connected undirected graph $G = (V, E)$, a nonnegative length function $\ell : E \mapsto \mathbb{R}^+$ on edges, denoting the cost to purchase that edge, and a nonnegative penalty (sometimes called the prize) function $\pi : V \mapsto \mathbb{R}^+$ for vertices. Recall that, given any subset E' of edges, we say E' spans the set $V(E')$ of vertices incident to edges in E'. In Prize-collecting Steiner Tree, we must select an edge set T such that $(V(T), T)$ is a tree, so as to minimize the combined cost

$$\ell(T) + \pi(\overline{V(T)}).$$

That is, we aim to minimize the total edge length of the tree plus the penalties of the vertices it does not span. Prize-collecting TSP is the same as Prize-collecting Steiner Tree, except that the set of edges should form a cycle instead of a tree. We also study the Prize-collecting Path (PC-Path) problem, in which the edges should form a path. In some of the literature, PC-Path is instead called Prize-collecting Stroll [CGRT03].

As mentioned in Section 2.3, even though a tree is usually defined as a graph $(V(T), T)$, we will slightly abuse terminology by referring to its edge set T as a tree. The same goes for cycles, paths, and forests.

The first approximation algorithms for the PCST and PCTSP problems were given by Bienstock et al. [BGSLW93], although PCTSP had been introduced earlier by Balas [Bal89]. Bienstock et al. achieved a factor of 3 for PCST and 2.5 for PCTSP by rounding the optimal solution to a linear-programming (LP) relaxation. Later, Goemans and Williamson [GW95] constructed primal-dual algorithms using the same LP relaxation to obtain a 2-approximation for both problems, building on work of Agrawal, Klein and Ravi [AKR95]. Chaudhuri et al. modified the Goemans-Williamson algorithm to achieve a 2-approximation for PC-Path [CGRT03]. There

has been a long dry spell with no improved approximation algorithms for PCST or PCTSP since the Goemans-Williamson results first appeared in 1992 [GW92]. Our work gives $(2-\epsilon)$-approximation algorithms for both problems, and for PC-Path as well.

By contrast, improvements for the ordinary Steiner Tree problem came rapidly after 1990, when Zelikovsky's 11/6-approximation for Steiner Tree became the first algorithm to beat the naïve 2-approximation [Zel92, Zel93]. Berman and Ramaiyer's improvement of Zelikovsky's $11/6 \approx 1.833$ bound to 1.746 appeared in 1992 [BR92]. The bound was improved to 1.693 by Zelikovsky himself in 1996 [Zel96], then to 1.667 by Prömel and Steger in 1997 [PS97], to 1.644 by Karpinski and Zelikovsky in 1997 [KZ97], to 1.598 by Hougardy and Prömel in 1999 [HP99], and to 1.550 (actually, $1 + \frac{1}{2}\ln 3 + \epsilon$, for any constant ϵ) by Robins and Zelikovsky in 2000 [RZ00, RZ05]. Most recently, Byrka et al. improved the bound to 1.387 (actually, $\ln 4 + \epsilon$, for any constant ϵ) in 2010 [BGRS10].

We hope for our work to trigger similar improvements for PCST, PCTSP, and PC-Path. Indeed, this process has already begun, with Goemans improving the ratio for PCTSP below 1.915 [Goe09].

Although the value of ϵ that we achieve in our $(2-\epsilon)$-approximations is small (less than 0.04 in all three cases), this is a conceptual breakthrough since the factor 2 was thought to be a barrier, at least for PCST. The natural LP relaxation for PCST used in [BGSLW93, GW95] is known to have an integrality gap of 2, even for the ordinary Steiner Tree problem. Thus, it cannot by itself provide a strong enough lower bound to prove an approximation factor better than 2. Since the preliminary version of this work appeared [ABHK09], Byrka et al. have shown a much more complicated LP relaxation to have an integrality gap of at most 1.550 for the ordinary Steiner Tree problem, and their 1.387-approximation is based primarily on rounding this LP [BGRS10]. However, their work has yet to lead to an improved approximation

ratio for PCST, other than via our use of their Steiner Tree algorithm as a black box, as we describe later.

PCST and PCTSP are two of the classic optimization problems, deserving of study in their own right. Moreover, work on the PCST has had a large impact, both in theory and practice. At AT&T, PCST code has been used in large-scale studies in access network design, both as described in Johnson, Minkoff and Phillips [JMP00], and in other unpublished applied work.

The impact of PCST within approximation algorithms is also far-reaching. As noted by Chudak, Roughgarden and Williamson [CRW04], PCST is a Lagrangian relaxation of the k-MST problem, which asks for the minimum-length tree spanning at least k vertices. Moreover, they note that the PCST algorithm of Goemans and Williamson (which we call `GW`) is not merely a 2-approximation algorithm, but rather a Lagrangian-preserving 2-approximation (we give a formal definition shortly). This implies the useful property that the total edge length of the tree T returned by `GW` is no more than twice the edge length of the cheapest tree that pays at most $\pi(\overline{V(T)})$ penalty. Suppose that one runs `GW` with all penalties for each vertex set to some constant λ, and it happens to return a tree T spanning exactly k vertices. Since the set of all trees that pay at most $\pi(\overline{V(T)}) = \lambda(|V| - k)$ penalty is precisely the set of trees spanning at least k vertices, T is a 2-approximate k-MST. This property has been used in a sequence of papers [Gar96, AR98, AK06] culminating in a 2-approximation algorithm for k-MST by Garg [Gar05]. It has also been used to improve the approximation ratio and running time of algorithms for the Minimum Latency problem [ALW08, CGRT03, AB10]. The technique of applying Lagrangian relaxation to a problem with hard constraints, then using a Lagrangian-preserving approximation algorithm on the relaxed problem to recover an approximation algorithm for the original problem, has been successfully applied to the k-Median and Uncapacitated Facility Location problems as well, in a sequence of papers

starting with Jain and Vazirani [JV01].

8.2 Overview of the algorithm

In this chapter, we use the Lagrangian-preserving guarantee of the GW algorithm in a different way. Tree T is a *Lagrangian-preserving* 2-approximate solution if

$$\ell(T) + 2\pi(\overline{V(T)}) \leq 2\ell(O) + 2\pi(\overline{V(O)}) = 2\mathsf{opt}, \tag{8.1}$$

where O is the optimal tree. To be an ordinary 2-approximation, it would suffice to satisfy (8.1) with the factor 2 on the left side changed to 1. Our Corollary 8.3.2 uses cost scaling to transfer the unbalanced approximation guarantee to the right side, as Charikar and Guha did for Uncapacitated Facility Location [CG05]. This transforms (8.1) into

$$\ell(T) + \pi(\overline{V(T)}) \leq 2\ell(O) + \pi(\overline{V(O)}) = 2\mathsf{opt} - \pi(\overline{V(O)}). \tag{8.2}$$

One way to view guarantees (8.1) and (8.2) is that GW essentially achieves an approximation ratio of 1 on the penalty term, and 2 on the tree term. Thus, if a constant fraction of opt comes from the penalty term, i.e., $\pi(\overline{V(O)}) \geq \epsilon\mathsf{opt}$, then GW achieves a $(2-\epsilon)$-approximation (when run on an instance with costs scaled appropriately). This frees us to search for a second solution that is guaranteed to be good, provided that $\pi(\overline{V(O)}) \leq \epsilon\mathsf{opt}$. Our algorithm can then return the better of the two solutions.

Although most do not explicitly work this way, we can reinterpret any PCST algorithm as making two decisions in sequence: first it decides what subset of vertices $S \subseteq V$ to span, then it constructs a tree T that spans exactly S, i.e., $V(T) = S$. There are three ways in which the algorithm could perform poorly on a given instance. First, the set of vertices it chooses to exclude from the tree could contribute too much

penalty relative to opt, i.e., $\pi(\overline{V(S)})$ is too large. Second, it could keep the penalty $\pi(\overline{V(S)})$ low, but its tree T could pay too much to span S. Since there are efficient algorithms for the Minimum Spanning Tree (MST) problem, this second pitfall is easily avoided by simply choosing T to be the MST of S. The third potential pitfall is that the algorithm chooses a set S such that even the MST of S is too expensive. Of course, most algorithms do not actually select S and T in sequence, but rather build a tree T that implicitly defines $S = V(T)$. While postprocessing the tree by taking the MST of S can avoid the second pitfall in practice, this trick is typically of no use in proving approximation guarantees since the usual way of showing that we have avoided the third pitfall is to build a provably inexpensive tree to begin with.

We adopt the following high-level strategy for generating our second solution. We will avoid the first pitfall by using GW to select a preliminary set S of vertices to span. To avoid the third pitfall, we will use an algorithm for the ordinary Steiner Tree problem to possibly enlarge S, while producing a cheap tree T spanning the new S (even if T is not the MST of S).

Suppose we can identify a set Q of vertices such that $\pi(Q)$ is a small fraction of opt, and we can prove that there is a tree that spans at least $V \setminus Q$ and costs only slightly more than opt. Now suppose we run any ρ-approximation algorithm ST for the ordinary Steiner Tree problem (with $\rho < 2$) on the instance in which the vertices $V \setminus Q$ are terminals. Then, the resulting tree will cost not much more than $\rho\,\mathsf{opt}$, and, since it spans at least $V \setminus Q$, it pays at most $\pi(Q)$ in penalties. Thus, this tree will be a $(2 - \epsilon)$-approximation, provided the hypothesized bounds related to Q are strong enough.

The key insight of our algorithm is that we can identify just such a set Q, provided that $\pi(\overline{V(O)})$ is small, which is precisely the case where we need our second solution to do well. Interestingly, we use the GW algorithm—in the proof of Superclustering Theorem—to compute Q before passing $V \setminus Q$ as terminals to a better Steiner Tree

algorithm ST. We use the algorithm ST as a black box: for the purpose of achieving a $(2-\epsilon)$ approximation factor for PCST, any ρ-approximation for ordinary Steiner Tree with $\rho < 2$ suffices, and, the smaller the value of ρ, the smaller the value we can prove for $2-\epsilon$ (see Theorem 8.4.4). We emphasize that in selecting the terminal set to send to ST, we do *not* simply use the set of vertices that GW chose to span, but rather a carefully chosen subset.

The key to this scheme is to prove the two desired properties of Q described above. Since PC-Superclustering identifies the set Q, naturally the proof heavily relies on Theorem 4.3.1. The upper bound on $\pi(Q)$ comes through Lemma 8.4.1 and Theorem 4.3.1 via LP duality, using the dual variables naturally generated while running PC-Superclustering. The deeper result, though, is the application of Theorem 4.3.1 in Lemma 8.4.2, which guarantee that there exists an inexpensive way to augment the optimal tree O to span whatever additional vertices are in $V \setminus Q$.

One novel feature of our analysis is that we use the details of the GW moat-growing algorithm to prove in Theorem 4.3.1 and Lemma 8.4.2 that there *exists* an inexpensive set of augmenting edges, but not to find it. All other applications of GW of which we are aware use it purely in an algorithmic sense to identify edges; this may be the first time it has been utilized solely to provide an existence proof.

Our algorithm for PCTSP is very similar, except that our black box is an approximation algorithm for TSP such as the $\frac{3}{2}$-approximation of Christofides [Chr76] instead of ST. For PC-Path-1 and PC-Path-2, our black boxes are approximation algorithms for the corresponding versions of Path-TSP, such as the algorithms of Hoogeveen, a $\frac{3}{2}$-approximation for Path-TSP1 and a $\frac{5}{3}$-approximation for Path-TSP2 [Hoo91].

8.3 A good case, and a motivating bad example

Our starting point is the following theorem regarding GW.

Theorem 8.3.1 ([FFFdP10, CRW04, GW95]). *The GW algorithm for* PCST *returns a tree T that satisfies*

$$\ell(T) + 2\pi(\overline{V(T)}) \leq 2\ell(O) + 2\pi(\overline{V(O)}) = 2\mathsf{opt}. \tag{8.3}$$

As before, O denotes the optimal tree. Notice that (8.3) is stronger than what would be necessary for GW to be a 2-approximation: there is a factor of 2 in front of $\pi(\overline{V(T)})$, where a 1 would suffice. Because of this property, GW is said to be a *Lagrangian-preserving* 2-approximation algorithm.[1]

We can now apply cost scaling to move the unbalanced approximation factor to the other side of the inequality. For all $\beta > 0$, let $\mathcal{I}_\beta$ denote the PCST instance derived from $\mathcal{I}$ by multiplying all penalties by β, and let opt_β denote the optimal objective value for this scaled instance. Thus, $\mathcal{I}_1 = \mathcal{I}$ and $\mathsf{opt}_1 = \mathsf{opt}$. Let $T^{\mathtt{GW}}$ denote the tree returned by GW on instance $\mathcal{I}_{\frac{1}{2}}$. Also, let δ satisfy $\pi(\overline{V(O)}) = \delta\mathsf{opt}$, i.e., δ denotes the fraction of opt contributed by the penalty term in the optimal solution to $\mathcal{I}$.

Corollary 8.3.2. *T^{GW} satisfies*

$$\ell(T^{GW}) + \pi(\overline{T^{GW}}) \leq 2\ell(O) + \pi(\overline{V(O)}) = (2-\delta)\mathsf{opt}. \tag{8.4}$$

[1] The term arises because of its connections to Lagrangian relaxation, which we will not explore here. Theorem 8.3.1 appears in exactly this form in [FFFdP10], which applies to the Johnson-Minkoff-Phillips variant of GW, exactly as we described it in Section 4.1.2. We also cite Chudak, Roughgarden and Williamson [CRW04] because they were the first to observe that the original version of the Goemans-Williamson algorithm from [GW95] already achieves the Lagrangian-preserving approximation guarantee, even though [GW95] claims only the ordinary 2-approximation. We also credit Goemans and Williamson [GW95] because they actually implicitly proved the stronger result, even though they neglected to make the stronger claim.

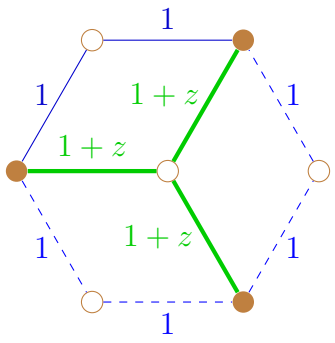


Figure 8.1: Bad example for GW, with $k = 3$. The k solid vertices are terminals, while the $k + 1$ hollow ones are Steiner vertices. The k bold spoke edges emanating from the hub vertex represent the optimal tree, of cost $k(1 + z)$, while the $2k - 2$ dashed edges represent the tree output by GW, of cost $2k - 2$.

Proof. Applying Theorem 8.3.1 to instance $\mathcal{I}_{\frac{1}{2}}$ yields

$$\begin{aligned}\ell(T^{\texttt{GW}}) + 2 \cdot \frac{1}{2}\pi(\overline{T^{\texttt{GW}}}) &\leq 2\mathsf{opt}_{\frac{1}{2}} \\ &\leq 2(\ell(O) + \frac{1}{2}\pi(\overline{V(O)})) = 2\ell(O) + \pi(\overline{V(O)}) \\ &= 2\mathsf{opt} - \pi(\overline{V(O)}) = (2 - \delta)\mathsf{opt},\end{aligned}$$

the second inequality holding because O is a feasible solution, even though it may not be optimal for $\mathcal{I}_{\frac{1}{2}}$. □

Thus, if δ is at least a constant ϵ, then $T^{\texttt{GW}}$ is a $(2-\epsilon)$-approximation. This accords with the intuition that the GW algorithm deals especially well with the penalty term. A naïve idea for obtaining a $(2-\epsilon)$-approximation algorithm when $\delta < \epsilon$ is to run GW (possibly after scaling the penalties) to obtain a tree T, then designate the vertex set $V(T)$ as the set of terminals and run a better Steiner Tree algorithm ST on that ordinary Steiner Tree instance. This does not work, and it is instructive to see an example of what goes wrong.

Given $k \geq 2$ and small but positive z, we construct an instance of the ordinary Steiner Tree problem, cast as a special case of PCST, where all penalties are

either 0 or ∞. Thus, scaling the penalties has no effect. Moreover, since opt cannot afford to pay an infinite penalty and all of the finite penalties are zero, we have $\pi(\overline{V(O)}) = 0$, so $\delta = 0$. Figure 8.1 depicts the instance for $k = 3$. There is a $2k$-cycle, alternating between terminals and Steiner vertices, connected by edges of cost 1. There is one *hub* Steiner vertex in the middle, with *spoke* edges of cost $1 + z$ to each of the terminals in the cycle. In this case, the tree T produced by GW is a path containing all cycle edges except for the pair on either side of one of the Steiner vertices. It costs $2k - 2$ and spans all but two Steiner vertices: one on the cycle, plus the hub vertex. Meanwhile, opt buys only the k spoke edges at total cost $k(1 + z)$. By contrast, it spans only one Steiner vertex, the hub. The ratio of these costs goes to 2 as $k \to \infty$ and $z \to 0$. Although $\ell(T)$ is essentially twice opt, T actually spans the set $V(T)$ *optimally*, even allowing ourselves to use the two vertices in $\overline{V(T)}$ as Steiner vertices. Thus, running ST cannot help. A better idea is required.

8.4 Our PCST algorithm

We now describe our PCST algorithm $\texttt{PCST}_\beta$, which is parametrized by a constant $\beta \geq 1$ to be optimized later. See Algorithm 8.1 for pseudocode. The algorithm is simple to describe, but its analysis is quite involved—most of the complication of proof is hidden in the proof of Theorem 4.3.1. We produce two trees and output the better one. As hinted in Section 8.3, the first tree is $T^{\texttt{GW}}$, the output of GW on instance $\mathcal{I}_{\frac{1}{2}}$. For the second tree, we identify a set of terminals defining an instance of the ordinary Steiner Tree problem, and run ST on it to generate a second solution $T^{\texttt{ST}}$. The bad example of Section 8.3 shows that we cannot simply take the set of vertices that GW spans as our terminal set. Instead, we run PC-Superclustering on $\mathcal{I}_\beta$ to obtain the set Q of exempt vertices. Then, we send the terminal set $V \setminus Q$ to ST. (Notice that Q is in fact the union of all clusters that ever died during PC-MoatGrowing run

Algorithm 8.1 $\texttt{PCST}_\beta(G, \pi, r)$

Input: graph $G(V, E)$, root vertex r, and penalties $\pi(v) \geq 0$
Output: tree T

1: Let $T^{\texttt{GW}} \leftarrow \texttt{GW}(G, \frac{1}{2}\pi, r)$ # run GW on $\mathcal{I}_{\frac{1}{2}}$
2: Let $(Q, T) \leftarrow \texttt{PC-Superclustering}(G, \beta\pi, r)$
3: Let $T^{\texttt{ST}} \leftarrow \texttt{ST}(G, V \setminus Q)$ # run ST on G with terminal set $V \setminus Q$
4: Let $T \leftarrow \arg\min_{T \in \{T^{\texttt{GW}}, T^{\texttt{ST}}\}} (\ell(T) + \pi(\overline{V(T)}))$
5: **return** T

on $\mathcal{I}_\beta$.) This concludes the description of algorithm $\texttt{PCST}_\beta$.[2]

Notice that choosing $V \setminus Q$ instead of $V(T)$ defeats the bad example from Section 8.3 because in that case Q is the set of all Steiner vertices, so excising Q gets rid of the $k - 1$ Steiner vertices on the cycle that had been part of $V(T)$ and were causing trouble.

Recall that δ is defined so that $\pi(\overline{V(O)}) = \delta\mathsf{opt}$, where O denotes the optimal tree. Since we have control over β but not over δ, our analysis can select the best possible β but must assume the worst δ. Lemma 8.3.2 shows that, if $\delta \geq \epsilon$ for some fixed $\epsilon > 0$, then $T^{\texttt{GW}}$ already provides a $(2-\epsilon)$-approximation. The big work is in showing that when $\delta < \epsilon$, the second solution, $T^{\texttt{ST}}$, provides a $(2 - \epsilon)$-approximation. The main ingredients of the proof, Lemmas 8.4.1 and 8.4.2, follow from Theorem 4.3.1.

Since $T^{\texttt{ST}}$ spans at least the terminal set $V \setminus Q$, it pays at most $\pi(Q)$ in penalty cost. Thus, we need to bound $\pi(Q)$. Lemma 8.4.1 shows that $\pi(Q)$ can be made negligible by choosing β large enough.

Lemma 8.4.1. *For the set Q in $\texttt{PCST}_\beta$,*

$$\pi(Q) \leq \left(\frac{1-\delta}{\beta} + \delta\right)\mathsf{opt}.$$

Proof. Recall that Q is the set of exempt vertices from PC-Superclustering run on instance $\mathcal{I}_\beta$. Hence, the first guarantee of Theorem 4.3.1 gives $\phi(Q) = \beta\pi(Q) \leq \mathsf{opt}_\beta$.

[2] Although $T^{\texttt{ST}}$ depends on β and $\texttt{PCST}_\beta$ depends on ST, we suppress this in the notation since we view β and ST as fixed.

Therefore,

$$\beta\pi(Q) \leq \mathsf{opt}_\beta \tag{8.5}$$

$$\leq \ell(O) + \beta\pi(\overline{V(O)}) \tag{8.6}$$

$$= (1-\delta)\mathsf{opt} + \beta\delta\mathsf{opt}, \tag{8.7}$$

where (8.6) follows because O is a feasible solution to $\mathcal{I}_\beta$, and (8.7) follows by the definition of δ. Rearranging terms yields the desired result. □

To bound $\ell(T^{\mathtt{ST}})$, we use the following key (nonalgorithmic) fact: starting from the optimal tree O, the *marginal* cost of extending this tree to connect to the terminals in $V \setminus Q \setminus V(O)$ is no larger than $(2\beta\delta)\mathsf{opt}$. This guarantees existentially that there is a tree that spans at least $V \setminus Q$ and is not much more expensive than opt since δ is small. Hence $\mathtt{ST}$ won't pay too much when fed terminal set $V \setminus Q$.

Lemma 8.4.2. *There exists a tree that costs at most* $(1+(2\beta-1)\delta)\mathsf{opt}$ *and spans at least* $V \setminus Q$.

Proof. We can apply second guarantee of Theorem 4.3.1 with $I = V(O)$ because the optimal tree O must include the root r. This yields a forest F with

$$\ell(F) \leq 2\phi(\overline{V(O)}) = 2(\beta\pi(\overline{V(O)})) = 2\beta(\delta\mathsf{opt}).$$

Augmenting the optimal tree O with F yields a tree spanning at least $V \setminus Q$ and possibly part of Q as well. Since $\ell(O) = (1-\delta)\mathsf{opt}$, we obtain $\ell(O \cup F) \leq (1+(2\beta-1)\delta)\mathsf{opt}$. □

It is critical for the sake of the above proof that we are considering the *rooted* version of PCST because this allows us to assume that $r \in V(O)$.

All that remains is to put our bounds together, and optimize the choice of β as a function of ρ.

Corollary 8.4.3. *The second solution T^{ST} in $\texttt{PCST}_\beta$ costs at most*

$$\left(\rho(1+(2\beta-1)\delta)+\frac{1-\delta}{\beta}+\delta\right)\mathsf{opt},$$

where ρ is the approximation ratio of algorithm $\texttt{ST}$ for Steiner Tree.

Proof. Lemma 8.4.2 proves that there exists a Steiner tree spanning at least terminal set $V \setminus Q$ and costing at most $(1+(2\beta-1)\delta)\mathsf{opt}$. Since $\texttt{ST}$ is a ρ-approximation algorithm, we have

$$\ell(T^{\texttt{ST}}) \leq \rho(1+(2\beta-1)\delta)\mathsf{opt}.$$

Since $T^{\texttt{ST}}$ spans at least $V \setminus Q$,

$$\pi(\overline{T^{\texttt{ST}}}) \leq \pi(Q) \leq \left(\frac{1-\delta}{\beta}+\delta\right)\mathsf{opt}$$

by Lemma 8.4.1. Summing these bounds yields the result. □

It is now a simple matter to prove the approximation ratio of our algorithm.

Theorem 8.4.4. *If the approximation ratio of the* Steiner Tree *algorithm $\texttt{ST}$ is $\rho < 2$, then $\texttt{PCST}_\beta$ with $\beta = \frac{2}{2-\rho}$ achieves an approximation ratio of at most $2-(\frac{2-\rho}{2+\rho})^2 < 2$ for* PCST.

Proof. Set $\beta = \frac{2}{2-\rho}$. Let $B^{\texttt{GW}} = 2-\delta$ and $B^{\texttt{ST}} = \rho(1+(2\beta-1)\delta)+\frac{1-\delta}{\beta}+\delta$ be the approximation ratios achieved by solutions $T^{\texttt{GW}}$ and $T^{\texttt{ST}}$, as proven by Corollaries 8.3.2 and 8.4.3, respectively. If $\delta \geq (\frac{2-\rho}{2+\rho})^2$, then $B^{\texttt{GW}} \leq 2-(\frac{2-\rho}{2+\rho})^2$, as desired. Otherwise,

$\delta < (\frac{2-\rho}{2+\rho})^2$, so

$$\begin{aligned}
B^{\mathtt{ST}} &= \rho + \frac{1}{\beta} + \delta(1 - \frac{1}{\beta} + \rho(2\beta - 1)) \\
&< \rho + \frac{2-\rho}{2} + \left(\frac{2-\rho}{2+\rho}\right)^2 \left(1 - \frac{2-\rho}{2} + \rho\left(\frac{4}{2-\rho} - 1\right)\right) \\
&= 1 + \frac{\rho}{2} + \left(\frac{2-\rho}{2+\rho}\right)^2 \left(-\frac{\rho}{2} + \frac{4\rho}{2-\rho}\right) \\
&= 1 + \frac{\rho}{2}\left(1 + \frac{(2-\rho)}{(2+\rho)^2}(-(2-\rho) + 8)\right) \\
&= 1 + \frac{\rho}{2(2+\rho)^2}\left((2+\rho)^2 + (2-\rho)(6+\rho)\right) \\
&= 1 + \frac{8\rho}{(2+\rho)^2} \\
&= 2 + \frac{8\rho - (2+\rho)^2}{(2+\rho)^2} \\
&= 2 - \left(\frac{2-\rho}{2+\rho}\right)^2,
\end{aligned}$$

as desired. Hence, the better of $T^{\mathtt{GW}}$ and $T^{\mathtt{ST}}$ gives an approximation ratio of at most $2 - (\frac{2-\rho}{2+\rho})^2$. □

The current best approximation algorithm for Steiner Tree, due to Byrka et al., achieves an approximation ratio arbitrarily close to $\ln 4 \leq 1.386295$ [BGRS10]. Using this value of ρ and $\beta = \frac{2}{2-\rho} \approx 3.258891$ yields an approximation ratio of less than 1.967155 for $\mathtt{PCST}_\beta$. Table 8.1 shows the approximation ratio obtained by $\mathtt{PCST}_\beta$, using several different Steiner Tree algorithms as our black box, ST.

We have not discovered a family of instances on which $\mathtt{PCST}_1$ fails to achieve an approximation ratio strictly below 2. Hence, we do not know whether it is strictly necessary to take $\beta > 1$. The bound on the tree length $\ell(T^{\mathtt{ST}})$ resulting from Lemma 8.4.2 gets worse as β increases, so it would be nice if we could take $\beta = 1$. However, we have found no better way to bound $\pi(\overline{T^{\mathtt{ST}}})$ than by using Lemma 8.4.1, which is why our algorithm must use a moderately large value of β. Finding a better way to sup-

Table 8.1: Approximation ratios implied by Theorem 8.4.4 for $\texttt{PCST}_\beta$ with $\beta = \frac{2}{2-\rho}$, for various algorithms $\texttt{ST}$, rounded up to the nearest 0.0001. The algorithm that returns the optimal Steiner tree is denoted $\texttt{opt}_{\texttt{ST}}$.

$\texttt{ST}$	ρ	apx. ratio of $\texttt{PCST}_\beta$
$\texttt{opt}_{\texttt{ST}}$	1	$\frac{17}{9} \leq 1.8889$
Byrka et al. [BGRS10]	$\ln 4 + \epsilon < 1.387$	1.9672
Robins and Zelikovsky [RZ05]	$1 + \frac{1}{2}\ln 3 + \epsilon < 1.550$	1.9839
Zelikovsky [Zel93]	$\frac{11}{6}$	$\frac{1057}{529} < 1.9982$

Algorithm 8.2 $\texttt{PCTSP}_\beta(G, \pi, r)$

Input: graph $G(V, E)$, root vertex r, and penalties $\pi(v) \geq 0$
Output: tour C

1: Let $C^{\texttt{GW}} \leftarrow$ output of Goemans-Williamson PCTSP algorithm on instance $\mathcal{I}_{\frac{1}{2}}$
2: Let $(Q, T) \leftarrow \texttt{PC-Superclustering}(G, \frac{\beta}{2}\pi, r)$
3: Let $C^{\texttt{TSP}} \leftarrow$ output of algorithm $\texttt{TSP}$ for Traveling Salesman instance $\mathcal{I}$ restricted to vertices $V \setminus Q$
4: Let $C \leftarrow \arg\min_{C \in \{C^{\texttt{GW}}, C^{\texttt{TSP}}\}} (\ell(C) + \pi(\overline{V(C)}))$
5: **return** C

press the penalty cost might lead to a substantial improvement in the approximation guarantee.

8.5 Our PCTSP algorithm

Our algorithm for PCTSP, called $\texttt{PCTSP}_\beta$, is nearly identical to $\texttt{PCST}_\beta$. See Algorithm 8.3 for pseudocode. Let $\mathcal{I}$ denote the PCTSP instance we wish to solve, and $\mathcal{I}_\beta$ be the same instance except with all penalties multiplied by β. First, we run the PCTSP algorithm of Goemans and Williamson [GW95] on $\mathcal{I}_{\frac{1}{2}}$ to generate a cycle $C^{\texttt{GW}}$. We then run $\texttt{PC-Superclustering}(G, \frac{\beta}{2}\pi, r)$ to find Q. Next we run any approximation algorithm $\texttt{TSP}$ for the ordinary Traveling Salesman problem on the vertex set $V \setminus Q$ to generate a second solution $C^{\texttt{TSP}}$. We then output the better of $C^{\texttt{GW}}$ and $C^{\texttt{TSP}}$.

The analysis of this algorithm hews very closely to the analysis of $\texttt{PCST}_\beta$. Let

O now denote the optimal PCTSP solution, opt its objective function value, and δ the fraction of opt contributed by the penalty $\pi(\overline{V(O)})$. First, there is an analog of Theorem 8.3.1.

Theorem 8.5.1 ([GW95]). *The* PCTSP *algorithm of Goemans and Williamson returns a tour C such that*

$$\ell(C) + 2\pi(\overline{V(C)}) \leq 2\ell(O) + 2\pi(\overline{V(O)}) = 2\mathsf{opt}.$$

The analog of Corollary 8.3.2 follows in exactly the same way, showing that $C^{\mathtt{GW}}$ is a $(2-\delta)$-approximation.

Next, we must bound $\pi(Q)$. The LP relaxation (4.1) that we used for PCST can be easily modified to model PCTSP, as follows.

$$\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} \ell(e) x_e + \sum_{v \in V} \pi(v) z_v & & && (8.8)\\
\text{subject to} \quad & \sum_{e \in \delta(S)} x_e \geq 2 - 2z_v & & \forall v \in S \subseteq V \setminus \{r\} && (8.9)\\
& x_e \geq 0 & & \forall e \in E && (8.10)\\
& z_v \geq 0 & & \forall v \in V, && (8.11)
\end{aligned}$$

whose dual is

$$\begin{aligned}
\text{maximize} \quad & 2 \sum_{v \in S \subseteq V \setminus \{r\}} y_{S,v} & & && (8.12)\\
\text{subject to} \quad & \sum_{\substack{S \subseteq V \\ e \in \delta(S)}} \sum_{v \in S} \leq \ell(e) & & \forall e \in E && (8.13)\\
& 2 \sum_{v \in S \subseteq V} y_{S,v} \leq \pi(v) & & \forall v \in V && (8.14)\\
& y_{S,v} \geq 0 & & \forall v \in S \subseteq V. && (8.15)
\end{aligned}$$

Compared to LP (4.1), in the primal constraints, the right side and the coefficient on the z_v terms change from 1 to 2. This is because every tour crosses the boundary of every set an even number of times. In the dual, this causes there to be a coefficient of 2 instead of 1 on the $y_{S,v}$ terms in the objective and the penalty packing constraints, but not the edge packing constraints. Thus, a dual solution y is feasible for the PCTSP instance $\mathcal{I}_\beta$ if and only if it is feasible for the PCST instance $\mathcal{I}_{\frac{\beta}{2}}$. This is why we ran `PC-Superclustering` with input matching $\mathcal{I}_{\frac{\beta}{2}}$, instead of $\mathcal{I}_\beta$.

Lemma 8.5.2. *For the set Q in PCTSP_β,*

$$\pi(Q) \leq \frac{\mathsf{opt}_\beta}{\beta} \leq \left(\frac{1-\delta}{\beta} + \delta\right)\mathsf{opt}.$$

Proof. As in the proof of Lemma 8.4.1, the dual solution y generated by `PC-Superclustering` on $\mathcal{I}_{\frac{\beta}{2}}$ is tight for the PCST penalty packing constraints on instance $\mathcal{I}_{\frac{\beta}{2}}$, so $\frac{\beta}{2}\pi(Q) = \sum_{S \subseteq D} y_S$. Because y is feasible for the PCST dual LP for instance $\mathcal{I}_{\frac{\beta}{2}}$, y is also feasible for the PCTSP dual LP for instance $\mathcal{I}_\beta$. Thus, by weak LP duality, $2\sum_{S \subseteq Q} y_S \leq 2\sum_{S \subseteq V-\{r\}} y_S \leq \mathsf{opt}_\beta \leq \ell(O) + \beta\pi(\overline{V(O)}) = (1-\delta)\mathsf{opt} + \beta\delta\mathsf{opt}$. Combining and rearranging gives the result. □

Next we have an analog to Lemma 8.4.2.

Lemma 8.5.3. *There exists a cycle that costs at most $(1 + (2\beta - 1)\delta)\mathsf{opt}$ and spans (exactly) $V \setminus Q$.*

Proof. Applying second condition of Theorem 4.3.1 to instance $\mathcal{I}_{\frac{\beta}{2}}$, with $I = V(O)$, yields a forest F with $\ell(F) \leq 2 \cdot \frac{\beta}{2}\pi(\overline{V(O)}) = \beta\delta\mathsf{opt}$. Let $2F$ denote the multiset that contains two copies of each edge in F. Augmenting the optimal cycle O by $2F$ yields an Eulerian graph spanning at least $V \setminus Q$. Since $\ell(O) = (1-\delta)\mathsf{opt}$, $\ell(O \cup (2F)) \leq (1 + (2\beta - 1)\delta)\mathsf{opt}$. To generate the desired cycle, we trace an Eulerian tour of the edge multiset $O \cup (2F)$, skipping over repeated vertices and all vertices in

Q, to obtain a cycle spanning exactly $V \setminus Q$. Because we assume the edge costs obey the triangle inequality, this shortcutting operation does not increase the cost. □

The rest of the analysis proceeds exactly as for $\texttt{PCST}_\beta$. Therefore, we obtain the following theorem.

Theorem 8.5.4. *If the approximation ratio of the* `TSP` *subroutine used is $\rho < 2$, then* $\texttt{PCTSP}_\beta$ *with parameter $\beta = \frac{2}{2-\rho}$ achieves an approximation ratio of at most $2 - (\frac{2-\rho}{2+\rho})^2$. In particular, using the $\frac{3}{2}$-approximation of Christofides [Chr76] yields an approximation ratio of at most $\frac{97}{49} < 1.979592$.*

8.6 Our Prize-collecting Path algorithms

Given any instance $\mathcal{I}$ of PC-Path and distinct vertices r and t, let $\mathcal{I}^r$ denote the same instance but with $\pi_r = \infty$, and $\mathcal{I}^{rt}$ denote the same instance but with $\pi_r = \pi_t = \infty$. For the PC-Path-1 problem with endpoint r fixed, instance $\mathcal{I}$ is equivalent to $\mathcal{I}^r$, since every feasible solution includes r, so avoids the penalty π_r. For PC-Path-2 with endpoints r and t fixed, instance $\mathcal{I}$ is equivalent to $\mathcal{I}^{rt}$. Subscripting an instance by β means to multiply all penalties by β.

Our algorithm for PC-Path-2, called $\texttt{PC-Path2}_\beta$, is nearly identical to $\texttt{PCTSP}_\beta$. Let $\mathcal{I}$ denote the PC-Path-2 instance we wish to solve, where the two specified endpoint vertices are the root r and the tail t. We run `CGRT`, the PC-Path-2 algorithm of Chaudhuri, Godfrey, Rao and Talwar [CGRT03], on $\mathcal{I}^{rt}_{\frac{1}{2}}$ to obtain our first solution, $P^{\texttt{CGRT}}$. Next, we run `PC-Superclustering` with input matching $\mathcal{I}^{rt}_{\frac{\beta}{2}}$ with root vertex r, to obtain Q. Recall that Q is the union of all clusters that ever died during the inner call to `PC-MoatGrowing`. Since $\pi(t) = \infty$ for this instance, the clusters containing vertex t cannot die, so $t \notin Q$. Next we run any algorithm `Path-TSP` for the ordinary Path-TSP2 problem on the vertex set $V \setminus Q$ with endpoints r and t, to generate a second solution Path-TSP. We then output the better of $P^{\texttt{CGRT}}$ and $P^{\texttt{Path-TSP}}$.

Algorithm 8.3 PC-Path2$_\beta(G, \pi, r, t)$

Input: graph $G(V, E)$, root vertex r, tail vertex t, and penalties $\pi(v) \geq 0$
Output: path P

1: Let $P^{\texttt{CGRT}} \leftarrow$ output of CGRT algorithm for PATH-TSP2 algorithm on instance $\mathcal{I}^{rt}_{\frac{1}{2}}$
2: Let $(Q, T) \leftarrow$ PC-Superclustering$(G, \frac{\beta}{2}\pi, r)$
3: Let $P^{\texttt{Path-TSP}} \leftarrow$ output of algorithm Path-TSP for PATH-TSP2 instance $\mathcal{I}^{rt}_{\frac{1}{2}}$ restricted to vertices $V \setminus Q$
4: Let $P \leftarrow \arg\min_{P \in \{P^{\texttt{CGRT}}, P^{\texttt{Path-TSP}}\}} (\ell(P) + \pi(\overline{V(P)}))$
5: **return** P

The analysis of this algorithm is nearly identical to that of PCTSP$_\beta$. Let O now denote the optimal PC-PATH-2 solution, opt its objective function value, and δ the fraction of opt contributed by the penalty $\pi(\overline{V(O)})$. First, there is an analog of Theorem 8.5.1.

Theorem 8.6.1 ([CGRT03])**.** *Algorithm CGRT returns a path P from r to t such that*

$$\ell(P) + 2\pi(\overline{V(P)}) \leq 2\ell(O) + 2\pi(\overline{V(O)}) = 2\mathsf{opt}.$$

To prove the analog of Lemma 8.5.2, we must slightly modify the LP relaxations (8.8) and (8.12) as follows.

$$\text{minimize} \quad \sum_{e \in E} \ell(e) x_e + \sum_{v \in V} \pi(v) z_v \tag{8.16}$$

$$\text{subject to} \quad \sum_{e \in \delta(S)} x_e \geq 2 - 2z_v \qquad \forall v \in S \subseteq V \setminus \{r, t\} \tag{8.17}$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subseteq V \setminus \{r\} : t \in S \tag{8.18}$$

$$x_e \geq 0 \qquad \forall e \in E \tag{8.19}$$

$$z_v \geq 0 \qquad \forall v \in V, \tag{8.20}$$

and the dual program

$$\text{maximize} \quad 2 \sum_{v \in S \subseteq V \setminus \{r,t\}} y_{S,v} + \sum_{v \in S \subseteq V \setminus \{r\} : t \in S} y_{S,v} \tag{8.21}$$

$$\text{subject to} \quad \sum_{\substack{S \subseteq V \\ e \in \delta(S)}} \sum_{v \in S} \leq \ell(e) \qquad \forall e \in E \tag{8.22}$$

$$2 \sum_{v \in S \subseteq V} y_{S,v} \leq \pi(v) \qquad \forall v \in V \setminus \{t\} \tag{8.23}$$

$$y_{S,v} \geq 0 \qquad \forall v \in S \subseteq V. \tag{8.24}$$

The only difference between this primal LP (8.16) and LP (8.8) for Prize-collecting TSP is that the constraint for each set $S \subseteq V - \{r\}$ such that $t \in S$ reflects the fact that every feasible path runs from r to t, and every such path must cross S an odd number of times. Using this LP relaxation, the proof is identical to the proof of Lemma 8.5.2. The rest of the analysis is identical to that for $\texttt{PCTSP}_\beta$, except that inside the proof of our analog to Lemma 8.5.3, we construct an Eulerian path from r to t instead of an Eulerian cycle. This yields the following theorem.

Theorem 8.6.2. *If the approximation ratio of the* Path-TSP2 *subroutine used is $\rho < 2$, then $\texttt{PC-Path2}_\beta$ with parameter $\beta = \frac{2}{2-\rho}$ achieves an approximation ratio of at most $2 - (\frac{2-\rho}{2+\rho})^2$. In particular, using the $\frac{5}{3}$-approximation of Hoogeveen [Hoo91] yields an approximation ratio of at most $\frac{241}{121} < 1.991736$.*

Our algorithm for PC-Path-1, called $\texttt{PC-Path1}_\beta$, is exactly the same as $\texttt{PC-Path2}_\beta$, but for two details. First, we guess the second endpoint t of the optimal path O, and run the algorithm for each choice of t, outputting the best solution. Second, we use an algorithm for Path-TSP1 as our black box `Path-TSP`, to find a path starting at r. Even though we have guessed t, we need not require $P^{\texttt{Path-TSP}}$ to end at t, which is why we can get away with using an algorithm for Path-TSP1 instead of Path-TSP2. Guessing t is necessary only so that the LP relaxation (8.16) will offer

a valid lower bound for opt. For the iteration in which t was guessed correctly, the analysis is identical to that for PC-Path2$_\beta$, and yields the following theorem.

Theorem 8.6.3. *If the approximation ratio of the* PATH-TSP1 *subroutine used is $\rho < 2$, then PC-Path1$_\beta$ with parameter $\beta = \frac{2}{2-\rho}$ achieves an approximation ratio of at most $2 - (\frac{2-\rho}{2+\rho})^2$. In particular, using the $\frac{3}{2}$-approximation of Hoogeveen [Hoo91] yields an approximation ratio of at most $\frac{97}{49} < 1.979592$.*

Chapter 9

Planar Steiner Forest

We give the first polynomial-time approximation scheme (PTAS) for the Steiner Forest problem on planar graphs and, more generally, on graphs of bounded genus. We employ the spanner framework introduced in Chapter 5, hence the first step is to build a *Steiner forest spanner* that allows us to reduce the problem to bounded-treewidth instances. Following the ideas introduced in Chapter 6, the spanner construction starts with a preprocessing that relies on the prize-collecting clustering paradigm: in particular, the main component of the preprocessing step, called `PC-Partition`, is based on `PC-Classify` and Theorem 4.2.1. `PC-Partition` breaks the input instance down into possibly several simpler subinstances that are easier to expand into a spanner; moreover, the terminals in different subinstances are far from each other, so, roughly speaking, they do not interact with each other. Each subinstance has a relatively inexpensive Steiner tree connecting all its terminals, and the subinstances can be solved (almost) separately.

Another building block in the PTAS for planar Steiner Forest is a PTAS for Steiner Forest on graphs of bounded treewidth. Surprisingly, the problem is NP-hard even on graphs of treewidth 3. Therefore, our PTAS for bounded-treewidth graphs needs a nontrivial combination of approximation arguments and dynamic pro-

Table 9.1: Complexity of Steiner Forest for different classes of graphs.

Graph class	Lower bound	Upper bound
Series-parallel graphs	P	P
Graphs of treewidth 3	NP-hard	PTAS
Bounded-treewidth graphs	NP-hard	PTAS
Planar graphs	NP-hard	PTAS
Bounded-genus graphs	NP-hard	PTAS
General graphs	APX-hard [CC02]	aprx(2) [AKR95]

gramming on the tree decomposition. We further show that Steiner Forest can be solved in polynomial time for series-parallel graphs (graphs of treewidth at most two) by a novel combination of dynamic programming and minimum-cut computations, completing our thorough complexity study of Steiner Forest in the range of bounded-treewidth graphs, planar graphs, and bounded-genus graphs.

The results of this chapter are based on a joint work with MohammadTaghi Hajiaghayi and Dániel Marx [BHM10a].

After giving a brief overview of previous results in Section 9.1, we present our polynomial-time algorithm for Steiner Forest on series-parallel graphs in Section 9.2. Then, we show in Section 9.3 that the problem becomes NP-hard if the treewidth reaches three. We give the PTAS for the case of bounded-treewidth graphs in Section 9.4 before explaining how to use it as a black box to obtain a PTAS for Steiner Forest on planar and bounded-genus instances in Section 9.5.

9.1 Background

The first and the best approximation factor for Steiner Forest is 2 due to Agrawal, Klein and Ravi [AKR95] (see also Goemans and Williamson [GW95]). Since the conference version of Agrawal, Klein and Ravi [AKR91] in 1991, there have been no improved approximation algorithms invented for Steiner Forest. Recently Bor-

radaile, Klein and Mathieu [BKM08] obtain a PTAS for Steiner Forest where the terminals are in the Euclidean plane, and the distance function is defined accordingly. They pose obtaining a PTAS for Steiner Forest in planar graphs, the natural generalization of the Euclidean case, as the main open problem. We settle this open problem by obtaining a PTAS for planar graphs (and more generally, for bounded-genus graphs) via a novel application of prize-collecting clustering paradigm.

The special case of the Steiner Forest problem when all pairs have a common terminal is the classical Steiner Tree problem, one of the first problems shown NP-hard by Karp [Kar72]. The problem remains hard even on planar graphs [GJ79]. In contrast to Steiner Forest, a long sequence of papers give approximation factors better than 2 for this problem [Zel92, Zel93, BR92, Zel96, PS97, KZ97, HP99, RZ05, BGRS10]; the current best approximation ratio is 1.39 [BGRS10]. Since the problem is APX-hard in general graphs [Kar72, Thi03], we do not expect to obtain a PTAS for this problem in general graphs. However, for the Euclidean Steiner Tree problem, the classic works of Arora [Aro98] and Mitchell [Mit99] present a PTAS. Obtaining a PTAS for Steiner Tree on planar graphs, the natural generalization of Euclidean Steiner Tree, remained a major open problem since the conference version of [Aro96] in 1996. Only in 2007, Borradaile, Mathieu, and Klein [BKM09] settle this problem with a nice technique of constructing *light spanners* for Steiner Tree in planar graphs. Here we also generalize this result to obtain light spanners for Steiner Forest; see Table 9.2.

Most approximation schemes for planar graph problems use (implicitly or explicitly) the fact that the problem is easy to solve on bounded-treewidth graphs—in fact, the ideas in Baker [Bak94] and the reformulations in [DHM07, Kle08] provide a general method of reducing many optimization problems on planar (and bounded-genus) graphs to bounded-treewidth graphs; see Section 5.3. In particular, a keystone black-box in the algorithm of [BKM09] for Steiner Tree is the result that, for every

Table 9.2: PTAS for Steiner Tree and Steiner Forest on Euclidean, planar and bounded-genus graph metrics. Notice that Steiner Forest is a generalization of Steiner Tree, and bounded-genus graphs generalize planar graphs that themselves generalize Euclidean metrics.

	Steiner Tree	**Steiner Forest**
Euclidean metrics	Arora [Aro98], Mitchell [Mit99]	Borradaile et al. [BKM08]
Planar graphs	Borradaile et al. [BKM09]	Bateni et al. [BHM10a]
Bounded-genus graphs	Borradaile et al. [BDT09]	Bateni et al. [BHM10a]

fixed value of k, the problem is polynomial-time solvable on graphs having treewidth at most k. There is a vast literature on algorithms for bounded-treewidth graphs and in most cases polynomial-time (or even linear-time) solvability follows from the well-understood standard technique of dynamic programming on tree decompositions. However, for Steiner Forest, the obvious way of using dynamic programming does not give a polynomial-time algorithm. The difficulty is that, unlike in Steiner Tree, a solution of Steiner Forest induces a partition on the set of terminals and a dynamic-programming algorithm needs to keep track of exponentially many such partitions. In fact, this approach seems to fail even for series-parallel graphs (that have treewidth at most 2); the complexity of the problem for series-parallel graphs was stated as an open question by [RP86]. We resolve this question by giving a polynomial-time algorithm for Steiner Forest on series-parallel graphs. The main idea is that, even though algorithms based on dynamic programming have to evaluate subproblems corresponding to exponentially many partitions, the function describing these exponentially many values turns out to be submodular, and it can be represented in a compact way by the cut function of a directed graph. On the other hand, Steiner Forest becomes NP-hard on graphs of treewidth at most 3 [Gas10]. Thus perhaps this is the first example when the complexity of a natural problem changes as treewidth increases from 2 to 3. In light of this hardness result, we investigate

the approximability of the problem on bounded-treewidth graphs and show that, for every fixed k, Steiner Forest admits a PTAS on graphs of treewidth at most k. The main idea of the PTAS is that, if the dynamic-programming algorithm considers only an appropriately constructed polynomial-size subset of the set of all partitions, then this produces a solution close to the optimum. Very roughly speaking, the partitions in this subset are constructed by choosing a set of center points and classifying the terminals according to the distance to the center points. Our PTAS for planar graphs (and, more generally, for bounded-genus graphs) uses this PTAS for bounded-treewidth graphs. This completes our thorough study of Steiner Forest in the range of bounded-treewidth graphs, planar graphs and bounded-genus graphs.

9.2 Steiner forest for series-parallel graphs

As defined in Section 2.3.1, a series-parallel graph can be built form elementary blocks using two operations: parallel connection and series connection. The algorithm of Theorem 3.2.2 uses dynamic programming on the construction of the series-parallel graph. For each subgraph arising in the construction, we find a minimum-length forest that connects some of the terminal pairs (i.e., satisfies the corresponding demands), connects a subset of the terminals to the "left exit point" of the subgraph, and connects the remaining terminals to the "right exit point" of the subgraph. The minimum length depends on the subset of terminals connected to the left exit point, thus it seems that we need to determine exponentially many values (one for each subset). Fortunately, it turns out that the minimum length is a submodular function of the subset. Furthermore, we show that this function can be represented by the cut function of a directed graph and this directed graph can be easily constructed if the directed graphs corresponding to the building blocks of the series-parallel subgraph are available. Thus, following the construction of the series-parallel graph, we can

build all these directed graphs and determine the value of the optimal solution by the computation of a minimum cut on the final directed graph.

We prove Theorem 3.2.2 in this section by constructing a polynomial-time algorithm to solve Steiner Forest on series-parallel graphs. It is well-known that the treewidth of a graph is at most 2 if and only if it is a subgraph of a series-parallel graph [Bod98]. Since setting the length of an edge to ∞ is essentially the same as deleting the edge, it follows that Steiner Forest can be solved in polynomial time on graphs with treewidth at most 2.

Let $(G, \mathcal{D})$ be an instance of Steiner forest where G is a series-parallel graph. For $i = 1, \ldots, m$, denote by $G_i(x_i, y_i)$ all the intermediary graphs appearing in the series-parallel construction of G. We assume that these graphs are ordered such that $G = G_m$, and, if G_i is obtained from G_{j_1} and G_{j_2}, then $j_1, j_2 < i$. Let $\mathcal{D}_i \subseteq \mathcal{D}$ contain those pairs $\{u, v\}$ where both vertices are in $V(G_i)$. Let A_i be those vertices $v \in V(G_i)$ for which there exists a pair $\{v, u\} \in \mathcal{D}$ with $u \notin V(G_i)$ (note that $A_m = \emptyset$ and $\mathcal{D}_m = \mathcal{D}$). For every G_i, we define two integer values a_i, b_i, and a function f_i as follows.

1. Let a_i be the minimum length of a solution F of the instance $(G_i, \mathcal{D}_i)$ with the additional requirements that x_i and y_i are connected in F and every vertex in A_i is in the same component as x_i and y_i.

2. Let G'_i be the graph obtained from G_i by identifying vertices x_i and y_i. Let b_i be the minimum length of a solution F of the instance $(G'_i, \mathcal{D}_i)$ with the additional requirement that every vertex of A_i is in the same component as $x_i = y_i$.

3. For every $S \subseteq A_i$, let $f_i(S)$ be the minimum length of a solution F of the instance $(G_i, \mathcal{D}_i)$ with the additional requirements that x_i and y_i are not connected, every $v \in S$ is in the same component as x_i, and every $v \in A_i \setminus S$ is in the same component as y_i. (If there is no such F, then $f_i(S) = \infty$.)

The main combinatorial property that allows us to solve the problem in polynomial time is that the functions f_i are submodular. We prove something stronger: the functions f_i can be represented in a compact way as the cut functions of certain directed graphs.

In this section, we use directed graphs as part of our algorithm. Recall that here the edges are ordered pairs of vertices; i.e., the edge (u, v) is different from the edge (v, u). In order to emphasize this distinction throughout, we use the notation $\overrightarrow{uv}$ to refer to an edge from u to v. This edge is said to be *leaving* u and *entering* v. If D is a directed graph with lengths on the edges and $X \subseteq V(D)$, then $\delta_D(X)$ denotes the total length of the edges of D leaving X (i.e., leaving a vertex in X and entering a vertex outside X). For $X, Y \subseteq V(D)$, we denote by $\lambda_D(A, B)$ the minimum length of a directed cut that separates A from B, i.e., the minimum of $\delta_D(X)$ taken over all $A \subseteq X \subseteq V(D) \setminus B$; if $A \cap B \neq \emptyset$, then $\lambda_D(A, B)$ is defined to be ∞.

Definition 9.2.1. *Let D_i be a directed graph with nonnegative edge lengths. Let s_i and t_i be two distinguished vertices of D_i, and let A_i be a subset of vertices of D_i. For a function $f_i : 2^{A_i} \mapsto \mathbb{R}^+$, we say that* (D_i, s_i, t_i, A_i) represents *f_i if $f_i(S) = \lambda_{D_i}(S \cup \{s_i\}, (A_i \setminus S) \cup \{t_i\})$ for every $S \subseteq A_i$. If s_i, t_i, A_i are clear from the context, then we simply say that D_i represents f_i.*

It is well-known that $\delta_G(X)$ is a submodular function on the subsets of $V(G)$; see, e.g., [Sch03, Section 44.1a]. Submodularity is a powerful unifying concept of combinatorial optimization: classical results on flows, cuts, matchings, and matroids can be considered as consequences of submodularity. The following (quite standard) proposition shows that, if a function can be represented in the sense of Definition 9.2.1, then the function is submodular. In the proof of Theorem 3.2.2, we show that every function f_i can be represented by a directed graph, thus it follows that every f_i is submodular. Although we do not use this observation directly, it explains in some sense why the problem is solvable in polynomial time.

Proposition 9.2.2. *If a function $f : 2^A \mapsto \mathbb{R}^+$ can be represented by (D, s, t, A) (in the sense of Definition 9.2.1), then f is submodular.*

Proof. Let $X, Y \subseteq A$ be arbitrary sets. Since D represents f, there exist appropriate sets X' and Y' with $\delta_D(X') = f(X)$ and $\delta_D(Y') = f(Y)$. Now we have

$$f(X) + f(Y) = \delta_D(X') + \delta_D(Y') \geq \delta_D(X' \cap Y') + \delta_D(X' \cup Y'), \tag{9.1}$$

where the inequality follows from the submodularity of δ_D. Observe that

$$(X \cap Y) \cup \{s\} \subseteq X' \cap Y' \subseteq V(D) \setminus [A \setminus (X \cap Y)] \cup \{t\} \tag{9.2}$$

since $X \cup \{s\} \subseteq X' \subseteq V(D) \setminus ((A \setminus X)$ and $Y \cup \{s\} \subseteq Y' \subseteq V(D) \setminus ((A \setminus Y)$. Thus we have $f(X \cap Y) \leq \delta_D(X' \cap Y')$. In a similar way, we have

$$(X \cup Y) \cup \{s\} \subseteq X' \cup Y' \subseteq V(D) \setminus [A \setminus (X \cup Y)] \cup \{t\} \tag{9.3}$$

that yields $f(X \cup Y) \leq \delta_D(X' \cup Y')$. Together with Inequality (9.1) obtained above, this proves that $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$. □

We now get to prove the main result of this section.

Proof of Theorem 3.2.2. We assume that in the given instance of Steiner forest each vertex appears only in at most one pair of $\mathcal{D}$. To achieve this, if a vertex v appears in $k > 1$ pairs, then we subdivide an arbitrary edge incident to v by $k - 1$ new vertices such that the length of each of the $k - 1$ edges on the path formed by v and the new vertices is 0. Replacing vertex v in each pair involving it by one of the new vertices does not change the problem, and achieves the said property. Furthermore, we assume that there is no trivial demand (v, v).

For every $i = 1, \ldots, m$, we compute the values a_i, and b_i, as well as a representation

D_i of f_i. In the optimal solution F for the instance $(G_m, \mathcal{D})$, vertices x_m and y_m are either connected or not. Thus the length of the optimal solution is the minimum of a_m and $f_m(\emptyset)$ (recall that $A_m = \emptyset$). The value of $f_m(\emptyset)$ can be easily determined by computing the minimum-length $\{s_m, t_m\}$ cut in D_m.

If G_i is a single edge e, then a_i and b_i are trivial to determine: a_i is the length of e and $b_i = 0$. The directed graph D_i representing f_i can be obtained from G_i by renaming x_i to s_i, renaming y_i to t_i, and either removing the edge e (if $\mathcal{D}_i = \emptyset$) or replacing e with a directed edge $\overrightarrow{s_i t_i}$ of length ∞ (if $\{x_i, y_i\} \in \mathcal{D}_i$).

If G_i is not a single edge, then it is constructed from some G_{j_1} and G_{j_2} by either series or parallel connection. Suppose that a_{j_p}, b_{j_p}, and D_{j_p} for $p = 1, 2$ are already known. We show how to compute a_i, b_i, and D_i in this case.

Parallel connection. Suppose that G_i is obtained from G_{j_1} and G_{j_2} by parallel connection. It is easy to see that $a_i = \min\{a_{j_1} + b_{j_2}, b_{j_1} + a_{j_2}\}$ and $b_i = b_{j_1} + b_{j_2}$. To obtain D_i, we join D_{j_1} and D_{j_2} by identifying s_{j_1} with s_{j_2} (call it s_i) and by identifying t_{j_1} with t_{j_2} (call it t_i). Furthermore, for every $\{u, v\} \in \mathcal{D}_i \setminus \{\mathcal{D}_{j_1} \cup \mathcal{D}_{j_2}\}$, we add directed edges $\overrightarrow{uv}$ and $\overrightarrow{vu}$ with length ∞.

To see that D_i represents f_i, suppose that F is the subgraph that realizes the value $f_i(S)$ for some $S \subseteq A_i$. We first show that there is an appropriate $X \subseteq V(D_i)$ certifying $\lambda_{D_i}(S \cup \{s\}, (A_i \setminus S) \cup \{t\}) \leq \ell(F)$. The graph F is the edge disjoint union of two graphs $F_1 \subseteq G_{j_1}$ and $F_2 \subseteq G_{j_2}$. For $p = 1, 2$, let $S^p \subseteq A_{j_p}$ be the set of those vertices that are connected to x_{j_p} in F_p, it is clear that F_p connects $A_{j_p} \setminus S^p$ to y_{j_p}. Since F_p does not connect x_i and y_i, we have that $\ell(F_p) \geq f_{j_p}(S^p)$. Since D_{j_p} represents f_{j_p}, there is a set X_p of vertices in D_{j_p} such that $S^p \cup \{s_{j_p}\} \subseteq X_p \subseteq V(D_i) \setminus ((A_{j_p} \setminus S^p) \cup \{t_{j_p}\})$, and $\delta_{D_{j_p}}(X_p) = f_{j_p}(S^p)$. We show that $\delta_{D_i}(X_1 \cup X_2) = \delta_{D_{j_1}}(X_1) + \delta_{D_{j_2}}(X_2)$. Since D_i is obtained from joining D_{j_1} and D_{j_2}, it suffices to verify that the edges with infinite length added after the join cannot leave $X_1 \cup X_2$. Suppose that there is such an edge $\overrightarrow{uv}$; assume without loss of generality that $u \in X_1$ and $v \in V(D_{j_2}) \setminus X_2$. This

means that $u \in S^1$ and $v \notin S^2$. Thus F connects u to x_i and v to y_i, implying that F does not connect u and v. However $\{u, v\} \in \mathcal{D}_i$ by the definition of $\mathcal{D}_i$, contradicting the assumption that F is a realization of $f_i(S)$. Therefore, for the set $X := X_1 \cup X_2$, we have

$$\delta_{D_i}(X) = \delta_{D_{j_1}}(X_1) + \delta_{D_{j_2}}(X_2) = f_{j_1}(S^1) + f_{j_2}(S^2) \leq \ell(F_1) + \ell(F_2) = \ell(F) = f_i(S),$$

proving the existence of the required X.

Suppose now that for some $S \subseteq A_i$, there is a set X with $S \cup \{s_i\} \subseteq X \subseteq V(D_i) \setminus ((A_i \setminus S) \cup \{t_i\})$. We have to show that $\delta_{D_i}(X) \geq f_i(S)$. If $\delta_{D_i}(X) = \infty$, then this is trivially true, thus we assume that $\delta_{D_i}(X)$ is finite. For $p = 1, 2$, let $X_p = X \cap V(D_{j_p})$ and $S^p = A_{j_p} \cap X_p$. As $\delta_{D_i}(X)$ is finite, the infinite edges added in the construction of D_i do not appear on the boundary of X, hence $\delta_{D_i}(X) = \delta_{D_{j_1}}(X_1) + \delta_{D_{j_2}}(X_2)$. Since D_{j_p} represents f_{j_p}, we know that $\delta_{D_{j_p}}(X_p) \geq f_{j_p}(S^p)$. Let F_p be a subgraph of G_{j_p} realizing $f_{j_p}(S^p)$. Let $F = F_1 \cup F_2$; we show that $\ell(F) \geq f_i(S)$ holds by verifying that F satisfies all the requirements in the definition of $f_i(S)$. It is clear that F does not connect x_i and y_i. Consider a pair $\{u, v\} \in \mathcal{D}_i$. If $\{u, v\} \in \mathcal{D}_{j_p}$, then F connects u and v. Otherwise, let $\{u, v\} \in \mathcal{D}_i \setminus \{\mathcal{D}_{j_1} \cup \mathcal{D}_{j_2}\}$ for some $u \in V(G_{j_1})$ and $v \in V(G_{j_2})$. Clearly, this means that $u \in A_{j_1}$ and $v \in A_{j_2}$. Suppose that F does not connect u and v, and, without loss of generality, assume that $u \in S^1$ and $v \notin S^2$. By definition of S^1 and S^2, it follows that $u \in X_1$ and $v \notin X_2$. This means that there is an edge $\overrightarrow{uv}$ of length ∞ in D_i, yielding $\delta_{D_i}(X) = \infty$, which contradicts our earlier assumption. Thus we can indeed assume $\ell(F) \geq f_i(S)$, which gives

$$\delta_{D_i}(X) = \delta_{D_{j_1}}(X_1) + \delta_{D_{j_2}}(X_2) \geq f_{j_1}(S^1) + f_{j_2}(S^2) = \ell(F_1) + \ell(F_2) = \ell(F) \geq f_i(S).$$

This completes the argument for the parallel connection.

Series connection. Suppose that G_i is obtained from G_{j_1} and G_{j_2} by series

connection and let $\mu := y_{j_1} = x_{j_2}$ be the middle vertex. It is easy to see that $a_i = a_{j_1} + a_{j_2}$ (i.e., vertex μ has to be connected to both x_i and y_i). To compute b_i, we argue as follows. Denote by $G^R_{j_2}$ the graph obtained from G_{j_2} by swapping the names of distinguished vertices x_{j_2} and y_{j_2}. Observe that the graph G'_i in the definition of b_i arises as the parallel connection of G_{j_1} and $G^R_{j_2}$. It is easy to see that $a^R_{j_2}$, $b^R_{j_2}$, and $f^R_{j_2}$ corresponding to $G^R_{j_2}$ can be defined as $a^R_{j_2} = a_{j_2}$, $b^R_{j_2} = b_{j_2}$, and $f^R_{j_2}(S) = f_{j_2}(A_{j_2} \setminus S)$. Furthermore, if D_{j_2} represents f_{j_2}, then the graph $D^R_{j_2}$ obtained from D_{j_2} by reversing the orientation of the edges and swapping the names of s_{j_2} and t_{j_2} represents $f^R_{j_2}$. Thus we have everything at our disposal to construct a directed graph D'_i that represents the function f'_i corresponding to the parallel connection of G_{j_1} and $G^R_{j_2}$. Now observe that to compute b_i we can consider two cases: either μ is connected to x_{j_1} and y_{j_2} or not. We take the minimum of the two values. The first case is simply $\min\left\{a_{j_1} + b^R_{j_2}, b_{j_1} + a^R_{j_2}\right\} = \min\left\{a_{j_1} + b_{j_2}, b_{j_1} + a_{j_2}\right\}$, and the second case is $f'_i(A_i)$: graph G'_i is isomorphic to the parallel connection of G_{j_1} and $G^R_{j_2}$ and the definition of b_i requires that A_i is connected to $x_i = y_i$. The value of $f'_i(A_i)$ can be determined by a simple minimum cut computation in D'_i.

Let $T_1 \subseteq A_{j_1}$ contain those vertices v for which there exists a pair $\{v, u\} \in \mathcal{D}_i$ with $u \in A_{j_2}$ and let $T_2 \subseteq A_{j_2}$ contain those vertices v for which there exists a pair $\{v, u\} \in \mathcal{D}_i$ with $u \in A_{j_1}$. Observe that $A_i = (A_{j_1} \setminus T_1) \cup (A_{j_2} \setminus T_2)$. (Here we are using the fact that each vertex is contained in at most one pair, thus $v \in T_1$ cannot be part of any pair $\{v, u\}$ with $u \notin V(D_i)$). To construct D_i, we connect D_{j_1} and D_{j_2} with an edge $\overrightarrow{t_{j_1} s_{j_2}}$ of length 0 and set $s_i := s_{j_1}$ and $t_i := t_{j_2}$. Furthermore, we introduce two new vertices γ_1, γ_2 and add the following edges (see Figure 9.1):

1. $\overrightarrow{s_{j_1}\gamma_1}$ with length a_{j_2},
2. $\overrightarrow{\gamma_1\gamma_2}$ with length $f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_2)$,
3. $\overrightarrow{\gamma_2 t_{j_2}}$ with length a_{j_1},

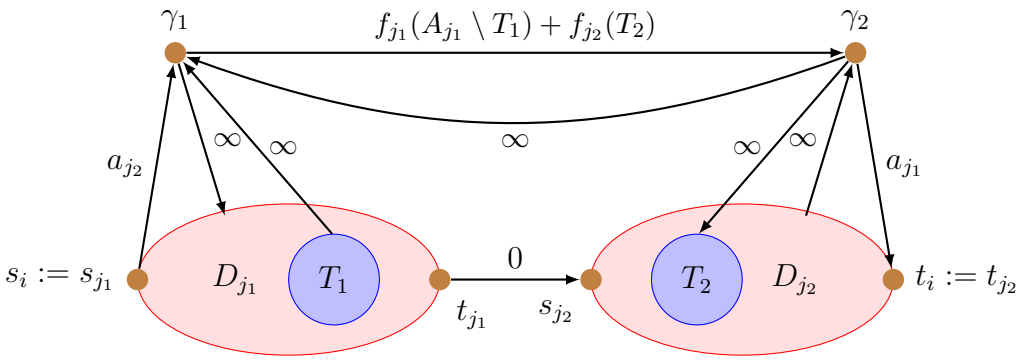


Figure 9.1: Construction of the auxiliary graph D_i for a series connection.

4. $\overrightarrow{\gamma_2\gamma_1}$ with length ∞,

5. $\overrightarrow{\gamma_1 v}$ with length ∞ for every $v \in V(D_{j_1})$,

6. $\overrightarrow{v\gamma_2}$ with length ∞ for every $v \in V(D_{j_2})$,

7. $\overrightarrow{v\gamma_1}$ with length ∞ for every $v \in T_1$, and

8. $\overrightarrow{\gamma_2 v}$ with length ∞ for every $v \in T_2$.

We prove the claim that D_i represents f_i in two parts.

Suppose that F is the subgraph that realizes the value $f_i(S)$ for some $S \subseteq A_i$; we first show that D_i has an appropriate cut with value at most $f_i(S)$. Subgraph F is the edge-disjoint union of subgraphs $F_1 \subseteq G_{j_1}$ and $F_2 \subseteq G_{j_2}$. We consider 3 cases: in subgraph F, vertex μ is either connected to neither x_{j_1} nor y_{j_2}, connected only to x_{j_1}, or connected only to y_{j_2} (recall that $x_i = x_{j_1}$, $\mu = y_{j_1} = x_{j_2}$, and $y_i = y_{j_2}$.

Case 1: μ is connected to neither x_{j_1} nor y_{j_2}. In this case, vertices of A_{j_1} are not connected to y_i and vertices of A_{j_2} are not connected to x_i, hence $S = A_i \cap A_{j_1} = A_{j_1} \setminus T_1$ is the only possibility. Furthermore, F connects both T_1 and T_2 to μ. It follows that $\ell(F) = \ell(F_1) + \ell(F_2) \geq f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_2)$. Set $X = V(D_{j_1}) \cup \{\gamma_1\}$: now we have $\delta_{D_i}(X) = f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_2) \leq \ell(F)$, X contains $(A_{j_1} \setminus T_1) \cup \{s_i\}$, and is disjoint from $(A_{j_2} \setminus T_2) \cup \{t_i\}$.

Case 2: μ is connected only to x_i. This is only possible if $A_{j_1} \setminus T_1 \subseteq S$. Clearly, $\ell(F_1) \geq a_{j_1}$. Subgraph F_2 has to connect every vertex in $(S \cap A_{j_2}) \cup T_2$ to $x_{j_2} = \mu$ and every vertex in $A_i \setminus S = A_{j_2} \setminus (S \cup T_2)$ to y_{j_2}. This implies $\ell(F_2) \geq f_{j_2}((S \cap A_{j_2}) \cup T_2)$. Let $X_2 \subseteq V(D_{j_2})$ be the corresponding cut in D_{j_2}. Set $X := X_2 \cup V(D_{j_1}) \cup \{\gamma_1, \gamma_2\}$; we have $\delta_{D_i}(X) = f_{j_2}((S \cap A_{j_2}) \cup T_2) + a_{j_1} \leq \ell(F_2) + a_{j_1} \leq \ell(F)$. (Note that no edge with infinite length leaves X since $T_2 \subseteq X_2$). As X contains S and contains none of the vertices in $A_i \setminus S$, the existence of the required cut is established.

Case 3: μ is connected only to y_i. The argument is very similar to case 2, but we include it for the sake of completeness. This case happens only if $A_{j_2} \setminus T_2 \subseteq A \setminus S$. Clearly, $\ell(F_2) \geq a_{j_2}$. Subgraph F_1 has to connect every vertex in $A_{j_1} \setminus S$ to $y_{j_1} = \mu$ and every vertex in S to x_{j_1}. This implies $\ell(F_1) \geq f_{j_1}(S)$. Let $X_1 \subseteq V(D_{j_1})$ be the corresponding cut in D_{j_1}. Set $X := X_1$; we have $\delta_{D_i}(X) = f_{j_1}(S) + a_{j_2} \leq \ell(F_2) + a_{j_2} \leq \ell(F)$. (Note that no edge with infinite length leaves X since $T_1 \cap X_1 = \emptyset$). As X contains S and contains none of the vertices in $A_i \setminus S$, the existence of the required cut is established.

Suppose now that for some $S \subseteq A_i$, there is a set X such that $S \cup \{s_i\} \subseteq X \subseteq V(D_i) \setminus ((A_i \setminus S) \cup \{t_i\})$. We show that $\delta_{D_i}(X) \geq f_i(S)$. There is nothing to show if $\delta_{D_i}(X) = \infty$. In particular, because of the edge $\overrightarrow{\gamma_2\gamma_1}$, the case $\gamma_2 \in X$ and $\gamma_1 \notin X$ is trivial. Thus we have to consider only 3 cases depending on which of γ_1, γ_2 are contained in X.

Case 1: $\gamma_1 \in X$, $\gamma_2 \notin X$. In this case, the edges having length ∞ ensure that $V(D_{j_1}) \subseteq X$ and $V(D_{j_2}) \cap X = \emptyset$, thus $\delta_{D_i}(X) = \ell(\overrightarrow{\gamma_1\gamma_2}) + \ell(\overrightarrow{t_{j_1}s_{j_2}}) = f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_{j_2})$. Notice that $S = A_{j_1} \setminus T_1$. In this case, it is easy to see that $f_i(S) \leq f_{j_1}(A_{j_1} \setminus T_1) + f_{j_2}(T_{j_2})$: taking the union of some F_1 realizing $f_{j_1}(A_{j_1} \setminus T_1)$ and some F_2 realizing $f_{j_2}(T_{j_2})$, we get a subgraph F realizing $f_i(S)$.

Case 2: $\gamma_1, \gamma_2 \in X$. The edges of infinite length leaving γ_1 ensure that $V(D_{j_1}) \subseteq X$. Furthermore, $\gamma_2 \in X$ ensures that $T_2 \subseteq X$. Let $X_2 := X \cap V(D_{j_2})$; we have

$X_2 \cap A_{j_2} = T_2 \cup (S \cap A_{j_2})$, which implies $\delta_{D_{j_2}}(X_2) \geq f_{j_2}(T_2 \cup (S \cap A_{j_2}))$. Observe that $\delta_{D_i}(X) = a_{j_1} + \delta_{D_{j_2}}(X_2)$ (where the term a_{j_1} comes from the edge $\overrightarrow{\gamma_2 t_{j_2}}$). Let F_1 be a subset of G_{j_1} realizing a_{j_1} and let F_2 be a subset of G_{j_2} realizing $f_{j_2}(T_2 \cup (S \cap A_{j_2}))$. Let $F := F_1 \cup F_2$, and note that F connects vertices S to x_i, vertices $A_i \setminus S$ to y_i, and vertices in $T_1 \cup T_2$ to μ. Thus $f_i(S) \leq \ell(F) = a_{j_1} + f_{j_2}(T_2 \cup (S \cap A_{j_2})) \leq \delta_{D_i}(X)$, as desired.

Case 3: $\gamma_1, \gamma_2 \notin X$. The argument is very similar to Case 2, but is included for the sake of completeness. The edges of infinite length entering γ_2 ensure that $V(D_{j_2}) \cap X = \emptyset$. Furthermore, $\gamma_1 \notin X$ ensures that $T_1 \cap X = \emptyset$. Let $X_1 := X$; we have $S \subseteq X_2$, which implies $\delta_{D_{j_1}}(X_1) \geq f_{j_1}(S)$. Observe that $\delta_{D_i}(X) = a_{j_2} + \delta_{D_{j_1}}(X_1)$ (where the term a_{j_2} comes from the edge $\overrightarrow{s_{j_1}\gamma_1}$). Let F_2 be a subset of G_{j_2} realizing a_{j_2} and let F_1 be a subset of G_{j_1} realizing $f_{j_1}(S)$. Let $F := F_1 \cup F_2$, and note that F connects vertices S to x_i, vertices $A_i \setminus S$ to y_i, and vertices in $T_1 \cup T_2$ to μ. Thus $f_i(S) \leq \ell(F) = a_{j_2} + f_{j_1}(S) \leq \delta_{D_i}(X)$, as desired. □

9.3 Steiner forest for graphs of treewidth three

In this section, we show that Steiner Forest is NP-hard on graphs with treewidth at least 3. Shortly before our work [BHM10a], this was proved independently by Gassner [Gas10], but our compact proof perhaps better explains the reason for the sharp contrast between the series-parallel and the treewidth 3 cases.

Consider the graph in Figure 10.1 and let us define the function f analogously to the function f_i in Section 9.2: for every $S \subseteq \{1, 2, 3\}$, let $f(S)$ be the minimum length of a subgraph F where x and y are not connected, t_i is connected to x for every $i \in S$, and t_i is connected to y for every $i \in \{1, 2, 3\} \setminus S$; if there is no such subgraph F, then define $f(S) = \infty$. It is easy to see that $f(\{1, 3\}) = f(\{2, 3\}) = f(\{1, 2, 3\})$, while $f(\{3\}) = \infty$. Thus, unlike in the case of series-parallel graphs, this function is

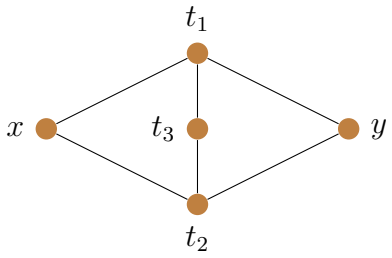


Figure 9.2: The graph used in the proof of Theorem 3.2.3. All edges have unit length.

not submodular.

We use the properties of the function f defined in the previous paragraph to obtain a hardness proof in a more or less "automatic" way. Let us define the Boolean relation $R(a, b, c) := (a = c) \vee (b = c)$. Observe that for any $S \subseteq \{1, 2, 3\}$, we have $f(S) = 3$ if $R(1 \in S, 2 \in S, 3 \in S) = 1$ and $f(S) = \infty$ otherwise. (Here, $i \in S$ for $i \in \{1, 2, 3\}$ indicates the Boolean variable that takes value 1 if and only if $i \in S$.) Thus, the gadget in Figure 10.1 in some sense represents the relation R and, as explained below, this is sufficient to construct an NP-hardness proof.

An *R-formula* is a conjunction of clauses, where each clause is the relation R applied to some Boolean variables or to the constants 0 and 1, e.g., $R(x_1, 0, x_4) \wedge R(0, x_2, x_1) \wedge R(x_3, x_2, 1)$. In the R-SAT problem, the input is an R-formula and it has to be decided whether the formula has a satisfying assignment.

Lemma 9.3.1. *R-SAT is NP-complete.*

Proof. Readers familiar with Schaefer's Dichotomy Theorem (more precisely, the version allowing constants [Sch78, Lemma 4.1]) can easily see that R-SAT is NP-complete. It is easy to verify that the relation R is neither weakly positive, weakly negative, affine, nor bijunctive. Thus, the result of Schaefer immediately implies that R-SAT is NP-complete.

For completeness, we give a simple self-contained proof here. The reduction is

from Not-All-Equal 3SAT[1], which is known to be NP-complete even if there are no negated literals [Sch78]. Given an NAE-3SAT formula, we replace each clause as follows. For each clause $\mathrm{NAE}(a, b, c)$, we introduce a new variable d and create the clauses $R(a, b, d) \wedge R(c, d, 0) \wedge R(c, d, 1)$. If $a = b = c$, then it is not possible that all three clauses are simultaneously satisfied; observe that the second and third clauses force $c \neq d$. On the other hand, if a, b, c do not have the same value, then all three clauses can be satisfied by an appropriate choice of d. Thus, the transformation from NAE-3SAT to R-SAT preserves satisfiability. □

The main idea of the following proof is that we can simulate arbitrarily many R-relations by joining in parallel copies of the graph shown in Figure 10.1. Here comes the NP-hardness proof.

Proof of Theorem 3.2.3. The proof is by reduction from R-SAT. Let ϕ be an R-formula having n variables and m clauses. We start the construction of the graph G by introducing two vertices v_0 and v_1. For each variable x_i of ϕ, we introduce a vertex x_i and connect it to both v_0 and v_1. We introduce 3 new vertices a_i, b_i, c_i corresponding to the i^{th} clause. Vertices a_i and b_i are connected to both v_0 and v_1, while c_i is adjacent only to a_i and b_i. If the i^{th} clause is $R(x_{i_1}, x_{i_2}, x_{i_3})$, then we add the 3 pairs $\{x_{i_1}, a_i\}$, $\{x_{i_2}, b_i\}$, and $\{x_{i_3}, c_i\}$ to $\mathcal{D}$. If the clause contains constants, then we use the vertices v_0 (as "false") and v_1 (as "true") instead of the vertices x_{i_1}, x_{i_2}, x_{i_3}; e.g., the clause $R(0, x_{i_2}, 1)$ yields the pairs $\{v_0, a_i\}$, $\{x_{i_2}, b_i\}$, and $\{v_1, c_i\}$. The length of every edge is 1. This completes the description of the graph G and the set of pairs $\mathcal{D}$.

We claim that the constructed instance of Steiner Forest has a solution with $n + 3m$ edges if and only if the R-formula ϕ is satisfiable. Suppose that ϕ has a

[1]In Not-All-Equal 3SAT, or NAE-3SAT for short, we are given a 3SAT instance with the extra restriction that a clause is not satisfied if *all* the literals in a clause are true. Similarly to 3SAT, the clause is not satisfied if all the literals are false, either. Thus, the literals in each clause have to take both true and false values.

satisfying assignment f. We construct F as follows. If $f(x_i) = 1$, then let us add edge (x_i, v_1) to F; if $f(x_i) = 0$, then let us add edge (x_i, v_0) to F. For each clause, we add 3 edges to F. Suppose that the i^{th} clause is $R(x_{i_1}, x_{i_2}, x_{i_3})$. We add one of (a_i, v_0) or (a_i, v_1) to F depending on the value of $f(x_{i_1})$ and we add one of (b_i, v_0) or (b_i, v_1) to F depending on the value of $f(x_{i_2})$. Since the clause is satisfied, either $f(x_{i_3}) = f(x_{i_1})$ or $f(x_{i_3}) = f(x_{i_2})$; we add (c_i, a_i) or (c_i, b_i) to F, respectively. (If $f(x_{i_3})$ is equal to both, then the choice is arbitrary). We proceed in an analogous manner for clauses containing constants. It is easy to verify that each pair is in the same connected component of F; in particular, all vertices corresponding to variables of the same value (true or false) form one connected component.

Suppose now that there is a solution F with length $n + 3m$. At least one edge is incident on each vertex x_i since it cannot be isolated in F. Each vertex a_i, b_i, c_i has to be connected to either v_0 or v_1, hence at least 3 edges of F are incident on these 3 vertices. As F has $n + 3m$ edges, it follows that exactly one edge is incident on each x_i, hence exactly 3 edges are incident on the set $\{a_i, b_i, c_i\}$. It follows that v_0 and v_1 are not connected in F. Define an assignment of ϕ by setting $f(x_i) = 0$ if and only if vertex x_i is in the same component of F as v_0. To verify that a clause $R(x_{i_1}, x_{i_2}, x_{i_3})$ is satisfied, observe that c_i is in the same component of F as either a_i or b_i. If c_i is in the same component as, say, a_i, then this component also contains x_{i_3} and x_{i_1}, implying $f(x_{i_3}) = f(x_{i_1})$ as required.

Finally, we claim that the graph of the above construction is planar and has treewidth at most three. Planarity can be easily verified. We propose a tree decomposition as follows to establish the treewidth bound. The root of the tree has a bag containing $\{v_0, v_1\}$. The root has a child for each variable x_i, with a bag containing $\{v_0, v_1, x_i\}$. In addition, there is a two-node path connected to the root corresponding to each clause and its gadget: let a_i, b_i, c_i be the vertices of the gadget. Then, the root of the tree decomposition has a child, whose bag is $\{v_0, v_1, a_i, b_i\}$, and has a child

of its own with a bag $\{a_i, b_i, c_i\}$. The largest bag has size four, and the endpoints of each edge of the graph appear together in at least one tree node. □

9.4 Steiner forest for bounded-treewidth graphs

The purpose of this section is to prove Theorem 3.2.4 by presenting a PTAS for Steiner Forest on graphs of bounded treewidth.

9.4.1 Groups

We define a notion of group that will be crucial in the description of the algorithm. A group is defined by a set S of center vertices, a set X of "interesting" vertices, and a maximum distance r; the *group* $\mathcal{G}_G(X, S, r)$ contains S and those vertices of X that are at distance (with respect to the shortest path on G) at most r from some vertex in S.

Lemma 9.4.1. *Let T be a Steiner tree of $X \subseteq V(G)$ with length $W = \ell(T)$. For every $\epsilon > 0$, there is a set $S \subseteq X$ of $O(1+1/\epsilon)$ vertices such that $X = \mathcal{G}_G(X, S, \epsilon W)$.*

Proof. Let us select vertices s_1, s_2, ... from X as long as possible, with the requirement that the distance of s_i is more than ϵW from every s_j, $1 \leq j < i$. Suppose s_t is the last vertex selected in this way. We claim that $t \leq 1 + 2/\epsilon$. Consider a shortest closed tour in G that visits the vertices $\{s_1, \ldots, s_t\}$ (not necessarily in the order of their indices). As the distance between any two such vertices is more than ϵW, the total length of the tour is more than $t\epsilon W$ (assuming that $t > 1$). On the other hand, all these vertices are on the tree T, and it is well-known that there is a closed tour that visits every vertex of the tree in such a way that every edge of the tree is traversed exactly twice and no other edge of the graph is used. Therefore, the shortest tour has length at most $2W$, and the claim $t \leq 2/\epsilon$ follows. □

The following consequence of the definition of group is easy to see.

Proposition 9.4.2. *If S_1, S_2, X_1, X_2 are subsets of vertices of G and r_1, r_2 are real numbers, then*

$$\mathcal{G}_G(X_1, S_1, r_1) \cup \mathcal{G}_G(X_2, S_2, r_2) \subseteq \mathcal{G}_G(X_1 \cup X_2, S_1 \cup S_2, \max\{r_1, r_2\}).$$

9.4.2 Conforming solutions

Let $(T, \mathcal{B})$ be a rooted nice tree decomposition of width k, let I be the nodes of T, and let $\mathcal{B} = \{B_i \mid i \in I\}$ be the bags of the decomposition. For every $i \in I$, let V_i be the set of vertices appearing in B_i or in the bag of a descendant of i. Let A_i be the set of *active vertices* at bag B_i: those vertices $v \in V_i$ for which there is a demand $\{v, w\} \in \mathcal{D}$ with $w \notin V_i$. Let $G_i := G[V_i]$. A Steiner forest F induces a partition $\pi_i(F)$ of A_i for every $i \in I$: let two vertices of A_i be in the same class of $\pi_i(F)$ if and only if they are in the same component of F. Note that if F is restricted to G_i, then a component of F can be split into up to $k+1$ components, thus $\pi_i(F)$ is a coarser partition than the partition defined by the components of the restriction of F to G_i. See Figure 9.5 for an example.

Let $\Pi = (\Pi_i)_{i \in I}$ be a collection such that Π_i is a set of partitions of A_i. If some Steiner forest F satisfies $\pi_i(F) \in \Pi_i$ for every $i \in I$, then we say that F *conforms* to Π.

The aim of this subsection is to give an algorithm for bounded-treewidth graphs that finds a minimum-length solution conforming to a given Π. For fixed k, the running time is polynomial in the size of the graph and the size of the collection Π on a graph with treewidth at most k. In Section 9.4.3, we construct a polynomial-size collection Π such that there is a $(1+\epsilon)$-approximate solution that conforms to Π. Putting together these two results, we get a PTAS for Steiner Forest on bounded-treewidth graphs.

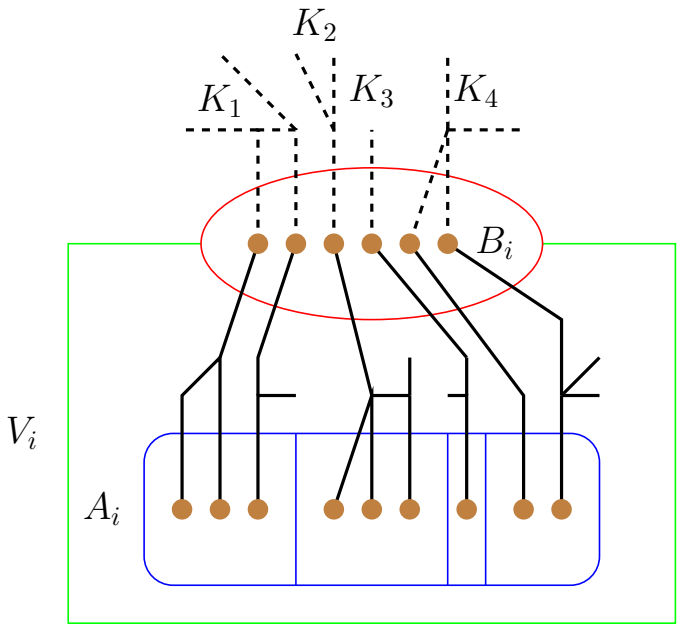


Figure 9.3: The 4 components K_1, K_2, K_3, K_3 of F partition A_i into 4 classes. Note that the restriction of F to V_i has 6 components.

Lemma 9.4.3. *For every fixed k, there is a polynomial time algorithm that, given a graph G with treewidth at most k and a collection Π, finds the minimum-length Steiner forest conforming to Π.*

The proof of Lemma 9.4.3 follows the standard dynamic programming approach, but it is not completely trivial. First, we use a technical trick that makes the presentation of the dynamic programming algorithm simpler. We can assume that every terminal vertex v has degree 1: otherwise, moving the terminal to a new degree 1 vertex v' attached to v with an edge (v, v') having length 0 does not change the problem and does not increase treewidth. Thus by Lemma 2.3.5, it can be assumed that we have a nice tree decomposition $(T, \mathcal{B})$ of width at most k where no terminal vertex is introduced and the join nodes contain no terminal vertices; this assumption simplifies the presentation. For the rest of the section, we fix such a tree decomposition and notation V_i, A_i, etc. refer to this fixed decomposition.

Recall that we can treat partitions as equivalence relations. If F is a subgraph of

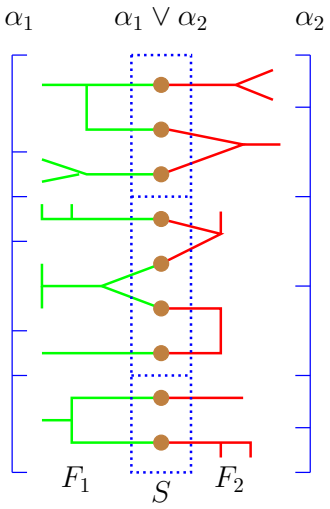


Figure 9.4: Forest F_1 induces partition α_1 on S; forest F_2 induces partition α_2; and the union of the two forests induces the partition $\alpha_1 \vee \alpha_2$.

G and $S \subseteq V(G)$, then F *induces* a partition α of S: $(x, y) \in \alpha$ if and only if x and y are in the same component of F (and every $x \in S \setminus V(F)$ forms its own class). Let F_1, F_2 be subgraphs of G, and suppose that F_1 and F_2 induce partitions α_1 and α_2 of a set $S \subseteq V(G)$, respectively. If F_1 and F_2 intersect only in S, then the partition induced by the union of F_1 and F_2 is exactly $\alpha_1 \vee \alpha_2$ (see Figure 9.4). Let β_i be a partition of B_i for some $i \in I$, and let F_i be a subgraph of $G[V_i]$. Then, we denote by $F_i + \beta_i$ the graph obtained from F_i by adding a new edge (x, y) for every $(x, y) \in \beta_i$. Note that $F_i + \beta_i$ is not necessarily a subgraph of $G[V_i]$.

Following the usual method of designing algorithms for bounded-treewidth graphs, we define several subproblems for each node $i \in I$. A subproblem at node i corresponds to finding a subgraph F_i in G_i satisfying certain properties: informally speaking, F_i is supposed to be the restriction of a Steiner forest F to V_i. The properties defining a subproblem prescribe how F_i should look from the "outside world" (i.e., from the part of G outside V_i), and they contain all the information necessary for deciding whether F_i can be extended, by edges outside V_i, to a conforming solution. Let us discuss briefly and informally what information these prescriptions should contain.

Clearly, the edges of F_i in B_i and the way F_i connects the vertices of B_i (i.e., the partition α of B_i induced by F_i) is part of this information. Furthermore, the way F_i partitions A_i should also be part of this information. However, there is a subtle detail that makes the description of our algorithm significantly more technical. The definition of $\pi_i(F) = \pi$ means that the components of F partition A_i in a certain way. But the restriction F_i of F to V_i might induce a finer partition of A_i than π: it is possible that two components of F_i are in the same component of F (see Figure 9.5). This means that we cannot require that the partition of A_i induced by F_i belongs to Π. We avoid this problem by "imagining" the partition β of B_i induced by the full solution F, and require that F_i partition A_i according to π if each class of β becomes connected somehow. In other words, instead of requiring that F_i itself partitions A_i in a certain way, we require that $F_i + \beta$ induce a certain partition.

Formally, each subproblem P is defined by a tuple $(i, H, \pi, \alpha, \beta, \mu)$, where

(S1) $i \in I$ is a node of T,

(S2) H is a spanning subgraph of $G[B_i]$ (i.e., contains all vertices of $G[B_i]$),

(S3) $\pi \in \Pi_i$ is a partition of A_i,

(S4) α, β are partitions of B_i, β is coarser than α, and α is coarser than the partition induced by the components of H, and

(S5) μ is an injective mapping from the classes of π to the classes of β.

The solution $c(i, H, \pi, \alpha, \beta, \mu)$ of a subproblem P is the minimum length of a subgraph F_i of $G[V_i]$ satisfying all of the following requirements.

(C1) $F_i[B_i] = H$ (which implies $B_i \subseteq V(F_i)$).

(C2) α is the partition of B_i induced by F_i.

(C3) The partition of A_i induced by $F_i + \beta$ is π.

(C4) For every descendant d of i (including $d = i$), the partition of A_d induced by $F_i + \beta$ belongs to Π_d.

(C5) If there is a terminal pair (x_1, x_2) with $x_1, x_2 \in V_i$, then they are connected in $F_i + \beta$.

(C6) Every $x \in A_i$ is in the component of $F_i + \beta$ containing $\mu(\pi(x))$.

We solve these subproblems by bottom-up dynamic programming. Let us discuss how to solve a subproblem depending on the type of i.

Leaf nodes i. If i is a leaf node, then the value of the solution is trivially 0.

Join node i having children i_1, i_2. Note that A_{i_1} and A_{i_2} are disjoint: the vertices of a join node are not terminal vertices. The set A_i is a subset of $A_{i_1} \cup A_{i_2}$ and it may be a proper subset; i.e., if there is a pair (x, y) with $x \in A_{i_1}$, $y \in A_{i_2}$, then x or y might not be in A_i.

We show that the value of the subproblem is

$$c(i, H, \pi, \alpha, \beta, \mu) = \min_{(J1),(J2),(J3),(J4)} \left[c(i_1, H, \pi^1, \alpha^1, \beta, \mu^1) + c(i_2, H, \pi^2, \alpha^2, \beta, \mu^2) - \ell(H)\right], \quad (9.4)$$

where the minimum is taken over all tuples satisfying, for $p = 1, 2$, all of the following conditions.

(J1) $\alpha^1 \vee \alpha^2 = \alpha$.

(J2) π and π^p are the same on $A_{i_p} \cap A_i$.

(J3) For every $v \in A_{i_p} \cap A_i$, $\mu(\pi(v)) = \mu^p(\pi^p(v))$. (Note that the classes of β are the domain of both μ and μ^p.)

(J4) If there is a terminal pair (x_1, x_2) with $x_1 \in A_1$ and $x_2 \in A_2$, then $\mu^1(\pi^1(x_1)) = \mu^2(\pi^2(x_2))$.

We will use the following observation repeatedly. Let F be a subgraph of G_i and let $F^p = F[V_{i_p}]$. Suppose that F induces partition α on B_i and β is coarser than α. Then two vertices of V_{i_p} are connected in $F + \beta$ if and only if they are connected in $F^p + \beta$. Indeed, F^{3-p} does not provide any additional connectivity compared to $F^p + \beta$: as β is coarser than α_{3-p}, if two vertices of B_i are connected by a path in F^{3-p}, then they are already adjacent in $F^p + \beta$.

We prove Equation (9.4) in two parts. First we show that any solution found by the formula is valid.

Proof of (9.4) *left* $\leq$ (9.4) *right.*

Let $P_1 = (i_1, H, \pi^1, \alpha^1, \beta, \mu^1)$ and $P_2 = (i_2, H, \pi^2, \alpha^2, \beta, \mu^2)$ be subproblems minimizing the right-hand side of (9.4), and let F^1 and F^2 be optimal solutions of P_1 and P_2, respectively. Let F be the union of subgraphs F^1 and F^2. It is clear that the length of F is exactly the right-hand side of (9.4): the common edges of F^1 and F^2 are exactly the edges of H. We show that F is a solution of P, i.e., F satisfies requirements (C1)–(C6).

(C1): Follows from $F^1[B_i] = F^2[B_i] = F[B_i] = H$.

(C2): Follows from (J1) and the fact that F^1 and F^2 intersect only in B_i.

(C3): First consider two vertices $x, y \in A_{i_p} \cap A_i$. Vertices x and y are connected in $F + \beta$ if and only if they are connected in $F^p + \beta$. By (C3) for F^p, this is equivalent to $(x, y) \in \pi^p$, which is further equivalent to $(x, y) \in \pi$ by (J2). Now suppose that $x \in A_{i_1} \cap A_i$ and $y \in A_{i_2} \cap A_i$. In this case, x and y are connected in $F + \beta$ if and only if there is a vertex of B_i reachable from x in $F^1 + \beta$ and from y in $F^2 + \beta$, or in other words, $\mu^1(\pi^1(x)) = \mu^2(\pi^2(y))$. By (J3), this is equivalent to $\mu(\pi(x)) = \mu(\pi(y))$, or $(x, y) \in \pi$ (as μ is injective).

(C4): If d is a descendant of i_p, then the statement follows using that (C4) holds for solution F^p of P^p and the fact that for every descendant d of i_p, $F_i + \beta$ and

$F + \beta$ induce the same partition of A_d. For $d = i$, the statement follows from the previous paragraph, i.e., from the fact that $F + \beta$ induces partition $\pi \in \Pi_i$ on A_i.

(C5): Consider a pair (x_1, x_2). If $x_1, x_2 \in V_{i_1}$ or $x_1, x_2 \in V_{i_2}$, then the statement follows from (C5) on F^1 or F^2. Suppose now that $x_1 \in V_{i_1}$ and $x_2 \in V_{i_2}$; in this case, we have $x_1 \in A_{i_1}$ and $x_2 \in A_{i_2}$. By (C6) on F^1 and F^2, x_p is connected to $\mu^p(\pi^p(x_p))$ in $F^p + \beta$. By (J4), we have $\mu^1(\pi^1(x_1)) = \mu^2(\pi^2(x_2))$, hence x_1 and x_2 are connected to the same class of β in $F + \beta$.

(C6): Consider an $x \in A_i$ that is in A_{i_p}. By condition (C6) on F^p, we have that x is connected in $F^p + \beta$ (and hence in $F + \beta$) to $\mu^p(\pi^p(v))$, which equals $\mu(\pi(v))$ by (J3).

Next to complete the proof of Equation (9.4), we show that any valid solution is discovered via the formula.

Proof of (9.4) *left* $\geq$ (9.4) *right.*

Let F be a solution of subproblem $(i, H, \pi, \alpha, \beta, \mu)$ and let F^p be the subgraph of F induced by V_{i_p}. To prove the inequality, we need to show three things. First, we have to define two tuples $(i_1, H, \pi^1, \alpha^1, \beta, \mu^1)$ and $(i_2, H, \pi^2, \alpha^2, \beta, \mu^2)$ that are valid subproblems, i.e., they satisfy (S1)–(S5). Second, we show that (J1)–(J4) hold for these subproblems. Third, we show that F^1 and F^2 are solutions for these subproblems (i.e., (C1)–(C6)), hence they can be used to give an upper bound on the right-hand side that matches the length of F.

Let α^p be the partition of B_i induced by the components of F^p; as F^1 and F^2 intersect only in B_i, we have $\alpha = \alpha^1 \vee \alpha^2$, ensuring (J1). Since β is coarser than α, it is coarser than both α^1 and α^2. Let π^p be the partition of A_{i_p} defined by $F + \beta$; we have $\pi_p \in \Pi_{i_p}$ by (C4) for F. Furthermore, by (C3) for F, π is the partition of A_i induced by $F + \beta$, hence it is clear that π and π^p are the same on $A_{i_p} \cap A_i$, so (J2)

holds. This also means that $F + \beta$ (or equivalently, $F^p + \beta$) connects a class of π^p to exactly one class of β; let μ^p be the corresponding mapping from the classes of π^p to β. Now (J4) is immediate. It is clear that the tuple $(i_p, H, \pi^p, \alpha^p, \beta, \mu^p)$ satisfies (S1)–(S5).

We show that F^p is a solution of subproblem $(i_p, H, \pi^p, \alpha^p, \beta, \mu^p)$. As the edges of H are shared by F^1 and F^2, it will follow that the right-hand side of (9.4) is not greater than the left hand side.

(C1): Obvious from the definition of F^1 and F^2.

(C2): Follows from the way α^p is defined.

(C3): Follows from the definition of π^p, and the fact that $F + \beta$ and $F^p + \beta$ induce the same partition on A_{i_p}.

(C4): Follows from (C4) on F and from the fact that $F + \beta$ and $F^p + \beta$ induces the same partition on A_d.

(C5): Suppose that $x_1, x_2 \in V_{i_p}$. Then by (C5) for F, x_1 and x_2 are connected in $F + \beta$, hence they are connected in $F_i + \beta$ as well.

(C6): Follows from the definition of μ^p.

Introduce node i of vertex v. Let j be the child of i. Since v is not a terminal vertex, we have $A_j = A_i$. Let F' be a subgraph of $G[V_j]$ and let F_S be obtained from F' by adding vertex v to F' and making v adjacent to $S \subseteq B_j$. If α' is the partition of B_j induced by the components of F', then we define the partition $\alpha'[v, S]$ of B_i to be the partition obtained by joining all the classes of α' that intersect S and adding v to this new class (if $S = \emptyset$, then $\{v\}$ is a class of $\alpha'[v, S]$). It is clear that $\alpha'[S, v]$ is the partition of B_i induced by F_S.

We show that the value of a subproblem is given by

$$c(i, H, \pi, \alpha, \beta, \mu) = \min_{(I1),(I2),(I3)} \left[c(j, H[B_j], \pi, \alpha', \beta', \mu') + \sum_{e \in \delta_H(v)} \ell(e) \right], \tag{9.5}$$

where the minimum is taken over all tuples satisfying the following.

(I1) $\alpha = \alpha'[v, S]$, where S is the set of neighbors of v in H.

(I2) β' is β restricted to B_j.

(I3) For every $x \in A_i$, $\mu(\pi(x))$ is the class of β containing $\mu'(\pi(x))$.

In a way similar to Equation (9.4), we prove Equation (9.5) in two parts. First we show that any solution found by the formula is valid.

Proof of (9.5) *left* $\leq$ (9.5) *right.*

Let F' be an optimal solution of subproblem $P' = (j, H[B_j], \pi, \alpha', \beta', \mu')$ satisfying (I1)-(I3). Let F be the graph obtained from F' by adding to it the edges of H incident to v; it is clear that the length of F is exactly the right-hand side of (9.5). Let us verify that (C1)–(C6) hold for F.

(C1): Immediate.

(C2): Holds because of (I1) and the way $\alpha'[v, S]$ was defined.

(C3)–(C5): Observe that $F + \beta$ connects two vertices of V_j if and only if $F' + \beta'$ does. Indeed, if a path in $F + \beta$ connects two vertices via vertex v, then the two neighbors x, y of v on the path are in the same class of β as v (using the fact that α and β are coarser than the partition induced by H), hence (I2) implies that x, y are in the same class of β' as well. In particular, for every descendant d of i, the components of $F + \beta$ and the components of $F' + \beta$ give the same partition of A_d.

(C6): Follows from (C6) for F' and from (I3).

Next to complete the proof of Equation (9.5), we show that any valid solution is discovered via the formula.

Proof of (9.5) *left* $\geq$ (9.5) *right.*

Let F be a solution of subproblem $(i, H, \pi, \alpha, \beta, \mu)$ and let F' be the subgraph of F induced by V_j. We define a tuple $(j, H[B_j], \pi, \alpha', \beta', \mu')$ that is a valid subproblem, show that it satisfies (I1)–(I3), and that F' is a solution of this subproblem.

Let α' be the partition of V_j induced by F' and let β' be the restriction of β on B_j; these definitions ensure that (I1) and (I2) hold. Let $\mu'(\pi(x)) = \mu(\pi(x)) \setminus \{v\}$, which is a class of β'; clearly, this ensures (I3). Note that this is well-defined, as it is not possible that $\mu(\pi(x))$ is a class of β consisting of only v: by (C6) for F, this would mean that v is the only vertex of B_i reachable from x in F. Since v is not a terminal vertex, $v \neq x$; thus, if v is reachable from x, then at least one neighbor of v has to be reachable from x as well.

Let us verify that (S1)–(S5) hold for the tuple $(j, H[B_j], \pi, \alpha', \beta', \mu')$. (S1) and (S2) clearly hold. (S3) follows from the fact that (C4) holds for F and $A_i = A_j$. To see that (S4) holds, observe that $(x, y) \in \alpha'$ implies $(x, y) \in \alpha$, which implies $(x, y) \in \beta$, which implies $(x, y) \in \beta'$. (S5) is clear from the definition of μ'.

The difference between the length of F and the length of F' is exactly $\sum_{e \in \delta_H(v)} \ell(e)$. Thus to show that the left-hand side of (9.5) is at most the right-hand side of (9.5), it is sufficient to show that F' is a solution of subproblem $(j, H[B_j], \pi, \alpha', \beta', \mu')$.

(C1)–(C2): Obvious.

(C3)–(C5): As in the other direction, follow from the fact that $F' + \beta'$ induces the same partition of V_j as $F + \beta$.

(C6): By the definition of μ', it is clear that $\mu'(\pi(x))$ is exactly the subset of B_j that is reachable from x in $F + \beta$ and hence in $F' + \beta'$.

Forget node i of vertex v. Let j be the child of i. We have $V_i = V_j$ and hence $A_i = A_j$. We show that the value of a subproblem is given by

$$c(i, H, \pi, \alpha, \beta, \mu) = \min_{(F1),(F2),(F3),(F4)} c(j, H', \pi, \alpha', \beta', \mu'), \tag{9.6}$$

where the minimum is taken over all tuples satisfying the following.

(F1) $H'[B_i] = H$.

(F2) α is the restriction of α' to B_i.

(F3) β is the restriction of β' to B_i and $(x, v) \in \beta'$ if and only if $(x, v) \in \alpha'$.

(F4) For every $x \in A_i$, $\mu(\pi(x))$ is the (nonempty) set $\mu'(\pi(x)) \setminus \{v\}$ (which implies that $\mu'(\pi(x))$ contains at least one vertex of B_i).

We prove Equation (9.6) in two parts. First we show that any solution found by the formula is valid.

Proof of (9.6) *left* $\leq$ (9.6) *right.*

Let F be a solution of $(j, H', \pi, \alpha', \beta', \mu')$. We show that F is a solution of $(j, H, \pi, \alpha, \beta, \mu)$ as well.

(C1): Clear because of (F1).

(C2): Clear because of (F2).

(C3)–(C5): We only need to observe that $F+\beta$ and $F+\beta'$ have the same components: since by (F3), $(x, v) \in \beta'$ implies $(x, v) \in \alpha'$, the neighbors of v in $F+\beta'$ are reachable from v in F, thus $F + \beta'$ does not add any further connectivity compared to $F + \beta$.

(C6): Observe that, if $\mu'(\pi(x))$ are the vertices of B_j reachable from x in $F+\beta'$, then $\mu(\pi(x)) = \mu'(\pi(x)) \setminus \{v\}$ are the vertices of B_i reachable from x in $F+\beta'$. We have already seen that $F+\beta$ and $F+\beta'$ have the same components, thus, the nonempty set $\mu(\pi(x))$ is indeed the subset of B_i reachable from x in $F+\beta$. Furthermore, by (F3), β is the restriction of β' on B_i, thus, if $\mu'(\pi(x))$ is a class of β', then $\mu(\pi(x))$ is a class of β.

Next to complete the proof of Equation (9.6), we show that any valid solution is discovered via the formula.

Proof of (9.6) *left* $\geq$ (9.6) *right.*

Let F be a solution of $(j, H, \pi, \alpha, \beta, \mu)$. We define a tuple $(j, H', \pi, \alpha', \beta', \mu')$ that is a subproblem, we show that (F1)–(F3) hold, and that F is a solution of this subproblem.

Let us define $H' = F[B_j]$ and let α' be the partition of B_j induced by the components of F; these definitions ensure that (F1) and (F2) hold. We define β' as the partition obtained by extending β to B_j such that v belongs to the class of β that contains a vertex $x \in B_i$ with $(x, v) \in \alpha'$ (as β is coarser than the partition induced by H, there is at most one such class; if there is no such class, then we let $\{v\}$ be a class of β'). It is clear that (F3) holds for this β'. Let us note that $F+\beta$ and $F+\beta'$ have the same connected components: if $(x, v) \in \beta'$, then x and v are connected in F. Let $\mu'(\pi(x))$ be the subset of B_j reachable from x in $F+\beta'$ (or equivalently, in $F+\beta$). It is clear that $\mu(\pi(x)) = \mu'(\pi(x)) \setminus \{v'\}$ holds, hence (F4) is satisfied.

Let us verify first that (S1)–(S5) hold for $(j, H', \pi, \alpha', \beta', \mu')$. (S1) and (S2) clearly holds. (S3) follows from the fact that (S3) holds for $(i, H, \pi, \alpha, \beta, \mu)$ and $A_i = A_j$. To see that (S4) holds, observe that, if $x, y \in B_i$, then $(x, y) \in \alpha'$ implies $(x, y) \in \alpha$, which implies $(x, y) \in \beta$, which implies $(x, y) \in \beta'$. Furthermore, if $(x, v) \in \alpha'$, then $(x, v) \in \beta'$ by the definition of β. (S5) is clear from the definition of μ'.

We show that F is a solution of $(j, H', \pi, \alpha', \beta', \mu')$.

(C1): Clear from the definition of H'.

(C2): Clear from the definition of α'.

(C3)–(C5): Follow from the fact that $F+\beta$ and $F+\beta'$ have the same connected components.

(C6): Follows from the definition of μ'.

This concludes the proof of Lemma 9.4.3.

9.4.3 Constructing the partitions

Recall that the collection $\Pi = (\Pi_i)_{i\in I}$ contains a set of partitions Π_i for each $i \in I$. We construct these sets Π_i in the following way. Each partition in Π_i is defined by a sequence $((S_1, r_1), \ldots, (S_p, r_p))$ of at most $k+1$ pairs and a partition ρ of $\{1, \ldots, p\}$. The pair (S_j, r_j) consists of a (seed) set S_j of $O((k+1)(1+1/\epsilon))$ vertices of G_i and a nonnegative real number r_j, which equals the distance between two vertices of G. This means that there are at most $|V(G)|^{O((k+1)(1+1/\epsilon))} \cdot |V(G)|^2$ possible pairs (S_j, r_j) and hence at most $|V(G)|^{O((k+1)^2(1+1/\epsilon))}$ different sequences. The number of possible partitions ρ is $O(k^k)$. Thus, if we construct Π_i by considering all possible sequences constructed from every possible choice of (S_j, r_j), the size of Π_i is polynomial in $|V(G)|$ for every fixed k and ϵ.

We construct the partition π corresponding to a particular sequence and ρ the following way. Each pair (S_j, r_j) can be used to define a group $R_j = \mathcal{G}_G(A_i, S_j, r_j)$ of A_i. Roughly speaking, for each class P of ρ, there is a corresponding class of π that contains the union of R_j for every $j \in P$. However, the actual definition is somewhat more complicated. We want π to be a partition, which means that the subsets of A_i corresponding to the different classes of ρ should be disjoint. In order to ensure disjointness, we define $R'_j := R_j \setminus \bigcup_{j'=1}^{j-1} R_{j'}$. The partition π of A_i is then constructed as follows: for each class P of ρ, we let $\bigcup_{j\in P} R'_j$ be a class of π. Note that these

classes are disjoint by construction. If these classes fully cover A_i, then we place the resulting partition π into Π_i; otherwise, the sequence does not define a partition. This finishes the construction of Π_i.

Before showing that there is a near-optimal solution conforming to the collection Π defined above, we need a further definition. For two vertices u and v, we denote by $u \prec v$ the fact that the topmost bag containing u is a proper descendant of the topmost bag containing v. Note that each bag is the topmost bag of at most one vertex in a nice tree decomposition. (Recall that we can assume that the root bag contains only a single vertex.) Thus, if u and v appear in the same bag, then $u \prec v$ or $v \prec u$ holds, i.e., this relation defines a total ordering of the vertices in a bag. We can extend this relation to connected subsets of vertices: for two disjoint connected sets K_1, K_2, $K_1 \prec K_2$ means that K_2 has a vertex v such that $u \prec v$ for every vertex $u \in K_1$, or in other words, $K_1 \prec K_2$ means that the topmost bag where vertices from K_1 appear is a proper descendant of the topmost bag where vertices from K_2 appear. If there is a bag containing vertices from both K_1 and K_2, then either $K_1 \prec K_2$ or $K_2 \prec K_1$ holds. The reason for this is that the bags containing vertices from $K_1 \cup K_2$ form a connected subtree of the tree decomposition, and if the topmost bag in this subtree contains vertex $v \in K_1 \cup K_2$, then $u \prec v$ for every other vertex u in $K_1 \cup K_2$.

Lemma 9.4.4. *There is a $(1 + k\epsilon)$-approximate solution conforming to Π.*

Proof. Let F be a minimum-length Steiner forest. We describe a procedure that adds further edges to F to transform it into a Steiner forest F' that conforms to Π, and has length at most $(1 + k\epsilon)\ell(F)$. We need a delicate charging argument to show that the total increase of the length is at most $k\epsilon \cdot \ell(F)$ during the procedure. In each step, we charge the increase of the length to an ordered pair (K_1, K_2) of components of F. We charge only to pairs (K_1, K_2) having the property that $K_1 \prec K_2$ and there is a bag containing vertices from both K_1 and K_2. Observe that, if B_i is the topmost bag where vertices from K_1 appear, then these properties imply that a vertex of K_2

appears in this bag as well. Otherwise, if every bag containing vertices of K_2 appears above B_i, then there is no bag containing vertices from both K_1 and K_2; if every bag containing vertices from K_2 appears below B_i, then $K_1 \prec K_2$ is not possible. Thus a component K_1 can be the first component of at most k such pairs (K_1, K_2): since the components are disjoint, the topmost bag containing vertices from K_1 can intersect at most k other components. We will charge a length increase of at most $\epsilon \cdot \ell(K_1)$ to the pair (K_1, K_2), thus, the total increase is at most $k\epsilon \cdot \ell(F)$. It is a crucial point of the proof that we charge to (pairs of) components of the original solution F, even after several modification steps, when the components of F' can be larger than the original components of F. Actually, in the proof to follow, we will refer to three different types of components.

(a) Components of the current solution F'.

(b) Each component of F' contains one or more components of F.

(c) If a component of F is restricted to the subset V_i, then it can split into up to $k+1$ components.

To emphasize the different meanings, and be clear as well, we use the terms a-component, b-component, and c-component.

Initially, we set $F' := F$ and it will be always true that F' is a supergraph of F, thus F' defines a partition of the b-components of F. Suppose there is a bag B_i such that the partition $\pi_i(F')$ of A_i induced by F' is not in Π_i. Let $K_1 \prec K_2 \prec \cdots \prec K_p$ be the b-components of F intersecting B_i, ordered by the relation $\prec$. Some of these b-components might be in the same a-component of F'; let ρ be the partition of $\{1, \ldots, p\}$ defined by F' on these b-components.

Let $A_{i,j}$ be the subset of A_i contained in K_j. The intersection of b-component K_j with V_i gives rise to at most $k+1$ c-components, each of length at most $\ell(K_j)$. Thus by Lemma 9.4.1 and Proposition 9.4.2, there is a set $S_j \subseteq V(K_j)$ of at most

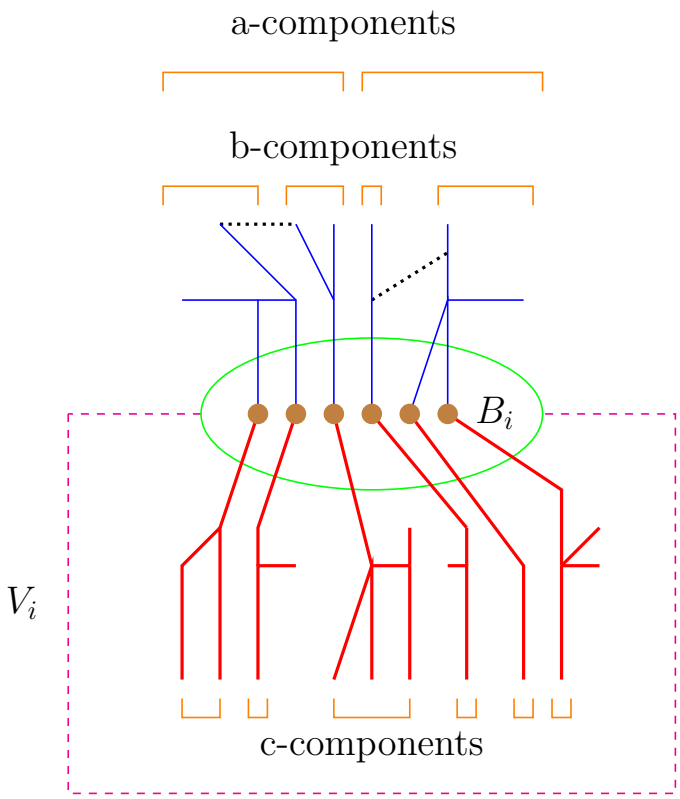


Figure 9.5: The original solution F (blue and red edges) consists of 4 b-components. Restricting F to V_i yields 6 c-components (red edges). Forest F', which is obtained from F by adding the two dotted edges, has two a-components.

$O((k+1)(1+1/\epsilon))$ vertices such that $A_{i,j} = \mathcal{G}_{G_i}(A_{i,j}, S_j, r_j)$ for some $r_j \leq \epsilon \cdot \ell(K_j)$. If the sequence $(S_1, r_1), \ldots, (S_p, r_p)$ and the partition ρ give rise to the partition $\pi_i(F')$, then $\pi_i(F') \in \Pi_i$. Otherwise, let us investigate the reason why this sequence and ρ do not define the partition $\pi_i(F')$. Let R_j, R'_j be defined as in the definition of Π_i, i.e., $R_j = \mathcal{G}_G(A_i, S_j, r_j)$ and $R'_j := R_j \setminus \bigcup_{j'=1}^{j-1} R_{j'}$. It is clear that $A_{i,j} \subseteq R_j$. Therefore, every vertex of A_i is contained in some R_j and hence in some R'_j. Thus, the sequence does define a partition π, but maybe a partition different from $\pi_i(F')$. Let $\rho(j)$ be the class of ρ containing j. If for every $1 \leq j \leq p$, every vertex of $A_{i,j}$ is contained in $\bigcup_{j' \in \rho(j)} R'_{j'}$, then π and $\pi_i(F')$ are the same. So suppose that some vertex $v \in A_{i,j}$ is not in $\bigcup_{j' \in \rho(j)} R'_{j'}$. As $v \in R_j$, this means that $v \in R_{j^*}$ for some $j^* \prec j$ and $j^* \notin \rho(j)$. The fact that $R_{j^*} = \mathcal{G}_{G_i}(A_i, S_{j^*}, r_{j^*})$ contains $v \in A_{i,j}$ means that there is a vertex $u \in S_{j^*}$ such that $d_{G_i}(u, v) \leq r_{j^*} \leq \epsilon \cdot \ell(K_{j^*})$. Note that u is a vertex of b-component K_{j^*} (as $u \in S_{j^*}$ and by definition S_{j^*} is a subset of $V(K_{j^*})$), and v is a vertex of b-component K_j. We modify F' by adding a shortest path that connects u and v. Clearly, this increases the length of F' by at most $\epsilon \cdot \ell(K_{j^*})$, which we charge to the pair (K_{j^*}, K_j) of b-components. Note that K_j and K_{j^*} both intersect the bag B_i and $K_{j^*} \prec K_j$, as required in the beginning of the proof. Furthermore, K_j and K_{j^*} are in the same a-component of F' after the modification, but not before. Thus we charge at most once to the pair (K_{j^*}, K_j).

Since the modification always extends F', the procedure described above terminates after a finite number of steps. At this point, every partition $\pi_i(F')$ belongs to the corresponding set Π_i, that is, the solution F' conforms to Π. □

9.5 Steiner forest for planar graphs

This section proves Theorem 3.2.5, the most general result of this chapter, which is a PTAS for Steiner Forest for bounded-genus graphs. To this end, we follow the

framework explained in Section 5.3; i.e., we first construct a Steiner forest spanner given the input graph G and the set of demands $\mathcal{D}$. The construction is based on the prize-collecting clustering paradigm—see Theorem 4.2.1—and the brick decomposition technique introduced in Chapter 6. The spanner framework then enables us to reduce the problem to a bounded-treewidth instance, for which a PTAS was given in Section 9.4.

After recalling some notation and definitions, we explain the preprocessing step in the construction of the spanner in Section 9.5.1, which invokes Procedure `PC-Classify` and its corresponding Theorem 4.2.1. Then, in Section 9.5.2 we explain the rest of the spanner construction, that is essentially the same as that of Borradaile et al. [BKM09] for planar Steiner Tree. Finally, we summarize in Section 9.5.3.

Recall from Definition 5.1.1 that a Steiner forest spanner is a subgraph of the given graph whose length is no more than a constant factor times the length of the optimal Steiner forest, and furthermore, it contains a nearly optimal Steiner forest. Denote by $\mathsf{opt}_{\mathcal{D}}(G)$ the minimum length of a Steiner forest of G satisfying (connecting) all the demands in $\mathcal{D}$. We sometimes use opt instead of $\mathsf{opt}_{\mathcal{D}}(G)$ when $\mathcal{D}$ and G are easily inferred from the context. More specifically, a subgraph H of G is a Steiner forest spanner with respect to demand set $\mathcal{D}$ and a parameter $\epsilon > 0$ if it has the following two properties.

Spanning Property: There is a forest in H that connects all demands in $\mathcal{D}$ and has length at most $(1+\epsilon)OPT_{\mathcal{D}}(G)$, namely, $\mathsf{opt}_{\mathcal{D}}(H) \leq (1+\epsilon)\mathsf{opt}_{\mathcal{D}}(G)$.

Shortness Property: The total length of H is not more than $f(\epsilon) \cdot OPT_{\mathcal{D}}(G)$.

Most of this section is devoted to proving the following theorem.

Theorem 9.5.1. *Given any fixed $\epsilon > 0$, a bounded-genus graph $G_{in}(V_{in}, E_{in})$ and demand pairs $\mathcal{D}$, we can compute in polynomial time a Steiner forest spanner H for G_{in} with respect to demand set $\mathcal{D}$.*

The algorithm that we propose achieves this in time $O(n^2 \log n)$. The entire algorithm for Steiner Forest runs in polynomial time but the exponent of the polynomial depends on ϵ and the genus of the input graph, so our algorithm is not an efficient PTAS. What leads to the large running time is the algorithm for the bounded-treewidth case. Improving the running time of that algorithm—ideally to obtain an efficient PTAS—immediately gives a better running time for the bounded-genus algorithm.

9.5.1 Preprocessing

A slight modification of the ideas of Borradaile et al. [BKM09] would give a spanner construction algorithm for Steiner Forest if one were able to find a good "backbone" graph to start with: i.e., a connected subgraph that spans all terminals (endpoints of every demand pair) with total length $O(\mathsf{opt})$.

Notice that the backbone graph can be easily found for Steiner Tree since any constant-factor approximate solution—in particular, that found from the Minimum Spanning Tree 2-approximation—can serve as the backbone.

For Steiner Forest, though, the existence of the backbone graph is not guaranteed. Consider a disconnected graph with two demand pairs, each in a connected component of its own. Although, the optimum may have a very small length, no subgraph of length $O(\mathsf{opt})$ can connect all the terminals together. This example is pretty easy to deal with since the optimum, we know, cannot connect different connected components to each other either. Therefore, the instance can be broken down into separate subinstances with different connected components and demand pairs inside each.

Now consider a more challenging instance. There are several demand pairs in our graph. Although any terminals from any pair of demand pairs can be connected to each other with a cost much smaller than opt, to connected *all* terminals to each

other is much more expensive than opt. In other words, unlike the above instance of the disconnected graph, it is not easy to rule out the optimum's connecting different demands to each other.

The following lemma introduces a relaxed notion of a backbone, which is sufficient for the rest of the spanner construction, and at the same time, not only can we prove that it always exists, but we can always find one in polynomial time as well.

Theorem 9.5.2. *Given a parameter $\epsilon > 0$, a graph $G_{in}(V_{in}, E_{in})$, and a set $\mathcal{D}$ of pairs of vertices, we can compute in polynomial time a set of trees $\{T_1, \ldots, T_k\}$, and a partition of demands $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$, with the following properties.*

1. *All the demands are covered, i.e., $\mathcal{D} = \bigcup_{i=1}^{k} \mathcal{D}_i$.*
2. *All the terminals in $\mathcal{D}_i$ are spanned by the tree T_i.*
3. *The sum of the lengths of all the trees T_i is no more than $(\frac{4}{\epsilon} + 2)\mathsf{opt}_{\mathcal{D}}(G_{in})$.*
4. *Let $\mathcal{D}^* \subseteq \mathcal{D}$ be an arbitrary subset of demands, and define $\mathcal{D}_i^* = \mathcal{D}^* \cap \mathcal{D}_i$. Then, we have $\sum_i \mathsf{opt}_{\mathcal{D}_i^*}(G_{in}) \leq (1+\epsilon)\mathsf{opt}_{\mathcal{D}^*}(G_{in})$. In particular, $\sum_i \mathsf{opt}_{\mathcal{D}_i}(G_{in}) \leq (1+\epsilon)\mathsf{opt}_{\mathcal{D}}(G_{in})$.*

The (second part of the) last condition implies that (up to a small factor $1+\epsilon$) it is possible to solve the demands $\mathcal{D}_i$ separately. Notice that this may lead to paying for portions of the solution more than once.

This chapter does not use the stronger form of the fourth condition of the theorem. The latter part of the condition suffices for the discussion of Steiner Forest algorithm, however, the stronger form is required in Chapter 10 where we look at Prize-collecting Steiner Forest.

We emphasize here that the above theorem works for any graph, and does not need a constant ϵ; however, it is the rest of the construction—see Section 9.5.2—that requires a small genus for the graph.

We now show that Procedure `PC-Partition` achieves the desired result of Theorem 9.5.2.

Algorithm 9.1 `PC-Partition`$(G_{in}, \mathcal{D})$

Input: graph $G_{in}(V_{in}, E_{in})$, and demands $\mathcal{D}$
Output: set of trees T_i with associated $\mathcal{D}_i$

1: $F^* \leftarrow$ 2-approximate solution for $\mathcal{D}$ (found, e.g., via [AKR95])
2: Let $T_1^*, T_2^*, \ldots$ denote the tree components of F^*
3: Contract each tree T_i^* to build a new graph $G(V, E)$
4: Define $\phi : V \mapsto \mathbb{R}^+$ such that $\phi(v) = \ell(T_i^*)$ if v is a supervertex formed by contracting some T_i^*, and $\phi(v) = 0$ otherwise.
5: $Z \leftarrow$ `PC-Classify`(G, ϕ)
6: Construct Z_{in} from Z by uncontracting all the trees T_i^*
7: $Z_{in} \leftarrow Z_{in} \cup F^*$
8: Let $\{T_i\}_i$ denote the tree components of Z_{in}
9: $\mathcal{D}_i \leftarrow \{(s,t) \in \mathcal{D} : s, t \in V(T_i)\}$ for each i
10: **return** $(\{T_i\}_i, \{\mathcal{D}_i\}_i)$

Proof of Theorem 9.5.2. We start with a 2-approximate solution F^* satisfying all the demands in $\mathcal{D}$; such a solution can be found, for instance, via Agrawal et al.'s Steiner Forest algorithm [AKR95]. In the following, we extend F^* by connecting some of its components to make the trees T_i. It is easy to see that this construction guarantees the first two conditions of the theorem. We work on a graph $G(V, E)$ formed from G_{in} by contracting each tree component of F^*. A potential $\phi(v)$ is assigned to each vertex v of G, which is $\frac{1}{\epsilon}$ times the length of a tree component T_i^* of F^* corresponding to v in case v is the contraction of T_i^*, and zero otherwise.

Let Z be the subgraph of G given by Theorem 4.2.1. Let Z_{in} be the subgraph of G_{in} obtained from Z by uncontracting the components of F^* and adding F^* to Z_{in}; as F^* is a solution for the input instance, Z_{in} is a solution as well. Let $T_1, \ldots, T_k$ be the tree components of Z_{in} and let $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be the set of demands spanned by these trees. It is clear that the first two conditions of the theorem hold; the first one holds since Z_{in} is a valid solution, and the second condition follows from the definition of $\mathcal{D}_i$. The length of Z_{in} is the length of F^* (which is at most $2\mathsf{opt}$) plus the length

of Z (which is at most $2\phi(V) \leq \frac{4}{\epsilon}\mathsf{opt}$), giving the third condition.

Let $\mathsf{opt}_{\mathcal{D}^*}$ be an optimal Steiner forest for demand set $\mathcal{D}^*$, and let L be the corresponding subgraph of G (obtained by contracting the components of F^*). Let Q be the set of vertices of G given by the second condition of Theorem 4.2.1. For every vertex in Q, there is a corresponding component of F^*; let Q_{in} be the forest of G_{in} composed of all these components. The way the potential ϕ was defined ensures $\ell(Q_{in}) = \epsilon\phi(Q) \leq \epsilon\ell(L) \leq \epsilon\ell(\mathsf{opt}_{\mathcal{D}^*})$, where the second inequality follows from condition 2(a) of Theorem 4.2.1.

To show that the last condition holds, for every $\mathcal{D}_i^*$ we construct a subgraph H_i that satisfies the demands in $\mathcal{D}_i^*$. For every demand in $\mathcal{D}_i^*$, if the component K of F^* satisfying the demand belongs to Q_{in}, then we place K into H_i; otherwise, we put the component of $\mathsf{opt}_{\mathcal{D}^*}$ satisfying the demand into H_i. Observe that each component of Q_{in} is used in at most one H_i: as Q_{in} is a subgraph of Z_{in}, all the demands satisfied by a component K of Q_{in} belong to the same $\mathcal{D}_i^*$. Furthermore, we claim that each component of $\mathsf{opt}_{\mathcal{D}^*}$ is used in at most one H_i. Suppose that a component K of $\mathsf{opt}_{\mathcal{D}^*}$ was used in both H_i and H_j, i.e., K satisfies a demand in $\mathcal{D}_i^*$ and a demand in $\mathcal{D}_j^*$. The components of F^* satisfying these two demands are *not* in Q_{in} (otherwise we would have put these components into H_i or H_j instead of K), thus they correspond to vertices $v_1, v_2 \notin Q$ in the contracted graph G. Thus L, the contracted version of $\mathsf{opt}_{\mathcal{D}^*}$, connects two vertices $v_1, v_2 \notin Q$. Condition 2(b) of Theorem 4.2.1 implies that v_1 and v_2 are in the same component of Z and hence the two demands are satisfied by the same component of Z_{in}. This contradicts that the two demands are in two different sets $\mathcal{D}_i^* \subseteq \mathcal{D}_i$ and $\mathcal{D}_j^* \subseteq \mathcal{D}_j$.

Since every component of Q_{in} and every component of $\mathsf{opt}_{\mathcal{D}^*}$ is used by at most one H_i, we have $\sum_{i=1}^{k} \ell(H_i) \leq \mathsf{opt}_{\mathcal{D}^*} + \ell(Q_{in}) \leq (1+\epsilon)\mathsf{opt}_{\mathcal{D}^*}$. This establishes the last condition of the theorem.

The weaker form of the condition follows by setting $\mathcal{D}^* = \mathcal{D}$. □

9.5.2 Spanner construction

Here we prove Theorem 9.5.1, the existence of a Steiner forest spanner for bounded-genus graphs. The proof follows the method explained in Chapter 6. Recall that four steps were outlined therein: preprocessing, brick decomposition, portal designation, and brick processing. We already described the preprocessing step, and the next three steps essentially mirror the construction in Borradaile et al. [BKM09], with certain differences in the analysis. Procedure `SF-Spanner` summarizes the spanner construction.

Algorithm 9.2 `SF-Spanner`$(G, \mathcal{D})$

Input: graph $G(V, E)$, and demands $\mathcal{D}$
Output: subgraph H

1: $(\{T_i\}_{i=1}^{k}, \{\mathcal{D}_i\}_{i=1}^{k}) \leftarrow$ `PC-Partition`$(G, \mathcal{D})$
2: **for** $i = 1$ **to** k **do**
3: $\quad \mathcal{B}_i \leftarrow$ `Brick-Decomposition`(G, T_i, ϵ)
4: $\quad H_i \leftarrow \mathcal{B}_i$
5: $\quad$ **for** brick $B \in \mathcal{B}_i$ **do**
6: $\quad\quad$ Let Π a set of portals on ∂B where distance of any boundary vertex to closest portal $\leq \frac{\ell(\partial B)}{\theta}$
7: $\quad\quad$ **for** each $\Pi' \subseteq \Pi$ **do**
8: $\quad\quad\quad$ $T \leftarrow$ optimal Steiner tree spanning Π' using only the boundary of and inside B, and assuming W- and E-boundaries of B have length zero (via, e.g., Erickson et al. [EMV87])
9: $\quad\quad\quad$ $H_i \leftarrow H_i \cup T$
10: $H \leftarrow \bigcup_i H_i$
11: **return** H

Roughly speaking, we first use `PC-Partition` to break up the instance into simpler ones, where each has a subset of demands all whose terminals can be connected with a relatively small cost. We then build the spanner separately for each subset, however, the analysis cannot be done independently, and has to be performed together. Let us focus on one set of demands $\mathcal{D}_i$ from `PC-Partition`, and its corresponding spanning tree T_i. We find a brick decomposition (see Section 6.2) for the terminals in $\mathcal{D}_i$ with respect to G and the parameter ϵ. Recall that each face of the brick decomposition

is called a brick. We designate a set of at most θ—that depends on ϵ, g—portals on the (N- and S-) boundary of each brick, and finally include all optimal Steiner trees connecting any subset of these portals. Erickson et al. [EMV87] shows how to find in polynomial time the optimal Steiner tree if all terminals are located on the outer face of a planar graph. The spanner consists of all the brick decompositions as well as every Steiner tree added in the last step.

Next we prove the two properties of the spanner: a bound on its length in Lemma 9.5.3 and its spanning property in Lemma 9.5.5. The following lemma is the main piece in proving the shortness property. Notice that $\sum_i \ell(T_i) = O(\mathsf{opt})$. Let Q_i denote all the terminals in $\mathcal{D}_i$.

Lemma 9.5.3. *Length of H_i is at most $f(\epsilon, g)\ell(T_i)$ for a universal certain function $f(\epsilon, g)$.*

Proof. H_i is made up of the mortar graph $\mathcal{B}_i$ (of the brick decomposition) and the Steiner trees added in brick processing step. We have $\ell(\mathcal{B}_i) \leq \gamma(\epsilon, g)\mathsf{span}(Q_i) \leq \gamma(\epsilon, g)\ell(T_i)$. The brick processing step adds for each brick B at most 2^θ Steiner trees, and each such Steiner tree has length no more than $\ell(\partial(B))$ because ∂B is a candidate solution. Since an edge of $\mathcal{B}_i$ may appear in the boundary of two bricks, the total addition due to these trees is at most $2^{\theta+1}\ell(\mathcal{B}_i)$. Therefore, $\ell(H_i) \leq (2^{\theta+1}+1)\ell(\mathcal{B}_i) \leq (2^{\theta+1}+1)\gamma(\epsilon, g)\ell(T_i) = f(\epsilon, g)\ell(T_i)$. □

Next we prove a property of H_i that is crucial in establishing the spanning property of H.

Lemma 9.5.4. *For any forest $F \subseteq G$, there exists a forest $F^* \subseteq H_i$ of length at most $(1+\epsilon)\ell(F) + \epsilon^2(3+\epsilon)\ell(T_i)$ that provides the same connectivity as F among Q_i.*

Proof. Add the set of all supercolumns of H_i to F to get F^1. Recall the total length of these supercolumns is at most $\epsilon^2\mathsf{span}(Q_i) \leq \epsilon^2\ell(T_i)$. Next, use Lemma 6.3.1 to replace the intersection of F^1 and each brick with another forest having the properties of the

lemma. Let F^2 be the new forest. The length of the solution increases to no more than a $1+\epsilon$ factor. Furthermore, as a result, F^2 crosses each brick at most α times. We claim, provided that θ is sufficiently large compared to α, we can ensure that moving these intersection points to the portals introduces no more than an ϵ factor in the length.

Consider a brick B. Connect each intersection point of the brick to its closest portal. Each connection on a brick B moves by at most $\ell(\partial(B))/\theta$. The total movement of each brick is at most $\alpha\ell(\partial(B))/\theta$ which is no more than $\epsilon^2\ell(\partial(B))/\gamma(\epsilon, g)$ if $\theta \geq \alpha\gamma(\epsilon, g)/\epsilon^2$. Therefore, the total additional length for all bricks of H_i is bounded by $2\epsilon^2\ell(T_i)$ because $\sum_{B\in\mathcal{B}_i} \ell(\partial(B)) \leq 2\ell(\mathcal{B}_i) \leq 2\gamma(\epsilon, g)\ell(T_i)$.

Finally, we replace the forests inside each brick B by the Steiner trees provisioned in the last step of our spanner construction. Take a brick B. Let $K_1, K_2, \ldots$ be the connected components of F^2 inside B. Each intersection point is connected to a portal of B. Replace each K_j by the optimal Steiner tree corresponding to this subset of portals. This procedure does not increase the length and produces a graph F^*.

Clearly, F^* provides (at least) the same connectivity as F among Q_i. In addition, F^* is a subgraph of H_i, and we can bound its length as follows.

Replacing the Steiner trees by the optimal Steiner trees between portals cannot increase the length, so, the only length increase comes from connections to portals. Thus, we get

$$\ell(F^*) \leq \ell(F^2) + 2\epsilon^2\ell(T_i). \tag{9.7}$$

From the above discussion,

$$\ell(F^2) \leq (1+\epsilon)\ell(F^1), \qquad \text{by Lemma 6.3.1, and} \tag{9.8}$$

$$\ell(F^1) \leq \ell(F) + \epsilon^2\ell(T_i) \qquad \text{due to supercolumns' length.} \tag{9.9}$$

Combining (9.7), (9.8) and (9.9) yields the desired result. □

Finally we prove the spanning property of H. Recall that H is formed by the union of the graphs H_i constructed above.

Lemma 9.5.5. $\mathsf{opt}_{\mathcal{D}}(H) \le (1 + c'\epsilon)\mathsf{opt}_{\mathcal{D}}(G)$ *for a universal constant* $c' > 0$.

Proof. Take the optimal solution opt. Find forests opt_i satisfying demands $\mathcal{D}_i$, i.e., $\ell(\mathsf{opt}_i) = \mathsf{opt}_{\mathcal{D}_i}(G)$. Theorem 9.5.2 guarantees $\sum_i \ell(\mathsf{opt}_i) \le (1+\epsilon)\mathsf{opt}$.

Consider one opt_i that serves the respective set of demands $\mathcal{D}_i$. Use Lemma 9.5.4 to obtain opt_i^* from each opt_i that provides the same connectivity, and has $\ell(\mathsf{opt}_i^*) \le (1+\epsilon)\mathsf{opt}_i + \epsilon^2(3+\epsilon)\ell(T_i)$. Define $\mathsf{opt}^* = \bigcup_i \mathsf{opt}_i^*$, whose length we bound as follows.

$$\ell(\mathsf{opt}^*) \le \sum_i \ell(\mathsf{opt}_i^*),$$

where the inequality may be strict due to common edges among different opt_i's,

$$\begin{aligned}
&\le \sum_i \left[(1+\epsilon)\ell(\mathsf{opt}_i) + \epsilon^2(3+\epsilon)\ell(T_i)\right] && \text{by Lemma 9.5.4}\\
&= \sum_i \left[(1+\epsilon)\ell(\mathsf{opt}_i)\right] + \sum_i \left[\epsilon^2(3+\epsilon)\ell(T_i)\right]\\
&\le (1+\epsilon)^2\mathsf{opt} + \epsilon^2(3+\epsilon)\sum_i \ell(T_i) && \text{by Theorem 9.5.2}\\
&\le (1+\epsilon)^2\mathsf{opt} + \epsilon^2(3+\epsilon)(4/\epsilon+2)\mathsf{opt} && \text{by Theorem 9.5.2}\\
&= \left[1 + 14\epsilon + 11\epsilon^2 + 2\epsilon^3\right]\mathsf{opt}\\
&\le (1+c'\epsilon)\mathsf{opt},
\end{aligned}$$

if we pick $c' = 27$ and assume $\epsilon \le 1$. □

With the above two lemmas, the proof of spanner construction is immediate.

Proof of Theorem 9.5.1. Procedure `SF-Spanner` gives the spanner. The spanning

property was established in Lemma 9.5.5. The length property follows from Lemma 9.5.3 and the face that $\sum_i \ell(T_i) = O(\mathsf{opt})$ from Theorem 9.5.2. □

Notice that the construction always produces a subgraph of the input, hence the bound on genus (or planarity) carries over.

We summarize the result of Lemmas 9.5.3, 9.5.4, and the construction of H_i from T_i in the following corollary. This abstraction is going to be useful in Chapter 10.

Corollary 9.5.6. *Given are a graph G of constant genus g, and a fixed constant $\lambda > 0$. A "spanner" graph H can be constructed from a subtree T of G in polynomial time such that*

1. *$\ell(H) = O(\ell(T))$, and*
2. *for any forest $F \subseteq G$, there exists a forest $F' \subseteq H$ of length at most $\ell(F)+\lambda\ell(T)$ that provides (at least) the same connectivity as F among vertices of T.*

Proof. We can assume $\ell(F) \leq \ell(T)$, otherwise we can replace F with T. The result then follows by an appropriate choice of ϵ such that $\lambda \geq \epsilon + \epsilon^2(3+\epsilon)$. □

9.5.3 The algorithm

Having proved the spanner result, we can present the PTAS for Steiner Forest on bounded-genus graphs here. `SF-PTAS` is the algorithm promised in Theorem 3.2.5.

Algorithm 9.3 `SF-PTAS`

Input: bounded-genus graph $G(V, E)$, and set of demands $\mathcal{D}$
Output: Steiner forest F satisfying $\mathcal{D}$

1: $H \leftarrow$ `SF-Spanner`$(G, \mathcal{D})$
2: $\zeta \leftarrow 2f(\epsilon)/\bar{\epsilon}$
3: $\epsilon \leftarrow \min(1, \bar{\epsilon}/6)$
4: Using Theorem 5.1.3, partition edges of H into $E_1, \ldots, E_\zeta$
5: $i^* \leftarrow \arg\min_i \ell(E_i)$
6: Find a $(1+\epsilon)$-approximate Steiner forest F^* of $\mathcal{D}$ in H/E_{i^*} via Theorem 3.2.4
7: **return** $F^* \cup E_{i^*}$

Proof of Theorem 3.2.5. Given are bounded-genus graph G, and a set of demand pairs $\mathcal{D}$. We build a Steiner forest spanner H using Theorem 9.5.1. For a suitable value of ζ whose precise value will be fixed below, we apply Theorem 5.1.3 to partition the edges of H into $E_1, E_2, \ldots, E_\zeta$. Let E_{i^*} be the set having the least total length. The total length of edges in E_{i^*} is at most $\ell(H)/\zeta$. Contracting E_{i^*} produces a graph H^* of treewidth $O(g^2\zeta)$.

Theorem 3.2.4 allows us to find a solution opt^* corresponding to H^*. Adding the edge set E_{i^*} clearly produces a solution for H whose length is at most $(1+\epsilon)\mathsf{opt}_{\mathcal{D}}(H)+\ell(H)/\zeta$. Letting $\epsilon = \min(1, \bar{\epsilon}/6)$ and $\zeta = 2f(\epsilon)/\bar{\epsilon}$ guarantees that the length of this solution is

$$
\begin{aligned}
&\leq (1+\epsilon)^2\mathsf{opt}_{\mathcal{D}}(G_{in}) + \ell(H)/\zeta && \text{by Theorem 9.5.1}\\
&\leq (1+\epsilon)^2\mathsf{opt}_{\mathcal{D}}(G_{in}) + \frac{\bar{\epsilon}}{2}\mathsf{opt}_{\mathcal{D}}(G_{in}) && \text{by Theorem 9.5.1 and choice of } \zeta\\
&(1+\bar{\epsilon})\mathsf{opt}_{\mathcal{D}}(G_{in}) && \text{by the choice of } \epsilon. \qquad \square
\end{aligned}
$$

The running time of the algorithm excluding the bounded-treewidth PTAS is bounded by $O(n^2 \log n)$. The parameter ζ above has a singly exponential dependence on ϵ. Yet, the running time of the current procedure for solving bounded-treewidth instances is not bounded by a low-degree polynomial; rather, ζ and ϵ appear in the exponent of the polynomial. Were we able to improve the running time of this procedure, we would obtain a PTAS that runs in time $O(n^2 \log n)$.

Chapter 10

Prize-collecting Network Design in Planar Graphs

This chapter is devoted to the discussion of several prize-collecting Steiner network problems on planar (and bounded-genus) graphs as well as their instances with small treewidth.

In fact, most of the material in this chapter comes from a joint work with Hajiaghayi and Marx [BHM10b]. This paper was merged with a work of Chekuri et al. [CEK10], and appeared as [BCE+11]. The only part of [BCE+11] that we do not present here is the simpler reduction for the special cases of PCST, PCTSP, and PCS—see [BCE+11, Section 3.1]—which is due to Chekuri et al. [CEK10].

Section 10.5 is based on a joint work with Hajiaghayi [BH10], however, in contrast to the discussion of that paper, we apply some of its techniques to the bounded-treewidth Multiplicative Prize-collecting Steiner Forest. We briefly highlight the main differences with the Euclidean case.

The rest of the chapter is organized as follows. Section 10.1 recalls some of the definitions, and goes over the relevant previous work. In Section 10.2 we show how the ideas of the previous chapter can be extended to give a reduction from the planar (and

bounded-genus) case of Submodular Prize-collecting Steiner Forest to its bounded-treewidth case (with a small loss in the approximation ratio). Section 10.3 then shows that the problem is APX-hard even when the treewidth is two (or on the two-dimensional Euclidean metric). This provides the first provable distinction in complexity of a network design problem and its prize-collecting variant. Nevertheless, we present in Section 10.4 PTASes for bounded-treewidth cases of several special cases of SPCSF—i.e., PCST, PCTSP, and PCS—that immediately give PTASes for their planar cases. We continue on this path by providing a PTAS for bounded-treewidth Multiplicative Prize-collecting Steiner Forest in Section 10.5.

10.1 Background

Recall that a typical network design problem is modeled as the problem of finding a minimum-length subgraph of a given graph G that satisfies certain "requests." The requests often correspond to connectivity requirements between some given pairs or sets of vertices.

In their prize-collecting variants, penalties (or prizes) are associated with requests that are not satisfied by the subgraph. These problems are interesting for several reasons. In particular, prize-collecting Steiner network design problems are well-known network design problems with several applications in expanding telecommunications networks (see, e.g., [JMP00, SCRS00]), cost sharing, and Lagrangian relaxation techniques (see, e.g., [JV01, CRW04]).

The prize-collecting problems generalize the underlying network design problems since one can set the penalties to ∞ which forces the solution to satisfy all requests. In particular, Prize-collecting Steiner Forest generalizes the well-studied Steiner Forest problem which is NP-hard and also APX-hard. The best known approximation ratio for Steiner Forest is $2 - \frac{2}{n}$ (n is the number of ver-

tices of the graph) due to Agrawal, Klein, and Ravi [AKR95] (see also [GW95] for a more general result and a simpler analysis). The special case of Prize-collecting Steiner Forest problem in which all sinks are identical is the (rooted) Prize-collecting Steiner Tree problem. In the nonrooted version of this problem, there is no specific sink (root); here, the goal is to find a tree connecting some sources and pay the penalty for the rest of them. We also study two variants of (nonrooted) Prize-collecting Steiner Tree, Prize-collecting TSP (PCTSP) and Prize-collecting Stroll (PCS), in which the set of edges should form a cycle and a path, respectively, instead of a tree. When in addition all penalties are ∞ in these prize-collecting problems, we have the classic APX-hard problems Steiner Tree, Traveling Salesman and Path-TSP (or Stroll) for which the best approximation factors in order are 1.39 [BGRS10], $\frac{3}{2}$ [Chr76], and $\frac{3}{2}$ [Hoo91].

The key difference between problems such as Prize-collecting Steiner Tree, Prize-collecting Steiner Forest and their special cases Steiner Tree and Steiner Forest is that we do not know a priori the set of demands that are to be satisfied/connected; satisfying more demands reduces the penalty, but increases the connection cost. This connection cost plus penalty nature of the objective function models realistic problems with multiple goals; for example, in network construction, one may wish to examine the tradeoff between the cost of serving clients and the potential profit from serving them. The impact of Prize-collecting Steiner Tree and Prize-collecting TSP within approximation algorithms is also far-reaching, especially in the study of other problems where the set of demands to be satisfied is not fixed: In the k-MST and k-Stroll problems [Gar96, AR98, CRW04, AK06, Gar05, CGRT03], the goal is to find a minimum-length tree or path containing at least k vertices, and in the Max-Prize-Tree and Orienteering problems [BCK+07, BBCM04, CKPar, NR07], the goal is to find a tree or path that contains as many vertices as possible, subject to a length constraint. In par-

ticular, Prize-collecting Steiner Tree is a Lagrangian relaxation of k-MST, and hence has played a crucial role in the design of algorithms for all the problems mentioned above. Thus, we are motivated to study prize-collecting problems both for their inherent theoretical and practical value, and because they are useful in the study of several other problems of interest.

Traveling Salesman, Steiner Tree, and Steiner Forest all have been studied extensively on planar graphs. Indeed, all these problems remain NP-hard even in this setting [GJ79]. However, obtaining a PTAS for each of these problems remained an open problem for several years. Grigni, Koutsoupias, and Papadimitriou [GKP95] obtained the first PTAS for Traveling Salesman on unweighted planar graphs in 1995; this was later generalized to weighted planar graphs [AGK+98] (and improved to linear time [Kle08]). Obtaining a PTAS for Steiner Tree on planar graphs remained elusive for almost 12 years until 2007 when Borradaile, Klein and Mathieu [BKM09] obtained the first PTAS for Steiner Tree on planar graphs using a new technique of contraction decomposition and building spanners (this borrowed ideas from earlier work of Klein on Subset TSP [Kle06]). Borradaile et al. [BKM09] posed obtaining a PTAS for Steiner Forest in planar graphs as the main open problem. Bateni, Hajiaghayi and Marx [BHM10a] very recently solved this open problem using a primal-dual technique for building spanners and obtaining PTASes by reducing the problem to bounded-treewidth graphs. Interestingly, Steiner Forest turns out to be NP-hard even on graphs of treewidth 3 and hence [BHM10a] had to devise a PTAS for the case of bounded-treewidth graphs in order to apply the general framework.

Obtaining PTASes for prize-collecting versions of the above network design problems was suggested as an open problem in [BHM10a, BH10]. The main technical difficulty in prize-collecting problems is that it is not a priori clear which requests are to be satisfied. In this paper, we resolve this difficulty for PCST, PCTSP,

Table 10.1: Complexity of PCSF and its special cases for different classes of graphs. Notice that, in each row of the table, positive results are given for the most general case applicable, and hardness results are mentioned for the most special case applicable. The problem in each row generalizes those in prior rows. The algorithms for Euclidean PCST, PCTSP, and PCS follow from the same ideas in Arora [Aro98], although the result is not mentioned there. Similarly, the algorithms for SPCST and SPCSF on trees can be reduced to Submodular Vertex Cover using the ideas in Hajiaghayi and Jain [HJ06], for which Goel et al. [GKTW09] give a 2-approximation algorithm.

	Small treewidth	**Euclidean**	**Small genus**	**General**
PCST, PCTSP, PCS	P [BHM10b]	PTAS [Aro98]	PTAS [BHM10b]	aprx$(2-c)$ [ABHK11, Goe09]
Multiplicative PCSF	PTAS §10.5	PTAS [BH10]	PTAS §10.5	aprx(2.54) [HJ06]
PCSF	P for trees [HJ06]; APX-hard [BHM10b] for treewidth(2)	APX-hard for treewidth(2) [BHM10b]		aprx(2.54) [HJ06]
Submodular PCST	aprx(2) for trees [GKTW09]; APX-hard for treewidth(2) [BHM10b]	APX-hard for treewidth(2) [BHM10b]		aprx(3) [HKKN10]
Submodular PCSF	aprx(2) for trees [GKTW09]; APX-hard for treewidth(2) [BHM10b]	APX-hard for treewidth(2) [BHM10b]		aprx(3) [HKKN10]

PCSF, and, even more generally, for SPCSF, by reducing these problems on planar graphs to the corresponding problems on graphs of bounded treewidth. More precisely we show that any α-approximation algorithm for these problems on graphs of bounded treewidth gives an $(\alpha + \epsilon)$-approximation algorithm for these problems on planar graphs and bounded-genus graphs, for any constant $\epsilon > 0$. Since PCST and PCTSP can be solved exactly on graphs of bounded treewidth using standard dynamic-programming techniques (as we discuss later in the paper), we immediately obtain PTASes for PCST and PCTSP on planar graphs (the same holds for PCS as well). In contrast, we show that PCSF is APX-hard already on series-parallel graphs, which are planar graphs of treewidth at most 2, ruling out a PTAS for planar Prize-collecting Steiner Forest (assuming $\mathsf{P} \neq \mathsf{NP}$). Apart from ruling out a PTAS for PCSF on planar graphs and bounded-treewidth graphs, this result is also interesting since it gives the first provable hardness separation between the approximability of a problem and its prize-collecting version: Steiner Forest and Prize-collecting Steiner Forest when restricted to planar graphs. We also show that Prize-collecting Steiner Forest is APX-hard on Euclidean instances, that is, when the input graph is induced by points in the Euclidean plane and the lengths are Euclidean distances.

10.2 Reduction to the bounded-treewidth case

Recall that an instance of Submodular Prize-collecting Steiner Forest (SPCSF) is described by a triple $(G, \mathcal{D}, \pi)$ where G is an undirected weighted graph (with edge length function ℓ), $\mathcal{D}$ is a set of $d_i = \{s_i, t_i\}$ demand pairs, and $\pi : 2^{\mathcal{D}} \mapsto \mathbb{R}^+$ is a monotone nonnegative submodular penalty function. A demand $d = \{s, t\}$ is *satisfied* by a subgraph F if and only if s, t are connected in F. If a forest F satisfies a subset $\mathcal{D}^{\mathsf{sat}}$ of the demands, its cost is defined as $\mathsf{cost}(F) := \ell(F) + \pi(\mathcal{D}^{\mathsf{unsat}})$, where

$\mathcal{D}^{\mathsf{unsat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{sat}}$ denotes the subset of unsatisfied demands.

Further recall that SPCTSP, SPCS and SPCST are submodular prize-collecting variants of Traveling Salesman, Stroll, and Steiner Tree, respectively. An instance of these problems is represented by $(G, \mathcal{D}, \pi)$ where all the demands $d = \{s, t\} \in \mathcal{D}$ share a common root vertex $r \in V(G)$.[1] A feasible solution F is a TSP tour, stroll, or Steiner tree, respectively, for a subset of demands, say $\mathcal{D}^{\mathsf{sat}} \subseteq \mathcal{D}$. The cost is then $\mathsf{cost}(F) := \ell(F) + \pi(\mathcal{D}^{\mathsf{unsat}})$, where $\mathcal{D}^{\mathsf{unsat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{sat}}$.

10.2.1 Overview of the reduction

We now outline the proof of Theorem 3.2.7 that presents a reduction from any bounded-genus instance of SPCSF to a bounded-treewidth instance. The reduction consists of three steps, and follows the spanner framework of Chapters 5, 6.

1. Given an instance of SPCSF, let opt denote the cost of an optimal solution. We construct a collection of trees $\{\hat{T}_1, \ldots, \hat{T}_k\}$ with two crucial properties:

   (a) The total length of the trees is bounded; $\sum_i \ell(\hat{T}_i) \leq f(\epsilon)\mathsf{opt}$, for some function f depending only on ϵ.

   (b) Paying the penalty for all demand pairs not contained in the same tree does not significantly increase the cost of an optimal solution. More formally, let $\hat{\mathcal{D}}$ denote the set of demand pairs which are not both contained in the same tree. There is a solution F such that, if $\overline{\mathcal{D}}$ is the set of demands not satisfied by F, $\ell(F) + \pi(\overline{\mathcal{D}} \cup \hat{\mathcal{D}}) \leq (1 + O(\epsilon))\mathsf{opt}$.

2. The collection of the trees can then be used as the backbone graphs to construct a spanner H. In particular, H is a subgraph of G, whose length is at most

[1] Both the rooted and nonrooted variants of these problems may be more naturally defined with single-vertex demands rather than demand pairs; having such a formulation, we can guess one vertex of the solution, designate it as the root and obtain the rooted formulation as defined here.

$f'(\epsilon, g)\mathsf{opt}$ where g is the genus of G, and contains a solution of cost $[1+O(\epsilon)]\mathsf{opt}$. The construction of this step is very similar to that in Section 9.5.

3. After constructing the spanner, we invoke Theorem 5.1.3 to obtain a bounded-treewidth graph, whose solution leads to a solution for the input instance $(G, \mathcal{D}, \pi)$. We finally use the α-approximation algorithm to solve the instance of SPCSF on this bounded-treewidth graph.

The second and third steps of the reduction are very similar to those in Chapter 9, thus we focus our attention on the first step. Recall that the additional difficulty in solving PCST, PCSF, PCSF, and related problems comes from not knowing which demands to connect. The first step implies that we can effectively focus our attention *only* on the demand pairs that have both vertices in the same tree component of some forest (i.e., the backbone graph). The core of the reduction, then, is obtaining the desired collection of trees, where our algorithm employs the prize-collecting clustering paradigm. For this application, the clustering technique is generalized as follows. First, we need to extend the ideas to work for prize-collecting variants of Steiner network problems. (This can indeed make the problem provably harder; see Theorem 3.2.6.) The standard prize-collecting clustering technique—e.g., `PC-Classify`—associates a potential value to each node and grows the corresponding clusters consuming these potentials. However, in order to extend it to the prize-collecting setting, we consider source-sink potentials. This means that there is some interaction between the potentials of different nodes. Secondly, we consider submodular penalty functions that model even more interaction between the demands. The extended prize-collecting clustering procedure has two phases. In the first phase, we have a source-sink moat-growing algorithm (`SubmodPC-Cluster`), and in the second phase, we have a single-node potential moat-growing (`PC-Classify`).

Before presenting the formal proof, we give an informal overview. The algorithm starts with a constant-approximate solution F^1, say, obtained using Hajiaghayi et

al. [HKKN10] who present a 3-approximation for SPCSF on general graphs. The forest F^1 satisfies a subset of demands, and we know the total penalty of unsatisfied demands is bounded.[2] The algorithm then tries to satisfy more demands by constructing a forest $F^2 \supseteq F^1$ whose length is bounded; see `RestrictDemands` in Section 10.2.2. This step heavily uses the `SubmodPC-Cluster` algorithm introduced in Section 4.4. At the end of this step, we can assume that the near-optimal solution does not satisfy the demands which are unsatisfied in F^2. Submodularity poses several difficulties in proving this property: ideally, we want to say that the cost paid by the optimal solution to satisfy these demands is significantly more than their penalty value. Surprisingly, this is not true. Nevertheless, we can prove that the *marginal penalty* of the demands satisfied in the near-optimal solution but not in F^2 can be charged to the cost the near-optimal solution pays in order to satisfy them. The next step of the reduction is to build a forest $F^3 \supseteq F^2$ of bounded length that may connect several components of F^2 together; see Section 10.2.3. This is done via `PC-Classify` (i.e., by assigning to each component of F^2 a potential proportional to its length, and then running a standard prize-collecting clustering). This guarantees that the near-optimal solution does not need to connect different components of F^3 to each other.

Another difference with the discussion of the previous chapter for planar Steiner Forest is that, unlike those reductions, we cannot solve the problem independently on each part of the spanner constructed from different backbones. The interaction between the demands due to the submodular penalty function requires us to solve the reduced instance in one piece.

[2] Looking at the internal working of the algorithm of Hajiaghayi et al. [HKKN10] and its analysis, we find out that the penalty portion of the cost is bounded by `opt` and the forest length portion is bounded by `2opt`. However, we look at the algorithm as a black box, providing a 3-approximation, and do not try to optimize the constants in our reduction. Therefore, we use the fact that the penalty portion of the cost is bounded by `3opt`, and so is the forest length portion.

It is very important for the reduction that the reduced bounded-treewidth instance have the same set $\mathcal{D}$ of demands and the same penalty function π. This enables us to use the reduction not only for SPCSF, but also for SPCST, PCSF, PCST, etc. In particular, the structure of the penalty function does not change through the reduction; for instance, the final instance is PCST instance if the original is, and similarly, it is multiplicative if the original instance is. Therefore, using the reduction theorem on a Multiplicative Prize-collecting Steiner Forest instance, for example, does not leave us with solving the bounded-treewidth instance of the more general Submodular Prize-collecting Steiner Forest.

We now start the formal proof of Theorem 3.2.7. Recall that in the beginning of this section, we outlined three steps of the reduction. The first step of the reduction is itself performed in two phases.

1. We start with an instance $(G, \mathcal{D}, \pi)$ of SPCSF. We first take out a subset, say $\mathcal{D}^{\mathsf{unsat}}$, of demands whose cost of satisfying, roughly speaking, is too much compared to their penalties. Thus, we can focus on the remaining demands, say $\mathcal{D}^{\mathsf{sat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{unsat}}$.

2. Afterwards, we partition the remaining demands $\mathcal{D}^{\mathsf{sat}}$ into $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_p$ such that, roughly speaking, the forest in the solution can be found independently for each demand set without a substantial increase in the length.

The first phase is carried out in the following theorem. The proof appears in Section 10.2.2, and uses `SubmodPC-Cluster` of Section 4.4. This phase allows us to focus on only a subset $\mathcal{D}^{\mathsf{sat}}$ of demands, and ignore the rest of the demands. The additional cost due to this is only $\epsilon\,\mathsf{opt}$.

Theorem 10.2.1. *Given an instance* $(G, \mathcal{D}, \pi)$ *of* SPCSF *(or* SPCTSP *or* SPCS*) and a parameter* $\epsilon > 0$*, we can construct in polynomial time a subgraph* F *of* G*,*

satisfying only a subset $\mathcal{D}^{\mathsf{sat}} \subseteq \mathcal{D}$ *of demands, in effect leaving* $\mathcal{D}^{\mathsf{unsat}} := \mathcal{D} \setminus \mathcal{D}^{\mathsf{sat}}$ *unsatisfied, such that*

1. $\ell(F) \leq (6\epsilon^{-1} + 3)\mathsf{opt}$, *and*
2. *the optimum of* $(G, \mathcal{D}^{\mathsf{sat}}, \pi')$ *is at most* $(1+\epsilon)\mathsf{opt}$ *where* $\pi'(D) := \pi(D \cup \mathcal{D}^{\mathsf{unsat}})$ *is defined for* $D \subseteq \mathcal{D}^{\mathsf{sat}}$.

At this point, we have a constant-approximate solution satisfying all the (remaining) demands. The second phase (which is required only for "forest" problems, and is irrelevant for PCTSP, e.g.) is a generalization and extension of `PC-Partition`—see Algorithm 9.1. We are trying to break the instance into smaller pieces. The solution to each piece is almost independent of the others, i.e., there is little interaction between them. The following theorem is proved in Section 10.2.3.

Theorem 10.2.2. *Given an instance* $(G, \mathcal{D}, \pi)$ *of* SPCSF, *a forest* F *satisfying all the demands, and a parameter* $\epsilon > 0$, *we can compute in polynomial time a set of trees* $\{\hat{T}_1, \ldots, \hat{T}_k\}$, *and a partition of demands* $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$, *with the following properties.*

1. *All the demands are covered, i.e.,* $\mathcal{D} = \bigcup_{i=1}^{k} \mathcal{D}_i$.
2. *The tree* $\hat{T}_i$ *spans all the terminals in* $\mathcal{D}_i$.
3. *The total length of the trees* $\hat{T}_i$ *is within a constant factor of the length of* F, *i.e.,* $\sum_{i=1}^{k} \ell(\hat{T}_i) \leq (\frac{4}{\epsilon} + 2)\ell(F)$.
4. *Let* $\mathcal{D}^*$ *be the subset of demands satisfied by* opt, *the optimal solution to* $(G, \mathcal{D}, \pi)$. *Then, we have* $\sum_i \mathsf{SteinerForest}(G, \mathcal{D}_i^*) \leq (1+\epsilon)\mathsf{SteinerForest}(G, \mathcal{D}^*)$, *where* $\mathcal{D}_i^* := \mathcal{D}^* \cap \mathcal{D}_i$, *and* $\mathsf{SteinerForest}(G, \mathcal{D})$ *denotes the length of a minimum Steiner forest in* G *satisfying the demands* $\mathcal{D}$.

The rest of the steps are similar to those for planar Steiner Forest reduction of Section 9.5. Before proving the above theorems, we show they are sufficient for carrying out the reduction.

Proof of Theorem 3.2.7. Start with an instance $(G, \mathcal{D}, \pi)$ of SPCSF or SPCTSP (the case of SPCS is very similar). Without loss of generality we present an approximation guarantee of $\alpha + O(1)\epsilon$. Find F, $\mathcal{D}^{\mathsf{sat}}$ and $\mathcal{D}^{\mathsf{unsat}}$ from applying Theorem 10.2.1 on $(G, \mathcal{D}, \pi)$. We know that F satisfies $\mathcal{D}^{\mathsf{sat}}$ and $\ell(F) = O(\mathsf{opt})$. Moreover, monotonicity of the penalty function ensures that the optimum of the corresponding prize-collecting problem with smaller demand set $\mathcal{D}^{\mathsf{sat}}$ costs at most opt.

For the moment, suppose the original instance is one of SPCTSP, and let opt' be the optimal solution for $(G, \mathcal{D}^{\mathsf{sat}}, \pi')$, with π' as defined in Theorem 10.2.1. (Note that $\mathsf{opt}' \leq (1+\epsilon)\mathsf{opt}$.) In addition, let T^0 denote the tree component of F containing the root vertex (i.e., the common sink of all demands). Since all the relevant terminals (i.e., endpoints of demands $\mathcal{D}^{\mathsf{sat}}$) appear in T^0 (which has length $O(\mathsf{opt}')$), the subset TSP spanner result of Klein [Kle06] allows us to construct from T^0 a graph H, of total length $O(\mathsf{opt}') = O(\mathsf{opt})$, that contains a near-optimal cycle satisfying the same demands as opt'; i.e., the length of this cycle is at most $1+\epsilon$ times that of opt'. Therefore, the instance $(H, \mathcal{D}^{\mathsf{sat}}, \pi')$, and as a result $(H, \mathcal{D}, \pi)$, has a solution of cost no more than $(1+\epsilon)^2\mathsf{opt}$. Applying Theorem 5.1.3 on H with sufficiently large (though still a constant) value for ζ, we can find a set E_d of edges of H such that H/E_d has a constant treewidth, and $\ell(E_d) \leq \epsilon\,\mathsf{opt}$. Clearly, removing the contracted edges from any cycle gives a cycle in H/E_d. It is well-known that any cycle in H/E_d can be augmented using at most two copies of each edge in E_d to obtain a cycle in H spanning a superset of the original vertices [Kle06, Kle08, AGK+98]. Therefore, the α-approximate solution for $(H/E_d, \mathcal{D}, \pi)$ gives an $(\alpha + O(\epsilon))$-approximate solution for the original instance $(G, \mathcal{D}, \pi)$.[3]

The rest of the proof applies to the case where the original instance is one of

[3] We are abusing the notation here, in that for the instance $(H/E_d, \mathcal{D}, \pi)$, the set of demands, and as a result π, need to modified to reflect the renaming of vertices. In particular, for any terminal adjacent to a contracted edge, we can add a dummy edge that is attached to the result of the contraction with the other endpoint being the replacement of the terminal. This way, $\mathcal{D}$ can be the same as the original, but the vertex set of H may have vertices outside $V(G)$, which are the result of edge contractions.

SPCSF. As in Theorem 10.2.1, define $\pi'(D) := \pi(D \cup \mathcal{D}^{\mathsf{unsat}})$ for all $D \subseteq \mathcal{D}$. Let opt' denote the optimal solution of $(G, \mathcal{D}^{\mathsf{sat}}, \pi')$ and its cost with some abuse of notation. We also use $\ell(\mathsf{opt}')$ and $\pi(\mathsf{opt}')$ to refer to the length of the forest in opt' and the penalty it pays, respectively. Clearly, $\mathsf{opt}' = \ell(\mathsf{opt}') + \pi(\mathsf{opt}') \leq (1+\epsilon)\mathsf{opt}$. Pick $\epsilon' < \epsilon \cdot \ell(F)/\mathsf{opt}$ and feed $(G, \mathcal{D}^{\mathsf{sat}}, \pi')$ along with F and ϵ' into Theorem 10.2.2 in order to obtain $\mathcal{D}_i$'s and $\hat{T}_i$'s for $i = 1, \ldots, k$. We have $\sum_i \ell(\hat{T}_i) = O(\ell(F)) = O(\mathsf{opt})$ since ϵ' is a constant. Define $\mathcal{D}_i^*$ as in the statement of Theorem 10.2.2, from which we have forests F_i (for $1 \leq i \leq k$) satisfying $\mathcal{D}_i^*$ such that $\sum_i \ell(F_i) \leq (1+\epsilon)\ell(\mathsf{opt}')$. Using Corollary 9.5.6 we can find subgraphs H_i from each backbone $\hat{T}_i$ (and a sufficiently small parameter $\lambda > 0$) such that there exists a forest $F_i^* \subseteq H_i$ satisfying $\mathcal{D}_i^*$ with $\ell(F_i^*) \leq \ell(F_i) + \lambda\ell(\hat{T}_i)$. Let $H = \bigcup_i H_i$ be the spanner graph. We have a forest $\bigcup_i F_i^* \subseteq H$ satisfying $\mathcal{D}^*$ with length at most $(1+2\epsilon)\ell(\mathsf{opt}')$ provided that $\lambda \leq \frac{1}{\mathsf{opt}'}\sum_i \ell(\hat{T}_i)$. Therefore, the optimum of $(H, \mathcal{D}, \pi)$ costs no more than $(1+2\epsilon)\ell(\mathsf{opt}') + \pi(\mathsf{opt}') \leq (1+2\epsilon)\mathsf{opt}' \leq (1+2\epsilon)(1+\epsilon)\mathsf{opt}$.

We next invoke Theorem 5.1.3 on H with a sufficiently large ζ to obtain $E_d \subseteq H$ such that $\ell(E_d) \leq \epsilon\,\mathsf{opt}$, and H/E_d has a constant treewidth. Let us find an α-approximate solution to the instance $(H/E_d, \mathcal{D}, \pi)$.[4] Clearly, this solution costs no more than $\alpha(1+\epsilon)^3\mathsf{opt}$, and can be extended to a solution for $(H, \mathcal{D}, \pi)$ by adding the edges E_d. The final solution is an $(\alpha + O(\epsilon))$-approximate solution of $(G, \mathcal{D}, \pi)$. □

10.2.2 Restricting demands

We prove Theorem 10.2.1 in this section. First, we obtain a constant-factor approximate solution F^+ (via the 3-approximation algorithm for general graphs [HKKN10]). Let $\mathcal{D}^+$ denote the demands satisfied by F^+. We denote by T_j^+ the connected components of F^+. For each demand $d = \{s, t\} \in \mathcal{D}^+$ we clearly have $\{s, t\} \subseteq V(T_j^+)$ for some j. However, for an unsatisfied demand $d' = \{s', t'\} \in \mathcal{D} \setminus \mathcal{D}^+$, the vertices s'

[4] The technicalities of the previous footnote apply here, too.

and t' belong to two different components of F^+. Construct G^* from G by reducing the length of edges of F^+ to zero. The new penalty function π^* is defined as follows:

$$\pi^*(D) := \epsilon^{-1}\pi(D) \qquad \text{for } D \subseteq \mathcal{D}. \tag{10.1}$$

Finally we run `SubmodPC-Cluster` on $(G^*, \mathcal{D}, \pi^*)$; see Algorithm 10.1.

Algorithm 10.1 `RestrictDemands`$(G, \mathcal{D}, \pi)$

Input: instance $(G, \mathcal{D}, \pi)$ of Submodular Prize-collecting Steiner Forest
Output: forest F and $\mathcal{D}^{\mathsf{unsat}}$

1: Use the algorithm of Hajiaghayi et al. [HKKN10] to find a 3-approximate solution: a forest F^+ satisfying subset $\mathcal{D}^+$ of demands.
2: Construct $G^*(V, E^*)$ in which E^* is the same as E except that the edges of F^+ have length zero in E^*.
3: Define π^* as Equation (10.1)
4: $(F, \mathcal{D}^{\mathsf{unsat}}, y) \leftarrow$ `SubmodPC-Cluster`$(G^*, \mathcal{D}, \pi^*)$
5: **return** $(F, \mathcal{D}^{\mathsf{unsat}})$

Now we show that the algorithm `RestrictDemands` outlined above satisfies the requirements of Theorem 10.2.1. Before doing so, we show how the length of a forest can be compared to the output vector y.

Lemma 10.2.3. *If a graph F satisfies a set $\mathcal{D}^{\mathsf{sat}}$ of demands, then $\ell(F) \geq y(\mathcal{D}^{\mathsf{sat}})$.*

This is quite intuitive. Recall that the y variables paint the edges of the graph. Consider a segment on edges corresponding to cluster S with color d. At least one edge of F *passes through* the cut $(S, \overline{S})$. Thus, a portion of the cost of F can be charged to $y_{S,d}$. Hence, the total cost of the graph F is at least as large as the total amount of colors paid for by $\mathcal{D}^{\mathsf{sat}}$. We now provide a formal proof.

Proof. The length of the graph F is

$$\sum_{e \in F} c_e \geq \sum_{e \in F} \sum_{S : e \in \delta(S)} y_S \qquad \text{by (4.14)}$$

$$
\begin{aligned}
&= \sum_S |F \cap \delta(S)| y_S \\
&\geq \sum_{S: F \cap \delta(S) \neq \emptyset} y_S \\
&= \sum_{S: F \cap \delta(S) \neq \emptyset} \sum_{d: d \odot S} y_{S,d} \\
&= \sum_d \sum_{\substack{S: d \odot S \\ F \cap \delta(S) \neq \emptyset}} y_{S,d} \\
&\geq \sum_{d \in \mathcal{D}^{\mathsf{sat}}} \sum_{\substack{S: d \odot S \\ F \cap \delta(S) \neq \emptyset}} y_{S,d} \\
&= \sum_{d \in \mathcal{D}^{\mathsf{sat}}} \sum_{S: d \odot S} y_{S,d},
\end{aligned}
$$

because $y_{S,d} = 0$ if $d \in \mathcal{D}^{\mathsf{sat}}$ and $F \cap \delta(S) = \emptyset$,

$$
= \sum_{d \in \mathcal{D}^{\mathsf{sat}}} y_d. \qquad \square
$$

Proof of Theorem 10.2.1. We know that $\ell(F^+) + \pi(\mathcal{D} \setminus \mathcal{D}^+) \leq 3\mathsf{opt}$ because we start with a 3-approximate solution. For any demand $d = (s, t)$, we know that y_d is not more than the distance of s, t in G^*. Since the distance between endpoints of d is zero if it is satisfied in $\mathcal{D}^+$, y_d is nonzero only if $d \in \mathcal{D} \setminus \mathcal{D}^+$. Thus, we have $y(\mathcal{D}) = y(\mathcal{D} \setminus \mathcal{D}^+) \leq \pi^*(\mathcal{D} \setminus \mathcal{D}^+)$ by Constraint (4.15). Theorem 4.4.1 gives $\ell(F)$ in G^*, denoted by $\ell_{G^*}(F)$, is at most $2y(\mathcal{D}) \leq 2\pi^*(\mathcal{D} \setminus \mathcal{D}^+) = 2\epsilon^{-1}\pi(\mathcal{D} \setminus \mathcal{D}^+) \leq 6\epsilon^{-1}\mathsf{opt}$. Therefore $\ell(F) = \ell(F^+) + \ell_{G^*}(F) \leq (6\epsilon^{-1} + 3)\mathsf{opt}$.

To establish the second condition of the theorem, take an optimal forest F' for $(G, \mathcal{D}, \pi)$: F' satisfies demands $\mathcal{D}^{\mathsf{opt}}$, and we have $\ell(F') + \pi(\mathcal{D} \setminus \mathcal{D}^{\mathsf{opt}}) = \mathsf{opt}$. Define $A := \mathcal{D}^{\mathsf{opt}} \setminus \mathcal{D}^{\mathsf{sat}}$ and $B := \mathcal{D}^{\mathsf{unsat}} \setminus A$. The penalty of F' under π' is $\pi((\mathcal{D} \setminus \mathcal{D}^{\mathsf{opt}}) \cup \mathcal{D}^{\mathsf{unsat}}) = \pi((\mathcal{D}^{\mathsf{sat}} \setminus \mathcal{D}^{\mathsf{opt}}) \cup A \cup B)$. Hence, the increase in penalty of F' due to changing from π to π' is $\pi((\mathcal{D}^{\mathsf{sat}} \setminus \mathcal{D}^{\mathsf{opt}}) \cup A \cup B) - \pi((\mathcal{D}^{\mathsf{sat}} \setminus \mathcal{D}^{\mathsf{opt}}) \cup B) \leq \pi(A \cup$

$B) - \pi(B)$ due to the diminishing returns property of submodular functions. We have $y(A \cup B) = \pi^*(A \cup B) = \epsilon^{-1}\pi(A \cup B)$ because $A \cup B = \mathcal{D}^{\mathsf{unsat}}$ is the set of dead demands of `SubmodPC-Cluster`; see the first condition of Theorem 4.4.1. We also have $\epsilon^{-1}\pi(B) = \pi^*(B) \geq y(B)$ because of Constraint (4.15). Therefore, the additional penalty is at most $\epsilon[y(A \cup B) - y(B)] = \epsilon y(A)$. Since F' satisfies the demands A, we have $y(A) \leq \ell(F') \leq \mathsf{opt}$ from Lemma 10.2.3. Therefore, the additional penalty is at most $\epsilon\mathsf{opt}$.

The extension to SPCTSP and SPCS is straight-forward once we observe that the cost of building a tour or a stroll[5] on a subset of vertices is at least the cost of constructing a Steiner tree on the same set. The algorithm in this case pretends it has an SPCST instance, and restricts the demand set accordingly. However, the extra penalty due to the ignored demands $\mathcal{D}^{\mathsf{unsat}}$ is charged to the Steiner tree cost which is no more than the TSP or stroll length. □

10.2.3 Restricting connectivity

In this section we prove that `PC-Partition`$(G, \mathcal{D})$ accomplishes the task outlined in Theorem 10.2.2.

Proof of Theorem 10.2.2. The first two conditions are the same as those in Theorem 9.5.2. The third condition follows from the fact that $\mathsf{opt}_{\mathcal{D}}(G) \leq \ell(F)$, where $\mathsf{opt}_{\mathcal{D}}(G)$ denotes the length of the optimal Steiner forest of demand set $\mathcal{D}$ in G. The last condition is a consequence of the stronger form of the fourth condition of Theorem 9.5.2. □

[5] A *stroll* is similar to a tour, except that it may start and end on different vertices.

10.3 APX-hardness for PCSF

10.3.1 Hardness for planar graphs of treewidth two

We first present the hardness proof for Prize-collecting Steiner Forest on a planar graph of treewidth two. The proof shows hardness for a very restricted class of graphs: short cycles passing through a single central vertex.

Proof of Theorem 3.2.6(1). We reduce an instance $\mathcal{I}$ of Minimum Vertex Cover on 3-regular graphs to an instance $\mathcal{I}'$ of Prize-collecting Steiner Forest on a planar graphs of treewidth two. The former is known to be APX-hard [AK00]. The instance $\mathcal{I}$ is defined by an undirected graph G. If n denotes the number of vertices of G, the number edges is $m = 3n/2$. We will denote the i^{th} vertex of G by v_i, the j^{th} edge by e_j, and the first and second endpoints of e_j by $e_j^{(1)}$ and $e_j^{(2)}$, respectively.

We now specify the reduction (illustrated in Figure 10.1); $\mathcal{I}'$ is represented by $(H, \mathcal{D}, \pi)$. The graph H consists of the vertices

- a_i for $1 \leq i \leq n$,
- b_j, c_j^1, c_j^2 for $1 \leq j \leq m$, and
- central vertex w,

with the edges

- $\{w, a_i\}$ of length 2 $(1 \leq i \leq n)$, and
- $\{w, c_j^1\}, \{w, c_j^2\}, \{c_j^1, b_j\}, \{c_j^2, b_j\}$ of length 1 $(1 \leq j \leq m)$.

The instance contains the following demands:

- $\{w, b_j\}$ with penalty 3 $(1 \leq j \leq m)$, and
- $\{a_i, c_j^p\}$ with penalty 1 if $v_i = e_j^{(p)}$ for some $1 \leq i \leq n$, $1 \leq j \leq m$, and $p \in \{1, 2\}$.

Thus the number of demands is exactly $m + 3n$, and each a_i appears in exactly 3 demands. We claim that the cost of the optimum of $\mathcal{I}'$ is exactly $2m + 2n + \tau(G)$, where $\tau(G)$ is the size of the minimum vertex cover in G. Note that, since $\tau(G) \geq m/3$ (as G is 3-regular), $2m + 2n + \tau(G)$ is at most a constant times $\tau(G)$. In order to prove the correctness of the reduction, we prove the following two statements.

(1) Given a vertex cover of size k for G, a solution of cost $2m + 2n + k$ can be constructed for $\mathcal{I}'$.

(2) Given a solution of cost at most $2m+2n+k$ for $\mathcal{I}'$, a vertex cover of size at most k can be constructed for G.

To prove (1), suppose that C is a vertex cover of size k for G. Let T be a tree of H that contains

- edge $\{w, a_i\}$ if and only if $v_i \notin C$,
- edges $\{w, c_j^1\}, \{c_j^1, b_j\}$ if and only if $e_j^1 \notin C$, and
- edges $\{w, c_j^2\}, \{c_j^2, b_j\}$ if and only if $e_j^1 \in C$.

The length of T is $2(n - k) + 2m$. Observe that all the demands $\{w, b_j\}$ are connected (via either c_j^1 or c_j^2). Furthermore, if $v_i \notin C$, then all three demands where a_i appears are satisfied: edge $\{w, a_i\}$ is in T and if $v_i = e_j^1$, then edge $\{w, c_j^1\}$ is in T as well. (Note that if $v_i = e_j^2$ and $v_i \notin C$, then $e_j^1 \in C$ must hold, and therefore $\{w, c_j^2\}$ is in T.) Thus, the total penalty paid by the solution T is at most $3k$, and hence the cost of the solution is at most $2n + 2m + k$, as claimed.

To prove (2), suppose that subgraph F of G is a solution to $\mathcal{I}'$ such that the sum of the length of F and the penalties of its unserved demands is at most $2m + 2n + k$. We can assume that, for every $1 \leq i \leq n$, vertex b_j can be reached from w via F: otherwise, we can decrease the penalty by 3 at the cost of adding two edges of length 1. Furthermore, we can assume that only one of c_j^1 and c_j^2 can be reached from w

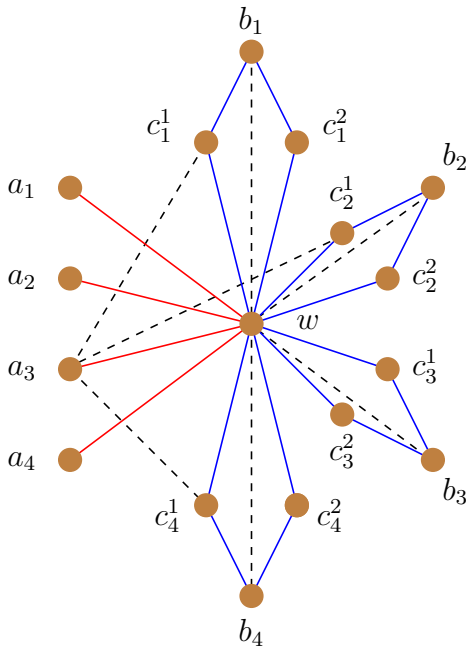


Figure 10.1: Illustrating the reduction from 3-regular Minimum Vertex Cover to Prize-collecting Steiner Forest. The solid edges represent the edges of the graph H—representing the PCSF instance—and the dashed edges depict the demands. See the description in the text for penalties and edge lengths. The graph G of the Minimum Vertex Cover instance consists of four vertices and four edges, where the edges e_1, e_2, e_4 are incident on v_3.

via F: otherwise, we can remove an edge without disconnecting b_j from w, thus the length decreases by 1 and the penalty increases by at most 1. Finally, we can assume that, if $\{w, a_i\} \in F$, then all 3 demands containing a_i are connected: otherwise, removing $\{w, a_i\}$ decreases the length by 2 and increases the penalty by at most 2.

Define C as follows. Let vertex v_i be in C if and only if $\{w, a_i\} \notin F$. We claim that C is a vertex cover of size at most k. To see that C is a vertex cover, consider an edge e_j. We have observed above that one of c_j^1 and c_j^2 cannot be reached from w. Without loss of generality, assume that c_j^1 cannot be reached from w and $e_j^{(1)} = v_i$. Then the demand $\{v_i, c_j^1\}$ is not connected by F. Therefore, not all 3 demands containing a_i are connected, which means (as observed above) that $\{w, a_i\} \notin F$. Thus $v_i \in C$ covers the edge e_j.

We next compute the size of C. Since every b_j can be reached from w and $\{w, a_i\} \in F$ if $v_i \notin C$, the length of F is at least $2m + 2(n - |C|)$. Furthermore, if $v_i \in C$, then $\{w, a_i\} \notin F$, which means that we have to pay the penalty for the 3 demands containing a_i. Therefore, the total cost of the solution is at least $2m + 2n + |C|$. We assumed that the cost of the solution is at most $2m + 2n + |C|$, thus $|C| \leq k$ follows, as promised.

Finally, we notice that H is a subgraph of a series-parallel graph—duplicating edges $\{w, a_i\}$ turns H into a collection of cycles of length two and four sharing the central vertex w. Therefore, it is planar and has treewidth at most two. □

10.3.2 Remarks about the reduction

It was previously noted that not only is the reduced instance a series-parallel graph, but it is indeed a very simple one. The graph only consists of cycles of length four and paths of length one sharing a common central vertex.

Besides, the solution to the instance can be assumed—as proved and used in the analysis—to be a tree rather than a forest of trees. Therefore, the difference between

the reduced instance and an instance of PCST is not in the ability of optimum of forming multiple connected components, but it is in the interaction between two endpoints of each demand. In particular, an alternative way to look at the demands is as single-sink demands (with the central vetex as the sink), however, the penalty function charges the solution even if only one of the endpoints of an original demand is not connected to the central vertex. As a result, the same hardness holds for Submodular Prize-collecting Steiner Tree since the interaction between the endpoints of a demand can be captured via a submodular function.

We emphasize that PCSF has two major differences with PCST: choosing the demands to satisfy, and deciding how to form the connected components. Although the machinery of the previous chapter can take care of the second difficulty, the complexity of the problem, as shown in the above reduction, turns out to stem from the first issue; i.e., what demands should be satisfied.

The above reduction gives the first provable separation between complexity of a natural network optimization problem and its prize-collecting variant. In fact, it was believed that the prize-collecting extension of a problem does not make it harder. We presented a PTAS for planar Steiner Forest in Chapter 9, however, the above reduction proves that PCSF does not admit any PTAS on planar graphs, unless P = NP. We show the separation holds in the two-dimensional Euclidean plane as well—the PTAS for this case is due to Borradaile et al. [BKM08], and our hardness proof follows.

10.3.3 Hardness for Euclidean metrics

The proof for the Euclidean version is very similar to the graph version. The main difference is that the central vertex w is replaced by a set of points arranged along a long vertical path.

Proof of Theorem 3.2.6(2). We reduce an instance $\mathcal{I}$ of Minimum Vertex Cover

on 3-regular graphs to an instance $\mathcal{I}'$ of Prize-collecting Steiner Forest on points in the Euclidean plane. If n denotes the number of vertices of the 3-regular graph G in $\mathcal{I}$, then the number edges is $m = 3n/2$. We will denote the i^{th} vertex of G by v_i, the j^{th} edge by e_j, and the first and second endpoints of e_j by $e_j^{(1)}$ and $e_j^{(2)}$, respectively.

We now specify the reduction (illustrated in Figure 10.2). Let us define $\mu := 10000(n+m)$ ("basic unit of cost"), $\mu_H = 10\mu$ ("horizontal length"), and $\mu_V = 100\mu$ ("vertical spacing"). Instance $\mathcal{I}'$ contains a set P of points with integer coordinates. There are two groups of vertices Z and $P \setminus Z$; set Z includes

- $(0, y)$ for every $-m\mu_V \leq y \leq n\mu_V$,
- (x, y) for every $1 \leq x \leq \mu_H$ and $y = i\mu_V$ for $1 \leq i \leq n$, and
- (x, y) and $(x, y + 4\mu)$ for every $0 \leq x \leq \mu_H$ and $y = -j\mu_V$ for $1 \leq j \leq m$,

whereas $P \setminus Z$ includes

- $a_i = (\mu_H + 2\mu, i\mu_V)$ for $1 \leq i \leq n$,
- $b_j = (\mu_H, -j\mu_V + 2\mu)$ for $1 \leq j \leq m$, and
- $c_j^1 = (\mu_H, -j\mu_V + \mu_)$, and $c_j^2 = (\mu_H, -j\mu_V + 3\mu)$ for $1 \leq j \leq m$.

Note that $|Z| = \mu_V(n+m) + 1 + \mu_H(n+2m)$. For the ease of notation, we define $w_i = (\mu_H, i\mu_V)$, $w_j^1 = (\mu_H, -j\mu_V)$, $w_j^2 = (\mu_H, -j\mu_V + 4\mu)$, that are all in Z.

The instance contains the following demands.

1. If (x, y) and $(x+1, y)$ are both in Z, there is a demand $\{(x, y), (x+1, y)\}$ with penalty 1. This applies to vertices (x, y) such that $y = i\mu_V$ or $y = -j\mu_V$ or $y = -j\mu_V + 4\mu$.
2. If (x, y) and $(x, y+1)$ are both in Z, there is a demand $\{(x, y), (x, y+1)\}$ with penalty 1. This applies to the points (x, y) with $x = 0$.

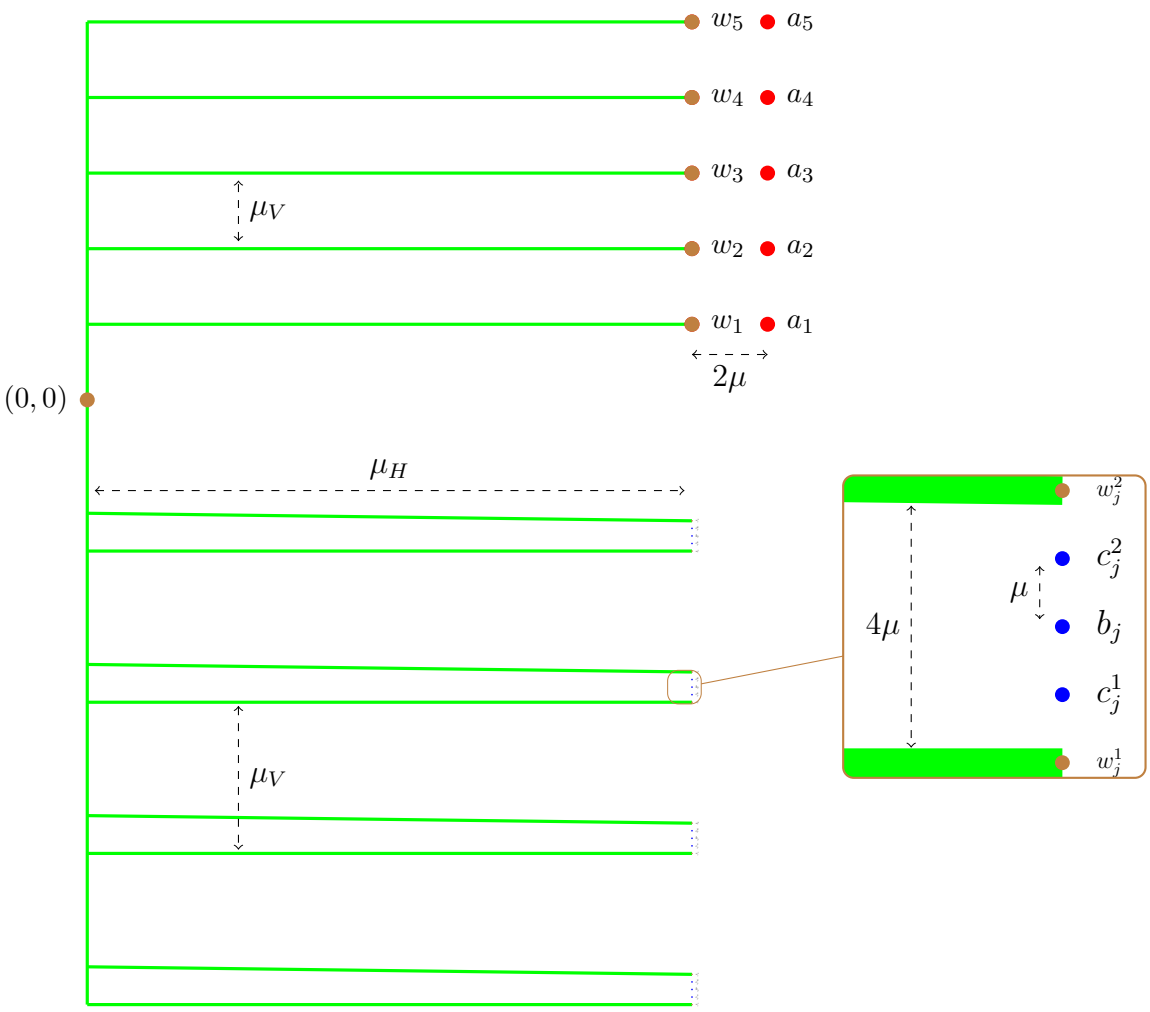


Figure 10.2: Illustrating the reduction from 3-regular Minimum Vertex Cover to Euclidean Prize-collecting Steiner Forest. Points with integer coordinates on the solid lines form the set Z. There are n horizontal lines corresponding to vertices of G, and m pairs of horizontal lines corresponding to its edges. These lines are used to simulate the role of the central vertex w in the planar graph reduction. Notice that distances in the figure do not always mirror the actual distances.

3. There is a demand $\{(0,0), b_j\}$ with penalty 3μ for $1 \leq j \leq n$.

4. If $v_i = e_j^{(p)}$ for some $1 \leq i \leq n$, $1 \leq j \leq m$, and $p \in \{1,2\}$, then $\{a_i, c_j^p\}$ is a demand with penalty $\mu - 10$.

The total number of demands is $|Z| - 1 + n + 3m$, and each a_i appears in exactly 3 demands. We claim that the cost of the optimum of $\mathcal{I}'$ is between $|Z| + \mu(2m + 2n + \tau(G))$ and $|Z| + \mu(2m + 2n + \tau(G)) - 100n$, where $\tau(G)$ is the size of the minimum vertex cover in G. Since $m = 3n/2$ and $\tau(G) \geq m/3$, we know that $|Z| + \mu(2m + 2n + \tau(G))$ is at most a constant factor larger than $\mu\tau(G)$. Therefore, a change in $\tau(G)$ by a constant factors translates to a constant-factor change in the optimum of $\mathcal{I}'$.

More precisely, in order to prove the correctness of the reduction, we prove the following two statements.

(1) Given a vertex cover of size k for G, a solution of cost at most $|Z| + (2m + 2n + k)\mu$ for $\mathcal{I}'$ can be constructed.

(2) Given a solution of cost at most $|Z| + (2m + 2n + k)\mu$ for $\mathcal{I}'$, a vertex cover of size at most k can be constructed for G.

To prove (1), suppose that C is a vertex cover of size k for G. Let F be the forest (actually, a tree) that contains

1. an edge $\{(x,y), (x+1,y)\}$ if both these points are in P,

2. an edge $\{(x,y), (x,y+1)\}$ if both these points are in P,

3. an edge $\{w_i, a_i\}$ if $v_i \notin C$,

4. edges $\{w_j^1, c_j^1\}$ and $\{c_j^1, b_j\}$ if $e_j^{(1)} \notin C$, and

5. edges $\{w_j^2, c_j^2\}$ and $\{c_j^2, b_j\}$ if $e_j^{(1)} \in C$.

The total length of F is $|Z| - 1 + 2\mu(n-k) + 2\mu m$, where $|Z| - 1$ accounts for the first two categories of edges above; $2\mu(n-k)$ accounts for the third category; and $2\mu m$ takes into account the last two categories. Observe that all the demands $\{(0,0), b_j\}$ are satisfied: F connects $(0,0)$ to all points (x,y) in P with $x = 0$, to w_j^1 and w_j^2 for $1 \leq j \leq m$; moreover, F connects one of w_j^1 or w_j^2 to b_j through the appropriate c_j^p. Furthermore, if $v_i \notin C$, then all three demands where a_i appears are satisfied. This can be seen as follows. First, a_i is in the same component as w_i and hence as every vertex of Z. If $v_i = e_j^{(1)}$, then there is a demand $\{a_i, c_j^1\}$ and c_j^1 is connected with w_j^1 (and hence with a_i). If $v_i = e_j^{(2)}$, then $v_i \notin C$ implies that $e_j^{(1)} \in C$ must hold—since C is a vertex cover—and therefore c_j^2 is connected to w_j^2, satisfying the demand $\{a_i, c_j^2\}$. Thus, the total penalty is at most $3k(\mu - 10)$, and hence the cost of the solution is at most $|Z| - 1 + (2m + 2n + k)\mu - 30k$, as claimed.

To prove (2), suppose that forest F is an optimal solution such that the sum of the length of F and the penalties it pays is at most $|Z| + \mu(2n + 2m + k)$. First, we can assume that every demand of the first two types is satisfied: if, say, $\{(x,y), (x+1,y)\}$ is not satisfied, then we can extend F by adding an edge of length 1, which decreases the penalty by at least 1. Thus, all points in Z points are in the same connected component K of F. We can also assume that every demand of the third type is satisfied: if $\{(0,0), b_j\}$ is not satisfied, we can decrease the penalty by 3μ at the cost of at most 2μ by adding edges $\{w_j^1, c_j^1\}$ and $\{c_j^1, b_j\}$, contradicting the optimality of F. Therefore, every vertex b_j is in the component K.

Let $Z' = \{(x,y) \in Z \mid x = 0 \vee x \geq 10\}$. Let R be the region of the plane at Manhattan distance at most 3 from Z'. Note that R consists of one "vertical" and $n + 2m$ "horizontal" components. See Figure 10.3a.

Claim 3. *The length of F inside R is at least $|Z'|$.*

Proof. We have seen above that a single component K of F contains every point of $P \cap R$ (because $P \cap R \subseteq Z$). The restriction of K to R gives rise to several components.

(a) R (b) R^+

Figure 10.3: Illustration of sets R and R^+ in the hardness proof for Euclidean PCSF. Notice that scales do not always mirror actual distances.

Consider such a component K' containing a subset $S \subseteq Z'$ of vertices. We show that the length of K' is at least $|S|$. The vertices of S lie on a horizontal or vertical line. This means that there are two vertices $s_1, s_2 \in S$ at distance $d \geq |S| - 1$. As K is not contained fully in any component of R, component K' has to contain a point s_3 on the boundary of R. As s_3 is at distance at least 3 from s_1 and s_2, it can be verified that any Steiner tree of s_1, s_2, s_3 has length at least $d + 1 = |S|$. Summing for every component K' of the restriction of K to R, we get that the length of K in R is at least $|Z'|$. □

Let R^+ be the region of space at Manhattan distance at most 3 from Z. See Figure 10.3b.

Claim 4. *The length of every component of $F \setminus R^+$ is at most 3μ.*

Proof. There are two types of components of $F \setminus R^+$: (1) those that contain a point of

P, and (2) those that do not contain such a point. Clearly, there are at most $n + 3m$ components of the first type (since $|P \setminus Z| \leq n + 3m$).

Suppose that there is a component D of the second type having length more than 3μ. In this case, we modify F to obtain a better solution as follows. Consider $F \setminus R^+$ (i.e., let us remove the part of F inside R^+) and let us remove every component of the second type. After that, let us add all the $|Z| - 1$ edges of the form $\{z_{x,y}, z_{x+1,y}\}$, $\{z_{x,y}, z_{x,y+1}\}$. Finally, for every component of the first type, if it intersects R^+, then let us choose a point of the component on the boundary of R^+ and connect this point to the nearest point of Z. It is clear that the new forest F' satisfies every demand satisfied by F: every point of P connected to Z remains connected to Z. Claim 3 and $R \subseteq R^+$ ensure that the length of $F \setminus R'$ is less than the length of F by at least $|Z'| = |Z| - 9(n + 2m)$. Removing components of the second type decreases the length by more than 3μ (as there are at least one such component having length more than 3μ). The edges connecting Z increase the length by $|Z| - 1$. Adding the new connections corresponding to the components of the first type increases the length by at most $3(n + 3m)$. Since $3\mu \geq 9(n + 2m) - 1 + 3(n + 3m)$, forest F' is a strictly better solution, which is a contradiction.

Suppose now that there is a component D of the first type with length more than 3μ. For $-m \leq s \leq n$, let R_s be the region of the plane at Manhattan distance at most 4μ from $(\mu_H, s\mu_V)$. Any point in $P \setminus Z$ falls within exactly one R_s. Observe that, for each s, all the points of $P \cap R_s$ can be connected to the nearest point of Z with a total length of at most 3μ. This implies that, if D intersects only one of these regions, say R_s, we can substitute D with a curve of length at most 3μ in such a way that every demand satisfied by F remains satisfied, contradicting the optimality of F. Suppose, therefore, that D intersects $t \geq 2$ of these regions; in this case, the length of D is at least $(t - 1)(\mu_V - 8\mu) > 90(t - 1)\mu \geq 45t\mu \geq 3t\mu$. Let us replace D by connecting every point of $P \cap D$ to the closest point of Z. The new connections

increase the length by at most $t \cdot 3\mu$, which is less than the length of D, contradicting the optimality of F. □

Observe that the distance of $a_i, a_{i'}$ for $i \neq i'$ is more than 3μ; so is the distance of a_i to the set $\{b_j, c_j^1, c_j^2\}$; similarly, the distance of any two sets $\{b_j, c_j^1, c_j^2\}$ and $\{b_{j'}, c_{j'}^1, c_{j'}^2\}$ for $j \neq j'$ is more than 3μ. It now follows from Claim 4 that, for every component D of $F \setminus R^+$, $D \cap P$ is either a single a_i, or a (possibly empty) subset of $\{b_j, c_j^1, c_j^2\}$. Therefore, every such component D intersects R^+: otherwise, D could be safely removed as it does not satisfy any demand. Next we show that it can be assumed that only one of c_j^1 and c_j^2 is in K. Otherwise, we can remove every component of $F \setminus R^+$ intersecting $\{c_j^1, c_j^2\}$ and replace them with the edges $\{w_j^1, c_j^1\}$ and $\{c_j^1, b_j\}$. The total length of the components removed in this way is at least $2\mu + \mu - 3$ (which is the minimum cost of connecting c_j^1 and c_j^2 to each other and to R^+), and the new edges have length 2μ. This transformation might disconnect the demand containing c_j^2, hence the penalty can increase by at most $\mu - 10$ only, contradicting the optimality of F.

We can assume that, if a_i is in K, all 3 demands containing a_i are connected: otherwise removing the component of $F \setminus R^+$ containing a_i decreases the length by at least $2\mu - 3$ and increases the penalty by at most $2(\mu - 10)$.

Let vertex v_i be in C if and only if a_i is not in component K. We claim that C is a vertex cover of size at most k. To see that C is a vertex cover, consider an edge e_j. We have observed above that one of c_j^1 and c_j^2 is not in K. If $c_j^1 \notin K$ and $e_j^{(1)} = v_i$, then the demand $\{a_i, c_j^1\}$ is not connected by F. Therefore, not all 3 demands containing a_i are connected, which implies (as observed above) that a_i is not in K. Thus, $v_i \in C$ covers the edge e_j. Similarly, $c_j^2 \notin K$ gives $e_j^{(2)} \in C$.

The length of $F \cap R^+$ is at least $|Z| - 9(n + 2m)$. Since every b_j is in K and a_i is in K if $v_i \notin C$, the length of $F \setminus R^+$ is at least $(2\mu - 3)m + (2\mu - 3)(n - |C|)$. Furthermore, if $v_i \in C$, we have to pay the penalty for the 3 demands containing a_i.

Therefore, the total cost of the solution is at least

$$|Z| - 9(n+2m) + (2\mu-3)m + (2\mu-3)(n-|C|) + 3|C|(\mu-10)$$

$$\geq |Z| + (2m+2n+|C|)\mu - 100n.$$

We assumed that the cost of the solution is at most $|Z|+(2m+2n+k)\mu$. As $\mu > 100n$, this is only possible if $|C| \leq k$. This concludes of the proof. □

10.4 PCST, PCTSP, and PCS

Theorem 3.2.7 states that solving any of these problems—i.e., PCST, PCTSP, or PCS—on bounded-treewidth graphs immediately yields a PTAS for its bounded-genus instances. The PTAS follows since the reduced problem has the same structure as that of the original instance: if it is a single-sink demand set, it remains so; and a PCTSP or PCS problem remains so after the reduction.

In the following, we present an exact algorithm for Prize-collecting Steiner Tree on bounded-treewidth instances. The algorithms for Prize-collecting TSP and Prize-collecting Stroll require only minor changes, and are not explicitly given here.

10.4.1 Bounded-treewidth PCST

Recall that the input consists of a graph $G(V,E)$ with length $\ell : E \mapsto \mathbb{R}^+$ on edges, as well as a penalty function $\pi : V \mapsto \mathbb{R}^+$ on vertices. In addition, we have a nice tree decomposition $(T, \mathcal{B})$ for G where each bag $B_i \in \mathcal{B}$ has size at most k for some constant k; i.e., the treewidth of G is at most $k-1$. We use I as the set of nodes of T. Let T_i denote the subtree of T rooted at $i \in I$. The vertices of B_i are called *portals* of T_i.

As hinted earlier, we employ the dynamic-programming technique to solve the problem. A dynamic-programming entry is specified by a tuple $(i, S, \mathcal{P})$ where

- $i \in I$ is a node in the tree decomposition,
- $S \subseteq B_i$ is a subset of portals of the subtree T_i, and
- $\mathcal{P}$ is a partition of S.

Let us denote by V_i the vertices corresponding to the subtree T_i, i.e., $V_i := \cup_{i' \in V(T_i)} B_{i'}$. Notice that, to avoid confusion, we refer to "nodes" of the tree decomposition and "vertices" of the original graph. A dynamic-programming entry $\texttt{DP}(i, S, \mathcal{P})$ takes up the least cost (i.e., length and penalty) of a subgraph H such that

- H uses only the edges both whose endpoints are in V_i,
- H connects the vertices in each part P_j of the partition $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$, and
- S is the subset of B_i whose penalty is not paid; moreover, if a vertex $v \in V_i$ is not connected to S via H, its penalty $\pi(v)$ is already paid in the total cost.

The final solution to the problem can be found as

$$\min_{S \subseteq B_r} \texttt{DP}(r, S, \{S\})$$

where r is the root of the tree decomposition, i.e., it does not matter which subset of the bag of the root is picked as long as they form a single component.

As is customary for dynamic-programming algorithms, we only describe how to obtain the value of the optimum rather the actual solution itself. Standard techniques can be used to store additional information in DP entries for recovering the final solution.

We next describe the DP algorithm for different nodes of T.

Leaf node i: there are two DP entries associated with $i \in I$, and we set them as $\texttt{DP}(i, \emptyset, \emptyset) = \pi(v)$ and $\texttt{DP}(i, \{v\}, \{\{v\}\}) = 0$.

Introduce node i is the parent of i', and we have $B_i = B_{i'} \cup \{v\}$. Then, $\texttt{DP}(i, S, \mathcal{P}) = \pi(v) + \texttt{DP}(i', S, \mathcal{P})$ if $v \notin S$. For the case $v \in S$, we have $\mathcal{P} = \{P_1, P_2, \dots\}$ such that $v \in P_1$. We set

$$\texttt{DP}(i, S, \mathcal{P}) = \min_{\mathcal{P}', S'} \left[\texttt{DP}(i', S \setminus \{v\}, \mathcal{P}') + \sum_{v' \in S'} \ell((v, v')) \right],$$

where

(I1) $\mathcal{P}'$ is a partition on $S \setminus \{v\}$; and

(I2) $\mathcal{P} = \mathcal{P}' \vee \{\{v\} \cup S'\}$.

The second term accounts for the cost of connecting v to all vertices in S', and these connections in addition to $\mathcal{P}'$ yield the connectivities in $\mathcal{P}$.

Forget node i is the parent of i', and we have $B_{i'} = B_i \cup \{v\}$. Then, we set

$$\texttt{DP}(i, S, \mathcal{P}) = \min \left[\texttt{DP}(i', S, \mathcal{P}), \min_{\mathcal{P}'} \texttt{DP}(i', S \cup \{v\}, \mathcal{P}') \right], \tag{10.2}$$

where $\mathcal{P}'$ is a partition of $S \cup \{v\}$ that induces $\mathcal{P}$ on S. The first term considers the case where we have already paid the penalty for v and do not plan to connect it in the final Steiner tree, whereas the second term takes into account the case where v is connected to one connected component of the partition.

Join node the node i has two children i_1 and i_2 with the same bags. We set

$$\texttt{DP}(i, S, \mathcal{P}) = \min_{\mathcal{P}_1, \mathcal{P}_2} \{\texttt{DP}(i_1, S, \mathcal{P}_1) + \texttt{DP}(i_2, S, \mathcal{P}_2) - \pi(B_i \setminus S)\}, \tag{10.3}$$

where the minimization goes over all pairs $\mathcal{P}_1$ and $\mathcal{P}_2$ such that $\mathcal{P} = \mathcal{P}_1 \vee \mathcal{P}_2$.

The last term cancels out the effect of paying the penalty of the unsatisfied terminals of B_i twice, i.e., once in each subtree.

It is straight-forward to verify that the value of each DP entry corresponds to a valid solution, i.e., the computed value is at least the desired one. Next we discuss why the computation indeed produces the desired result, by showing that any valid solution is discovered in the process.

The claim is trivial for the leaf nodes since there are only two possible DP entries at each leaf, and the values are computed correctly there.

For the DP entry $(i, S, \mathcal{P})$ at the introduce node i, that is the parent of i' with the introduced vertex v, we consider two cases. The solution for connecting S according to $\mathcal{P}$ does not use v since $v \notin S$, hence the same solution exists in subgraph $T_{i'}$. The only additional penalty the solution needs to pay at node i is that of v.

The second case is when $v \in S$. Let S' denote the neighbors of v in the subsolution; observe that they all have to be in the bag B_i. Removing the edges connecting v to S' refines the partition $\mathcal{P}$, and further removing v itself gives a partition $\mathcal{P}'$ on $S \setminus \{v\}$. Then, the solution cost comes from a subsolution for $\texttt{DP}(i', S \setminus \{v\}, \mathcal{P}')$ as well as the edges connecting v to its neighbors S'.

Now consider $(i, S, \mathcal{P})$ for a forget tree node i. If the solution does not use v, the first term of Equation (10.2) finds the desired solution. Otherwise, suppose v is indeed used in the solution. Thus, $S \cup \{v\}$ is part of the solution. Let $\mathcal{P}'$ be the partition the subsolution induces on $S \cup \{v\}$. Clearly, $\mathcal{P}'$ induces $\mathcal{P}$ on S. Therefore, the solution is discovered during the minimization.

Finally, we look at $(i, S, \mathcal{P})$ where i is a join node whose children are i_1 and i_2. If we simply take $\mathcal{P}_j$, for $j = 1, 2$, to be partitions induced by $\mathcal{P}$ on V_{i_j}, the solution may pay the cost of some edges more than once. Thus, we need to find $\mathcal{P}_1$ and $\mathcal{P}_2$ carefully. Let F be the intended subsolution corresponding to $\texttt{DP}(i, s, \mathcal{P})$. Let $F^* = F \setminus F[B_i]$ be the forest formed from F by removing edges with both endpoints in the bag B_i. Let

F_j, for $j = 1, 2$, be the forest $F^*[B_{i_j}]$ induced in each subtree. Let $\mathcal{P}^*_j$, for $j = 1, 2$, be the partition of vertices of S induced by F^*_j. Clearly, F^*_1 is a solution for $\texttt{DP}(i_1, S, \mathcal{D}^*_1)$, and $F^*_2 \cup F[B_i]$ is a solution for $\texttt{DP}(i_2, S, \mathcal{D}^*_2 \vee \mathcal{D})$. Notice that these two forests are disjoint. Therefore, their total length is equal to the length of F. That F is correctly discovered via Equation (10.3) follows because the only penalties from F that the two forest (F^*_1 and $F^*_2 \cup F[B_i]$) pay together are those in $B_i \setminus S$.

Notice that the number of DP entries is at most $k^{O(k)}|I|$. Since finding the value of each DP entry takes $k^{O(k)}$ time, we obtain an EPTAS for Prize-collecting Steiner Tree on bounded-treewidth graphs, that yields an EPTAS for the bounded-genus version as discussed above.

To extend the algorithm to Prize-collecting TSP, we augment the DP state to $(i, S, \mathcal{P}, Q)$ where Q contains one pair of vertices from each part of $\mathcal{P}$. The implication is that in each part $P \in \mathcal{P}$ with $(s, t) \in Q$ and $\{s, t\} \in P$, we have a path from s to t spanning P among B_i (and possibly some vertices of $V_i \setminus B_i$) such that the path only uses the edges of T_i. Dynamic programming stitches these paths together as it moves up the tree towards the root r. The final solution is $\min_{s \in S \subseteq B_r} \texttt{DP}(r, S, \{(s, s)\})$. The algorithm for Prize-collecting Stroll works in the same way except that the final solution can be founded in $\min_{s,t \in B_r, S \subseteq B_r} \texttt{DP}(r, S, \{(s, t)\})$ since we do not need to have a closed tour.

10.5 Multiplicative prize-collecting Steiner forest

Here we discuss Prize-collecting Steiner Forest where penalties (or prizes) have a multiplicative nature. In the symmetric case, each vertex v has a nonnegative real weight $w(v)$, and there is a demand with penalty $w(v_1)w(v_2)$ between any pair (v_1, v_2) of vertices. In the asymmetric case, each vertex v has two types of weights, namely $w_s(v)$ and $w_t(v)$, and the penalty for the demand (v_1, v_2) is $w_s(v_1)w_t(v_2)$. This

can model situations where there are two types of vertices and any demand should be between vertices of two different types.[6]

This setting is inspired by Product Multi-Commodity Flow in [LR99, Bon04, KS02], and its applications in wireless networks [MSL08] or routing [CKS04, CKS05]. However, the main motivation is the setting when the weights (after some normalization) denote the probability of a vertex appearing in a set of active vertices, and a demand will be realized between each pair of active vertices.

Theorem 3.2.7 reduces the bounded-genus variant of MPCSF to its bounded-treewidth version. We show in Section 10.5.4 how to obtain a PTAS for the latter (for arbitrary vertex weights, or an exact algorithm for small integer weights). This immediately implies a PTAS for bounded-genus MPCSF. The PTAS for the bounded-treewidth case is based on a bicriteria approximation algorithm for the fixed-prize variant of the problem, Π-MPCSF, in the bounded-treewidth setting. However, we emphasize that this does not lead to a PTAS for bounded-genus Π-MPCSF since the reduction only works for the prize-collecting setting, not for the fixed-prize setting. In fact, obtaining a PTAS for this problem—or its special case, k-MST—is still an interesting open question.

When the weights are polynomially small integers, we can extend the DP algorithm of the previous section to solve the fixed-prize (and as a result the prize-collecting version of) MPCSF for bounded-treewidth graphs even for asymmetric weights; see Section 10.5.1. Roughly speaking, this is done by storing in DP entries the weight of the component associated with each part of the partition the subsolution induces on the portals.

Then in Section 10.5.2 we discuss how granularization can be used to generalize the said algorithm to the case of arbitrary weights. The idea is to round the weights to integer multiples of some unit, so that addition and multiplication can be stored

[6]This can be generalized to more than two types, but in the interest of simplicity of getting the ideas across, we focus on the case of two types or weights.

approximately without compromising too much on the size of the DP table (and the algorithm's running time).

The asymmetric setting requires additional tricks and is discussed separately in Section 10.5.3. In particular, a single precision unit for the weights of all vertex sets may not be sufficient, and as the algorithm proceeds it uses different units for storing the weights. This idea is explained in Chapter 7 as dynamic granularization.

Section 10.5.4 gives the PTAS for MPCSF. This is simple if the weights are polynomially small integers since we can try all possible values for the collected prize to find the optimal choice. However, for the case of arbitrary weights, we show that, if the penalty portion of the optimal solution is comparable to the total penalties of all demands, polynomially many calls to the fixed-prize algorithm suffices for finding the optimum. If the penalty paid by the optimum is very small, though, most of the prize has to be collected. In this case, roughly speaking, we can find and leave aside most of the vertex weights and focus on the rest of the problem that can be solved using a modified version of the fixed-prize algorithm. Yet, unlike our previous work [BH10], we use this information to solve the original problem via several calls to the unmodified fixed-prize algorithm.

Finally in Section 10.5.5 we hint on the major differences of the bounded-treewidth and Euclidean settings.

10.5.1 Fixed prize from asymmetric small integer weights

Recall that the input consists of a graph $G(V, E)$ with length $\ell : E \mapsto \mathbb{R}^+$ on edges, as well as two weight functions $w_s, w_t : V \mapsto \mathbb{Z}^+$ on vertices. We are also given a nonnegative integer Π as a lower bound on how much prize we should collect. In addition, we have a nice tree decomposition $(T, \mathcal{B})$ for G where each bag $B_i \in \mathcal{B}$ has size at most k for some constant k; i.e., the treewidth of G is at most $k - 1$. We

assume in this section that the bag B_r of the root r of T is empty.[7] We use I as the set of nodes of T. Let T_i denote the subtree of T rooted at $i \in I$. The vertices of B_i are called *portals* of T_i. Define $W = \max[w_s(V), w_t(V)]$ as an upper bound on the (type-one or type-two) weights of all vertices.

As hinted earlier, we employ the dynamic-programming technique to solve the problem. A dynamic-programming entry is specified by a tuple $(i, \mathcal{P}, \omega_s, \omega_t, \sigma)$ where

- $i \in I$ is a node in the tree decomposition,
- $\mathcal{P}$ is a partition of B_i,
- ω_s, ω_t are functions mapping parts of $\mathcal{P}$ to nonnegative integers less than W, and
- σ is a nonnegative integer no more than W^2.

We sometimes abuse the notation $\omega_s(P) = \sum_{P_i \subseteq P} \omega_s(P_i)$ or $\omega_t(P) = \sum_{P_i \subseteq P} \omega_t(P_i)$ if ω_s, ω_t are defined on parts of $\mathcal{P} = \{P_1, P_2, \dots\}$. For any function ω_s, ω_t defined on the parts of a partition $\mathcal{P}$ of B_i, we use the notation $\hat{\omega}_s(P) = \omega_s(P) + w_s(P \cap B_i)$ and $\hat{\omega}_t(P) = \omega_t(P) + w_t(P \cap B_i)$ for any $P \subseteq \mathcal{P}$.

Let us denote by V_i the vertices corresponding to the subtree T_i, i.e., $V_i := \cup_{i' \in V(T_i)} B_{i'}$. Notice that, to avoid confusion, we refer to "nodes" of the tree decomposition and "vertices" of the original graph. A dynamic-programming entry $(i, \mathcal{P}, \omega_s, \omega_t, \sigma)$ takes up the least cost (i.e., length and penalty) of a subgraph H such that the following hold.

- H uses only the edges both whose endpoints are in V_i.
- H connects the vertices in each part P_j of the partition $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$.

[7] This can be easily obtained from any tree decomposition by adding a path, from the original root to a new root node, consisting of forget nodes only.

- $\omega_s(P_i)$ (respectively, $\omega_t(P_i)$) denotes the total type-one (respectively, type-two) weight of nonportals in the component of H containing P_i—i.e., we do not include the weight of P_i. Notice that $\hat{\omega}_s(P_i) = \omega_s(P_i) + w_s(P_i)$ holds the actual weight of all vertices in the component containing P_i; the same is true for $\hat{\omega}_t$.

- σ is the total prize collected by H from (satisfied) demands that are not connected to any portal in B_i. Notice that the total prize collected by H is then $\sigma + \sum_i \hat{\omega}_s(P_i)\hat{\omega}_t(P_i)$, however, for entries corresponding to the root of the tree decomposition, this is simply σ since $B_r = \emptyset$.

The definitions of $\sigma, \omega_s, \omega_t$ may seem strange at this point, but, not only do they simplify the presentation of the algorithm, but they are crucial for the case of arbitrary weights. Roughly speaking, with the natural definitions of $\sigma, \omega_s, \omega_t$, one would have to perform subtraction as well as addition on them to fill the DP table. This makes it very difficult, if not impossible, to carry out the charging via granularization.

The final solution to the problem can be found from

$$\min_{\sigma \geq \Pi} \mathtt{DP}(r, \{\}, \omega_s, \omega_t, \sigma),$$

where σ_s, σ_t are trivial functions (since their domains are empty).

As is customary for dynamic-programming algorithms, we only describe how to obtain the value of the optimum rather the actual solution itself. Standard techniques can be used to store additional information in DP entries for recovering the final solution.

We next describe the DP algorithm for different nodes of T.

Leaf node i: there is only one DP entry associated with $i \in I$, and we set it as $\mathtt{DP}(i, \{\{v\}\}, \omega_s, \omega_t, 0) = 0$ with ω_s, ω_t being the appropriate functions.

Introduce node i is the parent of i', and we have $B_i = B_{i'} \cup \{v\}$. We have $\mathcal{P} =$

$\{P_1, P_2, \dots\}$ such that $v \in P_1$. We set

$$\mathtt{DP}(i, \mathcal{P}, \omega_s, \omega_t, \sigma) = \min_{\mathcal{P}', S'} \left[\mathtt{DP}(i', \mathcal{P}', \omega'_s, \omega'_t, \sigma) + \sum_{v' \in S'} \ell((v, v')) \right], \tag{10.4}$$

where

(I1) $\mathcal{P}'$ is a partition on $B_i \setminus \{v\}$;

(I2) $\mathcal{P} = \mathcal{P}' \vee \mathcal{P}''$, where $\mathcal{P}''$ is a partition of B_i all whose elements are singletons except for a single part for $\{v\} \cup S'$;

(I3) $\omega_s(P_i) = \omega'_s(P_i \setminus \{v\})$; and

(I4) $\omega_t(P_i) = \omega'_t(P_i \setminus \{v\})$.

The second term of (10.4) accounts for the cost of connecting v to all vertices in S', and these connections in addition to $\mathcal{P}'$ yield the connectivities in $\mathcal{P}$.

Forget node i is the parent of i', and we have $B_{i'} = B_i \cup \{v\}$. Then, we set

$$\mathtt{DP}(i, \mathcal{P}, \omega_s, \omega_t, \sigma) = \min_{\mathcal{P}'} \mathtt{DP}(i', \mathcal{P}', \omega'_s, \omega'_t, \sigma'), \tag{10.5}$$

where the following hold.

(F1) $\mathcal{P}'$ is a partition of $B_i \cup \{v\}$ that induces $\mathcal{P}$ on B_i.

(F2) Let $v \in P'_1 \in \mathcal{P}'$. We have $\omega_s(P'_1) = \omega'_s(P'_1) + w_s(v)$ and $\omega_s(P'_i) = \omega'_s(P'_i)$ for $i > 1$.

(F3) We have $\omega_t(P'_1) = \omega'_t(P'_1) + w_t(v)$ and $\omega_t(P'_i) = \omega'_t(P'_i)$ for $i > 1$.

(F4) If $|P'_1| > 1$, then $\sigma = \sigma'$, otherwise we have $\sigma = \sigma' + \omega'_s(P'_1)\omega'_t(P'_1)$.

No edge is added in a forget node, hence no new connectivity or change in collected prize unless the forgotten vertex v forms a singleton part by itself in $\mathcal{P}'$.

Join node the node i has two children i_1 and i_2 with the same bags. We set

$$\mathtt{DP}(i, \mathcal{P}, \omega_s, \omega_t, \sigma) = \min_{\substack{\mathcal{P}_1, \mathcal{P}^2 \\ \omega_s^1, \omega_s^2 \\ \omega_t^1, \omega_t^2 \\ \sigma^1, \sigma^2}} \left\{ \mathtt{DP}(i_1, \mathcal{P}^1, \omega_s^1, \omega_t^1, \sigma^1) + \mathtt{DP}(i_2, \mathcal{P}^2, \omega_s^2, \omega_t^2, \sigma^2) \right\}, \tag{10.6}$$

where the following hold.

(J1) $\mathcal{P} = \mathcal{P}^1 \vee \mathcal{P}^2$.

(J2) Let $\mathcal{P}^1 = \{P_1^1, P_2^1, \dots\}$ and $\mathcal{P}^2 = \{P_1^2, P_2^2, \dots\}$. Then, we have $\omega_s(P_i) = \omega_s^1(P_i) + \omega_s^2(P_i)$.

(J3) $\omega_t(P_i) = \omega_t^1(P_i) + \omega_t^2(P_i)$.

(J4) $\sigma = \sigma^1 + \sigma^2$.

Now we verify that DP computes what it is supposed to do. This is trivial for leaf nodes. Before verifying the claim for the other three types of nodes, we observe that the conditions for ω_s, ω_t in the above are immediate consequences of the relationship between $\mathcal{P}, \mathcal{P}'$ and $\{v\}$. The same holds for computation of σ in the recursion. Notice that in an introduce (or joint) node, no vertex disappears, nor do nonportals get connected to each other. This happens for a forget node if v is in a singleton part of the partition (possibly with some nonportal vertices), hence we account for the disappearance of this part of the partition in σ.

It is now easy to see that the solution produced by DP is always a valid solution, i.e., costs no less than the desired solution. It suffices to show that any valid solution is discovered in the process.

For the DP entry $\mathtt{DP}(i, \mathcal{P}, \omega_s, \omega_t, \sigma)$ at the introduce node i, that is the parent of i' with the introduced vertex v, let S' denote the neighbors of v in the subsolution; observe that they all have to be in the bag B_i. Removing the edges connecting v to S' refines the partition $\mathcal{P}$, and further removing v itself gives a partition $\mathcal{P}'$ on $B_{i'} =$

$B_i \setminus \{v\}$. Thus, $\mathcal{P} = \mathcal{P}' \vee \mathcal{P}''$, where $\mathcal{P}''$ is defined in (I2). Then, for ω'_s, ω'_t as defined in (I3)-(I4), the solution cost comes from a subsolution for $\texttt{DP}(i', \mathcal{P}', \omega'_s, \omega'_t, \sigma)$—which is considered by DP since (I1)-(I4) hold—as well as the edges connecting v to its neighbors S'.

Now consider $\texttt{DP}(i, \mathcal{P}, \omega_s, \omega_t, \sigma)$ for a forget tree node i, with the intended solution F. Let $\mathcal{P}'$ be the partition of $B_{i'}$ induced by F. If $\omega'_s, \omega'_t, \sigma'$ are defined according to (F2)-F(4), F is a solution for $\texttt{DP}(i, \mathcal{P}', \omega'_s, \omega'_t, \sigma')$. Since (F1)-(F4) hold, the solution F is discovered during the minimization.

Finally, we look at $\texttt{DP}(i, \mathcal{P}, \omega_s, \omega_t, \sigma)$ where i is a join node whose children are i_1 and i_2. If we simply take $\mathcal{P}_j$, for $j = 1, 2$, to be partitions induced by $\mathcal{P}$ on V_{i_j}, the solution may pay the cost of some edges more than once. Thus, we need to find $\mathcal{P}_1$ and $\mathcal{P}_2$ carefully. Let F be the intended subsolution corresponding to $\texttt{DP}(i, \mathcal{P}, \omega_s, \omega_t, \sigma)$. Let $F^* = F \setminus F[B_i]$ be the forest formed from F by removing edges with both endpoints in the bag B_i. Let F_j, for $j = 1, 2$, be the forest $F^*[B_{i_j}]$ induced in each subtree. Let $\mathcal{P}^*_j$, for $j = 1, 2$, be the partition of vertices of B_i induced by F^*_j. Clearly, F^*_1 is a solution for $\texttt{DP}(i_1, \mathcal{P}^*_1, \omega^1_s, \omega^1_t, \sigma_1)$ for appropriately defined $\omega^1_s, \omega^1_t, \sigma_1$. Similarly, $F^*_2 \cup F[B_i]$ is a solution for $\texttt{DP}(i_2, \mathcal{P}^*_2 \vee \mathcal{P}, \omega^2_s, \omega^2_t, \sigma_2)$ for appropriately defined $\omega^2_s, \omega^2_t, \sigma_2$. Notice that these two forests are disjoint. Therefore, their total length is equal to the length of F. As discussed above, (J2)-(J4) follow from (J1) and the fact that F is the disjoint union of the two forests $F^*_1, F^*_2 \cup F[B_i]$. Therefore, F is correctly discovered via Equation (10.6).

The number of DP entries is at most $k^{O(k)} W^{O(k)} |I|$. Since finding the value of each DP entry takes $(kW)^{O(k)}$ time, we obtain a PTAS for Π-MPCSF on bounded-treewidth graphs.

10.5.2 Fixed prize from symmetric arbitrary weights

For the case of arbitrary weights, MPCSF generalizes the KNAPSACK problem. Thus, unless $\mathsf{P} = \mathsf{NP}$, we do not expect a polynomial-time algorithm that works for every value Π of collected prize. Instead, we relax the requirement to a bicriteria one. We want to find a solution whose cost is no more than opt, but collects almost as much prize as required (i.e., at least $(1 - \epsilon')\Pi$).

Let the original instance consist of G, ℓ, w, Π, where w is a nonnegative real weight function on vertices. Let $n = |V(G)|$ denote the number of vertices of the graph.

Maintaining the values of $\omega_s, \omega_t, \sigma$ during the algorithm only requires addition and zero initialization. Indeed, ω_s, ω_t represent the weight of a set (i.e., the forgotten vertices of each part of a partition of the portals) with size at most n. The only operation performed on them is (1) initialization to a singleton set, (2) copying, and (3) taking unions. A static granularization with precision $\theta = \frac{\epsilon'\sqrt{\Pi}}{2n}$ is sufficient to give, roughly speaking, an additive error of $\epsilon'\sqrt{\Pi}$ on the weight of each set. This gives an algorithm for the case of symmetric weights since the weights of sets are, roughly speaking, guaranteed to be within the range $[0, \sqrt{\Pi}]$. The prize in a set of weight A is A^2, and the error in the prize can be bounded in terms of the error in the weight. Therefore, static granularization suffices for storing values of $\omega_s, \omega_t, \sigma$ throughout, and the final error is bounded.

For asymmetric weights, static granularization runs into trouble because the final solution may be the result of product of a very small w_s term and a very large w_t term. Therefore, the said upper bound on these sets do not hold, nor is the additive error of w_s term sufficient for getting a bound on the error of the product term. Even worse, it may be the case that portions of the solution require employing a small precision unit for w_s and a large precision unit for w_t, whereas other portions of the solution require employing a small precision unit for w_t and a large precision unit for w_s; i.e., there is no fixed precision unit that works throughout the algorithm for

w_s, w_t. This is why we utilize dynamic granularization to obtain multiplicative errors. In particular, $(1-\epsilon')A \cdot (1-\epsilon')B \geq (1-2\epsilon')AB$ no matter how small or large A, B are.

Since the next section explains the algorithm for the more general case of asymmetric weights, we do not give any more details for the symmetric case here.

10.5.3 Fixed prize from asymmetric arbitrary weights

Run the algorithm of Section 10.5.1 with the following modification. Use dynamic granularization (Theorem 7.2.1) with parameters $U = V(G), \epsilon = \epsilon'/6, w = w_s, w_t$ to store ω_s, ω_t, and static granularization (Theorem 7.1.1) with parameters $n = 4|V(G)|, \epsilon = 4\epsilon'/6, \tau = \Pi$ for storing σ.

As mentioned earlier, ω_s, ω_t represent weights of sets of vertices. Leaf nodes initialize them with empty sets. The sets are copied in (I3), (I4). In a forget node, (F2) and (F3) either copy the sets or add one element to them. Union of two sets is taken during (J2), (J3). Finally, in the computation of $\hat{\omega}_s, \hat{\omega}_t$, set union is performed.

Let $\mathsf{dyn\text{-}granular}(\hat{\omega}_s(P_i))$ and $\mathsf{dyn\text{-}granular}(\hat{\omega}_t(P_i))$ denote the recovered values from dynamic granularization of $\omega_s(P_i)$ and $\omega_t(P_i)$. Theorem 7.2.1 guarantees that the error in any of these computations is small. In particular, we have

$$\left(1-\frac{\epsilon'}{6}\right)\hat{\omega}_s(P_i) \leq \mathsf{dyn\text{-}granular}(\hat{\omega}_s(P_i)) \leq \hat{\omega}_s(P_i), \tag{10.7}$$

$$\left(1-\frac{\epsilon'}{6}\right)\hat{\omega}_t(P_i) \leq \mathsf{dyn\text{-}granular}(\hat{\omega}_t(P_i)) \leq \hat{\omega}_t(P_i). \tag{10.8}$$

Therefore, when in (F4) we have $\sigma = \sigma' + \mathsf{dyn\text{-}granular}(\hat{\omega}_s(P_1'))\mathsf{dyn\text{-}granular}(\hat{\omega}_t(P_1'))$, the added term may have an error of $(2\epsilon'/6)\hat{\omega}_s(P_1')\hat{\omega}_s(P_1')$.

As for σ, we have an initialization to zero in leaf nodes. Introduce nodes copy σ, whereas forget nodes either simply copy or add one term to σ. A join node adds two σ elements to each other.

Induction on the size of subtrees shows that the number of addition operations performed on the σ term of any DP entry for $i \in I$ is less than the number of nodes in the subtree rooted at i, hence the computation for the final σ in root node r requires at most $|I|$ addition operations.

We prove that $|I| \leq 4n$. There are at most $2n$ nodes with one children, i.e., at most one introduce and one forget node for each vertex. Besides, there are at most n nodes with no children. Subtracting one for the root that does not have any parents, we obtain that the sum of all degrees of I is at least $n + 2(2n) + 3(|I| - 3n) - 1 = 3|I| - 4n - 1 \leq 2(|I| - 1)$ where the last inequality follows from T being a tree. Rearranging we get $|I| \leq 4n - 1$.

Therefore, Theorem 7.1.1 ensures that the final value of σ for a root node DP entry has error at most $4\epsilon'/6\Pi = 2\epsilon'/3\Pi$ with respect to the number added to σ. Those values for themselves may have a multiplicative $(1 - \epsilon'/3)$ error. Since for any feasible solution the total real sum forming σ should be at least Π, the recovered value from dynamic granularization is no less than $(1-\epsilon'/3)\Pi$, and then the recovered value of the corresponding σ is at least $(1 - \epsilon'/3)\Pi - (2\epsilon'/3)\Pi = (1 - \epsilon')\Pi$. This means that, if the final step of the DP looks for a solution with $\sigma \geq (1 - \epsilon'\Pi)$, it will find a solution whose cost is no more than opt. All the errors in the static and dynamic granularizations are one-sided, i.e., if they claim the value is σ, the actual value cannot be less. Therefore, the solution picked by the DP indeed collects a prize of at least $(1 - \epsilon')\Pi$ giving the bicriteria guarantee.

It is worth emphasizing that the cost guarantee is optimal, and the only approximation is there for the collected prize. Even for the $(1 - \epsilon')$ approximation on the collected prize, we can push the value of ϵ' to be inverse polynomially small since the dependence in the running time is polynomial in ϵ'^{-1}.

10.5.4 Prize-collecting tradeoff with arbitrary symmetric weights

We first observe that in the case of small integer weights, we can run the fixed-prize algorithm for all possible values of collected prize, and find the best solution (after adding the uncollected prize as the penalty to the cost of the forest). Therefore, polynomially many calls to the fixed-prize algorithm suffices in this case. In the following we focus on the case of arbitrary nonnegative real weights (in the symmetric case).

In the prize-collecting setting, recall, we pay for the length of the forest, and for the prizes not collected (usually called penalties). If the total weight of all vertices is Δ, the prize not collected is Δ^2 minus the collected prize. One difficulty here is to determine the correct range for the collected prize so that we can use the algorithm of Section 10.5.3. The trivial range is $[0, \Delta^2]$. However, the rounding precision we pick for the penalties should also take into account the length of the forest. If the cost of the intended solution is much smaller than Δ^2, we cannot simply go with rounding errors similar to $\epsilon\Delta/n$. Otherwise, the error caused due to rounding the penalties will be too large compared to the solution value. The problem, in other words, stems from the fact that approximating the collected prize does not necessarily results in a good approximation of the uncollected prize. This becomes crucial if the uncollected prize is small, and so is the cost of the prize-collecting instance in which case the error in the penalty is important.

The trick is to find an estimate of the solution value, and then consider two cases depending on how the cost compares to the total penalty (of satisfied and unsatisfied demands). Using a 3-approximation algorithm, we obtain a solution of value C. We are guaranteed that $\mathsf{opt} \geq C/3$. If $\Delta^2 \leq C/3$, the optimal solution does not collect any prize at all; i.e., it pays the penalties for all demands without building any connectivity. Otherwise, assume $\Delta^2 > C/3$. To beat the solution of value C, we

should collect a prize of at least $\Delta^2 - C$.

We first consider the simpler case when $C/\Delta^2 > 1/n^2$. For an $\epsilon' > 0$ whose precise value will be fixed below, we use the algorithm of Section 10.5.3 to find a bicriteria $(1, 1-\epsilon')$-approximate solution for collecting a prize Π; this is done for any Π which is a multiple of $\epsilon'\Delta^2$ in range $[(1-\epsilon')\Delta^2 - C, \Delta^2]$. We select the best one after adding the uncollected prize to each of these solutions. Suppose the optimal solution opt collects a prize Π'. Let $\ell(\mathsf{opt}) = \mathsf{opt} - (\Delta^2 - \Pi')$ be the length of the forest. Round Π' down to the next multiple of $\epsilon'\Delta^2$, say Π. Starting with prize value Π, the algorithm finds a solution that collects a prize of at least $(1-\epsilon')\Pi$ with forest cost at most $\ell(\mathsf{opt})$.

Claim 5. *The total cost of this solution is at most* $(1+\epsilon)\mathsf{opt}$ *if* $\epsilon' = \frac{\epsilon}{6n^2}$.

Proof. The total cost of this solution is

$$\ell(\mathsf{opt}) + \left[\Delta^2 - (1-\epsilon')\Pi\right] \leq \ell(\mathsf{opt}) + \left[\Delta^2 - (1-\epsilon')(\Pi' - n^2\epsilon' C)\right]$$

since $\epsilon'\Delta^2 \leq n^2\epsilon' C$,

$$\begin{aligned}
&\leq \mathsf{opt} + \epsilon'\Pi + n^2(\epsilon' - \epsilon'^2)C \\
&= \mathsf{opt} + \epsilon'\frac{\Pi'}{\mathsf{opt}}\mathsf{opt} + n^2(\epsilon' - \epsilon'^2)C \\
&\leq \mathsf{opt} + \epsilon'\frac{\Delta^2}{\mathsf{opt}}\mathsf{opt} + 3n^2\epsilon'\mathsf{opt} \\
&\leq \mathsf{opt} + 6\epsilon' n^2 \mathsf{opt} && (10.9) \\
&\leq \mathsf{opt} + \epsilon\mathsf{opt} && (10.10) \\
&= (1+\epsilon)\mathsf{opt},
\end{aligned}$$

where (10.9) follows from $\frac{\Delta^2}{\mathsf{opt}} \leq \frac{n^2 C}{C/3} = 3n^2$, and (10.10) uses the definition of ϵ'. □

The other case, i.e., $C/\Delta^2 \leq 1/n^2$, is more challenging. Notice that, in order to

carry out the same procedure in this case, ϵ' may not be bounded by $1/\mathsf{poly}(n)$ and thus the running time may not be polynomial. The solution, however, has to collect almost all the prize. Thus, one of the connected components includes almost all the vertex weights. We set aside a subset $\mathcal{B}$ of vertices of large weight. The vertices of $\mathcal{B}$ have to be connected in the solution, or else the paid penalty will be too large. Then, dynamic programming proceeds by ignoring the effect of these vertices and only keeping tabs on how many vertices from $\mathcal{B}$ exist in each component. At the end, we only take into account the solutions that gather *all* the vertices of $\mathcal{B}$ in one component and compute the actual cost of those solutions and pick the best one. In the following, we provide the details of our method and prove its correctness.

Let $\mathcal{B}$ be the set of all vertices whose weight is larger than nC/Δ. Notice that $\mathcal{B}$ is nonempty since $C/\Delta^2 \leq 1/n^2$.

Lemma 10.5.1. *All the vertices of $\mathcal{B}$ are connected in the optimal solution.*

Proof. There are at most n components, so there is a component, say $\mathcal{C}$, whose total weight is not less than Δ/n. We claim all the vertices of $\mathcal{B}$ are inside this component. The penalty paid by the optimal solution is at most $C \leq \Delta^2/n$. If there is any vertex of $\mathcal{B}$ outside $\mathcal{C}$, the penalty of the solution is more than $\Delta/n \cdot nC/\Delta = C$, yielding a contradiction. □

Next, we round up all the weights to the next multiple of $\theta = \epsilon' C/\Delta$ for vertices not in $\mathcal{B}$. Define opt' as the optimal solution of the resulting instance. Let $\ell(\mathsf{opt})$ be the length of the forest in opt, and define $\ell(\mathsf{opt}')$ similarly. Let $\pi(\mathsf{opt})$ and $\pi(\mathsf{opt}')$ denote the penalty paid by opt and opt', respectively. Assume that $\epsilon' \leq 1$.

Lemma 10.5.2. $\pi(\mathsf{opt}') \leq \pi(\mathsf{opt}) + 12n\epsilon'\mathsf{opt}$.

Proof. We recompute the penalties paid by opt using the rounded weights. The pair (s, t) not connected in opt is either of the two kinds: (1) one of s and t is in $\mathcal{B}$; or (2) none of them is in $\mathcal{B}$. The total rounding error for the penalties of the first type is

bounded by $n\Delta\theta$. There are at most n^2 pairs of the second type. Since the weights of these terminals are at most nC/Δ, the error is not more than $n^2[2(nC/\Delta)\theta + \theta^2]$. Hence, the total error is at most

$$\begin{aligned} n^2[2(nC/\Delta)\theta + \theta^2] + n\Delta\theta &\leq n^2\left[(\epsilon'^2 + 2n\epsilon')\frac{C^2}{\Delta^2}\right] + n\epsilon' C \\ &\leq n^2\left[3n\epsilon'\frac{C^2}{\Delta^2}\right] + n\epsilon' C && \text{because } \epsilon' \leq 1 \\ &= \left(3n^2\frac{C}{\Delta^2} + 1\right) n\epsilon' C \\ &\leq 4n\epsilon' C && \text{because } \frac{C}{\Delta^2} \leq \frac{1}{n^2}, \end{aligned}$$

which is no more than $12n\epsilon'\mathsf{opt}$ as desired. □

Therefore, it suffices to solve the new instance. We modify the instance further. Divide all weights by θ and all edge lengths by θ^2. We denote this instance by $\mathcal{I}_1$. Any approximation for $\mathcal{I}_1$ immediately translates to the original one. In the new instance, small-weight vertices—those not in $\mathcal{B}$—have integer weights in $[0, \tau]$, where $\tau = n/\epsilon'$. (We assume without loss of generality that τ is an integer). The large-weight vertices all have weights at least τ, and the weight is not necessarily an integer. Let $b = |\mathcal{B}|$, and set the weight of the vertices of $\mathcal{B}$ to $2n^2\tau^2$. Let this final instance be called $\mathcal{I}_2$. We run the fixed-prize algorithm on $\mathcal{I}_2$ for specific values for Π, and then infer the optimum of $\mathcal{I}_1$ from them.

We proved that all vertices of $\mathcal{B}$ should be in one component of the solution in $\mathcal{I}_1$. Any such solution collects a prize of at least $4b^2n^4\tau^4$. On the other hand, any solution that collects a prize of at least $4b^2n^4\tau^4$ must gather all vertices of $\mathcal{B}$ in one component since, otherwise, it collects no more than $4(b-1)^2n^4\tau^4 + 2bn^3\tau^3 + n^2\tau^2 < 4b^2n^4\tau^4$. We only call the fixed-prize algorithm with parameters $\Pi \geq 4b^2n^4\tau^4$. Let the component containing $\mathcal{B}$ be called the "big" component.

Consider a solution that gathers a weight $bn^2\tau^2 + s$ in the big component. It

collects a prize of $4b^2n^4\tau^4 + s \cdot 2bn^2\tau^2 + s^2 + p$ where $s^2 \leq s^2 + p \leq n^2\tau^2$ (since total prize corresponding to small-weight vertices cannot be more than $(n\tau)^2$). This ensures that any solution with prize $4b^2n^4\tau^4 + s \cdot 2bn^2\tau^2 + s^2 + p$ necessarily gathers a weight $bn^2\tau^2 + s$ in its big component.

As all weights in $\mathcal{I}_2$ are polynomially small integers, we can run the fixed-prize algorithm for any prize value Π. In particular, we do this for $\Pi = 4b^2n^4\tau^4 + s \cdot 2bn^2\tau^2 + s^2 + p$ for any $s \in [0, n\tau]$ and $p \in [0, n^2\tau^2 - s]$. After finding the best forest cost from the fixed-prize algorithm, we can recover the actual prize collected by the solution: knowing s, p, we can cancel out the effect of prize corresponding to big-weight vertices, and add the actual prize for these demands from the correct weights of vertices in $\mathcal{B}$. Adding the penalties with forest cost, we obtain the cost of each such solution. Taking a minimum over all cases gives the optimum for $\mathcal{I}_1$.

Notice that the running time depends polynomially on ϵ', hence we give an FPTAS for MPCSF on bounded-treewidth instances.

10.5.5 Euclidean setting

We studied MPCSF and Π-MPCSF on Euclidean metrics in [BH10]. We obtain bicriteria $(1+\epsilon, 1-\epsilon')$-approximation algorithms for the fixed-prize case, and a PTAS for the prize-collecting case. The latter is not an efficient PTAS since the fixed-prize algorithm has an exponential running time in terms of ϵ. The algorithm works, as is usual for many optimization problems on Euclidean metrics (e.g., [BKM08, Aro98]), on a random quadtree decomposition of the plane. Portals are placed on the boundary of each quadtree square. For Euclidean Steiner Forest, each square should further be decomposed into a grid of "cells." For each portal on the boundary, one should also include some information about what cells are connected to it. The ideas for the prize-collecting case are more or less the same as those presented above. Another point of departure between the two algorithms is that for the Euclidean case we have

to satisfy certain initial conditions by possibly decomposing the input into different pieces. For the bounded-treewidth case this has already been performed during the reduction from bounded-genus graphs.

Chapter 11

Open Problems

We conclude this thesis by mentioning some open problems.

The technique used in Chapter 8 (based on Theorem 4.3.1) may be applicable to other problems such as k-MST and the Submodular Prize-collecting Steiner Tree problem introduced in [HST05].

Improving the approximation ratio for Prize-collecting Steiner Tree further than that in the proof of Theorem 3.2.1 is a challenging question. This has been already done for Prize-collecting TSP by Goemans [Goe09]. In light of the recent improvement for Steiner Tree due to [BGRS10], there is hope that better approximation algorithms for PCST may exist. It is worth noting that even now we do not know of any linear-programming relaxation with an integrality gap better than two (i.e., $2 - c$ for constant c) for PCST. In particular, if the integrality of the LP relaxation used by Byrka et al. is shown to match the approximation guarantee they obtain, their LP can be augmented to give an LP relaxation for PCST with an integrality gap bounded away from two. In fact, this approach would beat Theorem 3.2.1.

All the demands in the problems we discussed in this thesis are for single connectivity. In order to extend the results to higher-connectivity requirements, the prize-

collecting technique, the spanner framework, and the spanner construction all need to be generalized. For constant-connectivity demands (that allows an edge to be used multiple times), all the three pieces seem to work with appropriate modifications—the spanner construction requires nontrivial ideas similar to those in [BK08]. Nevertheless, we do not know how to do this for superconstant connectivity.

The prize-collecting clustering technique, in conjunction to the spanner machinery, may be useful in obtaining better approximation ratios (PTAS for most) for several other problems on planar graphs. The list of these algorithm includes, but is not limited to, Facility Location, k-Median, k-MST, and Group Steiner Tree. In particular, although we still do not know how to carry out the reduction to bounded-treewidth instances, solving those reduced instances is easy for Facility Location, k-Median, and k-MST.

Recall that for PCSF the reduction works (Theorem 3.2.7), but we do not hope to obtain a PTAS for the bounded-treewidth case (Theorem 3.2.6). However, the treewidth reduction approach can still be useful for obtaining constant-factor approximations for planar graphs better than the 2.54-approximation algorithm of [HJ06] for general graphs. We pose it as an open question whether this is indeed possible for PCSF.

Nothing in the prize-collecting clustering theorems (i.e., Theorems 4.2.1, 4.3.1) is restricted to planar graphs. The theorems work with any graph, although the first theorem has only been used, as of now, in the context of building spanners for planar graphs. A recent development [DHK11] extends the spanner framework to H-minor-free graphs, hence, an analog of the spanner construction of Chapter 6 immediately results in a PTAS for Steiner Forest, PCST, PCTSP, etc. on H-minor-free graphs. In addition, it will provide a reduction from H-minor-free graphs to bounded-treewidth graphs for SPCSF.

A much harder question to answer is whether any of the spanner machinery,

or the prize-collecting clustering technique, can be modified to work for directed graph metrics. Unfortunately, Traveling Salesman and Steiner Tree seem much harder on directed graphs: the best known approximation ratio for the former problem is $O(\log n / \log \log n)$, whereas the latter is known not to be approximable (unless $\mathsf{P} = \mathsf{NP}$) to within polylogarithmic factors. This also renders the use of the spanner framework, as is, impossible since those ideas require a constant-approximate solution to build the spanner on.

The issue of running time is important not only from a practical point of view, but also from a theoretical perspective. The focus of this thesis was on the theoretical side where, in particular, we are interested in improving the running time of the planar Steiner Forest algorithm of Chapter 9. Currently, the PTAS for the bounded-treewidth case is not efficient, as the exponent of the polynomial depends on ϵ. This stems from the way we construct the collection of partitions. A nice question is whether we can find a smaller number of partitions, so as to obtain an EPTAS.

Bibliography

[AABV95] Baruch Awerbuch, Yossi Azar, Avrim Blum, and Santosh Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *Proceedings of the* 27^{th} *Annual ACM Symposium on Theory of Computing*, pages 277–283. ACM, 1995.

[AB10] Aaron Archer and Anna Blasiak. Improved approximation algorithms for the minimum latency problem via prize-collecting strolls. In *Proceedings of the* 21^{st} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 429–447, 2010.

[ABHK09] Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. In *Proceedings of the* 50^{th} *Annual Symposium on Foundations of Computer Science*, pages 427–436, 2009.

[ABHK11] Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM Journal of Computing*, 40(2):309–332, 2011.

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowksi. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.

[AFZ06] Julián Aráoz, Elena Fernández, and Cristina Zoltan. Privatized rural postman problems. *Computers and Operations Research*, 33:3432–3449, December 2006.

[AGK+98] Sanjeev Arora, Michelangelo Grigni, David Karger, Philip N. Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proceedings of the* 9^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 33–41, 1998.

[AK00] Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1-2):123–134, 2000.

[AK06] Sanjeev Arora and George Karakostas. A $2 + \epsilon$ approximation algorithm for the k-MST problem. *Mathematical Programming*, 107(3):491–504, 2006.

[AKCF+04] Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings of the* 6^{th} *Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics*, pages 62–69, 2004.

[AKR91] Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. In *Proceedings of the* 23^{rd} *Annual ACM Symposium on Theory of Computing*, pages 134–144, New York, NY, United States, 1991. ACM.

[AKR95] Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal of Computing*, 24(3):440–456, 1995.

[AKR03] Eyal Amir, Robert Krauthgamer, and Satish Rao. Constant factor approximation of vertex-cuts in planar graphs. In *Proceedings of the* 35^{th} *Annual ACM Symposium on Theory of Computing*, pages 90–99. ACM, 2003.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160:781–793, 2004.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45:501–555, May 1998.

[ALW08] Aaron Archer, Asaf Levin, and David P. Williamson. A faster, better approximation algorithm for the minimum latency problem. *SIAM Journal of Computing*, 37(5):1472–1498, 2008.

[And73] Michael R. Anderberg. *Cluster analysis for applications*. Academic Press, Inc., New York, NY, United States, 1973.

[AR98] Sunil Arya and Hariharan Ramesh. A 2.5 factor approximation algorithm for the k-MST problem. *Information Processing Letters*, 65(3):117–118, 1998.

[Aro96] Sanjeev Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the* 37^{th} *Annual Symposium on Foundations of Computer Science*, page 2, Washington, DC, United States, 1996. IEEE Computer Society.

[Aro98] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45:70–122, January 1998.

[Bak94] Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

[Bal89] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[BBCM04] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows. In *Proceedings of the* 36^{th} *Annual ACM Symposium on Theory of Computing*, pages 166–174, New York, NY, United States, 2004. ACM.

[BC04] Daniel Boley and Dongwei Cao. Training support vector machine using adaptive clustering. In *Proceedings of the* 4^{th} *SIAM International Conference on Data Mining*, pages 126–137, New York, NY, United States, 2004. Society for Industrial and Applied Mathematics.

[BC10] MohammadHossein Bateni and Julia Chuzhoy. Approximation algorithms for the directed k-tour and k-stroll problems. In *Proceedings of the* 13^{th} *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 6302 of *Lecture Notes in Computer Science*, pages 25–38. Springer, 2010.

[BCC+10] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *Proceedings of the* 42^{nd} *Annual ACM Symposium on Theory of Computing*, pages 201–210, 2010.

[BCE+11] MohammadHossein Bateni, Chandra Chekuri, Alina Ene, Mohammad-Taghi Hajiaghayi, Nitish Korula, and Dániel Marx. Prize-collecting Steiner problems on planar graphs. In *Proceedings of the* 22^{nd} *Annual ACM-SIAM*

Symposium on Discrete Algorithms, pages 1028–1049, Philadelphia, PA, United States, 2011. Society for Industrial and Applied Mathematics.

[BCG09a] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. MaxMin allocation via degree lower-bounded arborescences. In *Proceedings of the* 41^{nd} *Annual ACM Symposium on Theory of Computing*, pages 543–552, 2009.

[BCG09b] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. New approximation algorithms for degree lower-bounded arborescences and Max-Min allocation. Technical Report TR-848-09, Princeton University, March 2009.

[BCK+07] Avrim Blum, Shuchi Chawla, David Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal of Computing*, 37(2):653–670, 2007. Preliminary version in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 46–55, 2003.

[BDT09] Glencora Borradaile, Erik D. Demaine, and Siamak Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded genus graphs. In *Proceedings of the* 26^{th} *International Symposium on Theoretical Aspects of Computer Science*, pages 171–182, New York, NY, United States, 2009. Springer.

[BGHK09] MohammadHossein Bateni, Lukasz Golab, MohammadTaghi Hajiaghayi, and Howard J. Karloff. Scheduling to minimize staleness and stretch in real-time data warehouses. In *Proceedings of the* 21^{st} *Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 29–38. ACM, 2009.

[BGHS09] MohammadHossein Bateni, Alexandre Gerber, MohammadTaghi Hajiaghayi, and Subhabrata Sen. Multi-VPN optimization for scalable routing via relaying. In *Proceedings of the* 28^{th} *IEEE International Conference on Computer Communications*, pages 2756–2760. IEEE Computer Society, 2009.

[BGHS10] MohammadHossein Bateni, Alexandre Gerber, MohammadTaghi Hajiaghayi, and Subhabrata Sen. Multi-VPN optimization for scalable routing via relaying. *IEEE/ACM Transactions on Networking*, 18(5):1544–1556, 2010.

[BGRS10] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved LP-based approximation for Steiner tree. In *Proceedings of the* 42^{nd} *Annual ACM Symposium on Theory of Computing*, pages 583–592, 2010.

[BGSLW93] Daniel Bienstock, Michel X. Goemans, David Simchi-Levi, and David P. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.

[BH09a] MohammadHossein Bateni and MohammadTaghi Hajiaghayi. Assignment problem in content distribution networks: unsplittable hard-capacitated facility location. In *Proceedings of the* 20^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–814, 2009.

[BH09b] MohammadHossein Bateni and MohammadTaghi Hajiaghayi. A note on the subadditive network design problem. *Operations Research Letters*, 37(5):339–344, 2009.

[BH10] MohammadHossein Bateni and MohammadTaghi Hajiaghayi. Euclidean prize-collecting Steiner forest. In *Proceedings of the* 9^{th} *Latin American Theoretical Informatics Symposium*, volume 6034 of *Lecture Notes in Computer Science*, pages 503–514. Springer, 2010.

[BHIM10] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Nicole Immorlica, and Hamid Mahini. The cooperative game theory foundations of network bargaining games. In *Proceedings of the* 37^{th} *International Colloquium on Automata, Languages and Programming*, volume 6198 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2010.

[BHJP11] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Sina Jafaropur, and Dan Pei. Towards an efficient algorithmic framework for pricing cellular data service. In *Proceedings of the* 30^{th} *IEEE International Conference on Computer Communications*. IEEE, 2011.

[BHKM11] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Philip N. Klein, and Claire Mathieu. A polynomial-time approximation scheme for planar multiway cut, 2011. manuscript.

[BHM10a] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. In *Proceedings of the* 42^{nd} *Annual ACM Symposium on Theory of Computing*, pages 211–220, 2010. To appear in SIAM Journal of Computing.

[BHM10b] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Prize-collecting network design on planar graphs. *CoRR*, abs/1006.4339, 2010.

[BHZ10] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Morteza Zadimoghaddam. Submodular secretary problem and extensions. In *Proceedings of the* 13^{th} *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 39–52, 2010.

[BK08] Glencora Borradaile and Philip N. Klein. The two-edge connectivity survivable network problem in planar graphs. In *Proceedings of the* 35^{th} *International Colloquium on Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 485–501. Springer, 2008.

[BKM08] Glencora Borradaile, Philip N. Klein, and Claire Mathieu. A polynomial-time approximation scheme for Euclidean Steiner forest. In *Proceedings of the* 49^{th} *Annual Symposium on Foundations of Computer Science*, pages 115–124, 2008.

[BKM09] Glencora Borradaile, Philip N. Klein, and Claire Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3), 2009.

[Bod93] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernica*, 11(1-2):1–22, 1993.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing*, 25(6):1305–1317, 1996.

[Bod98] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[Bon04] Paul Bonsma. Sparsest cuts and concurrent flows in product graphs. *Discrete Applied Mathematics*, 136(2-3):173–182, 2004.

[BR92] Piotr Berman and Viswanathan Ramaiyer. Improved approximations for the Steiner tree problem. In *Proceedings of the* 3^{rd} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 325–334, 1992.

[BRV96] Avrim Blum, R. Ravi, and Santosh Vempala. A constant-factor approximation algorithm for the k MST problem (extended abstract). In *Proceedings*

of the 28^{th} *Annual ACM Symposium on Theory of Computing*, pages 442–448, 1996.

[BS08] Surender Baswana and Sandeep Sen. Algorithms for spanners in weighted graphs. In *Encyclopedia of Algorithms*. Springer-Verlag, 2008.

[BS11] Prosenjit Bose and Michiel Smid. On plane geometric spanners: A survey and open problems, 2011.

[BT97] Hans L. Bodlaender and Dimitrios M. Thilikos. Treewidth for graphs with small chordality. *Discrete Applied Mathematics*, 79(1-3):45–61, 1997.

[CC02] Miroslav Chlebik and Janka Chlebikova. Approximation hardness of the Steiner tree problem on graphs. In *Proceedings of the* 8^{th} *Scandinavian Workshop on Algorithm Theory*, pages 170–179. Springer, 2002.

[CC07] Sergio Cabello and Erin W. Chambers. Multiple source shortest paths in a genus g graph. In *Proceedings of the* 18^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 89–97, Philadelphia, PA, United States, 2007. Society for Industrial and Applied Mathematics.

[CEK10] Chandra Chekuri, Alina Ene, and Nitish Korula. Prize-collecting Steiner tree and forest in planar graphs. *CoRR*, abs/1006.4357, 2010.

[CFN77] G. Cornuejols, M. L. Fisher, and G. Nemhauser. Location of bank accounts to optimize oat: An analytic study of exact and approximate algorithms. *Management Science*, 23:789810, 1977.

[CG05] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM Journal of Computing*, 34(4):803–824, 2005.

[CGRT03] Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, Trees, and Minimum Latency Tours. In *Proceedings of the* 44^{th} *Annual*

Symposium on Foundations of Computer Science, pages 36–45. IEEE Computer Society, 2003.

[Che86] Paul L. Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the* 2^{nd} *Annual Symposium on Computational Geometry*, pages 169–177, 1986.

[Chr76] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling-salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.

[CKP10a] Deeparnab Chakrabarty, Jochen Könemann, and David Pritchard. Hypergraphic LP relaxations for Steiner trees. In *Proceedings of the* 14^{th} *International Conference on Integer Programming and Combinatorial Optimization*, volume 6080 of *Lecture Notes in Computer Science*, pages 383–396. Springer, 2010.

[CKP10b] Deeparnab Chakrabarty, Jochen Könemann, and David Pritchard. Integrality gap of the hypergraphic relaxation of Steiner trees: A short proof of a 1.55 upper bound. *Operations Research Letters*, 38(6):567–570, 2010.

[CKPar] Chandra Chekuri, Nitish Korula, and Martin Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms*, to appear. Preliminary version in *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 661–670, 2008.

[CKR+03] Shuchi Chawla, D. Kitchin, Uday Rajan, R. Ravi, and Amitabh Sinha. Profit guaranteeing mechanisms for multicast networks. In *Proceedings of the* 4^{th} *ACM Conference on Electronic Commerce*, pages 190–191. ACM, 2003.

[CKS04] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Edge-disjoint paths in planar graphs. In *Proceedings of the* 45^{th} *Annual Symposium on Foundations of Computer Science*, pages 71–80, 2004.

[CKS05] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proceedings of the* 37^{th} *Annual ACM Symposium on Theory of Computing*, pages 183–192, New York, NY, United States, 2005. ACM.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3 edition, 2009.

[CM05] Sergio Cabello and Bojan Mohar. Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. In *Proceedings of the* 13^{th} *Annual European Symposium of Algorithms*, pages 131–142, New York, NY, United States, 2005. Springer.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the* 3^{rd} *Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[CRR01] S. A. Canuto, Mauricio G. C. Resende, and Celso C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38(1):50–58, 2001.

[CRW01] Fabián A. Chudak, Tim Roughgarden, and David P. Williamson. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. In *Proceedings of the* 8^{th} *International Conference on Integer Programming and Combinatorial Optimization*, pages 60–70, London, UK, 2001. Springer-Verlag.

[CRW04] Fabián A. Chudak, Tim Roughgarden, and David P. Williamson. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. *Mathematical Programming*, 100:411–421, 2004.

[dCLMR03] Alexandre Salles da Cunha, Abilio Lucena, Nelson Maculan, and Mauricio G. C. Resende. A relax and cut algorithm for the prize collecting Steiner problem in graphs. In *Proceedings of Mathematical Programming in Rio*, pages 72–78, 2003.

[DDO+04] Matt DeVos, Guoli Ding, Bogdan Oporowski, Daniel P. Sanders, Bruce A. Reed, Paul D. Seymour, and Dirk Vertigan. Excluding any graph as a minor allows a low tree-width 2-coloring. *Journal of Combinatorial Theory, Series B*, 91(1):25–41, 2004.

[DH08a] Erik D. Demaine and MohammadTaghi Hajiaghayi. Approximation schemes for planar graph problems (1983, 1984; Baker). In *Encyclopedia of Algorithms*, pages 59–62. Springer-Verlag, 2008.

[DH08b] Erik D. Demaine and MohammadTaghi Hajiaghayi. Bidimensionality (2004; Demaine, Fomin, Hajiaghayi, Thilikos). In *Encyclopedia of Algorithms*, pages 88–90. Springer-Verlag, 2008.

[DHK05] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *Proceedings of the* 46^{th} *Annual Symposium on Foundations of Computer Science*, pages 637–646, 2005.

[DHK11] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in H-minor-free graphs and algorithmic applications. In *Proceedings of the* 43^{rd} *Annual ACM Symposium on Theory of Computing*, page to appear, 2011.

[DHM07] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Bojan Mohar. Approximation algorithms via contraction decomposition. In *Proceedings of the*

18^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 278–287, 2007.

[Duf65] Richard J. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10:303–318, 1965.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Edm03] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi, editors, *Combinatorial optimization - Eureka, you shrink!*, pages 11–26. Springer-Verlag, New York, NY, United States, 2003.

[EGLV98] Stefan Engevall, Maud Göthe-Lundgren, and Peter Värbrand. A strong lower bound for the node weighted Steiner tree problem. *Networks*, 31(1):11–17, 1998.

[Elk08] Michael Elkin. Sparse graph spanners. In *Encyclopedia of Algorithms*. Springer-Verlag, 2008.

[EMV87] Ranel E. Erickson, Clyde L. Monma, , and Arthur F. Veinott. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12:634–664, 1987.

[Epp00] David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

[Eul41] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741.

[FFFdP10] Paulo Feofiloff, Cristina G. Fernandes, Carlos Eduardo Ferreira, and José Coelho de Pina. A note on Johnson, Minkoff and Phillips' algorithm for the prize-collecting Steiner tree problem. *arXiv:1004.1437v2 [cs.DS]*, 2010.

[FGL+96] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43:268–292, March 1996.

[FHL08] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal of Computing*, 38(2):629–657, 2008.

[Fle00] Lisa Fleischer. Recent progress in submodular function minimization. *OPTIMA: Mathematical Programming Society Newsletter*, 64:1–11, 2000.

[FMV07] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. In *Proceedings of the* 48^{th} *Annual Symposium on Foundations of Computer Science*, pages 461–471, 2007.

[FNW78] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions—ii. *Mathematical Programming Studies*, 8:7387, 1978.

[FP03] Eric J. Friedman and David C. Parkes. Pricing WiFi at Starbucks: issues in online mechanism design. In *Proceedings of the* 4^{th} *ACM Conference on Electronic Commerce*, pages 240–241. ACM, 2003.

[Fra93] András Frank. Applications of submodular functions. In *Surveys in Combinatorics*, pages 85–136. London Mathematical Society, 1993.

[Gar96] Naveen Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the* 37^{th} *Annual Symposium on Foundations of Computer Science*, pages 302–309, 1996.

[Gar05] Naveen Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *Proceedings of the* 37^{th} *Annual ACM Symposium on Theory of Computing*, pages 396–402, 2005.

[Gas10] Elisabeth Gassner. The Steiner forest problem revisited. *Journal of Discrete Algorithms*, 8(2):154–163, 2010.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GK11] Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3 – 20, 2011.

[GKP95] Michelangelo Grigni, Elias Koutsoupias, and Christos H. Papadimitriou. An approximation scheme for planar graph TSP. In *Proceedings of the* 36^{th} *Annual Symposium on Foundations of Computer Science*, page 640, Washington, DC, United States, 1995. IEEE Computer Society.

[GKT51] David Gale, Harold W. Kuhn, and Albert W. Tucker. Linear programming and the theory of games. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 317–329. Wiley, New York, NY, United States, 1951.

[GKTW09] Gagan Goel, Chinmay Karande, Pushkar Tripathi, and Lei Wang. Approximability of combinatorial problems with multi-agent submodular cost functions. In *Proceedings of the* 50^{th} *Annual Symposium on Foundations of Computer Science*, pages 755–764, 2009.

[GLS88] Martin Grötschel, Laszlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

[GNS08] Joachim Gudmundsson, Giri Narasimhan, and Michiel H. M. Smid. Geometric spanners. In *Encyclopedia of Algorithms*. Springer-Verlag, 2008.

[Goe09] Michel X. Goemans. Combining approximation algorithms for the prize-collecting TSP. *arXiv:0910.0553v1 [cs.DS]*, 2009.

[GP68] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[GP02] Harold N. Gabow and Seth Pettie. The dynamic vertex minimum problem and its application to clustering-type approximation algorithms. In *Proceedings of the* 8^{th} *Scandinavian Workshop on Algorithm Theory*, pages 190–199, 2002.

[GRST10] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *Proceedings of the* 6^{th} *International Workshop on Internet and Network Economics*, pages 246–257, 2010.

[GW92] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the* 3^{rd} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 307–316, 1992.

[GW95] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal of Computing*, 24:296–317, 1995.

[Haj05] MohammadTaghi Hajiaghayi. *The Bidimensionality Theory and Its Algorithmic Applications*. PhD thesis, Massachusetts Institute of Technology, June 2005.

[Hie73] Carl Hierholzer. über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechnung zu umfahren. *Mathematische Annalen*, 6:3032, 1873.

[HJ06] MohammadTaghi Hajiaghayi and Kamal Jain. The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema. In *Proceedings of the* 17^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 631–640, New York, NY, United States, 2006. ACM.

[HK70] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[HK71] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[HKKN10] MohammadTaghi Hajiaghayi, Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. Prize-collecting Steiner network problems. In *Proceedings of the* 14^{th} *International Conference on Integer Programming and Combinatorial Optimization*, volume 6080 of *Lecture Notes in Computer Science*, pages 71–84. Springer, 2010.

[Hoo91] J. A. Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10:291–295, July 1991.

[HP99] Stefan Hougardy and Hans Jürgen Prömel. A 1.598 approximation algorithm for the Steiner problem in graphs. In *Proceedings of the* 10^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–453, Philadelphia, PA, United States, 1999. Society for Industrial and Applied Mathematics.

[HST05] Ara Hayrapetyan, Chaitanya Swamy, and Éva Tardos. Network design for information networks. In *Proceedings of the* 16^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 933–942, 2005.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[JF96] Anil K. Jain and Patrick J. Flynn. Image segmentation using clustering. In N. Ahuja and K. Bowyer, editors, *Advances in Image Understanding: A Festschrift for Azriel Rosenfeld*, pages 65–83. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[JMP00] David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting Steiner tree problem: theory and practice. In *Proceedings of the* 11^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000.

[JP95] Michael Jünger and William R. Pulleyblank. New primal and dual matching heuristics. *Algorithmica*, 13(4):357–386, 1995.

[JV01] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[Kar72] Richard Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the* 16^{th} *Annual ACM Symposium on Theory of Computing*, pages 302–311, New York, NY, United States, 1984. ACM.

[KG92] J. Keil and Carl Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992.

[Kha79] Leonid G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR*, 244:1093–1096, 1979. translated in Soviet Mathematics Doklady 20:1 (1979), p. 191-194.

[Kle06] Philip N. Klein. A subset spanner for planar graphs, with application to subset TSP. In *Proceedings of the* 38^{th} *Annual ACM Symposium on Theory of Computing*, pages 749–756, 2006.

[Kle08] Philip N. Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal of Computing*, 37(6):1926–1952, 2008.

[Klo96] Ton Kloks. Treewidth of circle graphs. *International Journal of Foundations of Computer Science*, 7(2):111–120, 1996.

[KPT11] Jochen Könemann, David Pritchard, and Kunlun Tan. A partition-based relaxation for Steiner trees. *Mathematical Programming*, 127(2):345–370, 2011.

[KR91] Arkady Kanevsky and Vijaya Ramachandran. Improved algorithms for graph four-connectivity. *Journal of Computer and System Sciences*, 42(3):288–306, 1991. Preliminary version in Twenty-Eighth IEEE Symposium on Foundations of Computer Science (1987).

[KS02] Petr Kolman and Christian Scheideler. Improved bounds for the unsplittable flow problem. In *Proceedings of the* 13^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 184–193, Philadelphia, PA, United States, 2002. Society for Industrial and Applied Mathematics.

[KT11] Ken-ichi Kawarabayashi and Mikkel Thorup. Minimum k-way cut of bounded size is fixed-parameter tractable. *CoRR*, abs/1101.4689, 2011.

[Kur30] Casimir Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[KZ97] Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the Steiner tree problems. *Journal of Combinatorial Optimization*, 1(1):47–65, 1997.

[LCFX07] Bin Li, Mingmin Chi, Jianping Fan, and Xiangyang Xue. Support cluster machine. In *Proceedings of the* 24^{th} *International Conference on Machine Learning*, pages 505–512, New York, NY, United States, 2007. ACM.

[Lev73] Leonid Levin. Universal search problems. *Problems of Information Transmission*, 9:265–266, 1973. translated into English by Trakhtenbrot, B. A. (1984).

[LLP96] Youngho Lee, Byung Ha Lim, and June S. Park. A hub location problem in designing digital data service networks: Lagrangian relaxation approach. *Location Science*, 4(3):185 – 194, 1996.

[LMNS10] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM Journal on Discrete Mathematics*, 23(4):2053–2078, 2010.

[LR99] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[LSV09] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. In *Proceedings of the* 12^{th} *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 244–257, 2009.

[LT79] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[LT80] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM Journal of Computing*, 9(3):615–627, 1980.

[LW67] G. N. Lance and W. T. Williams. A general theory of classificatory sorting strategies 1. Hierarchical systems. *Computer Journal*, 9:373–380, 1967.

[LWP+06] Ivana Ljubić, René Weiskircher, Ulrich Pferschy, Gunnar W. Klau, Petra Mutzel, and Matteo Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming*, 105:427–449, 2006.

[Mar07] Dániel Marx. Precoloring extension on chordal graphs. In A. Bondy, J. Fonlupt, J.-L. Fouquet, J.-C. Fournier, and J.L. Ramirez Alfonsin, editors, *Graph Theory in Paris. Proceedings of a Conference in Memory of Claude Berge*, Trends in Mathematics, pages 255–270. Birkhäuser, Basel, Switzerland, 2007.

[Mcc05] S. Thomas Mccormick. Submodular function minimization. In *Handbook on Discrete Optimization*, pages 321–391. Elsevier, 2005.

[Mit99] Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal of Computing*, 28(4):1298–1309, 1999.

[MSL08] Ritesh Madan, Devavrat Shah, and Olivier Leveque. Product multicommodity flow in wireless networks. *IEEE Transactions on Information Theory*, 54(4):1460–1476, 2008.

[MSST07] Nina Mishra, Robert Schreiber, Isabelle Stanton, and Robert Endre Tarjan. Clustering social networks. In *Proceedings of the* 5^{th} *International Workshop on Algorithms and Models for the Web-Graph*, pages 56–67. Springer, 2007.

[MT01] Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*. Johns Hopkins University Press, Baltimore, MD, United States, 2001.

[N47] John von Neumann. On a maximization problem. Manuscript, Institute for Advanced Studies, Princeton University, Princeton, NJ 08544, USA, 1947.

[NR07] Viswanath Nagarajan and R. Ravi. Poly-logarithmic approximation algorithms for directed vehicle routing problems. In *Proceedings of the* 10^{th} *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 257–270, 2007.

[NSW08] Chandrashekhar Nagarajan, Yogeshwer Sharma, and David P. Williamson. Approximation algorithms for prize-collecting network design problems with general connectivity requirements. In *Proceedings of the* 6^{th} *International Workshop on Approximation and Online Algorithms*, pages 174–187, 2008.

[NW78] George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3:177–188, 1978.

[NWF78] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14:265–294, 1978.

[OGS11] Shayan Oveis Gharan and Amin Saberi. Asymmetric traveling salesman problem on graphs with bounded genus. In *Proceedings of the* 22^{nd} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 967–975, 2011.

[PS89] David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13, 1989.

[PS97] Hans Jürgen Prömel and Angelika Steger. RNC-approximation algorithms for the Steiner problem. In *Proceedings of the* 14^{th} *Annual Symposium on Theoretical Aspects of Computer Science*, pages 559–570, 1997.

[Ras92] Edie M. Rasmussen. Clustering algorithms. In *Information Retrieval: Data Structures & Algorithms*, pages 419–442. Prentice-Hall, 1992.

[RP86] M. B. Richey and R. Gary Parker. On multiple Steiner subgraph problems. *Networks*, 16(4):423–438, 1986.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[RS94] Neil Robertson and Paul D. Seymour. Graph minors. XI. circuits on a surface. *Journal of Combinatorial Theory, Series B*, 60(1):72–106, 1994.

[RSM+94] R. Ravi, Ravi Sundaram, Madhav V. Marathe, Daniel J. Rosenkrantz, and S. S. Ravi. Spanning trees—short or small. In *Proceedings of the* 5^{rd} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 546–555, 1994.

[RV95] Sridhar Rajagopalan and Vijay V. Vazirani. Logarithmic approximation of minimum weight k trees. Unpublished Manuscript, 1995.

[RZ00] Gabriel Robins and Alexander Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the* 11^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779, 2000.

[RZ05] Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19(1):122–134, 2005.

[Sal91] Gerard Salton. Developments in automatic text retrieval. *Science*, 253(5023):974–980, August 1991.

[SC08] Karsten Steinhaeuser and Nitesh V. Chawla. Community detection in a large real-world social network. In Huan Liu, John J. Salerno, and Michael J. Young, editors, *Social Computing, Behavioral Modeling, and Prediction*, pages 168–175. Springer US, 2008.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the* 10^{th} *Annual ACM Symposium on Theory of Computing*, pages 216–226, New York, NY, United States, 1978. ACM.

[Sch86] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, United States, 1986.

[Sch03] Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency.* Springer, 2003.

[SCRS00] F. Sibel Salman, Joseph Cheriyan, R. Ravi, and S. Subramanian. Approximating the single-sink link-installation problem in network design. *SIAM Journal on Optimization*, 11(3):595–610, 2000.

[SSW07] Yogeshwer Sharma, Chaitanya Swamy, and David P. Williamson. Approximation algorithms for prize collecting forest problems with submodular penalty functions. In *Proceedings of the* 18^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1275–1284, 2007.

[Thi03] Martin Thimm. On the approximability of the Steiner tree problem. *Theoretical Computer Science*, 295(1-3):387–402, 2003.

[Von07] Jan Vondrák. *Submodularity in Combinatorial Optimization.* PhD thesis, Charles University, 2007.

[Wag37] Klaus W. Wagner. über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114:570–590, 1937.

[Wes00] Douglas B. West. *Introduction to Graph Theory (2nd Edition)*. Prentice-Hall, August 2000.

[Win87] Pawel Winter. Steiner problem in networks: a survey. *Networks*, 17:129–167, April 1987.

[YLZ06] Jinhui Yuan, Jianmin Li, and Bo Zhang. Learning concepts from large scale imbalanced data sets using support cluster machines. In *Proceedings of the* 14^{th} *Annual ACM International Conference on Multimedia*, pages 441–450, New York, NY, United States, 2006. ACM.

[YYH03] Hwanjo Yu, Jiong Yang, and Jiawei Han. Classifying large data sets using SVMs with hierarchical clusters. In *Proceedings of the* 9^{th} *International Conference on Knowledge Discovery and Data Mining*, pages 306–315, New York, NY, United States, 2003. ACM.

[Zel92] Alexander Zelikovsky. An 11/6-approximation for the Steiner problem on graphs. In *Proceedings of the* 4^{th} *Czechoslovakian Symposium on Combinatorics, Graphs, and Complexity (1990) published in Annals of Discrete Mathematics*, volume 51, pages 351–354, 1992.

[Zel93] Alexander Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9(5):463–470, 1993.

[Zel96] Alexander Zelikovsky. Better approximation bounds for the network and Euclidean Steiner tree problems. Technical report, University of Virginia, Charlottesville, VA, USA, 1996.