COST SHARING AND APPROXIMATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Martin Pál

January 2005

COST SHARING AND APPROXIMATION

Martin Pál, Ph.D.

Cornell University 2005

Central to this thesis are problems in which a group of users can benefit from building and jointly using some kind of infrastructure, be it a set of supply depots, service stations, or a communication or transportation network. We study two important questions related to these kinds of scenarios: (1) how to build the shared facility that satisfies the needs of a given set of users in a cost-effective way, and (2) how to split the cost of the shared facility among the participating users in a fair and reasonable way.

In the first part of the thesis, we seek to design cost sharing functions with desirable game-theoretic properties. We are looking for cost sharing functions that are fair, and encourage cooperation among users. This is captured in the notion of cross-monotonicity: it says that a cost share of any user should never increase as more people join the system, and never decrease when players leave. Towards this end, we develop a new technique to generate such cross-monotonic cost shares using a primal-dual type process, and use it to design cross-monotonic cost shares for several NP-hard optimization problems.

In the second part we proceed in a slightly different direction, applying cost sharing to the design of approximation algorithms. We consider a class of two-stage stochastic problems with recourse. In these problems, we can buy building blocks (edges, facilities, vertices. . .) in two stages. In the first stage, the elements are

relatively inexpensive, but we do not know the requirements of users we will have to serve (we only have a probabilistic forecast of their demands). In the second stage, the actual demands are revealed, and we must buy enough elements (now at a higher price) to satisfy all user demands.

We show that whenever the underlying deterministic problem admits a certain type of cost sharing, an extremely simple strategy gives us good approximation guarantees: in the first stage, take several samples from the forecasted distribution, and build a solution that covers all the sampled clients at the low price. When the real users materialize, augment the first stage solution to cover the actual demands. In this way we obtain constant approximation algorithms for stochastic versions of problems like Uncapacitated Facility Location, Steiner tree, Steiner Forest or Vertex Cover.

BIOGRAPHICAL SKETCH

Martin Pál was born on the 4th of January, 1977 in Brezno, in a wonderful country once called Czechoslovakia. The country managed to get rid of the communist rule in 1989, but got split into two by two unscrupulous guys who both wanted to be prime ministers. As a result, in 1993 Martin became a citizen of Slovakia.

Martin graduated form high school named after a Slovak mathematician Jur Hronec in 1995 and received a “magister” degree from a university named after the “Great Didactic” John Amos Comenius in June 2000. He hopes to receive a PhD from Cornell University, named after an American telegraph-builder Ezra Cornell, in January 2005.

In his previous life he used to do a lot of competitive fencing, and everyday commute to college got him addicted to biking along the beautiful banks of the Danube river. Besides biking, he spent most of his free time as a volunteer running the Correspondence Seminar in Programming, a mail contest for high school kids. He got married recently and did not yet have a single opportunity to regret this move.

ACKNOWLEDGEMENTS

Thanks. It sounds that I might have made it at last. Thanks to everybody who has helped me to pull through all these years. Thanks to my advisor, Éva Tardos,for being with me through all my whinnings and complaints and excuses and for all the nudges necessary to get me all the way to this point. And, well, thanks for all the advice without which I would be hopelessly lost. Thanks to David Shmoys, who has shown me what it takes to be a great teacher; his scheduling course may well be the best class that I've ever taken. Thanks to Johannes Gehrke and Jon Kleinberg for serving on my committee; I only wish I had used more opportunities to benefit from their guidance.

Thanks to my housemates Mark Sandler, Alex Slivkins and Ivan Lysiuk for stimulating after-midnight discussions on a number of scientific an non-scientific topics. It was great to be a part of a purely theoretical office with Tom Wexler, Ara Hayrapetyan and Zoya Svitkina – I will miss you guys.

I am glad to have met a number of wonderful people at conferences and other meetings; their example keeps convincing me that a career as a computer scientist can be more interesting than that of a shepherd or a bus driver after all. I am much indebted to my co-authors and collaborators; fruits of collaboration with Luca Becchetti, Anupam Gupta, Jochen Könemann, Amit Kumar, Stefano Leonardi, R. Ravi, Tim Roughgarden, Amitabh Sinha and Éva Tardos are the substance of this thesis. Special thanks to Anupam, who has shown me that long-distance collaboration can not only be successful and productive, but also a lot of fun.

Finally, thanks to Lucia, who had the patience to wait for me for four long years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

This thesis is about the interplay of two areas: combinatorial optimization and cost sharing, the art of dividing up the costs associated with building and maintaining a shared property among the group of agents who benefit from it. Our prototypical example would be a computer or communication network, built to satisfy the needs of a number of users. The first question we need to address is: how to design a good network, one that satisfies all requirements, while minimizing the overall cost? What is the cost of the optimal network? Once the cost is determined, the second question arises: how to distribute this cost among all users that participate in the network in a fair way?

1.1 Cost Sharing

On an abstract level, a cost sharing problem is specified by a set (universe) of users (called also players, agents or clients interchangeably) U and a *cost function* $c : 2^U \mapsto \mathbb{R}^+$. The cost function specifies, for any given set $S \subseteq U$ of users how much would it cost to build a solution that satisfies all users in S.

Example 1.1 (building a bridge) Suppose each user $i \in U$ has a need to regularly cross a river. To satisfy this need, a bridge has to be built, at cost M. Once the bridge has been built, any number of users can use it at no additional cost. Of course, if no users are served, there is no point in spending money on building the bridge. This would ssugest a cost function of the form

$$c(S) = \begin{cases} 0 & \text{if } S = \emptyset \\ M & \text{if } S \neq \emptyset. \end{cases}$$

■

Instead specifying the cost function explicitly as in Example 1.1, we will usually have to deal with cost functions defined implicitly, as the solution to a combinatorial optimization problem.

Example 1.2 (spanning tree) Suppose our users want to receive periodical updates from a news provider. The provider is located at a source node r, and our goal is to build a network that connects all users to the source node. A reasonable idea would be to build a broadcast tree rooted at the source node r that spans all the users. Assuming that distances among clients can be modeled by an undirected weighted graph, a minimum spanning tree on the clients and the source node would be a good candidate for our broadcast network. Hence the resulting cost function would be given by

$$c(S) = \mathsf{Cost}(\mathrm{MST}(S \cup \{r\})),$$

where $\mathrm{MST}(S \cup \{s\})$ denotes a minimum cost spanning tree on the vertices $S \cup \{s\}$. ■

Throughout this thesis, we will consider a number of combinatorial optimization problems Π. Like in Example 1.2, each problem will be specified by a universe of *users* U (called also clients, customers, or players interchangeably), each user having a requirement she wants to be satisfied. For a given subset $S \subseteq U$ of users, the problem $\Pi(S)$ is to find a solution Sol that satisfies the requirements of every user in S and has minimum cost. The minimum cost solutions to the optimization problem naturally define a cost function:

$$c(S) = \mathsf{Cost}(\Pi(S)).$$

The goal of cost sharing is to devise a scheme that specifies how to divide the cost (or profit, for profit sharing schemes) among the agents participating in a given enterprise.

A *cost allocation* is simply a vector $x = \{x_i\}_{i \in U}$ of non-negative real numbers with one coordinate for every user $i \in U$. The value x_i specifies how much the user i is supposed to contribute towards the common goal.[1]

Studying cost allocations for a fixed set of users has its own merits. In this thesis, however, we will be more interested in relationships among cost allocations for different sets of users, and changes in the cost allocation as the underlying set of clients changes. Towards this end, we define a cost sharing function that can be thought of as a collection of cost allocations for all possible sets of users.

Definition 1.3 A *cost sharing function* (sometimes called also a cost sharing method) is any function $\xi : 2^U \times U \mapsto \mathbb{R}^+$.

For a given set of users S, the value $\xi(S, i)$ specifies the cost share of user i, assuming that S is the set of users being served. By convention, if $i \notin S$, we define $\xi(S, i)$ to be 0. Thus, if we fix a set of clients S, then $\xi(S, \cdot)$ (more precisely, the vector $\{\xi(S, i)\}_{i \in S}$) is a cost allocation for the client set S.

[1]Some works, most notably game theory textbooks, require any cost allocation to satisfy $\sum_{i \in U} x_i = c(U)$. For our purposes it will be more convenient to defer this requirement to a separate definition (Def. 1.6) and allow cost allocations to be arbitrary non-negative vectors.

Remark 1.4 Definition 1.3 allows some rather uninteresting cost sharing functions. For example, an all-zero function is a valid cost sharing function according to this definition. We shall impose restrictions on cost sharing functions (Definitions 1.6 and 1.7) that relate the cost sharing function to the actual cost, as well as two additional properties studied in depth in subsequent chapters: cross-monotonicity (Def. 3.1) and strictness (Defs. 4.2 and 4.14)

Example 1.5 (Cost sharing for MST) One possible cost sharing function for the MINIMUM SPANNING TREE problem from Example 1.2 is the following. For any set $S \subseteq U$ of clients, let T_S be the minimum cost tree spanning the vertices $S \cup \{r\}$ (in case there are multiple minimum spanning trees for $S \cup \{r\}$, pick T_S according to some consistent tie-breaking rule, e.g. lexicographically smallest). For each vertex (user) $i \in S$ (that is, any vertex except the root) there is a unique first edge $e(i)$ on the path from i to r (in other words, e is the edge between i and its parent). We can define a cost sharing function simply by demanding that every user pays the cost of its parent edge: $\xi(S, i) = c(e(i))$. ■

The cost sharing function from Example 1.5, although extremely simple, will be useful in Section 4.4 to derive an approximation algorithm for the stochastic Steiner tree problem. The details on how to do this appear in Chapter 4. For now, let us point out one desirable (and almost obvious) property of our cost sharing function: the cost shares should pay for the cost of the solution we construct.

Definition 1.6 A cost allocation x is *budget balanced*, if $\sum_{i \in U} x_i = c(U)$. A cost sharing function $\xi : 2^U \times U \mapsto R^+$ is *budget balanced* (with respect to a cost function $c(\cdot)$), if for each set $S \subseteq U$, the cost allocation $\xi(S, \cdot)$ is budget balanced, that is, if $\sum_{i \in S} \xi(S, i) = c(S)$ for all $S \subseteq U$.

It is easy to check that the cost sharing function ξ from Example 1.5 is budget balanced. Throughout this thesis, we shall strive to design budget balanced cost sharing functions; however in many cases this turns out to be an unattainable goal. Instead, we may have to settle for a somewhat lesser goal: the sum of cost shares should come within some multiplicative factor of the overall cost. In the following definition, $\alpha \geq 1$ is a real number.

Definition 1.7 A cost allocation x is α-*approximately budget balanced*, if

$$\sum_{i \in U} x_i \leq c(U) \leq \alpha \sum_{i \in U} x_i.$$

A cost sharing function $\xi : 2^U \times U \mapsto R^+$ is α-*approximately budget balanced* (with respect to a cost function $c(\cdot)$), if the cost allocation $\xi(S, \cdot)$ is α-approximately budget balanced, i.e. if

$$\sum_{i \in S} \xi(S, i) \leq c(S) \leq \alpha \sum_{i \in S} \xi(S, i)$$

for all $S \subseteq U$.

It will often be convenient to separate budget balance into two independent properties: the first part is called competitiveness, while the later (approximate) cost recovery.

Definition 1.8 A cost sharing function $\xi : 2^U \times U \mapsto R^+$ is *competitive* (with respect to a cost function $c(\cdot)$), if $\sum_{i \in S} \xi(S, i) \leq c(S)$, and α-*cost recovering* (or satisfying α-*cost recovery*) if $c(S) \leq \alpha \sum_{i \in S} \xi(S, i)$ for all $S \subseteq U$. Analogous definition holds for cost allocations.

The cost function from Example 1.2 can be computed in polynomial time: indeed, there are many efficient polynomial time algorithms for computing a minimum cost spanning tree (see, e.g. [26]). Unfortunately for most problems of our interest, computing the cost function is NP-hard: for example, if we replace minimum cost spanning tree by a *Steiner* tree, we already arrive at an NP-hard optimization problem. This is bad news in two ways. From a mathematical standpoint, a function that is NP-hard to compute is often very hard to understand and usually lacks any succint mathematical description. It also means that in general we cannot hope to discover structural properties of the cost functions that could aid us in the design of the "right" cost sharing functions.

From the standpoint of a computer scientist, NP hardness means that something has to be sacrificed: either we give up polynomial running time, or live with algorithms that do not always produce the optimal answer. Our choice is to go with approximation algorithms: they are an important paradigm by itself, and the linear programming tools they use (especially primal-dual algorithms) are particularly useful in designing good cost shares.

The following chapter, Chapter 2, serves as a quick overview of some established primal-dual approximation algorithms and techniques that we will be building upon in subsequent chapters.

In Chapter 3, we develop methods to compute cost sharing functions guaranteeing that the cost share of any individual user never increases as the pool of served users grows. This property, called *cross-monotonicity*, has useful game theoretic implications: the existence of such cost sharing suggests that the available resources can be effectively shared, as there is no competition for a scarce resource. New players are welcome to join, as their arrival can only help old users utilize their service at a lower price. Cross-monotonic cost sharing is also the main ingredient in the construction of some truthful mechanisms.

Chapter 4 defines *strict* cost shares. Informally, strictness states that if we want to extend an old network to accomodate the needs of a group of newly arrived users, the cost shares of the new users cover the expenses of this extension. We shall see that strict cost shares are a useful tool for designing approximation algorithms. We study a class of two-stage stochastic optimization problems and show that whenever the underlying deterministic problem admits a certain cost sharing scheme, we can lift an approximation algorithm for the deterministic version and use it to obtain an approximation algorithm for the stochastic problem.

Finally, in Chapter 5 we use construct strict[2] cost sharing for the Steiner Forest problem, which enables us to give an approximation algorithm for the MULTICOMMODITY RENT OR BUY PROBLEM, a network design problem with economy of scale.

[2]Technically, we obtain only a weaker form of strictness, that is still sufficient for our purposes.

Chapter 2

Primal-dual algorithms and cost sharing

Nearly all cost sharing functions we know how to construct come from some kind of dual-growing process. In this chapter, we would like to mention two of them. The minimum spanning tree algorithm of Kruskal [64] is usually not viewed as a primal-dual algorithm; however, analyzing it in this way leads us to a different way of defining cost sharing functions for trees (and forests). The second process is part of a simple 2-approximation algorithm for the Vertex Cover problem. A third dual-growing algorithm should perhaps be mentioned here: the 3-approximation algorithm of Jain and Vazirani [52] for Metric Facility Location. After some deliberation, we decided to leave its discussion to Chapter 3, where we make heavy use of it.

There is a linear program hidden behind every primal-dual algorithm. While sometimes it is more convenient to work with this linear program directly, more often than not we try to define our algorithms and work with them without an explicit appeal to the underlying linear program.

2.1 Minimum spanning trees

Given an undirected graph $G = (V, E)$ with non-negative weights on edges, the Minimum Spanning Tree problem is to select a set of edges of minimum cost so that there is a path between any pair of vertices using only the selected edges. There is a number of efficient algorithms for finding minimum spanning trees, both sequential [22, 79, 64] and distributed [7]. Here we focus on the well-known algorithm of Kruskal [64], and interpret it as a primal-dual algorithm. In the following, we shall always assume that the graph G is connected.

Kruskal's algorithm sorts all edges of the graph in the order of increasing length, and considers them in this order. It maintains a list of selected edges that is initially empty. For each edge $e = (uv)$ in order it checks whether there is a path between vertices u and v consisting of only selected edges. If yes, edge e is discarded; if not, e is added to the list of selected edges. In both cases, the algorithm proceeds with the next edge in the ordering. The algorithm stops when it exhausts all edges.

It is a well known fact that Kruskal's algorithm is a valid MST algorithm.

Theorem 2.1 (see, e.g.[26]) *Assuming the graph G is connected, the list of selected edges forms a minimum spanning tree of G.*

Let $e_1, e_2, \ldots, e_{n-1}$ be the list of edges output by Kruskal's algorithm ($n = |V|$ is the number of vertices of the graph). An obvious way to compute the cost of the MST is to add up the cost of individual edges:

$$\mathsf{Cost}(\mathrm{MST}) = \sum_{i=1}^{n-1} c(e_i). \tag{2.1}$$

Let us suggest a different way of computing the cost. We can think of all the edges of G as being laid on a real line, with each edge e covering the interval $[0, c(e)]$. Imagine a pointer t moving continuously from zero to plus infinity: whenever the pointer hits the rightmost point of an edge e (which happens at time $t = c(e)$) the edge is processed by the algorithm (some sort of tie-breaking has to be employed for edges of equal length). The contribution of each edge is equal to the length of the interval it covers (some intervals will be covered multiple times). To express the cost of the MST, we can divide the real line into smaller sub-intervals, and add up the lengths of the sub-intervals, counting each sub-interval with multiplicity equal to the number of times it is covered.

To find out how many times is a given point $t \geq 0$ on the real line covered by selected edges, let us look at the state of the algorithm after processing all edges with $c(e) < t$. Let G_t be a subgraph of G consisting of all edges of length less than t. Note that two vertices u, v belong to the same component of G_t if and only if they are connected by a path of edges already selected by the algorithm. Since addition of each selected edge reduces the number of components by one, and in the end there will be only one large component (i.e. $G = G_\infty$), it follows that at time t, the number of edges that Kruskal's algorithm will yet select (which is equal to the number of edges that cover t) is one less than the number of components of G_t. If we use $\kappa(G_t)$ to denote the number of components of G_t, we can write

$$\mathsf{Cost}(\mathrm{MST}) = \int_0^\infty \kappa(G_t) - 1 \,\mathrm{d}t. \tag{2.2}$$

Example 2.2 (Cost sharing for MST II) Equation 2.2 allows us to attribute the cost of the MST differently, allowing us to define an alternative to the cost sharing function from Example 1.5. This alternative cost sharing function will satisfy *cross-monotonicity*, a property useful for designing strategyproof cost-sharing mechanisms as we shall see in Chapter 3.

Equation 2.2 says that to sustain the growth of the spanning tree, we have to pour in $\kappa(G_t) - 1$ "money" per time unit at any given time t. In other words, we have to pay one unit of money per time step for each component of G_t except one. Bearing with the setup of Example 1.5, it is clear that the exempt component should be the component containing the root vertex: the root itself is not a player and hence should not be charged. It would also not be fair to charge users who are already connected for the growth of the rest of the tree. For each of the non-root components, we shall split the cost of growth equally among the users (vertices) in that component.[1]

To compute the cost share of a particular user, let $C_{j,t}$ denote the component that user j belongs to at time t. If we define

$$a_j(t) = \begin{cases} \frac{1}{|C_{j,t}|} & \text{if } s \notin C_{j,t} \\ 0 & \text{if } s \in C_{j,t}, \end{cases} , \tag{2.3}$$

we can write the cost share of the user j as

$$\xi(S,i) = \int_0^\infty a_j t \, \mathrm{d}t. \tag{2.4}$$

One way to think about (2.3) is that the root vertex s has infinite weight, and hence $|C_{j,t}| = \infty$ if $s \in C_{j,t}$.

By the preceeding discussion, it is clear that the cost sharing function ξ recovers the exact cost of the MST and hence is budget-balanced. Moreover, ξ satisfies Definition 3.1 which will prove useful in design of strategyproof mechanisms in Chapter 3.

2.2 Steiner trees

A spanning tree of a set $S \subseteq V$ of vertices is a tree that spans *exactly* the vertices in S; whereas a *Steiner tree* of the set S is a tree that spans all vertices of S, but is allowed to pass through other intermediate vertices (called also Steiner vertices). The Minimum Steiner Tree problem is then, given an undirected graph G with weights on the edges, and a set S of vertices, to find a Steiner tree of S that has minimum cost.

[1]There is no particular reason (except a desire for symmetry) for equal split. In fact, Jain and Vazirani [51] propose a whole class of cost sharing functions generated obtained by varying the splitting rule.

Unlike minimum spanning tree, the minimum Steiner forest problem is NP-hard; hence we cannot hope to obtain an algorithm that solves it optimally in polynomial time. The currently best approximation algorithm for Minimum Steiner tree by Robins and Zelikovsky [82] achieves approximation ratio 1.55; here we sketch a simple 2-approximation that better fits our purposes.

Clearly, every spanning tree of S is a valid Steiner tree of S; and (if the underlying graph is a metric), every Steiner tree can be converted into a spanning tree on the same set of terminals while at most doubling its cost. This suggests the following approximation algorithm for the Steiner Tree problem:

1. Compute the *metric closure* G' of the graph G. The metric closure G' of G is a complete undirected graph with each edge (u, v) of G' having cost equal to the cost of the shortest path from u to v in G.

2. Find a minimum spanning tree T on the required vertices S in the graph G'.

3. Convert T into a valid Steiner tree in G by replacing each edge $e = (u, v) \in T$ by its shortest u-v path in G. Note that this operation does not increase the cost of the tree.

We state the following theorem without a proof (the proof is easy and can be found in most algorithms textbooks, see e.g. [90]).

Theorem 2.3 *The above algorithm is a 2-approximation algorithm for the* Steiner tree *problem. Hence, the cost sharing function given by Equation (2.4) (multiplied by* $1/2$*) is a 2-budget-balanced cost sharing function for the Steiner Tree problem.*

2.3 Vertex cover

Vertex Cover is one of the oldest NP-hard problems — it has been proven NP-hard by Karp in his seminal paper [56]. It is very simple to state. Yet, although it has number of simple approximation algorithms, none of them achieves approximation ratio better than (essentially) 2 (see [90, 6]).

The Vertex Cover problem is given by an undirected graph $G = (V, E)$ and a non-negative cost $c : V \mapsto R^+$ on vertices. The goal is to find a minimum cost vertex cover, that is, a set of vertices $V' \subseteq V$ such that every edge $e \in E$ has at least one endpoint incident at a vertex from V' (that is, e is *covered*).

Vertex Cover has a number of simple approximation algorithms using a maximal matching or a depth-first search tree of the graph (for unweighted graphs), the local-ratio technique of Bar-Yehuda and Even [12], as well as LP-rounding and primal-dual approaches. The simple primal-dual algorithm presented below best fits our purposes, because it outputs useful cost shares.

The algorithm maintains a variable α_e for each edge e, that are initially set to zero. We keep increasing all variables at an uniform rate, until the edges incident to a vertex v can pay the opening cost of v, that is, until $\sum_{e\in\delta(v)} \alpha_e = c_v$. At this point, we declare the vertex v *tight*, and we *freeze* all edges incident to v. We stop raising the variables of frozen edges. We keep raising variables of unfrozen edges until another vertex goes tight, another edges freeze and repeat this process until all edges are frozen. In the end, we output the set C of frozen vertices as the vertex cover, and define the cost share $\xi(E, e)$ of each edge to be equal to α_e at the end of the algorithm.

Claim 2.4 *The set of tight vertices C is a valid vertex cover.*

Proof. Each edge is frozen only when one of its endpoints becomes tight. Since in the end, all edges are frozen, each edge must be incident to a tight vertex. ■

Claim 2.5 *The sum of cost shares $\sum_{e\in E}\alpha_e$ is a lower bound on the cost of an optimal vertex cover C^*.*

Proof. Fix the set C^*. Let us assign each edge $e = (u, v)$ to one of its endpoints, that is in the vertex cover C^*. If both endpoints are in C^*, assign e to one of them arbitrarily. Note that since C^* is a valid vertex cover, each edge is assigned. For each vertex v, let $A(v)$ be the set of edges assigned to v.

$$\sum_{e\in E} = \sum_{v\in V}\sum_{e\in A(v)} \leq \sum_{v\in V} c_e, \tag{2.5}$$

where the inequality follows from the fact that $\sum_{e\in\delta(v)}\alpha_e \leq c_v$, i.e, no vertex is ever overpaid by the cost shares. ■

Claim 2.6 *The cost of the set of tight vertices C is at most $2\sum_{e\in E}\alpha_e$.*

Proof. For each tight vertex we have $c_e = \sum_{e\in\delta(v)}\alpha_e$. Hence the total cost of the vertex cover is

$$\sum_{v\in C}\sum_{e\in\delta(v)}\alpha_e \leq 2\sum_{e\in E}\alpha_e, \tag{2.6}$$

where the inequality holds because on the left hand side, each edge is counted at most twice. ■

Chapter 3

Cost sharing and incentive compatibility

The bulk of this chapter is based on the paper, *Group Strategyproof Mechanisms via Primal-dual Algorithms* by Pál and Tardos [77]. Section 3.2.3 is based on a project for a game theory class [76] taught by Eric Friedman.

3.1 Introduction

Large networks, such as the Internet, are developed, built, and operated by co-operation of a large number of Autonomous Systems, all of whom act selfishly. These independent "agents" are willing to work together when such cooperation helps them achieve their goals. In this chapter we study the *cost sharing* problem, when a group of agents agrees to install or maintain a jointly used facility, such as a shared network, and split the cost among themselves. We picture the agents coming to an agreement with an independent service provider, who promises to set up the jointly used service in return for certain payments from the participating users.

Ideally, the users and the provider should arrive at a contract that is *self-enforcing*, that is, an agreement honored by all parties simply because violating it does not benefit anybody. We are interested in identifying situations where such self-enforcing contract (at least in an appropriately defined approximate sense) is possible.

The question we are trying to answer is how to agree on a set $S \subseteq U$ of users that will be part of the network, and how to determine the payments to the provider so that all players would be content with the decision. In particular we want that no agent would have an incentive to change her mind about whether to participate in the agreement, and that no subgroup of users would want to deviate because a cheaper alternative has become available.

If the set S of users being served is fixed, what remains is the problem of determining the vector of payments for users in S. If, however, we allow S to change, we need to worry about the changes of the user's cost shares, and the impact of this change on their behavior. In this chapter, we study cost shares with the natural property that when the set of served users expands, the cost share of any current user can either decrease or stay the same, but can never go up. Cost shares with this property are called *cross-monotonic*. Intuitively, cross-monotonicity encourages cooperation among the users: as adding new participants can only lower the cost, there is no reason for a hostile action of any user against newcommers, or any current users with the intention of removing them form the game.

Definition 3.1 A cost sharing function $\xi : 2^u \times U \mapsto R^+$ is *cross-monotonic*, if for all users $i \in U$, and any two sets of users S, S' such that $i \in S \subseteq S' \subseteq U$ we have

$$\xi(S, i) \geq \xi(S', i).$$

Modeling the users

Suppose that each potential user $i \in U$ has a private *utility* u_i of being connected to the network. The user is not willing to pay more than her utility for the benefit of being a member of the network, and if she is asked to pay a share that is higher than her utility, she prefers to stay disconnected.

Unfortunately for many games of interest, cross-monotonic budget-balanced cost sharing methods do not exist. It is well known that such methods provide cost allocations that lie in the *core*, a well studied concept in the game theory literature [74], and that the core is empty for these games. Motivated by the research effort on *approximate core* (e.g., [31, 23, 37]), we relax budget balance by require the users only to recover certain fraction of the total cost.

Definition 3.2 A cost sharing function $\xi : 2^U \times U \mapsto R^+$ satisfies α-*approximate cost recovery*, if

$$\sum_{i \in S} \xi(S, i) \geq c(S)/\alpha$$

for all $S \subseteq U$. A competitive cost sharing function that recovers an $1/\alpha$-fraction of the cost is said to be α-*approximately budget balanced.*

Remark 3.3 We have chosen to relax cost recovery instead of competitiveness in order to be consistent with previous work on approximate core. If we chose to relax competitiveness instead, all our cost sharing methods would be multiplied by the factor α.

3.1.1 Our Results

We give a general technique for using primal-dual approximation algorithms to generate cross-monotonic, competitive and approximately cost recovering cost shares. The cost shares are closely related to the dual solution generated by the algorithm, and hence the competitiveness and cost recovery properties are a consequence of dual feasibility and the approximation guarantee of the algorithm. However, cost-sharing directly based on linear programming duals does not usually deal well with individual utilities: the resulting cost-sharing is not cross-monotonic, and hence does not lead to group strategyproof mechanisms. Our technique changes the primal-dual approximation algorithm by striving to make the cost-sharing (the dual growing process) as smooth as possible, avoiding sudden arbitrary jumps and changes in the growth that could harm cross-monotonicity, even at the cost of somewhat weakening the approximation ratio.

We apply our technique to two network design games. First, we illustrate it by giving a cross-monotonic cost sharing method for the *Facility Location* game, and then proceed to the main result of the paper, which is a cross-monotonic cost sharing method for *Single Source Rent-or-Buy Network Design* (also known as a simple Access Network Design game). Both cost sharing methods are competitive, while the former recovers 1/3 and the latter 1/15 of the cost of the optimal solution. The bounds on cost recovery are shown by explicitly constructing a feasible solution to the respective problems.

3.1.2 The problems

The *facility location* problem is given by a set of potential servers (facilities) F, and a set $S \subseteq U$ of users. The problem is to open a subset of the facilities, paying amount f_p for each open facility p, and build a link from each user $j \in S$ to some open facility, given that the cost for connecting a client j to facility p is c_{jp}. We make the common assumption that c is a metric on $F \cup S$ (more precisely, on $F \cup U$). Let $c^*_{\mathrm{FL}}(S)$ denote the cost of the minimum cost solution for the set of users S.

In the *single source rent or buy* problem, we are given a set s of users and a source s, residing in a metric space defined by a graph G, and a number M. The goal is to build a tree that connects all users to the source, so that each edge e of the tree can either be *bought* at cost Mc_e, or *rented* at cost c_e times the number of users that use the edge on their way to the sink. Similarly, let $c^*_{\mathrm{RoB}}(S)$ be the cost of the optimum solution.

In the corresponding games users are trying to decide on the set S of users who should participate in the network, so that the participants can (approximately) share the cost.

3.1.3 Related work

There is a large body of literature on cooperative games, mostly focused on fair cost allocation (see e.g. [74]). A cost allocation is a vector x assigning each user $i \in U$ her share. It is *fair*, if no subset of users I is charged more than the optimum cost, i.e. $\sum_{j\in I} x(j) \le c^*(I)$ for every $I \subseteq U$. If it also recovers or approximately recovers the cost, it is said to be in the *core* or *approximate core*.

It is easy to see that a cross-monotonic and competitive cost sharing method automatically gives us fair allocations for all $S \subseteq U$:

Claim 3.4 *Let $\xi : 2^U \times U \mapsto \mathbb{R}^+$ be a cross-monotonic competitive cost sharing function. Then for any set $T \subseteq U$ of users, the vector $\xi(T, \cdot)$ is a fair cost allocation.*

Proof. We need to show that no subset $S \subseteq T$ of users is paying more than the cost $c(S)$ of serving them. This is a direct consequence of cross-monotonicity:

$$\sum_{j\in S} \xi(T,j) \le \sum_{j\in S} \xi(S,j) \le c(S).$$

The connection between the linear programming dual and cost-sharing is well known, and has been used to build core and approximate core allocations for various problems, including facility location [23, 37] and TSP games [31]. In fact, the classic Bondareva-Shapley theorem [21, 85] implies that for so called covering games (in which the cost $c^*(\cdot)$ is given by the minimum cost solution to a covering integer program), the core is nonempty if and only if the linear relaxation of the game-defining IP has no integrality gap.

Moulin and Shenker [75] show that cross-monotonic cost-sharing leads to group strategyproof mechanisms. They study the following simple mechanism for selecting the set of users S who should participate in the network: Start with assuming that $S = U$. Repeatedly offer every user j who remains in S membership in the network at price $\xi(S, j)$, and keep removing users who decline at the price offered. Stop when all remaining users in S accept the offer. Assuming that ξ is a cross-monotonic cost sharing method, they prove that this mechanism induces users to play truthfully, that is, accept whenever the price offered is less than or equal to u_j and decline otherwise. They prove that this mechanism is not only *strategyproof*, but also *group strategyproof*, that is, no user or coalition of users can benefit by deviating from the truthful strategy.

Moulin and Shenker [75] show that the classical Shapley value is in fact a cross-monotonic cost-sharing method whenever the cost function $c^*(\cdot)$ is a submodular function of the set S (see also [53]). The optimum solution to facility location, Rent-or-Buy network design, or their LP relaxations are in general not submodular.

The only known cross-monotonic cost-sharing for a non-submodular cost function is the cost-sharing of Kent and Skorin-Kapov [57] and Jain and Vazirani [51] for the spanning tree game. The spanning tree heuristic gives a 2-approximation for the Steiner tree problem, hence the spanning tree cost-sharing method is also a 2-approximate budget-balanced cross-monotonic cost-sharing for the Steiner tree game, which is a special case of the above Network Design problem with $M = 1$.

Jain and Vazirani [51] pose the question of designing cross-monotonic cost sharing for other combinatorial games. Recently, Devanur, Mihail and Vazirani [28] used a standard primal-dual algorithm to provide a strategyproof cost sharing mechanism for facility location. However, the resulting cost-sharing is *not* cross-monotonic, and their mechanism is only individually strategyproof, not group-strategyproof. In contrast, our technique modifies the algorithm to make the cost shares cross-monotonic (and hence the resulting mechanism group strategyproof).

Much of the previous work on network design games [4, 10, 30, 48] focuses on the Nash equilibria of the corresponding non-cooperative games. In its nature, Nash equilibrium is a non-cooperative concept: the agents simply observe each others behavior, and respond accordingly. Typically Nash solutions break down if we allow the agents to cooperate. In contrast, our aim is to design a game that allows the agents to follow a very simple truthful strategy, and such that the final solution adopted by the agents is stable, meaning that no group of agents has interest to change it.

From the plethora of approximation algorithms for facility location we mention only the primal-dual 3-approximation algorithm of Jain and Vazirani [52], and the currently best, 1.52-approximation algorithm of Mahdian et al. [69].

The first constant factor approximation algorithm for the Single Source Rent-or-Buy Network Design problem is due to Karger and Minkoff [55] and Guha, Meyerson and Munagala [39] (see also [40]). Both algorithms use a variant of facility location to gather the clients, and then build a Steiner tree on the set of gathering points. Swamy and Kumar [88] gave a primal-dual 4.55-approximation algorithm, that is also built analogously as a 2-stage algorithm (gathering followed by a Steiner tree). Recently, Gupta et al. [44] gave a simple randomized

3.55-approximation algorithm. Gupta et al. [47] announced that a derandomized version of this algorithm also yields cross-monotonic cost shares for the Single Source Rent-or-Buy game.

3.1.4 Our Techniques

Any cost sharing method ξ' can be made cross-monotonic by replacing it by a new function $\xi(S,j) = \sup_{T \supseteq S} \xi(T,j)$. It is not hard to see that $\xi(S,j)$ is population monotonic. However, the new function may be much bigger than the original one; and may not be easily computable. In Section 3.2.3 we give one example where this approach succeeds; but we suspect that its general usefulness is limited.

Most of this chapter is dedicated to designing primal-dual algorithms that generate cross-monotonic cost shares. Our algorithms are simple dual-growing processes that guarantee population monotonicity in a way analogous to the cross-monotonic cost-shares $\xi'(S,j)$ above. Known primal-dual approximation algorithms all are based on a "fair" way of raising cost-shares: all users increase their cost-contributions uniformly until they are satisfied (the cost contributed so far can pay for a facility they can connect to). The fact that these primal-dual algorithms raise all players cost-shares evenly suggests that they may be useful for building fair cost-shares. However, the resulting cost-shares are *not* population monotonic: adding new users close to an existing agent i, makes it easier for agent i to be satisfied, and once agent i stops raising her share, all remaining agents can be negatively effected.

The idea that allows us to compute monotone cost-shares is to have a "ghost" share for every user. Although in the original algorithm we are forced to stop raising the user's contribution in order not to over-charge some set of users, we still raise the ghost's share until all users are satisfied. For example in the facility location problem, considered in Section 3.2, the extra contribution from the ghosts results in opening more facilities and users getting connected earlier than in the original algorithm. This idea trivially ensures cross-monotonicity, as adding more users (and their ghosts) can only cause more facilities to be open, and each individual user satisfied earlier. The main challenge is to show that even that most facilities are only virtually paid for by the ghosts, the real cost shares still pay enough for constructing a feasible solution.

In Section 3.3 we consider the Rent-or-Buy network design game. Most known approximation algorithms for this problem build the solution in a two phase process: first they use a facility location algorithm to gather (or cluster) users into larger groups, and then build a Steiner-tree on the set of gathering points. It would be natural to try combining the cross-monotonic cost-sharing for the facility location problem of Section 3.2 with the spanning tree cost-sharing of Kent and Skorin-Kapov [57] and Jain and Vazirani [51]. However, combining cost-sharing for such a two phase process does not lead to cross-monotonic cost-shares: decisions made in the gathering stage affect the way players contribute towards building the tree in an unpredictable way, making it impossible to guarantee cross-monotonicity of the second stage.

In Section 3.3 we extend the idea of ghosts to the Rent or Buy network design game. We use the idea of a two stage process, gathering demand followed by building a Steiner tree on the gathering points. To overcome the difficulty making a two-phase process cross-monotonic, we run the two processes simultaneously, adding all potential gathering points into the second stage as soon as they are created by the first stage. Again, cross-monotonicity is easy: the more users we have, the more gathering points get opened, and earlier. More gathering points means that also the tree gets built faster, only reducing the cost share of each individual user.

As in the facility location case, the main challenge is to prove that the cost shares can pay for a feasible solution. Note that since the first phase may open much more gathering points than necessary (in fact, every point reachable by enough users eventially becomes a gathering point), the cost of building a Steiner tree on all of them would be prohibitively large. We thus need to select only a suitable subset of the points, and relate the processes that builds a tree on the selected points to the process on all potential points to show that the cost shares generated by the latter process can pay for the tree built by the former.

3.2 The Facility Location Game

In this section we develop our cross-monotonic cost-sharing method for the metric facility location problem. The facility location problem gives rise to very simple and elegant primal-dual algorithms, and hence it provides a clean example to illustrate how our "ghost" method turns a primal-dual algorithm into a cross-monotonic cost-sharing method. In the next section we use these ideas to develop a cost-sharing method for a more complex network design problem.

The *Metric Facility Location* game is given by a set F of possible *facilities*, a universe U of users and a metric $c(\cdot,\cdot)$ on $F \cup U$. Each facility $p \in F$ has *opening cost* f_p. Given a subset $S \subseteq U$ of users, the problem is to open a subset Q of facilities, and connect each user in S to an open facility. The cost of a solution for a given set of users S is the sum of the costs of facilities we open plus the sum of the costs for connecting clients to facilities. The cost of connecting user j is equal to her distance from the closest open facility. Formally, the cost $c(S)$ of a solution for the set of users S is

$$\sum_{q \in Q} f_q + \sum_{j \in S} \min_{q \in Q} c(j, q),$$

where Q is the set of facilities we open. The players in this game are the users, and our goal is to construct a solution and make each user pay certain amount to recover a constant fraction of the cost of the solution constructed.

3.2.1 A Linear programming formulation

Consider the following integer program for metric facility location, due to Balinski [11]. It has a variable y_p for each facility p indicating whether facility p is open, and indicator variable x_{pj} for each client-facility pair specifying whether client j is connected to facility p.

$$\text{minimize} \sum_{j \in S, p \in F} x_{pj} c(p, j) + \sum_{p \in F} y_p f_p \tag{3.1}$$

$$\sum_{p \in F} x_{pj} \geq 1 \qquad \forall j \in S$$

$$y_i - x_{ij} \geq 0 \qquad \forall p \in F, j \in S$$

$$x_{ij}, y_i \in \{0, 1\}$$

We obtain a LP relaxation of this program by replacing the last set of constraints by $x_{ij}, y_i \geq 0$. The dual LP is:

$$\text{maximize} \sum_j \alpha_j \tag{3.2}$$

$$\alpha_j \leq c_{pj} + \beta_{pj} \qquad \forall p \in F, j \in S$$

$$\sum_j \beta_{pj} \leq f_p \qquad \forall p \in F$$

The dual program has a variable α_j corresponding to each usert that can instantly be recognized as a candidate cost share for that user. The variable β_{pj} is best understood as a contribution of user j towards the cost of opening p. The first set of constraints then says that the cost share of each user can be split into a payment towards the connection cost (c_{pj}) and a contribution towards opening cost (β_{pj}). The second set of constraints says that the sum of contributions to any facility can never exceed its opening cost.

To obtain useful cost shares, we need to agree which solution to the LP (3.2) we wish to use. One option would be to use an optimal solution to the LP. However, there may not be a unique optimal solution. Even if there is, it can have undesirable properties, like assigning zero cost shares to some clients, and it can be unstable (a small perturbation, such as adding a client, can change the structure of the optimal solution radically). To get a better hold of the dual solution, we will use a primal-dual algorithm to generate it. The first and best known primal-dual algorithm for facility location is due to Jain and Vazirani [52], which we proceed to describe in the next section. The J-V algorithm does not give rise to cross-monotonic cost shares; in the following two sections we give two approaches to making them cross-monotonic.

3.2.2 The Jain-Vazirani algorithm

The Jain-Vazirani algorithm proceeds in two phases. In the first, dual-growing phase it tries to generate as large dual as possible. In the second phase it uses the dual constructed to pick a set of facilities to open and connect users to open facilities. Here we focus mainly on the first phase of the algorithm.

Phase 1 The algorithm starts with all dual variables α_j set to zero. Each client raises its dual variable at unit rate until it gets *satisfied* by being connected to an open facility. The β_{pj} variables merely respond to this growth in a minimalistic way to maintain dual feasibility. Initially, we define each client to be *unsatisfied.*

An edge between a client j and a facility p is said to be *tight* if $\alpha_j \geq c(j,p)$. In the beginning, there are no tight edges. As we raise the clients' variables α_j, eventually some α_j reaches the distance $c(j,p)$ to a facility p. At this point, we have to start growing the βpj variable in order to maintain dual feasibility. That is, at all times we maintain the invariant $\beta_{pj} = \max(0, \alpha_j - c_{pj})$.

A facility p is said to be *paid for*, if $\sum_j \beta pj = f_p$. When a facility is p paid for, the algorithm declares it temporarily open. It also declares all clients having a tight edge to this facility to be *satisfied* and facility p is declared to be a *wittness* for all these clients. We stop raising the dual variables of satisfied clients when they get satisfied. If an unconnected client j gets a tight edge to an already paid for facility, it is also declared satisfied and we stop raising its dual variable. The first phase ends when all clients are satisfied.

Note that for each user j, the dual variable α_j is equal either to the opening time $t(p)$ of its wittness facility (in case j has been contributing towards p) or to the distance $c(j,p)$ (in case p got paid for before the edge jp got tight). In any event, $\alpha_j \geq \max(c_{j,p}, t(p))$.

Phase 2 In the second phase, the J-V algorithm selects a subset of the set F_t of temporarily open facilities to open permanently. To ensure that the cost of opened facilities can be charged to the clients's dual variables, it never opens two facilities that have received positive contributions from the same client. We omit the details of this construction. However, Jain and Vazirani have proved the following.

Theorem 3.5 ([52]) *The solution output by Phase 2 of the J-V algorithm has cost at most $3\sum_{j\in S}\alpha_j$, and hence the J-V algorithm is a 3-approximation algorithm for metric facility location.*

A reader wishing to see the full algorithm as well as a proof of Theorem 3.5 is urged to read the original paper [52], or the textbook [90].

The dual variables produced by the J-V algorithm look almost like cost shares. Indeed, we can define a cost sharing function ξ' in the following way: given a set S of clients, we run the first phase of the J-V algorithm to obtain dual variables α_j, and define $\xi(S, j) = \alpha_j$ for $j \in S$. By Theorem 3.5, the function ξ is 3-approximately budget balanced. Is it also cross-monotonic? The following example provides a negative answer to this question.

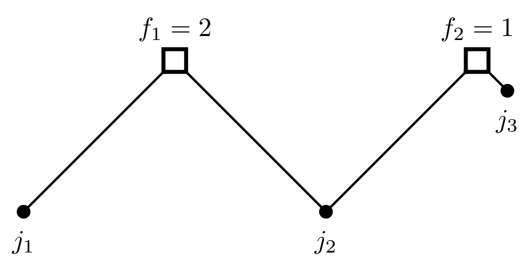

Figure 3.1: The Jain-Vazirani algorithm is not cross-monotonic.

Example 3.6 Consider an instance with two facilities p and q with $f_p = 2$, $f_q = 1$, and two clients, j_1 and j_2. Let the distances be $c(j_1, p) = c(j_2, p) = c(j_2, q) = 1$ (see Figure 3.1), the remaining distances implied by the triangle inequality. The J-V algorithm would raise both dual variables α_{j_1} and α_{j_2} until the facility p gets paid for, where it declares both clients satisfied and stops. Hence each client ends up paying for its distance to p, and the cost of p is split evenly among them.The resulting cost shares are $\xi'(\{j_1, j_2\}, j_1) = \xi'(\{j_1, j_2\}, j_2) = 2$.

Now suppose we add a new client j_3 located near facility q, i.e. $c(j_3, q) = 0$. Now the picture is different: user j_3 starts to contribute to q immediately, causing q to be paid for by time 1. Thus, user j_2 will get satisfied at time 1 by connecting to q, and stops raising α_{j_2}. That means, j_2 does not contribute to p at all, which means j_1 has to pay for p by herself. Thus, $\xi'(\{j_1, j_2, j_3\}, j_1) = 3 > \xi'(\{j_1, j_2\}, j_1)$.

3.2.3 The supremum trick

There is an easy way to make any cost sharing function ξ' cross-monotonic. Consider the following function ξ:

$$\xi(S, i) = \min_{T: i \in T \subseteq S} \xi'(T, i). \tag{3.3}$$

Instead of looking at only the amount $\xi'(S, i)$ that i is supposed to pay, we look at the cost share of i in all possible subsets of S, and make i pay the smallest of these cost shares. It is easy to see that the cost sharing function ξ is cross-monotonic: the larger the set S becomes, the larger range of subsets S' is that the minimum in Equation 3.3 is taken over. There are two obvious problems with this scheme. The first problem is that in order to compute the minimum, we may need to examine $2^{|S|-1}$ values of ξ'. The second problem is that the new cost shares may be much smaller, and may not recover a significant fraction of the cost, even if the original function did.

Formula 3.3 has a natural “dual” version

$$\xi(S, i) = \sup_{T: S \subseteq T} \xi'(T, i). \tag{3.4}$$

The supremum is taken "over all supersets" of S. We have a little bit of freedom here to specify what "all supersets" means: a simple option is to fix a finite universe U in advance and take the supremum over sets T such that $S \subseteq T \subseteq U$.

In general, Formula 3.4 suffers from problems similar to that of Formula 3.3: the supremum may be much larger than the original cost shares (potentially violating competitiveness), and computing the supremum means going through exponentially many (even infinitely many, if we allow U infinite). However, in the case of metric facility location, we can get past these obstacles. In particular, the "supremization" of the cost sharing function ξ' computed by the J-V algorithm is approximately budget-balanced and computable in polynomial time. As a first step in this direction, let us prove the following lemma.

Lemma 3.7 *Let j be a user, and S, T two sets of users such that $i \in S \subseteq T \subseteq U$. Then*

$$\xi'(T, j) \leq 3\xi'(S, j).$$

Proof. Consider running two instances of the J-V algorithm on the client sets S and T. Assume $\xi'(S, j) < \xi'(T, j)$, otherwise the lemma trivially holds. This means that in th execution on S, the client j got connected to some open facility p earlier than in the execution on the set T. Since j is still not connected in the execution on T, it must be that the facility p is still not open in that execution. How could that be? The only explanation is that there is a client j', that used to contribute to facility p in the execution on S, but contributed less in the execution on T. Hence in the execution on T, the client j' must have connected before time $\xi'(S, j)$ to some facility p'. But then,

$$c(j, p') \leq c(j, p) + c(p, j') + c(j', p') \leq 3\xi'(S, j).$$

It means that in the execution T', client j will get connected to facility p' by time $3\xi'(S,j)$, which is an upper bound on $\xi'(T,j)$ ■

Claim 3.8 *The cost sharing function ξ defined by*

$$\xi(S,j) = \frac{1}{3} \sup_{T: S \subseteq T \subseteq U} \xi'(T,j)$$

is a competitive, 9-budget-balanced cross-monotonic cost sharing method for metric facility location.

Proof. Cross-monotonicity follows from discussion of Formula (3.4). Budget balance follows from Theorem 3.5 and Lemma 3.7. ■

Computing $\xi(\cdot,\cdot)$

Our next goal is to show that the cost sharing function ξ from Claim 3.8 can be computed in polynomial time. In fact, we show that $\xi(S,j)$ is the optimum of an explicitly constructed linear program.

Consider a set S of users, and two users, j, k from this set. If j and k are physically close to each other, we would expect them to behave similarly in the J-V algorithm: they should connect to facilities at similar times, and hence the times when they get satisfied should not differ too much. This intuition is captured in the following lemma.

Lemma 3.9 *Let j and k be two clients, and consider the execution of the J-V algorithm on some set of clients S such that $j, k \in S$. Let α_j, α_k be the dual variables computed by the J-V algorithm corresponding to these two clients. Then, $|\alpha_j - \alpha_k| \leq c(j,k)$.*

Proof. Without loss of generality assume $\alpha_j \leq \alpha_k$. Let p be the wittness facility for j, that is, the facility that caused j to stop growing its dual variable. Then, we know that both the opening time $t(p)$ of p and the distance $c(j,p)$ are at most α_j. Thus, by the time $\max(c(k,p), t(p)) \leq \alpha_j + c(j,k)$, either α_k has stopped growing, or it has achieved a tight edge to an open facility (namely, facility p) ■

To compute the supremum according to Equation 3.4, consider a fixed set of users S, and imagine the J-V algorithm running on a larger set $T \supseteq S$ of users. How does the set of possible vectors of dual variables $(\alpha_j)_{j \in S}$ restricted to S look like? It is not hard to verify that any such vector must be a feasible solution to the linear program (3.2). In addition, we know that each pair of dual variables must satisfy Lemma 3.9. This motivates the following definition. Let $\alpha_S = (\alpha_j)_{j \in S}$ denote the vector of dual variables restricted to the set of users S. We define $A(S)$ to be the set of tuples α_S for which there exist some β_{pj} satisfying

$$\begin{array}{rcll} \alpha_j - \alpha_k & \leq & c_{jk} & \forall j,k \in S \\ \alpha_i - \beta_{pj} & \leq & c_{pj} & \forall p \in F, j \in S \\ \sum_j \beta_{pj} & \leq & f_p & \forall p \in F \\ \alpha_j & \geq & 0 & j \in S \\ \beta_{pj} & \geq & 0 & j \in S, p \in F \end{array} \tag{3.5}$$

Clearly, the vector of cost shares α_S generated by the execution of the J-V algorithm on any $T \supseteq S$ has to be in $A(S)$. Interestingly, by adding enough clients in strategic locations, we can achieve (at least in the limit) any vector in $A(S)$ to come as an output of the J-V algorithm. To make sure we can achieve this, we resort to a little cheat. We allow adding new expensive facilities at strategic locations. If we make the new facilities sufficiently expensive ($M = |S| \max_{j,p} c(j,p) + f_p$ is enough), these new facilities will never become paid for, and hence will not affect the behavior of the J-V algorithm (the optimum solution also remains the same).

Thus, the addition of these facilities does not change the cost shares $\xi'(S, \cdot)$, but helps to ensure that

Lemma 3.10 *For any set of users S and any strictly positive vector $\alpha'_S \in A(S)$, we can construct a superset $T \supseteq S$ of users, and add $|S|$ sufficiently expensive facilities, such that the duals of the J-V algorithm on the new input restricted to S will be equal to α'_S.*

Proof. For each client $j \in S$, we add a new facility p_j of sufficiently high cost M, such that $c(j, p_j) = \alpha'_j$ (the remaining distances follow from the triangle inequality). We add M/α'_j new clients[1] at distance zero from f_p.

Now, let us observe the execution of the J-V algorithm on the new instance. We use α_j to denote the dual variables constructed by this execution of the algorithm. Note that each of the new facilities p_j gets opened exactly at time $t(p_j) = \alpha_j$. This means that the dual variable α_j of each user $j \in S$ stops growing at latest by time α'_j. Thus $\alpha_j \leq \alpha'_j$.

We claim that each client $j \in S$ gets satisfied exactly at time α'_j, and hence $\alpha_j = \alpha'_j$. To see this, assume the contrary and let j be a client who gets satisfied at time $t < \alpha'_j$. This can happen only because j got a tight edge to a facility p that got opened at or before time t. We claim that p cannot be any of the new facilities $p_{j'}$. To see this, note that the distance $c(j, p_{j'})$ is too large: $c(j, p_{j'}) = c(j, j') + c(j', p_{j'}) = c(j.j') + \alpha'_{j'}$, which by the first inequality of (3.5) is greater or equal to $\alpha'_j > t$. Similarly, p cannot be any of the old facilities, and that follows form the fact that α'_S is a feasible solution to the dual program 3.2. ■

Thus, finding the maximum α_j over all dual vectors in $A(S)$ gives us the desired

[1] If this is not an integer, round M to the nearest multiple of α'_j first.

cost share.

Theorem 3.11 *The cost sharing function defined by*

$$\xi(S, j) = \frac{1}{3} \max \{\alpha_j \mid \alpha_S \in A(S)\} \tag{3.6}$$

is cross-monotonic, competitive, and recovers a $1/9$ *fraction of the cost of the solution of metric facility location.*

Our main contribution for the facility location game is to build cross-monotonic cost-shares that recover a constant fraction of the cost of the optimal solution. To define the cost-shares we use a "ghost-process" that can be viewed as a smoothed version of a primal-dual approximation algorithm, such as Jain and Vazirani [51]. We will use a facility location algorithm by Mettu and Plaxton [70] for building a primal solution. Their algorithm is a greedy in nature and does not come with a dual solution or cost shares. As a byproduct, our analysis shows that the algorithm of Mettu and Plaxton can be interpreted as a primal-dual algorithm. Archer et al. [5] have also independently given a primal-dual analysis of this algorithm.

3.2.4 Smoothing out the dual: the ghost process

Now we proceed to describe a dual-growing process that smoothes out the sudden stops in the Jain-Vazirani algorithm. As Example 3.6 shows, non-cross-monotonicity is caused by clients who suddenly stop raising their dual variable, thus failing to help others to achieve their goal soon enough. The algorithm we propose avoids this problem by keeping to raise shadows (*ghosts*) of dual variables even after the real variable was forced to stop.

First we define a *ghost process*, and use it to define cost shares for each user. For each facility p we define a neighborhood S_p of clients, the smallest set of clients such that charging each of them the same amount $\alpha = t(p)$ can cover the cost of their connection, and enough is left over to pay for opening the facility. We will not be able to charge all users in this set the amount α as they may have closer other facility that they would rather contribute to. But our plan is to make the users in the set S_p pay enough so that if we choose to open facility p we will be able to recover a constant fraction of the cost.

The *ghost process* grows a ghost for every user. The ghost of user j is a ball with j as its center that is growing uniformly to infinity. The radius of the ghost at any time $t \geq 0$ is equal to t. The ghost of a user j *touches* a facility p at time t if $c(j,p) \leq t$. All ghosts that touch a facility start *filling* the facility. The contribution of the ghost j at time t towards filling facility p is $\kappa_{jp} = \max\{0, t - c(j,p)\}$. A facility p is declared *full*, if $\sum_{j \in S} \kappa_{jp} \geq f_p$. Let $t(p)$ denote the time when p becomes full. Let S_p be the set of users that contributed towards filling p, i.e. all users that are at distance less than $t(p)$ from p. Note that charging each user in S_p the amount $\alpha = t(p)$ we would recover exactly the cost of opening facility p and the cost of connecting all users in S_p to p, as $f_p = \sum_{j \in S_p}(t(p) - c(j,p))$, by definition.

Cost shares

We raise the cost share α_j of each user j at unit rate until j's ghost touches some full facility or some facility it is touching becomes full. More formally,

$$\alpha_j = \min_{p \in F} \left\{ \max(t(p), c(j,p)) \right\}. \tag{3.7}$$

Another way to think about this is that each facility p makes an offer to serve j at cost $\alpha_j^{(p)} \max(t(p), c(j,p))$, and the client then chooses the facility that can serve her at the lowest cost. Note that by adding more users, each facility can only get filled more quickly, and hence each individual user can only become satisfied earlier. Hence

Lemma 3.12 *The cost shares generated by the above process are cross-monotonic.*

Deciding which facilities to open

We give two alternative rules for constructing the set of open facilities. Both rules construct a feasible solution of cost bounded by thrice the sum of cost shares. The first rule can be viewed as a restatement of the facility location algorithm due to Mettu and Plaxton [70]. We give the second rule because it may look more natural in this context; in Chapter 4 we shall prove that the the cost shares α are strict with respect to this rule.

- **Mettu-Plaxton Rule:** For each facility, decide whether to open it or not at the time it gets filled. Open the facility p if and only if at time $t(p)$, there is no already open facility q such that $c(p, q) \leq 2t(p)$. This way we avoid opening many facilities crowded in one place.

- **Modified Mettu-Plaxton Rule:** The rationale behind this rule is that we only want to open well-funded facilities, that is, facilities for which its clients' contributions are at least partially backed by the contributing clients' cost shares. We define a facility p to be *well-funded*, if $\alpha_j \geq t(p)/3$ for all $j \in S_p$. Each time a facility p gets filled, the modified rule decides to open p if (1) it is well funded, and (2) there is no already open facility q such that $c(p, q) \leq 2t(p)$.

This simple fact holds regardless of which rule we use to open facilities.

Claim 3.13 *For any facility p, there is an opened facility q such that $c(p,q) \leq 2t(p)$ (possibly $p = q$, if p is open).*

Proof. If the original Mettu-Plaxton rule is used, the claim is immediate. For the modified rule, consider a closed facility p. By a way of contradiction, suppose the claim does not hold, and let p be the facility with minimum $t(p)$ for which the claim fails. The claim clearly holds for any p that is well-funded; thus consider a facility p that is not well funded. For any such facility, there must be a client $j_0 \in S_p$ such that $\alpha_j < t(p)/3$. Now, the reason why α_j stopped growing must be a facility q' such that $c(j,q') \leq \alpha_j$ and $t(q') \leq \alpha_j$. Since we picked p to be the facility with minimal $t(p)$ for which the claim does not hold and $t(q') \leq \alpha_j < t(p)/3$, the claim must hold for q'. Hence there is an open facility q such that $c(q,q') \leq 2t(q')$. But then, the triangle inequality gives us

$$c(p,q) \leq c(p,q') + c(q',j) + c(j,p) \leq 2t(q') + \alpha_j + t(p) \leq 2t(p)$$

. ■

Analysis

First we note that the cost shares α_j give a feasible solution to the dual LP (3.1) of cost $\sum_{j \in S} \alpha_j$. As this is a lower bound on the cost of any solution, it follows that $\sum_{j \in S} \alpha_j \geq c(S)$.

Theorem 3.14 (Competitiveness) *Every solution to the facility location problem has cost at least $\sum_{j \in S} \alpha_j$.*

Next we will relate the amount of funds collected to the cost of the solution constructed by the original or the modified Mettu-Plaxton rule. This not only provides us with a proof that our cost shares are budget balanced, but also gives an alternate proof of the approximation guarantee of the algorithm of Mettu and Plaxton.

Recall that for each facility p we defined the set S_p of clients "contributing to" p. The following argument holds for either opening rule. Consider two open facilities p and q, and let $t(p) \geq t(q)$. Since we decided to open p, it must be far enough from q: $c(p,q) \geq 2t(p) \geq t(p) + t(q)$, and hence

Fact 3.15 *If facilities p and q are both open, then their contributor sets S_p and S_q are disjoint.*

This means that every user contributes towards filling of at most one open facility. For the purposes of the analysis, we assign all users in the set S_p to the facility p, for every open facility p. We assign each user that does not belong to the set S_p of any open facility p to its closest open facility.

We claim that the users in the set S_p of every open facility p have paid enough to cover 1/3 of the cost of p as well as 1/3 of their connection costs. By definition of S_p, $f_p = \sum_{j \in S_p}(t(p) - c(j,p))$. After rearranging, we obtain that the opening cost f_p plus the connection cost $\sum_{j \in S_p} c(j,p)$ is equal to $|S_p|t(p)$. To prove our claim it suffices to show that the cost share of each user is at least $t(p)/3$.

Lemma 3.16 *Let p be an open facility, and let $j \in S_p$ be a user. Then $3\alpha_j \geq t(p)$.*

Proof. The lemma trivially holds if the modified Mettu-Plaxton rule is used. For the original rule, consider a user $j \in S_p$. For contradiction, assume that $\alpha_j < t(p)/3$. Let q be the first full facility that j touched, i.e. $c(j,q) \leq \alpha_j$ and $t(q) \leq \alpha_j$. If q is open, we get a contradiction, since $c(p,q) \leq t(p) + \alpha_j < 2t(p)$. If q is not open, there is an open facility q' such that $c(q,q') \leq 2t(q) \leq 2\alpha_j$. See Figure 3.2. Again, we get a contradiction, because $c(p,q') \leq c(p,j) + c(j,q) + c(q,q') \leq$

$t(p) + \alpha_j + 2t(q) \leq t(p) + 3\alpha_j < 2t(p)$. Note that $p \neq q'$, since $t(q') \leq \alpha < t(p)$.

■

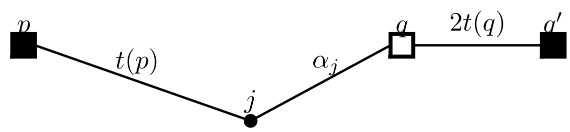

Figure 3.2: If $\alpha_j < t(p)/3$, then $c(p, q') < 2t(p)$.

Theorem 3.17 (Cost Recovery) *The cost of the solution constructed by either original or modified Mettu-Plaxton rule is at most* $3\sum_{j \in S} \alpha_j$*, and hence the above algorithm is a 3-approximate cross-monotonic cost sharing method for metric facility location.*

Proof. Lemma 3.16 shows that the clients in the sets S_q pay for a 1/3rd of their connection costs and the cost of the open facilities. Lemma 3.13 implies that for every user that does not belong to any S_q, there is an open facility within distance $3\alpha_j$. ■

3.2.5 Fault tolerant facility location

The Fault tolerant facility location problem generalizes regular facility location by allowing clients to have connections to multiple facilities. In addition to facilities and clients, an integer $r \geq 1$ is given as a part of the input. The goal is to open a subset of facilities, and connect each user to r distinct open facilities (the rationale is that even if any $r - 1$facilities fail, each client will still be connected). As usual, the cost of the solution is the sum of opening costs, plus the connection costs $c_{j,p}$ for each client j and each of the r facilities p that j is connected to.

The ghost process from Section 3.2.4 transfers almost without change to fault tolerant facility location. Again, we will have a ghost for every user j that will be just a ball centered at j with radius equal to the current time t. A ghost j contributes $\kappa_{jp} = \max\{0, t - c(j,p)\}$ to each facility it touches, and the opening time $t(p)$ of facility p is the minimum time such that $\sum_{j \in S} \kappa_{jp} \geq f_p$.

In the non-fault-tolerant case, each client got to pick one facility p, and then had to pay cost share equal to $\alpha_j^{(p)} = \max(t(p), c(j,p))$. In fault tolerant facility location, each client wishes to have connections to r distinct facilities; we thus let her choose the least costly set of r distinct facilities to contribute to. This gives us the cost share

$$\alpha_j = \min_{\{p_1, p_2, \ldots, p_r\} \subseteq F} \sum_{i=1}^{r} \alpha_j^{(p_i)}. \tag{3.8}$$

Cross-monotonicity of such shares is easy to see: adding users can only decrease $\alpha_j^{(p)}$, and in turn, the cost share. We will not elaborate on competitiveness, but it is not difficult to verify that α_j form a feasible dual solution to the fault tolerant facility location LP (see e.g. [41] for an LP formulation).

Opening facilities To decide which facilities to open, we again consider all facilities in the order of increasing $t(p)$. For each faciltity p, we decide to open it if there are fewer than r already opened facilities within distance $2t(p)$ from p. We assign each client to the r closest open facilities. Note that by construction, we get the following equivalent of Fact 3.15.

Fact 3.18 *No client is contributing to more than r open facilities.*

Analysis Consider user j. Let $p_1, p_2, \ldots, p_r$ be the r facilities that determine the cost share of j. That is, $p_1, p_2, \ldots, p_r$ are the r facilities with lowest $\alpha_j^{(p)}$. Let us assume they are ordered by increasing $\alpha_j^{(p_i)}$. According to Equation (3.8), the cost share of j is $\alpha_j = \sum_{i=1}^r \alpha_j^{(p_i)}$.

On the other side of the equation, we take the $k \leq r$ facilities that user j contributes to. If $k < r$, we complete the list by adding the closest $r - k$ open facilities that j did not contribute to. Let $q_1, q_2, \ldots, q_r$ be the list of k open facilities that j contributed to, plus $r - k$ closest open facilities that were filled before j had a chance to reach them, ordered by increasing $\alpha_j^{(p_i)}$. Note that the lists $\{p_i\}_{i=1}^r$ and $\{q_i\}_{i=1}^r$ may not be disjoint.

To bound the cost of our solution, it suffices to argue that each $\alpha_j^{(p_i)}$ is sufficient to pay for 1/3 of the cost of connecting j to q_j plus 1/3 of the contribution offered to q_j by j. Hence, it suffices to argue that $3\alpha_j^{(p_i)} \geq \alpha_j^{(q_i)}$ for $i = 1 \ldots, r$.

For the sake of contradiction, let l be the first index such that $3\alpha_j^{(p_l)} < \alpha_j^{(q_l)}$. Note that this means, that immediately before time $\tau = \alpha_j^{(q_l)}$, there were fewer than $l \leq r$ open facilities within distance τ from j. Thus, this means that when each of the facilities $p_1, p_2, \ldots, p_l$ were considered, there were fewer than r open facilities within distance $\tau - c(j, p_i) \geq \tau - \alpha_j^{(p_i)} \geq 2t(p_i)$ from p_i! Hence, according to our opening rule, each of the facilities $p_1, p_2, \ldots, p_l$ must have been declared open. But this means that $p_1 = q_1$, $p_2 = q_2$,..., $p_l = q_l$, and hence $\alpha_j^{(p_l)} = \alpha_j^{(q_l)}$, contradicting our original assumption.

Theorem 3.19 (Cost recovery) *Each user j can pay 1/3 of her contributions to open facilities, plus 1/3 of her connection costs. Hence the cost shares α_j recover 1/3 of the cost of our solution.*

3.3 Single Source Rent-or-Buy

The Single Source Rent-or-Buy game is given by an undirected graph $G = (V, E)$ with costs c_e on edges, a designated *source* vertex s, and a parameter M. We are given a set S of *users*, each user j being located at some vertex $v \in V$ (we allow more users to be located at the same node). If no confusion arises, we use j instead of its location v.

The goal is to build a network that connects each user to the source s by a path. Each edge e in the network can either be *bought* at cost Mc_e, or rented at cost c_e. A bought link can support any number of paths, while on a rental link we must pay the cost c_e for each path that uses the edge. A network is feasible, if it can support a path from each user j to the source s. The cost of the network is the sum costs of all bought and rented edges. Let $c^*(S)$ denote the cost of the optimal network supporting the set of users S.

In this section we present our main result, a cost sharing method for the rent or buy game.

Theorem 3.20 *There is a cross-monotonic cost sharing method for the single source rent or buy game that is competitive and recovers at least* $1/15$ *of the cost of a feasible network that can be constructed in polynomial time.*

It is not hard to see that the bought edges in the optimum rent or buy network form a Steiner tree with s at its root, and that the rental edges form a shortest path connection from each user S to the closest point in the tree. Several approximation algorithms [55, 39, 88] for this problem exploit the optimal structure to construct a solution in two phases: first they use a facility location-like algorithm to gather the users in a few *center* locations, and rent a path from each user to such a center.

When enough demand is gathered at each center, a Steiner tree is built, connecting all centers to the source.

It would be tempting to use our scheme for facility location to share the rental costs, and use the cost sharing method of Jain and Vazirani [51] to share the cost of buying the tree among the centers. Naturally, the contribution of each center would be split among the users connected to it. However, the gathering algorithm has to make decisions on where to choose the centers and how to assign users to them. These decisions (necessarily influenced by the addition of new users) may have arbitrary effects on what center a particular user is assigned to and the number of other users it is sharing the center with, that could make the share of an individual user heavily non-crossmonotonic.

We solve this problem by giving a process that avoids making decisions altogether. As in the facility location case, we grow a ball around every user. Since there is no opening cost, we open a center at any location p that has been reached by M or more user balls. A regular primal-dual algorithm, such as [88] would freeze all users that got connected to the open center, and stop growing their balls.

To ensure cross-monotonicity, we use the same trick as in the previous section: continue growing a ghost ball around every user, even after the user gets connected. Note that since the ghost balls keep contributing to opening new centers, much more centers are opened than necessary. This poses a new problem: each user may be connected to multiple centers, and the cost of building a Steiner tree on all centers may be prohibitively large.

We deal with this problem in two different ways. When computing the cost shares, we pretend that we are building a Steiner tree on all centers. We split the cost of each center and try to charge it equally to all users connected to the center.

Now, since a user may be connected to multiple centers, she gets to choose which center to contribute to (we assume each user wants to contribute only to the center where the required contribution is smallest).

Since most centers are not contributed to by enough users (or any users at all), we can not hope that the user contributions would recover a constant fraction of the cost of a Steiner tree on all centers. To build a tree that the cost shares can pay for, we select only a subset of the centers, and assign each user to only one center. Now we have forced the user to contribute only to the center he was assigned to, but we still can only ask a contribution that is equal to the minimum of her potential contributions to any possible center.

Technically, the hard part is to relate the two processes: one that pretends to build a Steiner tree on all centers, used to compute the cost shares, and a standard primal-dual process that we use to build a real tree on a subset of the centers, and show that the cost shares grown by the first process pay for the solution constructed by the second.

After a simplifying assumption, we describe the heart of the algorithm, which is the ghost growing process. Then we describe how to generate the cost shares, and how to find a feasible solution that can be partially paid for by the cost shares. Finally, we show competitiveness by giving a lower bound on the cost of any feasible solution.

A Simplifying Assumption. It will be convenient to think of each edge e as a line segment of length c_e, containing a continuum of points. We use the term *location* to refer to both original vertices and intermediate points. We extend the shortest path metric c to a metric on locations in the obvious way. For the rest of this section, we will assume that *an edge segment can be bought or rented between arbitrary pair of locations.*

As shown in [88], a solution that is allowed to buy and rent edge segments arbitrarily can be transformed into one that does not use intermediate points of no greater cost. This is because any intermediate point p on edge $e = (uv)$ that is an endpoint of some paths can be shifted towards u or towards v without increasing the cost. We move p until it merges with another intermediate endpoint or until it reaches one of the ends of the edge. This way, we can remove all intermediate endpoints one by one.

3.3.1 The ghost process

The ghost process we consider here is similar to the process considered in Section 3.2. Hence, each user j has a ghost, which is a ball $B(j,t)$ with j as its center and radius equal to the current time t.

The difference is that instead of thinking of the ghosts as contributing towards opening a facility, we will be looking for locations that are potential gathering points. Good candidates are locations that have been reached by M or more ghosts. Let us denote the (time varying) set of all such locations by $\mathcal{C}$. Note that at any time, the set $\mathcal{C}$ can be represented as a collection of vertices, edges and segments of edges.[2]

A *connected component* of $\mathcal{C}$ is any inclusion-maximal subset C of $\mathcal{C}$ such that any two locations $p_1, p_2 \in C$ are joined by a path (that is, a continuous curve) lying completely within C.

[2]It is easy to see that each edge is subdivided into at most $2\min\{M, |S|\} + 1$ segments, and hence this representation is of polynomial size.

Let us take a closer look at the behavior of the set $\mathcal{C}$ in time. In the beginning, $\mathcal{C}$ is empty, and there are no components. As time progresses, components begin to appear. Each component starts as a single location p, that has been first reached by some set S of users, $|S| \geq M$. With increasing time, the set of locations reachable by the users in S locally looks like a uniformly growing ball with center p (note that the set $\bigcap_{j \in S} B(j, t)$ of locations reachable by all users in S may look as a union of several balls, as the same set S may initiate multiple components of $\mathcal{C}$). As these balls grow, two or more of them may eventually touch, joining into a single component. In general, pairs of components will be connecting up and merging. Hence at any time, each component C of $\mathcal{C}$ can be expressed as an union of uniformly growing balls of different radii, each of which started as an independent component at a single location p.[3] Note that since the ghosts grow indefinitely, eventually the set $\mathcal{C}$ will consist of a single component.

We would like to point out a correspondence between the way the set $\mathcal{C}$ grows and the standard primal-dual algorithm for Steiner tree (see [1], and also [38]), that will be useful later. The algorithm maintains a list of components. Initially, each terminal vertex is in a separate component. It grows a ball uniformly around every vertex, and whenever the balls of two vertices touch, it builds the edge joining the two vertices, merging the two components. The only difference is that while the Steiner tree algorithm starts with a fixed set of components, in the ghost growing process components may get created at arbitrary points in time. We will make use of this connection in Section 3.3.4 to show that the cost shares can pay for a constant fraction of the buying cost in a solution we construct.

[3] At most $\min\{M, |S|\}$ separate components can be created on any edge e, hence the number of balls in the union is polynomial.

3.3.2 Cost shares

To specify how the cost shares are computed, we need some definitions. We say that a user j at time t is *connected* to a component C of $\mathcal{C}$, if $B(j,t) \cup C \neq \emptyset$. The user is *satisfied* if it is connected to a component containing the source vertex s. For each user j, let $t(j)$ denote the time when the user first becomes connected to a component and let $t'(j)$ the time when j becomes satisfied.

Roughly speaking, we should make each user pay for the rental costs associated with her. This suggests that we should charge each user amount proportional to the time the user spent unconnected. Further, each user connected to a component C should contribute an equal share to the cost of growing the component. If a user is connected to multiple components, we let her contribute only to the component where his share is smallest.

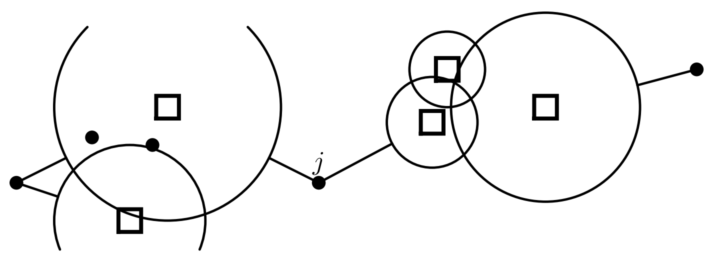

Figure 3.3: A client connected to multiple growing components.

For each component C of $\mathcal{C}$, let $S(C)$ denote the set of users connected to C. For a connected user j, let $a_j(t)$ be the maximum size $|S(C)|$ over all components C that j is connected to at time t. Let us define a function f_j for each user j as follows: $f_j(t) = 1$ for $0 \leq t < t(j)$, $f_j(t) = M/a_j(t)$ for $t(j) \leq t < t'(j)$, and $f_j(t) = 0$ for $t > t'(j)$. We define two versions of cost shares α_j and α'_j for each user j. The final cost share, determined in Theorem 3.30, will be a weighted sum of the shares we now define, and balanced to optimize the fraction of the cost recovered.

$$\alpha_j = \int_0^{t'(j)} f_j(t)dt \quad \text{and} \quad \alpha'_j = \int_0^{t(j)} f_j(t)dt = t(j).$$

It is easy to establish the first of the three desired properties of our cost sharing method.

Theorem 3.21 *The cost shares α and α' are cross-monotonic functions of the set S.*

Proof sketch. By adding more users, we can only make the set $\mathcal{C}$ larger, and hence make each user connect earlier. Moreover, each component of $\mathcal{C}$ can only increase by adding more users, as increases the number of users connected to that component, hence the cost shares can only grow slower. ■

3.3.3 Building a solution

The goal of the following two sections is to prove that the cost shares α and α' can recover a constant fraction of the cost of a network supporting the given set S of users. In this section, we give a way to construct such a feasible network, and in the next section we use the similarity between the behavior of the set $\mathcal{C}$ and the primal-dual Steiner tree algorithm to compare the cost of the solution constructed with the cost shares.

Remember from the discussion in Section 3.3.1 that each new component of the set $\mathcal{C}$ started out as a single point p. We will call such starting points *centers.* For a location p, let $t(p)$ denote the first time p appeared in the set $\mathcal{C}$. Note that $\mathcal{C}$ at a given time t is just a union of balls $B(p, t - t(p))$, with a ball of radius $t - t(p)$ around each center p.

We want to use the centers as gathering points. However, to be able to pay for the Steiner tree on the centers, we must select only a subset of the centers to open, and assign each user to contribute to at most one open center. We do the selection in a similar way we did for facility location.

For each center p we decide whether to open it at the time the center is created. We declare p open, if at time $t(p)$, there is no open center within radius $2t(p)$ of p. As in the facility location case, this ensures that no user j can be in the set $S_p = \{j \in S \mid c(j,p) \leq t(p)\}$ of users contributing towards creating p for two distinct open centers p. The following result carries over from the previous section.

Lemma 3.22 *Let p be an open center, and let $j \in S_p$. Then $3\alpha'_j \geq t(p)$. For a user j that does not belong to any S_p, there is an open center p such that $c(j,p) \leq 3\alpha'_j$.*

The network we construct is thus: we buy a Steiner tree that connects open centers to the source s, and rent a path from each user to its closest open center. Lemma 3.22 shows that the cost shares α' can pay for 1/3 of the rental cost of the network. The next section is devoted to showing that the shares α can pay for a constant fraction of the tree we build.

3.3.4 Bounding the tree cost

For the purposes of analysis we assume that the primal-dual algorithm of Agrawal et al. is used [1] (see also [38]) to build a Steiner tree on the open centers. The algorithm starts with each center in a separate component. It grows a ball uniformly around every center. Every time two balls touch, a check is performed if the centers of the balls are in the same component. If not, a shortest path between the ball centers is bought, merging the two components into one. In addition, the center p of the first ball to reach the source vertex s is allowed to buy a shortest path between p and s.

We can think of the algorithm as incurring cost of M per time unit for growing each component that does not contain the root vertex s. (Growing the component containing s is free.) That is, if $a(t)$ denotes the number of non-root components at time t, the total cost of growing the components is $M \int_0^\infty a(t)dt$, where the integral is over the entire execution of the algorithm. By standard arguments, the cost of the tree constructed is at most two times the growing cost.

For each Steiner component C' we define a (time varying) set $\mathrm{Contrib}(C')$ of users that will be responsible for maintaining a steady flow of funding at least $M/3$ per time unit from the time increments $f_j(t)$ of their cost shares. For an open center p, let $S_p(t)$ denote the set of users that are within distance t from p. Note that $S_p = S_p(t(p))$. Let C' be a Steiner component at some given time t. We define

$$\mathrm{Contrib}(C') = \bigcup_{p \in C'} S_p(\max\{t, t(p)\}).$$

We invite the reader to verify that no user contributes to more than one component at any time. Indeed, if a user j contributes to two open centers p and q at some time, by triangle inequality p and q are close enough to belong to the same Steiner component. Hence

Lemma 3.23 *At every time t, the contributor sets of two distinct Steiner components C'_1, C'_2 are disjoint.*

Consider a user j that belongs to a contributor set of a Steiner component C'. In the ghost process, this user may be connected to several components of $\mathcal{C}$, and some of these components may have a large number of users connected to them. So even if the component C' is "small", that is, does not have many contributors, it may happen that most of its contributors all have a connection to a large $\mathcal{C}$-component, and hence their contributions $f_j(t)$ are not enough to support the growth of the component C' at time t. However, what we will be able to say is that later, namely at time $3t$, the component C' will have grown large enough to cover the area that the original "large" components occupied at time t and connect to all users that were connected to the "large" components at time t. These users together are able to pay enough for the growth of C' at time $3t$ with their growth rates $f_j(t)$ at time t.

Lemma 3.24 *Let j be a user such that at time t, $j \in \text{Contrib}(C')$ for some Steiner component C'. Then $f_j(t/3) \geq M/|\text{Contrib}(C')|$.*

Before proving 3.24, let us state a useful fact that essentially carries over from the facility location secition.

Fact 3.25 *For location $p \in \mathcal{C}$, there is an opened center q (possibly $q = p$) such that $c(p, q) \leq 2t(p)$.*

Lemma 3.26 *Suppose at time t, users i and j are connected to the same component C of the set $\mathcal{C}$. Then at time $3t$, i and j belong to the contributor set of the same Steiner component C'.*

Proof. Let i and j be connected to a component C of $\mathcal{C}$ at some time t. Note that C is a union of balls around (opened or unopened) centers. User i is connected to C, hence there must be a center q with a ball $B(q, t - t(q)$ that intersects the ghost of i. Similarly there is a center q' and a ball $B(q', t - t(q'))$ intersecting the ghost of j. Since the centers q and q' belong to the same component, there must be a sequence of centers $q_1 = q, q_2, \ldots, q_k = q'$, with balls $B_\ell = B(q_\ell, t - t(q_\ell))$ such that the balls of every two consecutive centers intesect.

By Fact 3.25, there is a sequence of (not necessarily distinct) open centers $p_1, \ldots, p_k$, such that for every ℓ, $c(p_\ell, q_\ell) \leq 2t(q_\ell)$. See Figure 3.4 for an example. It is easy to verify that the distance $c(i, p_1) \leq 2t(q_1) + t \leq 3t$, and analogously $c(p_k, j) \leq 3t$. The distance $c(p_\ell, p_{\ell+1})$ can be bounded by $4t$ as

$$c(p_\ell, p_{\ell+1}) \leq c(p_\ell, q_\ell) + c(q_\ell, q_{\ell+1}) + c(q_{\ell+1}, p_{\ell+1}) \leq$$

$$\leq 2t(q_\ell) + (t - t(q_\ell)) + (t - t(q_{\ell+1})) + 2t(q_{\ell+1}) \leq 4t.$$

Hence each pair of consecutive balls $B(p_\ell, 3t)$ must intersect, proving that p_1 and p_k are in the same Steiner component at time $3t$. By definition of Contrib, both i and j must be in the contributor set of this Steiner component. ■

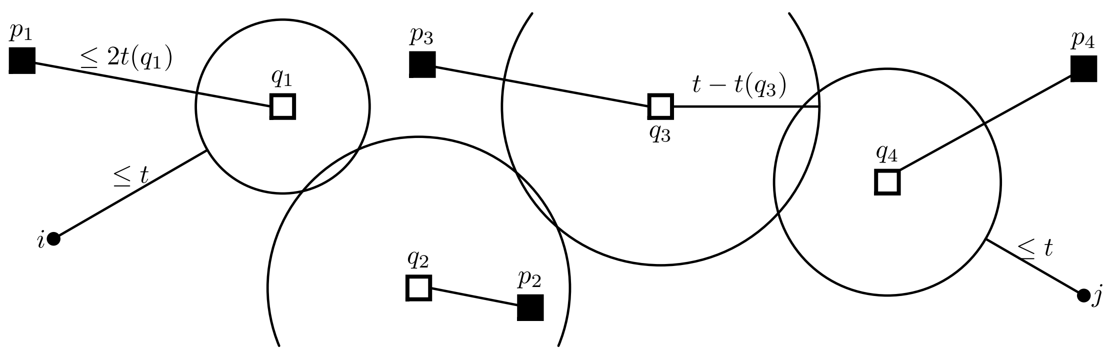

Figure 3.4: Clients i and j connected to a component C at time t.

Proof of Lemma 3.24. Note that since $|S_p| \geq M$ for any open center p, $|\text{Contrib}(C')|$ is always at least M, and hence $M/|\text{Contrib}(C')| \leq 1$. If j is not connected at time $t/3$, then $f_j(t/3) = 1$ and the inequality trivially holds.

If j is connected at time $t/3$, $f_j(t/3) = M/|S(C)|$ for some component C of $\mathcal{C}$. By Lemma 3.26, $\text{Contrib}(C')$ at time t contains all users that were contributing to

C at time $t/3$. ■

Using Lemma 3.24, we can bound the cost of the tree built.

Lemma 3.27 *The cost of the tree we buy is at most* $6\sum_j \alpha_j$.

Proof. At any time t, the users are able to collect $Ma(t)$ revenue from their contributions at time $t/3$. Hence

$$\int_0^\infty Ma(t)dt \le \int_0^\infty \sum_j f_j(t/3)dt = 3\sum_{j\in S} \alpha_j.$$

Since $M\int_0^\infty a(t)dt$ can pay for a half of the tree, the buying costs of our solution does not exceed $6\sum_j \alpha_j$. ■

3.3.5 Cost Recovery and Competitiveness

Theorem 3.28 (Cost recovery) *The cost of the solution constructed is at most* $\sum_j (3\alpha'_j + 6\alpha_j)$.

Proof. Lemma 3.27 bounds the buying cost by $6\sum_j \alpha_j$, while by Lemma 3.22, the rental costs can be bounded by $3\sum_j \alpha'_j$. ■

Theorem 3.29 (Competitiveness) *Every feasible solution to the single source rent or buy instance has cost at least* $\max\left(\frac{1}{2}\sum_{j\in S}\alpha_j,\ \sum_{j\in S}\alpha'_j\right)$.

One way to prove Theorem 3.29 is to write down an IP formulation of the single source rent or buy problem and give feasible dual solutions of cost $\sum_j \alpha_j/2$ and $\sum_j \alpha'_j$ respectively. In fact, the vectors $\alpha/2$ and α' can be completed to a feasible solution of the dual given by [88]. We sketch a direct proof of this theorem, which we believe is more intuitive.

Proof of Theorem 3.29. Consider an arbitrary solution SOL to the Single Source Rent-or-Buy instance. We will show that its cost must be greater or equal to $\frac{1}{2}\sum_{j\in S}\alpha_j$. (the proof for $\sum_{j\in S}\alpha'_j$ is simpler and is omitted).

Suppose we came up with a set of lengths $d_{j,e}$ of edges for each user j with the following properties:

1. For every edge e and every user j, $d_{j,e} \leq c_e$.
2. For every edge e, $\sum_{j\in S} d_{je} \leq 2Mc_e$.
3. The shortest path from every user j to the source s in the metric d_j has length at least α_j.

Again, consider an arbitrary solution SOL. Since every user must have a path to s, we have

$$\alpha_j \leq \sum_{e \text{ bought}} d_{j,e} + \sum_{e \text{ rented for } j} d_{j,e}.$$

Summing the above inequalities over all $j \in S$ we obtain

$$\sum_j \alpha_j \leq \sum_{e \text{ bought}} \sum_j d_{j,e} + \sum_j \sum_{e \text{ rented for } j} d_{j,e}.$$

We can write the cost of the solution as

$$\mathrm{cost(SOL)} = \sum_{e \text{ bought}} Mc_e + \sum_j \sum_{e \text{ rented for } j} c_e.$$

From properties (1) and (2) we obtain that

$$\begin{aligned} \sum_{e \text{ bought}} \sum_j d_{j,e} &\leq 2 \sum_{e \text{ bought}} Mc_e \\ \sum_j \sum_{e \text{ rented for } j} d_{j,e} &\leq \sum_j \sum_{e \text{ rented for } j} c_e. \end{aligned}$$

Combining these inequalities we obtain $\sum_j \alpha_j \leq 2\,\mathrm{cost(SOL)}$.

It remains to show that such $d_{j,e}$ exist. For a user j, let $S_j(t)$ be the set of locations *reachable* by j. A location p is reachable by j, if it lies inside j's ghost (i.e. $c(j,p) \leq t$), or it lies in a component that j is connected to. An edge e is *crossed* by the set S_j, if it has one endpoint inside s and one outside. We start with all $d_{j,e} = 0$. At every time t, we increase the length $d_{j,e}$ of each edge e that is crossed by S_j at rate $f_j(t)$.

Since at every time point we increase the length of all edges in some j-s cut at rate $f_j(t)$, the length of the shortest j-s path in the d_j metric will be equal to $\int_0^\infty f_j(t)dt = \alpha_j$. The boundary of each S_j moves at speed greater or equal to 1 (it may jump when components merge), hence no edge can accumulate $d_{j,e}$ that is greater than its original length c_e. Finally, consider property (2). Each edge e can be crossed by two types of boundaries, corresponding to ghosts of users and ghosts of components. Each location on e can be crossed by at most M user ghosts, contributing at most one unit of length per unit of time each, and at most one component ghost, contributing M units of length per unit of time. Hence, $\sum_j d_{j,e} \leq Mc_e + Mc_e = 2Mc_e$. ■

Combining Lemma 3.21, Theorem 3.28 and Theorem 3.29 we obtain

Theorem 3.30 *The cost shares $\xi(S,j) = 1/5\alpha'_j + 2/5\alpha_j$ are cross-monotonic, competitive and recover $1/15$ fraction of the cost of the solution constructed.*

3.4 Recent developments

There has been a lot of recent interest in cross-monotonic cost sharing methods among researchers. Let us mention some papers, most of them are in preparation or being submitted for publication.

Immorlica, Mahdian and Mirrokni [50] have developed an elegant technique for proving lower bounds on cost recovery, based on selecting a random "counterexample" subinstance of a problem from an instance with high symmetry. Using this technique they prove, among other things, that the 3-budget balanced cost sharing method for facility location from Section 3.2 is indeed optimal, and no α-budget balanced cross monotonic cost sharing exists for $\alpha < 3$. They also show that there is no $o(n)$-balanced cost sharing for the set cover problem, and no $o(\sqrt[3]{n})$-balanced cost sharing function for vertex cover.

Leonardi and Schäfer [68] derive a constant-budget balanced cross-monotonic cost sharing function for the connected facility location problem by generalizing our rent-or-buy algorithm from Section 3.3, while Xu [91] obtains a 6-budget-balanced cross-monotonic cost sharing for the multilevel facility location problem using the ghost technique.

Finally, Könemann et al. [63] recently obtained 2-budget-balanced cost sharing for the Steiner Network problem, resolving an open question whether Steiner network admits a constant-budget balanced cross-monotonic cost sharing.

Chapter 4

Approximation via Cost Sharing

In this chapter we explore an interesting and promising application of cost sharing to the design and analysis of approximation algorithms. We introduce the notion of *strict* cost sharing methods and show how they lead to simple yet powerful algorithms for a number of combinatorial optimization problems.

The first ideas leading to the technique of boosted sampling appear in a paper by Gupta, Kumar and Roughgarden [44]. Gupta, Kumar, Pál and Roughgarden [43] introduced the notion of strict cost shares in the context of an approximation algorithm for the Multicommodity Rent or Buy problem (which is the subject of Chapter 5). This chapter is based on a paper by Gupta, Pál, Ravi and Sinha [45], who generalized the notion of strictness, formally introduced the boosted sampling technique and used it to design approximation algorithms for stochastic versions of a number of classical NP-hard optimization problems.

4.1 Introduction

Infrastructure planning problems involve making decisions under uncertainty about future requirements; while more effective decisions can be made after the actual set of clients have materialized, the decision-making costs are inflated if deferred until then. The following simple two-stage model captures this tradeoff effectively. Future requirements are uncertain, but are assumed to be drawn from a known probability distribution (e.g., from demand forecasts, industry outlooks). In light of this information, an *anticipatory* part of the solution may be constructed in a first-stage at the current costs. Subsequently, the requirements facing the planner materialize in the form of a client set (drawn from the distribution), and the first-stage solution must be *augmented* to satisfy the revealed requirements. The elements chosen in this second stage are costlier than when chosen earlier, reflecting the need for careful (first-stage) planning. Given the uncertainty of the requirements, the traditional minimum-cost goal may be adapted to minimize the total *expected* cost of the solution.

As an example, consider the STOCHASTIC STEINER TREE problem that specifies an inflation parameter σ and a probability distribution π on the set of terminal nodes (which are clients) that have to be connected to the root in a given rooted discrete metric space. A subset of edges E_0 may be bought by paying the original lengths in the first stage. Once the actual set of terminals S is revealed, we must then buy the recourse edges E_S at σ times their lengths so that S is connected to the root by edges in $E_0 \cup E_S$. The objective is to minimize $c(E_0) + \mathbf{E}[\sigma\, c(E_S)]$. Here the expectation is over π, the randomness in the set of terminals revealed.

The framework is that of *two-stage stochastic optimization with recourse* [61, 60, 84] which may be paraphrased as "On Monday, we only know the input distribution on the clients, and we can buy some resources. On Tuesday, the client set is now completely specified, but things are now more expensive (in our case, by a factor σ); we <u>must</u> buy any additional resources needed to get a feasible solution to the instance."

Following this framework, in STOCHASTIC VERTEX COVER, the clients are edges to be covered, and we are given a probability distribution over sets of edges that will arrive; vertices become σ times more expensive after these edges are revealed. The STOCHASTIC FACILITY LOCATION problem on a metric space containing clients and facilities with opening costs defines a probability distribution over the set of clients that will require connection to open facilities. Opening facilities becomes σ times costlier in the second stage. The objective, in addition to expected cost of opening facilities, also includes expected connection costs of the revealed clients to their closest open facilities.

Our Results. In this paper, we give a simple yet general framework to translate approximation algorithms for deterministic optimization problems[1] into approximation algorithms for corresponding stochastic versions with second-stage inflation parameter σ. Given an α-approximation algorithm for the classical problem, one can use it in the following framework:

1. *Boosted Sampling:* Sample σ times from the distribution π to get sets of clients $D_1, \ldots, D_\sigma$.
2. *Building First Stage Solution:* Build an α-approximate solution for the clients $D = \cup_i D_i$.
3. *Building Recourse:* When actual future in the form of a set S of clients appears (with probability $\pi(S)$), augment the solution of Step 2 to a feasible solution for S.

Note that *we do not need to know the distribution* π *explicitly*; it could be a black-box from which we can draw samples. (In practice, these samples could be drawn from market predictions, or from Monte-Carlo simulations.) Thus we can sidestep the often-lethal problem of handling an exponential number of scenarios.

Informal Main Result 1.1 If the α-approximation algorithm $\mathcal{A}$ satisfies some technical properties (the problem is *sub-additive*, and $\mathcal{A}$ admits a β-strict *cost-*

[1] While the approximation algorithm solves the deterministic counterpart of the problem as opposed to the *stochastic* one, there is no requirement for this algorithm itself to be deterministic, e.g., randomized approximation algorithms can just as well be used in our framework.

sharing function[2]), then the above framework yields an $\alpha + \beta$ approximation for the stochastic version of the problem.

The framework is laid out in Section 4.2 and the formal result is Theorem 4.4. Using this framework, we show that stochastic variants of Steiner Tree, Facility Location, and Vertex Cover have constant-factor approximation algorithms; the details are in Sections 4.4-4.5.

We also consider the special case of *independent decisions*; in this, each client j has a probability π_j of arrival *independent* of other clients, and the probability $\pi(S)$ of the set S materializing is given by $\prod_{j \in S} \pi_j \prod_{j \notin S}(1 - \pi_j)$. For this model, we can also give a 8-approximation for the stochastic version of the Steiner Forest problem and improve the approximation ratios of the corresponding versions of Vertex Cover and Facility location to 3 and 6 respectively.

While a natural approach to utilizing an approximation algorithm for a deterministic problem is to set the client requirements at their expected value according to π, we note that this approach cannot yield bounded approximation ratios even in simple examples. Rather, using the full power of sampling in building the first stage solution gives a provably good solution as we demonstrate.

Related Work. The study of stochastic optimization [20, 54] dates back to the work of Dantzig [27] and Beale [17] in 1955; these papers defined the notions of *stochastic linear programming.* Stochastic LPs have been very widely studied since, and several gradient-based and decomposition-based approaches are known for two-stage versions of stochastic linear programming. On the other hand, only moderate progress has been reported for stochastic integer (and mixed-integer) programming in both theoretical and computational domains; see [84, 61] for details.

[2]These terms will soon be defined, in Definitions 4.1 and 4.2 respectively.

While stochastic scheduling problems have been studied extensively in the literature [78], the papers often try to identify cases where some standard scheduling policies yield optimal results, or the results hold for some special distributions (e.g., where job sizes are exponentially distributed), or focus on problems for which the deterministic versions are polynomial-time solvable. There are, of course, exceptions; see, e.g., [62, 36, 73, 87].

Very recently, there has been a surge of interest in stochastic versions of NP-hard problems. Dye et al. [29] demonstrate that while deterministic version of a problem called single resource service-provision admits a FPTAS, allowing stochastic demands makes it strongly NP-hard. They also give an approximation algorithm for the stochastic problem. Ravi and Sinha [81], and independently, by Immorlica et al. [49] look at versions of our model with some restrictions on the distribution π, while we consider *arbitrary distributions* π. In particular, they consider the following cases.

- the *scenario model*, where the distribution π has its support on a family $\mathcal{F}$ of possible subsets explicitly given as part of the input (and hence the algorithms are allowed to take time $\text{poly}(|\mathcal{F}|, n)$), and
- the *independent decisions* model, where each element j has an associated probability π_j, and the probability of a set $\pi(S) = \prod_{j \in S} \pi_j \prod_{j \notin S}(1 - \pi_j)$. (I.e., the sets are chosen by flipping a coin independently for each element.)

Since our algorithms in Sections 4.4 and 4.5 work for *arbitrary distributions*, our theorems hold in both the above models as well. In particular, our 3.55- and 3-approximations for stochastic Steiner Tree and Vertex Cover in the independent model improve upon the $O(\log n)$- and 6.3-approximations in [49] respectively.

One can define (as in [81]) other stochastic variants of the problems we define here: e.g., one can imagine that there are multiple inflation parameters, and that instead of all things getting dearer by σ, different parts of the problem change in different ways. This work leaves open the question of whether our framework can be extended to handle such multiple-parameter stochastic problems.

Stochastic Steiner Tree appears similar to the *maybecast* problem of Karger and Minkoff [55]; however, the latter is a single-stage optimization problem. Finally, though some of our techniques, including strict cost-shares come from the work of [44, 43], the problems considered there are deterministic optimization problems.

4.2 Model and Notation

We define an abstract combinatorial optimization problem that we will adapt to a stochastic setting. To define Π, a combinatorial optimization problem, let U be the universe of *clients* (or requirements), and let X be the set of *elements* that we can purchase. For any $F \subseteq X$, let $c(F)$ denote the *cost* of the element set F. Given a set of clients $S \subseteq U$, let $\mathsf{Sols}(S) \subseteq 2^X$ be the set of *feasible solutions* for the client set S. The *deterministic version* $\mathsf{Det}(\Pi)$ specifies a fixed subset of clients $S \subseteq U$, and the objective is to return $F \in \mathsf{Sols}(S)$ with least cost. We denote by $\mathsf{OPT}(S)$ a solution in $\mathsf{Sols}(S)$ of minimum cost.

Definition 4.1 We require all our problems Π to satisfy *sub-additivity.* This property states that if S and S' are two sets of clients with solutions $F \in \mathsf{Sols}(S)$ and $F' \in \mathsf{Sols}(S')$, then we must have that (i) $S \cup S'$ is a legal set of clients for Π, and (ii) $F \cup F' \in \mathsf{Sols}(S \cup S')$.

For example, the Steiner Tree problem on a graph $G = (V, E)$; the clients $U = V$ are the set of vertices, and the elements X are edges E of the graph. To ensure sub-additivity, we require a *root vertex* r. The cost of a set of edges $F \subseteq X$ is $c(F) = \sum_{e \in F} c_e$. Given a set $S \subseteq V$ of *terminals*, the solutions are $\mathsf{Sols}(S) = \{T \mid T \text{ connects all vertices of } S \cup \{r\}\}$.

Given any problem Π, we can define a variant adapted from the framework of *two-stage stochastic programming with recourse* as follows.

- There will be *two stages* of purchasing. Let $\sigma \geq 1$ be a given *inflation parameter*; every element $x \in X$ costs c_x in the first stage but costs $\sigma\, c_x$ in the second.
- In the first stage, the algorithm is only given access to an oracle that can draw from the probability distribution $\pi : 2^U \to [0, 1]$ in time $poly(|U|)$. It can then construct a *first-stage solution* by buying a set of elements F_0 at cost $c(F_0)$.
- In the second stage, one set $S \subseteq U$ of clients is *realized* according to the distribution π; i.e., the probability that S is realized is $\pi(S)$. We assume that this set S is conditionally independent of any of our actions in the first stage. Now the *second-stage solution* (also called the *recourse*) consists of a set of elements F_S purchased at the inflated cost $\sigma\, c(F_S)$. The union of the sets $F_0 \cup F_S$ must lie in $\mathsf{Sols}(S)$, i.e., the first and second stage solutions taken together give a feasible solution for the realized set of clients.

The objective of an algorithm for the stochastic problem $\mathsf{Stoc}(\Pi)$ is to select a set F_0, and then, given a set S from the distribution π, to select F_S to minimize the expected cost of the solution:

$$c(F_0) + \sum_{S \subseteq U} \pi(S) \sigma c(F_S). \tag{4.1}$$

Hence the stochastic version of the Steiner Tree problem allows us to purchase some edges F_0 in the first stage, and once the set of realized clients $S \subseteq V$ is revealed, to buy some more edges F_S so that $F_0 \cup F_S$ contains a tree spanning S.

4.2.1 Cost sharing functions

We now define the *cost-shares* that are used crucially to bound the performance of our approximation algorithms. Loosely, a cost-sharing function is one that divides the cost of a solution $F \in \mathsf{Sols}(S)$ among the client set S. Cost-sharing functions have long been used in the context of game-theory [51, 53, 75, 77, 92]. We will use (a slight variant of) a cost-sharing function defined recently by Gupta et al. [43]: in contrast to previous cost-sharing mechanisms, this cost-sharing function is defined relative to a fixed (approximation) algorithm for the problem Π.

Definition 4.2 Given an α-approximation algorithm $\mathcal{A}$ for the problem Π, the function $\xi : 2^U \times U \to \mathbb{R}_{\geq 0}$ is a *β-strict cost-sharing function* if the following properties hold:

P1. For a set $S \subseteq U$, $\xi(S, j) > 0$ only for $j \in S$.

P2. For a set $S \subseteq U$, $\sum_{j \in S} \xi(S, j) \leq c(\mathsf{OPT}(S))$. **(fairness)**

P3. If $S' = S \uplus T$, then $\sum_{j \in T} \xi(S', j) \geq (1/\beta)\times$ cost of *augmenting* the solution $\mathcal{A}(S)$ to a solution in $\mathsf{Sols}(S')$. **(strictness)**

Formally, there must exist a polynomial time algorithm $\mathsf{Aug}_{\mathcal{A}}$ which can augment $\mathcal{A}(S)$ to a solution in $\mathsf{Sols}(S')$ at cost at most $\beta \sum_{j \in T} \xi(S', j)$.

Define the function $\xi(S, A)$ as the sum $\sum_{j \in A} \xi(S, j)$.

Remark 4.3 Note that one possible algorithm $\mathsf{Aug}_{\mathcal{A}}$ is obtained by just zeroing out the costs of elements already picked in $\mathcal{A}(S)$, and running $\mathcal{A}$ again. In all cases we consider, there are natural algorithms $\mathcal{A}$ for which this $\mathsf{Aug}_{\mathcal{A}}$ ensures strictness; however, in this paper, we choose algorithms $\mathcal{A}$ and $\mathsf{Aug}_{\mathcal{A}}$ that complement each other and give better approximation ratios.

4.3 Approximation via Cost Sharing

In this section, we give a general technique for converting an approximation algorithm $\mathcal{A}$ for a deterministic problem $\mathsf{Det}(\Pi)$ into an approximation algorithm for the stochastic version $\mathsf{Stoc}(\Pi)$, provided the problem Π satisfies sub-additivity, and there is a cost-sharing function ξ that is strict w.r.t. $\mathcal{A}$.

4.3.1 Algorithm: Boosted Sampling

Given an instance of a stochastic problem $\mathsf{Stoc}(\Pi)$, the goal of the first stage is to buy the elements that will be useful for the unknown client set realized in the second stage. Since our algorithm is not clairvoyant and hence cannot see the future, the next best thing it can do is to sample from the distribution π, and use the samples as an indication of what the future holds. This simple idea is the basis of our method.

A naïve attempt would be to sample once from the distribution and use the set obtained as our prediction for the future: however, this is not aggressive enough in that it ignores the fact that the future is more expensive by a factor of σ. In fact, as $\sigma \to \infty$, the optimal solution would be to assume that every client in U will be realized and must be accounted for in the first stage itself. Motivated by these concerns, the algorithm for the problem $\mathsf{Stoc}(\Pi)$ is stated below, in terms of the α-approximation algorithm $\mathcal{A}$ for Π, which comes equipped with a β-strict cost-sharing function (and hence an associated augmentation algorithm $\mathsf{Aug}_{\mathcal{A}}$).

Our algorithm requires a number of samples that is linear in σ. Indeed, if all we have is a sampling access to the distribution π, it is not hard to see that $\Omega(\sigma)$ samples are needed. In many practical situations, this may not be a concern, unless the value of σ is exorbitant. In some cases (e.g., see Section 4.6), additional information about the distribution π can be used to make the running time independent of σ. Let $\lfloor\sigma\rfloor$ denote σ rounded down to the nearest integer.

Algorithm $\mathsf{Boost\text{-}and\text{-}Sample}(\Pi)$

1. Draw $\lfloor\sigma\rfloor$ independent samples $D_1, D_2, \ldots, D_{\lfloor\sigma\rfloor}$ of the realized clients by sampling from the distribution π. Let $D = \cup_i D_i$.
2. Using the algorithm $\mathcal{A}$, construct an α-approximate *first-stage solution* $F_0 \in \mathsf{Sols}(D)$.
3. If the client set S is realized in the second stage, use the augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ from (P3) to compute F_S such that $F_0 \cup F_S \in \mathsf{Sols}(S)$.

The following theorem is the main result of the paper:

Theorem 4.4 *Consider a combinatorial optimization problem* Π *that is sub-additive, and let* $\mathcal{A}$ *be an* α*-approximation algorithm for its deterministic version* $\mathsf{Det}(\Pi)$ *that admits a* β*-strict cost sharing function. Then* $\mathsf{Boost\text{-}and\text{-}Sample}(\Pi)$ *is an* $(\alpha+\beta)$*-approximation algorithm for* $\mathsf{Stoc}(\Pi)$.

We will prove Theorem 4.4 in the rest of the section. In the next section, we illustrate an application of this theorem to obtain an approximation algorithm for the Stochastic Steiner Tree. In subsequent sections, we go on to consider several other problems, and show that their approximation algorithms and attendant cost sharing functions provide approximation algorithms for the corresponding stochastic versions.

Proof of Theorem 4.4: To simplify the notation, we shall assume that σ is an integer, i.e. $\sigma = \lfloor \sigma \rfloor$. It is straightforward to verify that the proof can be carried out for arbitrary real value of σ.

We will bound the expected costs of our first and second-stage solutions separately. Let F_0^* be the first-stage component of the optimal solution, and F_S^* be the second-stage component if the set of realized clients is S. Hence

$$\text{the optimal cost } Z^* = c(F_0^*) + \textstyle\sum_S \pi(S)\, \sigma\, c(F_S^*). \tag{4.2}$$

Let us denote $Z_0^* = c(F_0^*)$ and $Z_r^* = \sum_S \pi(S)\, \sigma\, c(F_S^*)$.

First stage: We claim that there is an element $\widehat{F}_1 \in \mathsf{Sols}(D)$ such that $\mathbf{E}[c(\widehat{F}_1)] \leq Z^*$. Indeed, define $\widehat{F}_1 = F_0^* \cup F_{D_1}^* \cup F_{D_2}^* \cup \ldots \cup F_{D_\sigma}^*$. The fact that $\widehat{F}_1 \in \mathsf{Sols}(D)$ follows from sub-additivity of the problem Π. Therefore,

$$\begin{aligned} \mathbf{E}_D[c(\widehat{F}_1)] &\leq c(F_0^*) + \mathbf{E}_D[\textstyle\sum_{i=1}^{\sigma} c(F_{D_i}^*)] \\ &= c(F_0^*) + \textstyle\sum_{i=1}^{\sigma} \mathbf{E}_{D_i}[c(F_{D_i}^*)] \\ &= c(F_0^*) + \sigma \textstyle\sum_S \pi(S)\, c(F_S^*) \quad = Z^*, \end{aligned}$$

the penultimate equality following from the fact that each of the D_i's are chosen from the probability distribution π. Since we have an α-approximation algorithm for $\mathsf{Det}(\Pi)$, our solution F_0 satisfies $\mathbf{E}[c(F_0)] \leq \alpha c(\widehat{F}_1)$, which in turn is at most $\alpha\, Z^*$, bounding our first stage costs.

Second stage: Let S be the set of realized clients, and let F_S be the result of our algorithm $\mathsf{Aug}_{\mathcal{A}}$ such that $F_0 \cup F_S \in \mathsf{Sols}(S)$. We need to bound our expected second stage cost, which is $\sigma\, \mathbf{E}[c(F_S)]$. The β-strictness of the cost sharing function ξ implies that $c(F_S) \leq \beta\, \xi(D \cup S, S)$. In fact, we even have $c(F_S) \leq \beta\, \xi(D \cup S, S \setminus D)$.

Consider the following alternate probabilistic process to generate the sets D_i and the set S: Draw $\sigma + 1$ independent samples $\widehat{D}_1, \widehat{D}_2, \ldots, \widehat{D}_{\sigma+1}$ from the distribution π. Now choose a random value K uniformly from $\{1, 2, \ldots, \sigma+1\}$, and set $S = \widehat{D}_K$ and $D = \cup_{i \neq K} \widehat{D}_i$. This process is identically distributed to the original process, since we are picking the sets independently. Let $\widehat{\mathbb{D}}$ be the union of *all* the $\widehat{D}_i$'s, and let $\widehat{\mathbb{D}}_{-i}$ be the union $\cup_{l \neq i} \widehat{D}_l$ of all the sets except $\widehat{D}_i$.

Since the cost sharing function is fair (Property P2), we have

$$\textstyle\sum_{i=1}^{\sigma+1} \xi(\widehat{\mathbb{D}}, \widehat{D}_i \setminus \widehat{\mathbb{D}}_{-i}) \leq c(\mathsf{OPT}(\widehat{\mathbb{D}})).$$

By our random choice of K, we get

$$\mathbf{E}_K[\xi(\widehat{\mathbb{D}}, \widehat{D}_K \setminus \widehat{\mathbb{D}}_{-K})] \leq \tfrac{1}{\sigma+1} c(\mathsf{OPT}(\widehat{\mathbb{D}}))$$

Since the alternate process is probabilistically identical to the one we used to pick D and S,

$$\begin{aligned} \mathbf{E}_{D,S}[\xi(D \cup S, S \setminus D)] &= \mathbf{E}_{\widehat{\mathbb{D}},K}[\xi(\widehat{\mathbb{D}}, \widehat{D}_K \setminus \widehat{\mathbb{D}}_{-K})] \\ &\leq \tfrac{1}{\sigma+1}\, \mathbf{E}_{\widehat{\mathbb{D}}}[c(\mathsf{OPT}(\widehat{\mathbb{D}}))] \end{aligned} \tag{4.3}$$

To complete the argument, we now show that $\mathbf{E}_{\widehat{\mathbb{D}}}[c(\mathsf{OPT}(\widehat{\mathbb{D}}))] \leq \frac{\sigma+1}{\sigma} Z^*$. To derive a feasible solution to $\widehat{\mathbb{D}}$, define $\widehat{F}_2 = F_0^* \cup F^*_{\widehat{D}_1} \cup F^*_{\widehat{D}_2} \cup \ldots \cup F^*_{\widehat{D}_{\sigma+1}}$. Again, the fact that $\widehat{F}_2 \in \mathsf{Sols}(\widehat{\mathbb{D}})$ follows from sub-additivity of Π. Thus we have

$$\begin{aligned} \mathbf{E}_{\widehat{\mathbb{D}}}[c(\mathsf{OPT}(\widehat{\mathbb{D}}))] &\leq c(F_0^*) + \textstyle\sum_{i=1}^{\sigma+1} \mathbf{E}_{\widehat{D}_i}[c(F^*_{\widehat{D}_i})] \\ &\leq Z_0^* + (\sigma+1)\, \tfrac{Z_r^*}{\sigma} \\ &\leq \tfrac{\sigma+1}{\sigma}\,(Z_0^* + Z_r^*) = \tfrac{\sigma+1}{\sigma}\, Z^*. \end{aligned} \tag{4.4}$$

Thus $\mathbf{E}_S[c(F_S)] \leq \beta\, \mathbf{E}_{D,S}[\xi(D \cup S, S \setminus D)]$, which using (4.3) and (4.4), is bounded by $\frac{\beta}{\sigma} Z^*$. Finally, since our second stage cost is $\sigma\, c(F_S)$, our expected second stage cost is $\mathbf{E}_S[\sigma\, c(F_S)] \leq \beta\, Z^*$.

Putting together the first and second stage costs gives the bound claimed in the theorem. ■

4.4 Stochastic Steiner Trees

In the classical deterministic Steiner Tree problem, we are given a set of vertices V, and the costs c_e on edges satisfy the triangle inequality. (This assumption is without loss of generality, since we can take the metric completion of the graph.) We assume there is a designated *root* vertex r. Given a set of terminals (i.e., the clients), the goal is to buy a set of edges (the elements) of minimum cost so that the terminals and the root r lie in a connected component. Note that the presence of the root ensures that the problem is sub-additive.

Now let us consider the problem Stoc(Steiner Tree): in the first stage, we can buy some edges F_0 at cost $\sum_{e \in F_0} c_e$. In the second stage, a set of terminals $S \subseteq V$ is realized with probability $\pi(S)$, after which we may buy some more edges to connect the terminals to the root; however, these edges must be bought at cost $\sigma\, c_e$ each.

Theorem 4.5 *There exists a* 2*-approximate algorithm* $\mathcal{A}$ *for* Steiner Tree*, along with a cost sharing function* ξ *that is* 2*-strict for* $\mathcal{A}$*.*

Proof. The algorithm $\mathcal{A}$ is simply Prim's algorithm [79] for minimum spanning tree; given a set of terminals S, it ignores the vertices not in $S \cup \{r\}$, and builds an MST on $S \cup \{r\}$. It is well-known that the cost $c(\mathcal{A}(S))$ of any MST is within a factor of 2 of the cost of the optimal Steiner tree $\mathsf{OPT}(S)$.

Given an MST $\mathcal{A}(S)$ on the set of terminals S, let us imagine it to be rooted at r; for $j \in S$, set $\xi(S, j)$ to be *half* the cost of the edge connecting j to its parent in $\mathcal{A}(S)$, which we call j's *parental edge* in $\mathcal{A}(S)$. Clearly, if $j \notin S$, then $\xi(S, j) = 0$. By definition, $\sum_{j \in S} \xi(S, j) = \frac{1}{2}\, c(\mathcal{A}(S))$; since the MST is a 2-approximation to the Steiner tree problem, this implies that $\frac{1}{2}\, c(\mathcal{A}(S)) \leq c(\mathsf{OPT}(S))$, and hence ξ is fair.

Finally, to prove the 2-strictness, consider a set $S' = S \uplus T$. The augmenting procedure $\mathsf{Aug}_{\mathcal{A}}$ basically zeroes out the edges of $\mathcal{A}(S)$ and runs Prim's algorithm; i.e., it takes the solution $\mathcal{A}(S)$, and for each $j \in T$, adds the parental edge of j in $\mathcal{A}(S \cup T)$. We claim this gives a solution in $\mathsf{Sols}(S \cup T)$. (Indeed, each vertex $j \in T$ whose parent in $\mathcal{A}(S \cup T)$ was in $S \cup \{r\}$ will now be connected to $\mathcal{A}(S)$, and hence to r; the general argument follows by a simple induction.) Since these edges cost $2 \times \sum_{j \in T} \xi(S \cup T, j)$, we have proved the theorem.

Note that the argument for strictness only required that each vertex in the solution $\mathcal{A}(S)$ was connected to the root, and hence the same cost shares are also 2-strict for *any* heuristic for Steiner Tree. Now, using the 1.55-approximation for Steiner Tree [82] and Theorem 4.4, we obtain the following improved theorem:

Theorem 4.6 *There is a* 1.55*-approximation algorithm for* Steiner Tree*, along with a cost-sharing function* ξ *that is* 2*-strict for it. Hence, there is a* 3.55*-approximation algorithm for* Stoc(Steiner Tree)*.*

This improves on the $O(\log n)$-approximation for the *independent decisions* version of Stoc(Steiner Tree) given by Immorlica et al. [49].

4.5 Other Applications

This section will be devoted to looking at several other (deterministic) problems Π; for each problem, we will give an α-approximation algorithm $\mathcal{A}$ and its accompanying β-strict cost-share function.

4.5.1 Facility Location

An instance of Facility Location is given by a set of facilities F and a set of clients S. The distances c_{ij} between any pair of points i, j from $F \cup S$ form a metric. Each facility p has opening cost f_p; the goal is to open a subset of facilities F' to minimize the opening costs plus the sum of distances from each client to its closest open facility:

$$\sum_{p \in F'} f_p + \sum_{j \in S} c(j, F').$$

The main result for Stoc(Facility Location) is the following:

Theorem 4.7 *The cost sharing function ξ given by Pál and Tardos is 5.45-strict for the 3-approximation algorithm of Mettu and Plaxton. Hence, there is a 8.45-approximation algorithm for* Stoc(Facility Location).

Proof. Let us briefly review the algorithm of Mettu and Plaxton [70], and the cost sharing defined by Pál and Tardos [77]. Let S be a set of clients. For a facility p, let $B(p, \tau)$ be a ball with center p and radius τ. We define the *opening time* $t_p(S)$ of a facility p w.r.t. the set of clients S to be the unique radius τ such that

$$f_p = \sum_{j \in B(p,\tau) \cap S} (\tau - c(j, p)). \tag{4.5}$$

Let the set $C_p(S) = \{j \in S \mid c(j, p) < t_p(S)\}$ be called the *contributing set* for p. Note that if we charge each client in C_p the amount $t_p(S)$, we exactly recover the facility cost of p plus the cost of assigning clients in C_p to p. We drop the set of clients from the notation, and say t_p instead of $t_p(S)$ when there is no danger of confusion. The cost shares of clients are then defined as

$$\xi(S, j) = \min_{p \in F} \{ \max(t_p(S), c(j, p)) \}. \tag{4.6}$$

Intuitively, the contribution of user j towards the facility p should be either t_p if $j \in C_p$, or the connection cost $c(j,p)$ if $j \notin C_p$. The client can (and does) choose to contribute only to the least demanding facility; the facility p for which this minimum is attained is called the *primary* facility of j (in the run on S). A facility p is said to be *well-funded* if $\xi(S,j) \geq t_p(S)/3$ for all $j \in C_p$.

The algorithm $\mathcal{A}$ we use is a slight modification of the algorithm of Mettu and Plaxton; given a set of clients S, $\mathcal{A}$ considers all the well-funded facilities p in order of increasing opening time $t_p(S)$. For each such (well-funded) facility p, the algorithm declares it *open* if there are no previously opened facilities within a radius $2\,t_p(S)$ around p.

For each open facility p, $\mathcal{A}$ assigns all clients in C_p to p. (By construction, the sets C_p for open facilities p are disjoint.) It then assigns each client not lying in any C_p to its closest open facility. The following facts can be derived from the arguments in [70] and [77]:

1. For each open facility p, the cost shares $\xi(S, C_p)$ of the clients in C_p pay $1/3$ of f_p plus their assignment cost. (See [77, Lemma 2.4] and the preceding discussion therein.)
2. For each facility p, there exists a *well-funded* facility q (possibly $p = q$) such that $c(p,q) \leq 2\,(t_p - t_q)$. (Note that it must be that $t_q \leq t_p$.)
3. For each facility p, there exists an *open* facility q within a distance of $2t_p$. Hence, if p is a primary facility for some client j, then $c(j,q) \leq 3\xi(S,j)$.

To show strictness of ξ, we must specify the algorithm $\mathsf{Aug}_{\mathcal{A}}$ which augments a solution $\mathcal{A}(S)$ to cover a set of new clients T with $S \cap T = \emptyset$. In the following, let $C_p = C_p(S \cup T)$ denote the contributor set of a facility p in the run $\mathcal{A}(S \cup T)$. Similarly, when we say a facility p is well-funded, we mean that p is well-funded in the run $\mathcal{A}(S \cup T)$. A facility p is called *T-heavy* if $|C_p \cap T| \geq b|C_p|$ (where the parameter $b \in (0,1)$ will be specified later), and is *T-light* otherwise. Note that a T-light facility must have $|C_p \cap S| \geq (1-b)|C_p|$.

Claim 4.8 *If p is a T-light facility, then*

$$t_p(S) \leq \tfrac{1}{1-b}\, t_p(S \cup T) \; - \; \tfrac{1}{|C_p \cap S|} \textstyle\sum_{j \in C_p \cap T} c(j,p).$$

Proof. Consider the set $C_p = \{j \in S \cup T \mid c(j,p) < t_p\}$; by the definition (4.5),

$$f_p + \sum_{j \in C_p} c(j,p) = |C_p|\, t_p(S \cup T)$$

Since p is T-light, $|C_p \cap S| \geq (1-b)\,|C_p|$.

$$\begin{aligned} f_p + \sum_{j \in C_p \cap S} c(j,p) &= |C_p|\, t_p(S \cup T) - \sum_{j \in C_p \cap T} c(j,p) \\ &\leq |C_p \cap S|\, \frac{t_p(S \cup T)}{1-b} - \sum_{j \in C_p \cap T} c(j,p). \end{aligned}$$

which means facility p was already paid for at time

$$t_p(S \cup T)/(1-b) - \sum_{j \in C_p \cap T} c(j,p)/(|C_p \cap S|)$$

in the run $\mathcal{A}(S)$, proving the claim.

Augmentation procedure $\mathsf{Aug}_{\mathcal{A}}$: To augment $\mathcal{A}(S)$ to cover T as well, we pick a subset of well-funded T-heavy facilities to open greedily in a manner very similar to that in $\mathcal{A}(S \cup T)$: we consider all well-funded T-heavy facilities in order of increasing $t_p(S \cup T)$, and open a facility p if there is no facility q already open within a radius $2\, t_p(S \cup T)$ of p. (Note that q may have been opened in either $\mathcal{A}(S)$, or in the augmenting phase before p was considered.) We never open any T-light or non-well-funded facilities. At the end of this procedure, for a client $j \in C_p$ whose p is open, we assign j to p; else we assign j to the closest open facility.

Claim 4.9 *The augmentation cost for a set T is at most* $(3+\sqrt{6})\sum_{j\in T}\xi(S\cup T,j)$.

Proof. Firstly, consider any well-funded T-heavy facility p. Since p is T-heavy, the shares of clients in $C_p \cap T$ can pay for a $b/3$ fraction of the facility cost plus their own connection costs. Hence we must consider clients j whose primary facility p is either not well-funded or not T-heavy. We claim that in both cases there must be a facility close to p opened either by $\mathcal{A}(S)$, or in the augmenting phase. Note that by the properties of the algorithm $\mathcal{A}$, there is a well-funded facility q such that $t_q(S\cup T) \leq t_p(S\cup T)$ and $c(p,q) \leq 2\,(t_p(S\cup T) - t_q(S\cup T))$.

Now, if q is T-heavy, by the properties of our augmentation procedure, there must a facility r that was open in the augmentation step such that $c(q,r) \leq 2\,t_q(S\cup T)$. On the other hand, if q is T-light, we have that $t_q(S) \leq t_q(S\cup T)/(1-b)$ by Claim 4.8 above. Thus, in the run $\mathcal{A}(S)$, there must be an open facility r such that $c(q,r) \leq 2\,t_q(S) \leq (2/(1-b))\,t_q(S\cup T)$.

In both cases, the assignment cost of the client j is bounded by

$$
\begin{aligned}
c(j,r) &\leq c(j,p) + c(p,q) + c(q,r) \\
&\leq c(j,p) + 2(t_p(S\cup T) - t_q(S\cup T)) + \frac{2}{1-b}\,t_q(S\cup T) \\
&\leq c(j,p) + \frac{2}{1-b}\,t_p(S\cup T) \\
&\leq (1+\frac{2}{1-b})\,\xi(S\cup T,j).
\end{aligned}
$$

To balance $3/b$ and $(1+2/(1-b))$, we can now pick $b = 3-\sqrt{6}$ to get the desired result.

Since $\beta = 3+\sqrt{6} \leq 5.45$, this proves the theorem.

4.5.2 Vertex Cover

In the Vertex Cover problem, we are given a graph $G = (V, E)$ with costs c_v on vertices. The clients are the edges, and our goal is to choose a subset V' of the vertices so that each edge is *covered*; i.e., at least one of its adjacent vertices is chosen. In the stochastic version, we pay c_v for picking a vertex v in the first phase, and $\sigma\, c_v$ for picking it in the second phase. We will prove the following theorem:

Theorem 4.10 *There is a* 8*-approximation algorithm for* $\mathsf{Stoc}(\textsc{Vertex Cover})$.

Before we do this, let us define a version of the problem called *Relaxed Stochastic Vertex Cover*. In the relaxed version of the stochastic problem, we are allowed to make payments to a vertex in both stages, with $p^1(v)$ and $p^2(v)$ being payments made to v in the first and second stage respectively. A vertex v is chosen if and only if $p^1(v) + p^2(v)/\sigma \geq c_v$. Again, given a set of realized edges S, the set of chosen vertices must form a feasible vertex cover for S. The cost of our solution is just the sum of payments, i.e. $\sum_{v \in V} p^1(v) + p^2(v)$, and the goal is again to minimize the expected cost.

Note that by requiring that $p^1(v) \in \{0, c_v\}$ and $p^2(v) \in \{0, \sigma c_v\}$, we get back to our usual stochastic framework, and hence the relaxed problem allows us to make partial commitments to vertices in the first stage. However, it turns out that we can convert any algorithm $\mathcal{A}$ for the relaxed problem into an algorithm $\mathcal{A}'$ for the unrelaxed version with the same expected cost. Indeed, if $p^1(v)$ is the amount of money placed on vertex v by $\mathcal{A}$ in the first stage, the algorithm $\mathcal{A}'$ picks the vertex v in its first stage with probability $\min\{1, p^1(v)/c_v\}$. In the second stage, $\mathcal{A}'$ selects the vertex v if v was selected by $\mathcal{A}$ (that is, $p_v^1 + p_v^2/\sigma \geq c_v$), and if $\mathcal{A}'$ has not already selected it in the first stage. By linearity of expectations, the expected cost incurred by $\mathcal{A}'$ in each phase is at most the cost incurred by $\mathcal{A}$ in that phase. Thus it suffices to give an algorithm and a cost sharing function for relaxed vertex cover.

The Algorithm $\mathcal{A}$: We use a standard primal-dual 2-approximation algorithm $\mathcal{A}$ for vertex cover. Let $S \subseteq E$ be the set of edges in the instance. For each edge e, we have a dual variable y_e, initially set to 0. We simultaneously raise all dual variables at a uniform rate. A vertex v becomes *tight* when the duals of edges adjacent to it can pay its cost, i.e. when $\sum_{e \in \delta(v)} y_e = c_v$. When a vertex v becomes tight, we *freeze* all edges adjacent to it, i.e., we stop raising their dual variables. We continue raising the dual variables of all unfrozen edges, until all edges become frozen.

Output: The algorithm places payments $p(v) = \sum_{e \in \delta_v} y_e$ on each vertex $v \in V$. Since each edge is adjacent to some tight vertex v, it has been paid c_v and hence bought outright; thus the solution is feasible for S.

The Cost Shares: Define $\xi(S, e) = y_e$; since each edge pays both its endpoints, it holds that $\sum_{v \in V} p(v) = 2 \sum_{e \in S} y_e$. Furthermore, $\sum_e \xi(S, e)$ is just the LP dual value, and hence at most $\mathsf{OPT}(S)$.

Clearly, the algorithm $\mathcal{A}$ is a 2-approximation for the vertex cover problem. To prove Theorem 4.10, it suffices to prove the strictness of ξ for $\mathcal{A}$.

Theorem 4.11 *The cost shares ξ defined above are 6-strict with respect to the algorithm $\mathcal{A}$.*

Proof. Let S and T be two disjoint sets of edges. To augment the solution $\mathcal{A}(S)$ to handle T as well, the augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ looks at the (relaxed) payment function $p : V \to \mathbb{R}_{\geq 0}$ for the set of edges S, and runs the algorithm $\mathcal{A}$

on the set of edges T with the reduced costs $c'_v = c_v - p(v)$. To prove strictness, we need to compare this augmentation cost to the cost share $\xi(S \cup T, T)$. To this end, we shall compare several runs of $\mathcal{A}$ on different related inputs.

- **Run** R_1**:** This is the run of $\mathcal{A}$ with original costs c_v on the set $S \cup T$. Let y^1_e be the duals produced. Let us define payments $p^1_S(v) = \sum_{e \in \delta(v) \cap S} y_e$ and $p^1_T(v) = \sum_{e \in \delta(v) \cap T} y_e$ respectively.

  Note that $p^1 = p^1_{S \cup T} = p^1_S + p^1_T$ is exactly the payment function computed by $\mathcal{A}$. Furthermore, this is the run that computes the cost-shares $\xi(S \cup T, T)$.

- **Runs** R_S **and** R_T**:** The run R_S is the run of $\mathcal{A}$ on the set of edges S, but with costs $c^S = c - p^1_T$ (i.e., reduced by the payments of T in R_1). Similarly the run R_T is on the edges T, with reduced costs $c^T = c - p^1_S$.

- **Run** R_2**:** This is $\mathcal{A}$'s run on the edge set S, with original costs c, and hence corresponds to the actual run of the first stage. Let y^2 be the duals and p^2 the payments computed.

- **Run** R_3**:** This is on the edge set T, with reduced costs $c^3 = c - p^2$; hence, this corresponds to the augmentation step for T. Again, y^3 and p^3 are the duals and the payments.

By the definition of R_S, the freezing time of all edges $e \in S$ in the two runs R_S and R_1 is the same; hence the dual y^S is just the dual y^1 restricted to the set S, and $p^1_S = p^S$. Similarly, the dual y^T from the run R^T is identical to the dual y^1 restricted to T, and $p^1_T = p^T$. We claim that, to prove the theorem, it suffices to prove the following claim:

Claim 4.12 $\sum_{v \in V} p^3(v) \leq 3 \sum_{v \in V} p^1_T(v)$.

Before we prove this claim, let us see how it proves Theorem 4.11. Note that cost of the augmentation run R_3 is exactly $\sum_v p^3(v)$, while the cost shares

$$\xi(S \uplus T, T) = \sum_{e \in T} y_e = \frac{1}{2} \sum_{v \in V} p^1_T(v).$$

Hence $\xi(S \cup T, T)$ can defray at least one-sixth of the cost of the run R_3, proving the theorem.

Proof of Claim 4.12: The proof relies on the following Lipschitz-type property of the algorithm $\mathcal{A}$: imagine the vertex costs to be vectors in $\mathbb{R}^{|V|}$, and suppose the costs change by an amount ϵ (in their L_1-distance), then we claim that the payments do not change by more than $2\,\epsilon$. Formally,

Claim 4.13 (Lipschitz continuity) *Consider two runs R and $\widehat{R}$ of $\mathcal{A}$ with the same edge set S on two different cost vectors c and $\widehat{c}$, and let p and $\widehat{p}$ be the two vectors of payments computed. If we define Δ so that $(p - \widehat{p}) = (c - \widehat{c}) + \Delta$, then $\|\Delta\|_1 \leq \|c - \widehat{c}\|_1$.*

The proof of the Lipschitz condition follows below; but let us use it to complete this proof. First, we use it to compare the runs R_S and R_2 (both being defined on the edge set S): define Δ_1 so that

$$p^2 - p^1_S = c - (c - p^1_T) + \Delta_1 = p^1_T + \Delta_1; \tag{4.7}$$

then Claim 4.13 implies that $\|\Delta_1\|_1 \leq \|p^1_T\|_1$. The second application of Claim 4.13 is to the runs R_3 and R_T (both on the edge set T); it implies that if Δ_2 is such that

$$p^3 - p^1_T = (c - p^2) - (c - p^1_S) + \Delta_2 = p^1_S - p^2 + \Delta_2, \tag{4.8}$$

then $\|\Delta_2\|_1 \leq \|p_S^1 - p^2\|_1$; this, by (4.7), is at most $\|p_T^1\|_1 + \|\Delta_1\|_1 \leq 2\|p_T^1\|_1$. Furthermore, plugging (4.7) into (4.8), we get

$$p^3 - p_T^1 = -(p_T^1 + \Delta_1) + \Delta_2.$$

Simplifying, this gives $\|p^3\|_1 = \|\Delta_1\|_1 + \|\Delta_2\|_1 \leq 3\|p_T^1\|_1$. ■

Proof of Claim 4.13: Consider the two runs R and $\widehat{R}$ of $\mathcal{A}$ on the two cost vectors c and $\widehat{c}$ being executed in parallel. Let $p_t(v)$ and $\widehat{p_t}(v)$ be the payments towards vertex v accumulated in the respective runs until time t. We claim that the quantity $\Phi(t) = \sum_{v \in V} |(c(v) - p_t(v)) - (\widehat{c}(v) - \widehat{p_t}(v))|$ never increases as a function of t. Since $\Phi(0) = \|c - \widehat{c}\|_1$ and $\Phi(\infty) = \|(p - \widehat{p}) - (c - \widehat{c})\|_1 = \|\Delta\|_1$, this will prove Claim 4.13.

Consider any edge $e = \{u, v\}$ at time t in both runs. If e is not frozen in either run, it causes both $p(u)$ and $\widehat{p}(u)$ to increase at unit rate; the same arguments hold for v. Since u is not tight in either run, $c(u) - p_t(u) > 0$ and $\widehat{c}(u) - \widehat{p_t}(u) > 0$, and edge e contributes to both terms equally; hence it is currently contributing at rate zero to the difference $(c(u) - p_t(u)) - (\widehat{c}(u) - \widehat{p_t}(u))$. If e is frozen in both runs, its current rate of contribution is zero as well.

Now suppose that e is frozen in only one of the runs; say, it is frozen in the run R but not in the run $\widehat{R}$ (the other case is symmetric). That means one of its endpoints must be tight in R; w.l.o.g., assume the tight vertex is u. Thus $c(u) - p_t(u) = 0$. In the run $\widehat{R}$, the contribution of e makes the term $\widehat{c}(u) - \widehat{p_t}(u) = |(c(u) - p_t(u)) - (\widehat{c}(u) - \widehat{p_t}(u))|$ *decrease* at unit rate. However, its contribution towards v, and hence towards the term $|c(v) - p_t(v) - (\widehat{c}(v) - \widehat{p_t}(v))|$ increases at a rate of at most 1 in the worst case. Hence, the quantity Φ never increases. ■

4.6 Independent Decisions

The *independent decisions* model was defined in Section 1 as the model when each client $j \in U$ has a probability π_j of requiring service *independently* of all other clients. For this special case of our model, we show that a weaker version of strict cost-shares is sufficient to obtain algorithms for stochastic problems. This allows us to obtain approximations for some more problems (e.g., for Steiner Network), and obtain stronger results for others (e.g., for Vertex Cover and Facility Location) in the independent decisions model.

Given a problem Π, we use $\mathsf{Ind}(\Pi)$ to denote the stochastic extension of Π in this independent decisions model. For this section, we will need the following weaker definition of strictness that holds only for additions of a *single* client.

Definition 4.14 Given an α-approximation algorithm $\mathcal{A}$ for the problem Π, the function $\xi : 2^U \times U \to \mathbb{R}_{\geq 0}$ is a *β-uni-strict cost-sharing function* if properties (P1), (P2) hold in conjunction with:

P3′. If $S' = S \uplus \{j\}$, then $\xi(S', j) \geq (1/\beta) \times$ cost of *augmenting* the solution $\mathcal{A}(S)$ to a solution in $\mathsf{Sols}(S')$. **(uni-strictness)**

Again, we need a *poly-time algorithm* $\mathsf{Aug}_{\mathcal{A}}$ that does the augmentation with cost at most $\beta\, \xi(S', j)$.

4.6.1 The (Even Simpler) Algorithm Ind-Boost

Let us define the (yet simpler) algorithm for the independent case:

Algorithm $\mathsf{Ind\text{-}Boost}(\Pi)$

1. Choose a set D by picking each element $j \in U$ with probability $\sigma\, \pi_j$ independently.

2. Using the algorithm $\mathcal{A}$, construct an α-approximate solution $F_0 \in \mathsf{Sols}(D)$.

3. Let S be the set of clients realized in the second stage. For each client $j \in S$, use the augmentation algorithm $\mathsf{Aug}_{\mathcal{A}}$ of (P3′) to compute F_j such that $F_0 \cup F_j \in \mathsf{Sols}(D \cup \{j\})$. Output $F_S = \bigcup_{j \in S} F_j$ as the second stage solution. Note that by subadditivity, $F_0 \cup F_S \in \mathsf{Sols}(S)$.

Note that $\mathsf{Ind\text{-}Boost}(\Pi)$ can be implemented in polynomial time regardless of how large σ is. Here is a version of the main Theorem 4.4 for the independent decisions model. This version assumes only the weaker property of uni-strictness, hence it is useful when fully strict cost sharing is not known, or when uni-strictness leads to better approximation guarantees.

Theorem 4.15 *Consider a deterministic combinatorial optimization problem* Π *that is sub-additive, and let* $\mathcal{A}$ *be an* α*-approximation algorithm for* Π *with a* β*-uni-strict cost sharing function. Then* $\mathsf{Ind\text{-}Boost}(\Pi)$ *is an* $(\alpha+\beta)$*-approximation algorithm for* $\mathsf{Ind}(\Pi)$*.*

Proof. While it is possible to prove this result closely following the lines of that for Theorem 4.3.1, we give a slightly different proof here.

First, some notation: let $\pi(S) = \prod_{j \in S} \pi_j \prod_{j \notin S}(1 - \pi_j)$. Let F_0^* be the first-stage component of the optimal solution, and F_S^* be the second-stage component if the set of realized clients is S, and let Z^* be defined as in Equation (4.2).

First stage: Again, we claim that there is $\widehat{F}_1 \in \mathsf{Sols}(D)$ such that $\mathbf{E}[c(\widehat{F}_1)] \leq Z^*$; the actual proof is by a slightly different "coupling" argument. As a thought experiment, throw elements of D into σ sets $D_1, \ldots, D_\sigma$ independently and uniformly at random. Now, since D was picked by sampling U at rate $\sigma\,\pi_j$, each D_i is distributed as though we sampled element $j \in U$ with probability π_j. (The contents of different D_i's are correlated negatively, but we will only use linearity of expectations.)

Define $\widehat{F}_1 = F_0^* \cup F_{D_1}^* \cup F_{D_2}^* \cup \ldots \cup F_{D_\sigma}^*$. Again, $\widehat{F}_1 \in \mathsf{Sols}(D)$ from sub-additivity, and

$$\begin{aligned}
\mathbf{E}_D[c(\widehat{F}_1)] &\leq c(F_0^*) + \mathbf{E}_D[\sum_{i=1}^{\sigma} c(F_{D_i}^*)] \\
&= c(F_0^*) + \sum_{i=1}^{\sigma} \mathbf{E}_{D_i}[c(F_{D_i}^*)] \\
&= c(F_0^*) + \sigma \sum_S \pi(S)\, c(F_S^*) \qquad = Z^*.
\end{aligned}$$

Now an α-approximation algorithm for $\mathsf{Det}(\Pi)$ gives us a solution F_0 with expected cost $\mathbf{E}[c(F_0)] \leq \alpha c(\widehat{F}_1) \leq \alpha\, Z^*$, bounding our first stage costs.

Second stage: Let S be the set of realized clients, and let $F_S = \bigcup_{j \in S} F_j$ be the result of our algorithm $\mathsf{Aug}_{\mathcal{A}}$. Note that for all $j \in S$, $F_0 \cup F_j \in \mathsf{Sols}(\{j\})$, thus by subadditivity $F_0 \cup F_S \in \mathsf{Sols}(S)$. We need to bound our expected second stage cost, which is $\sigma\, \mathbf{E}[c(F_S)]$, which we will bound by the expected *first stage* cost.

Define ϕ_j for an element $j \in U$ to be the random variable $\phi_j = \xi(D, j)$, and ψ_j to be the cost of augmenting a solution for D to include j as well, in the case that $j \in S$. (Hence, $\psi_j = c(F_j)$ if $j \in S$ and $\psi_j = 0$ if $j \notin S$.) Let $X_j = \sigma\psi_j - \beta\, \phi_j$.

Now let us condition on all the first-stage coin-tosses $\mathcal{T}$ in U except for j's toss. Let $D_{\mathcal{T}}$ be all the vertices picked according to $\mathcal{T}$ (which does not include j), and consider the expected value of X_j over the first-stage toss for j, and the tosses of the realized set S.

$$\mathbf{E}_{D,S}[\sigma\, \psi_j |\ \mathcal{T}\ ] = \sigma \times \pi_j \times (1 - \sigma\, \pi_j)\, c(F_j) \quad \text{and} \tag{4.9}$$

$$\mathbf{E}_D[\beta\, \phi_j |\ \mathcal{T}\ ] = \beta \times \sigma\ \times \pi_j \times \xi(D_{\mathcal{T}} \uplus \{j\}, j). \tag{4.10}$$

By uni-strictness of $\mathcal{A}$, (4.10) is at least (4.9), and hence $\mathbf{E}_{D,S}[X_j |\ \mathcal{T}\ ] \leq 0$. Since this holds for all $\mathcal{T}$, $\mathbf{E}_{D,S}[X_j] \leq 0$ unconditionally, and thus

$$\mathbf{E}_{D,S}[\psi_j] \leq \frac{\beta}{\sigma} \mathbf{E}_D[\phi_j]. \tag{4.11}$$

Note that Properties (P1) and (P2) of the cost shares ξ imply that

$$\begin{aligned} \sum_{j \in U} \mathbf{E}_D[\phi_j] = \sum_{j \in U} \mathbf{E}_D[\xi(D,j)] = \mathbf{E}_D[\sum_{j \in D} \xi(D,j)] \\ \leq \mathbf{E}_D[c(OPT(D))] \leq Z^*. \end{aligned} \tag{4.12}$$

Furthermore, $\mathbf{E}_{D,S}[F_S] \leq \sum_j \mathbf{E}_{D,S}[\psi_j]$ follows by sub-additivity; using this, (4.11) and (4.12), we get that expected second-stage cost $\sigma \mathbf{E}[c(F_S)] \leq \beta\, Z^*$, proving the result.

4.6.2 Steiner forest

The Steiner forest problem is a generalization of the Steiner tree problem, and is defined over an edge-weighted graph. A client u is now a pair (s_i, t_i) of vertices, and given a set of clients S, a feasible solution consists of a set of edges F such that for each $(s_i, t_i) \in S$, there is a path from s_i to t_i in F. The problem is easily verified to be sub-additive.

In Chapter 5, we will give a $(2+\sqrt{2})$-approximation algorithm for this problem, that admits a $(2+\sqrt{2})$-strict cost sharing function. We note that there are algorithms for the Steiner Forest problem with approximation guarantee of 2 [1, 38], but they do not admit a uni-strict cost sharing function. We believe that the techniques of Chapter 5 can lead to fully strict cost-sharing for Steiner Network, although we are still lacking a proof at this moment.

Theorem 4.16 *There is a* $(4+\sqrt{8} < 6.83$*-approximation algorithm for* Ind(STEINER FOREST).

4.6.3 Unrooted Steiner Tree

The Unrooted Steiner Tree problem is, given a set S of users, to build a minimum cost Steiner tree spanning these users. In contrast to the regular Steiner Tree problem, here the tree does not have to include the root vertex. Note that while deterministic Steiner tree and Unrooted Steiner tree are equivalent problems, this is no longer the case with stochastic versions of these problems.

In particular, Stochastic Steiner Tree is not sub-additive (Def. 4.1). Indeed, the input consisting of a single user $\{i\}$, the solution consisting of an empty set of edges is feasible, and the same holds for the single user $\{j\}$. However, the union of these two empty solutions is not feasible for the union of the inputs (any solution for the input $\{i, j\}$ must contain an i-j path, and hence be non-empty).

It turns out, however, that stochastic Steiner tree is a special case of a generalized version of stochastic Steiner forest. Although we are not able to construct a fully strict cost sharing for Steiner forest, we can take advantage of the fact that the second stage solution for Steiner tree is connected. In Section 5.5 we give cost sharing for Steiner forest that is strict with respect to this special type of augmentations.

Theorem 4.17 *There is a* $4+\sqrt{8}$*-approximation algorithm for* Ind(Group Steiner Tree).

4.6.4 Facility Location

Improved results may be obtained for other problems which have already been studied in Section 4.5.

Theorem 4.18 *There is a* 3*-approximation algorithm* $\mathcal{A}$ *for the facility location* *problem, along with cost-shares ξ that are 3-uni-strict w.r.t. $\mathcal{A}$. Hence, there is a 6-approximation for* Ind(Facility Location).

Proof. The proof closely follows that of Theorem 4.7, which the reader is urged to peruse. Here, we will be concerned with the special case of the singleton set $T = \{j\}$.

Consider the run $\mathcal{A}(S \cup \{j\})$, and let p be the primary facility of j in this run. Here is the augmentation procedure $\mathsf{Aug}_{\mathcal{A}}$: if p is open in the run $\mathcal{A}(S)$, it simply assigns j to p. If p is closed, it has two options: if p is $\{j\}$-heavy, it opens p and assigns j to it. Otherwise, it assigns j to the closest facility opened in $\mathcal{A}(S)$.

We claim that the augmentation cost is at most $3\xi(S \cup \{j\}, j)$. Indeed, if we decide to open j's primary facility p, $\xi(S \cup \{j\}, j)$ can pay for a b-fraction of the facility cost of p plus assignment cost of j. If not, Claim 4.8 implies that $t_p(S) \leq t_p(S \cup \{j\}) \frac{|C_p|}{|C_p \cap S|} - \frac{c(j,p)}{|C_p \cap S|}$. We know that there is an open facility r within distance $2\, t_p(S)$ from p, and so reroute j to r. The connection cost in this case is at most

$$c(j,p) + 2\left(t_p(S \cup \{j\})\, \frac{|C_p|}{|C_p \cap S|} - \frac{c(j,p)}{|C_p \cap S|}\right)$$

which is at most $3 \max(c(j,p), t_p(S \cup \{j\})) = 3\,\xi(S \cup \{j\}, j)$. Since we need to minimize $\max\{1/b, 3\}$, the best value is $b = 1/3$, finishing the proof.

4.6.5 Vertex Cover

We present the following improvement on the 6.3-approximation given for Ind(Vertex Cover) given by Immorlica et al. [49]. As discussed in Section 4.5.2, to obtain an approximation algorithm for Stoc(Vertex Cover), it is enough to consider the relaxed version of the problem, where we are allowed to make arbitrary payments p^1 and p^2 to vertices in the two stages, with the vertex v being bought if

$p^1(v) + p^2(v)/\sigma \geq c_v$. As mentioned there, results for this relaxed problem can be easily transferred back to obtain an algorithm in the standard model: this is done by choosing a vertex with probability $p^1(v)/c_v$ in the first stage, and then picking it in the second stage if $p^1(v) + p^2(v)/\sigma \geq c_v$ and it was not already picked.

Theorem 4.19 *There is a 2-approximation algorithm $\mathcal{A}$ for the relaxed Vertex Cover problem, and cost-shares ξ that are 1-strict with respect to $\mathcal{A}$, giving us a 3-approximation for* Ind(VERTEX COVER)*.*

Proof. The algorithm $\mathcal{A}$, as well as the cost shares ξ, are the same as in Section 4.5.2. To augment a solution $\mathcal{A}(S)$ on the addition of the edge $e = \{u, v\}$, the augmentation procedure $\mathsf{Aug}_{\mathcal{A}}$ opens the endpoint whose reduced cost is less. I.e., if the payments in $\mathcal{A}(S)$ are denoted by p, we pay $\delta = \min(c_u - p(u), c_v - p(v))$ to the vertex from $\{u, v\}$ that achieves this minimum and open it. Proving strictness is now equivalent to proving that $\delta \leq \xi(S \cup \{e\}, e)$.

Indeed, consider the runs $\mathcal{A}(S)$ and $\mathcal{A}(S \cup \{e\})$. Both runs behave identically till some endpoint of e, say u, goes tight in the latter run. At that point, the payment made by other edges to u in $\mathcal{A}(S \cup \{e\})$ is exactly $c_u - \xi(S \cup \{e\}, e)$. Since the two runs were identical till now, u has received this payment in $\mathcal{A}(S)$ as well, and hence $p(u) \geq c_u - \xi(S \cup \{e\}, e)$. Hence, $\xi(S \cup \{e\}, e) \geq c_u - p(u) \geq \delta$, proving the theorem.

4.7 Recent developments

Some interesting papers appeared recently on the topic of two-stage stochastic optimization. Gupta, Ravi and Sinha [46] develop LP-rounding techniques for stochastic Steiner tree and related problems. Shmoys and Swamy [86]show that linear programming relaxations of a broad class of stochastic problems, although exponential in size, can be solved within a desired precision using the machinery of the ellipsoid method, and show how to round this solution for problems such as set cover or facility location.

Flaxman, Frieze and Krivelevich [32] derive asymptotic formulas on the cost of a two-stage stochastic minimum spanning tree under the assumption that all edge costs are drawn uniformly at random from the unit interval. They show that any algorithm for the stochastic problem has to do strictly worse in expectation than an omniscient algorithm that can see the second stage costs before making decisions in the first stage. They also show that the expected cost of a natural simple heuristic for the stochastic problem, is within 17% of the optimal cost.

Gupta, Srinivasan and Tardos [47] observe that the boosted sampling scheme preserves cross-monotonicity: if the cost sharing functio ξ is cross-monotonic, the expected cost share of each user will be cross-monotonic as well. In general, it is not clear how to compute these expectations; however, at some expense in approximation guarantee, [47] have been able to derandomize the computation in case of Steiner trees, obtaining a cross-monotonic cost sharing function with better budget balance than in Section 3.3.

Chapter 5

Multicommodity Rent-or-Buy

Most of this chapter has appeared in a paper *Approximation via cost sharing: A simple approximation algorithm for the multicommodity rent or buy problem* by Gupta, Kumar, Pál and Roughgarden [43]. Improvements in the analysis of the Multicommodity Rent or Buy algorithm are due to Becchetti, Könemann, Leonardi and Pál [18].

5.1 Introduction

We study the *multicommodity rent-or-buy* (MRoB) problem, a network design problem with economy of scale. In this problem, we are given an undirected graph $G = (V, E)$ with non-negative weights c_e on the edges e and a set $\mathcal{D} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of vertex pairs called *demand pairs*. We seek a minimum-cost way of installing sufficient capacity on the edges E so that a prescribed amount of flow can be sent simultaneously from each source s_i to the corresponding sink t_i. The cost of installing capacity on an edge is given by a simple concave function: capacity can be *rented*, with cost incurred on a per-unit of capacity basis, or *bought*, which allows unlimited use after payment of a large fixed cost. Precisely, there are positive parameters μ and M, with the cost of renting capacity equal to μ times the capacity required (per unit length), and the cost of buying capacity equal to M (per unit length). By scaling, there is no loss of generality in assuming that $\mu = 1$.

The MRoB problem is a simple model of network design with *economies of scale*, and is a central special case of the more general *buy-at-bulk network design* problem, where the cost of installing capacity can be described by an arbitrary concave function. In addition, the MRoB problem naturally arises as a subroutine in multicommodity versions of the *connected facility location* problem and the *maybecast* problem of Karger and Minkoff [55]; see [65] for further details on these applications.

The MRoB problem is easily seen to be NP- and MAX SNP-hard (for example, it contains the Steiner tree problem as a special case [19]), and researchers have therefore sought approximation algorithms for the problem.[1] For many years, the best known performance guarantee for MRoB was the $O(\log n)$-approximation algorithm due to Awerbuch and Azar [8] and Bartal [14] ($n = |V|$ denotes the number of nodes in the network). The first constant-factor approximation algorithm for the problem was recently given by Kumar et al. [65]. However, both the analysis and the primal-dual algorithm of [65] are quite complicated, and the performance guarantee shown for the algorithm is, while constant, extremely large.[2] The problem of obtaining an algorithm with constant performance guarantee for MRoB while using only a transparent algorithm and/or obtaining a reasonable constant has since remained open.

In this chapter we will show that the deterministic rent-or-buy problem can be modeled as a special case of the corresponding two stage stochastic buy-only problem that we studied in Chapter 4. In our case, the deterministic buy-only problem is the *Steiner network* problem: given a graph G with non-negative weights and a

[1]Recall a *c-approximation algorithm* for a minimization problem runs in polynomial time and returns a solution no more than c times as costly as an optimal solution. The value c is the *approximation ratio* or *performance guarantee* of the algorithm.

[2]While this constant was neither optimized nor estimated in [65], the approach of that paper only seems suitable to proving a performance guarantee of at least several hundred.

set of users (demand pairs) $S = ((s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k))$, find a minimum-cost subgraph $F \subseteq G$ such that every demand pair is connected in F.

The boosted sampling approach has been first used by Gupta et al. [44] to obtain an algorithm for a rent-or-buy type problem; however, without the cost sharing machinery of Chapter 4 they were only obtain an algorithm for the single-sink version of the problem.

5.1.1 Our Results

The main technical result of this chapter is obtaining a $(2+\sqrt{2})$-uni-strict cost sharing function for the Steiner network problem, together with an attending $2+\sqrt{2}$-approximation algorithm for the problem. Plugging this into the framework of Chapter 4, we get an algorithm for stochastic Steiner network for a restricted class of distributions. Finally, we obtain an algorithm for the multicommodity rent-or-buy problem by modelling it as a special case of the stochastic Steiner network problem.

A key ingredient for our result is a simple but novel extension of the primal-dual algorithms of Agrawal et al. [1] and Goemans and Williamson [38] for the Steiner forest problem.

Our performance guarantee of $4+\sqrt{8} \leq 6.83$ is probably not the best achievable for the MRoB problem, but it identifies rent-or-buy as one of the more tractable problems with economy of scale. Recently, Andrews [2] has shown that the multicommodity buy-at-bulk network design problem, where the cost of installing capacity on edges is an arbitrary concave function is hard to approximate within a factor better than $O((\log n)^{1/4})$. By restricting ourselves to a special case, constant factor approximability can usually be recovered. The single-sink buy-at-bulk problem, which requires all commodities to share a single sink can be approximated within 73 [44]. Keeping the single-sink assumption and placing further restrictions on the function describing the cost of capacity yields the *Access Network Design* problem of Andrews and Zhang [3], for which the best known approximation ratio is 68 [72]. The single-sink special case of MRoB is approximable within 3.55 [44]).

5.1.2 Related Work

The only previous constant-factor approximation algorithm for the MRoB problem is due to Kumar et al. [65]. Additional papers that considered multicommodity network design with economies of scale are Awerbuch and Azar [8] and Bartal [14], who gave an $O(\log n)$-approximation to the more general multicommodity buy-at-bulk problem. The special case of MRoB where all commodities share the same sink, and the closely related *connected facility location* problem, have been extensively studied [42, 44, 55, 58, 59, 66, 67, 80, 88]. The randomized 3.55-approximation algorithm of Gupta et al. [44] is the best known approximation algorithm for the problem. Swamy and Kumar [88] achieve a performance guarantee of 4.55 with a deterministic algorithm. Several more general problems that retain the single-sink assumption have also been intensively studied in recent years, including the Access Network Design problem [3, 39, 40, 72], the single-sink buy-at-bulk network design problem [34, 40, 44, 83, 89], and the still more general problems where the capacity cost function can be edge-dependent [25, 71] or unknown to the algorithm [35]. The best known approximation ratios for these four problems are 68 [72], 73 [44], $O(\log n)$ [25, 71], and $O(\log n)$ [35], respectively.

Finally, our high-level algorithm of randomly reducing the MRoB problem to the Steiner forest problem, followed by computing shortest paths, is similar to and partially inspired by previous work that gave online algorithms with polylogarithmic competitive ratios for many rent-or-buy-type problems [9, 13, 15, 16].

5.2 Rent-or-Buy as stochastic Steiner forest

By a *Steiner forest* for $(G, \mathcal{D})$, we mean a subgraph F of G so that, for each demand pair $(s, t) \in \mathcal{D}$, there is an s-t path in F. For such a subgraph F, $c(F) = \sum_{e \in F} c_e$ denotes its overall cost. Since we are only interested in solutions with small cost and edge costs are non-negative, we can always assume that F is a forest.

Consider the stochastic version $\mathsf{Stoc}(\textsc{Steiner forest})$ defined in Chapter 4. Consider a simple distribution π:

$$\pi(S) = \frac{1}{|D|} \qquad \text{if } S = \{(s_i, t_i)\} \tag{5.1}$$

$$\pi(S) = 0 \qquad \text{if } |S| \neq 1$$

That is, with probability 1, exactly one demand pair appears, and each demand pair is equally likely.

We claim that an instance of *stochastic Steiner forest* with second-stage inflation ratio σ is equivalent to an instance of multicommodity rent-or-buy problem with $M = n/\sigma$. Indeed, edges built by the first stage stochastic solution correspond to the edges bought by the solution to the rent-or-buy problem, and edges installed in the second stage in scenario where $\{(s_i, t_i)\}$ is the materialized demand correspond to the edges rented by the (s_i, t_i) demand pair. It is easy to check that the costs of these two solutions are proportional: each edge bought costs c_e and Mc_e for stochastic Steiner forest and multicommodity rent-or-buy respectively; a rented edge costs σc_e with probability $1/n$ (that is, $\sigma/nc_e = 1/Mc_e$ in expectation) and c_e respectively.

Hence, we can solve MRoB by running the boosted sampling algorithm with the distribution π. To claim an approximation ratio using Theorem 4.4, we need to exhibit an approximation algorithm with an attending strict cost sharing function. Although we believe that a natural cost sharing function is strict with respect to our algorithm, we have only been able to prove uni-strictness. Fortunately, it turns out uni-strictness is all we need: since the distribution π never allows more than one user materialize at a time, we only need to care about single demand-pair augmentations.

Theorem 5.1 *The boosted sampling framework turns any α-approximation algorithm for Steiner forest that admits a β-uni-strict cost sharing function into an $(\alpha + \beta)$-approximation algorithm for the multicommodity rent or buy problem.*

5.2.1 Some Definitions

In the following text, it will be crucial for us to compare executions of algorithms and cost shares on inputs with varying sets of clients, but also on different underlying graphs. To accomodate this need, we specify the graph in question with an additional parameter to the cost sharing function. We use $\xi(G, \mathcal{D}, (s,t))$ to denote the cost share of a demand pair (s,t), assuming that the edge costs are given by the graph G, and that the set $\mathcal{D}$ of demand pairs is being served (we assume $(s,t) \in \mathcal{D}$).

Let $d_G(\cdot,\cdot)$ denote the shortest-path distance in G (w.r.t. edge costs in G). Given a subset $E' \subseteq E$ of edges, G/E' is the graph obtained from G by contracting the edges in E'. Note that the cheapest way of connecting vertices s and t by renting edges, given that edges E' have already been bought, is precisely $d_{G/E'}(s,t)$. Using this notation, we restate the uni-stricntess property as follows.

Definition 5.2 Let $\mathcal{A}$ be a deterministic algorithm that, given instance $(G, \mathcal{D})$, computes a Steiner forest. A cost-sharing method is *β-uni-strict for $\mathcal{A}$* if for all $(G, \mathcal{D})$ and $(s,t) \in \mathcal{D}$, the cost $\xi(G, \mathcal{D}, (s,t))$ assigned to (s,t) by ξ is at least a $1/\beta$ fraction of $d_{G/F}(s,t)$, where F is the Steiner forest returned for $(G, \mathcal{D} \setminus \{(s,t)\})$ by algorithm $\mathcal{A}$.[3]

It is not clear *a priori* that strict cost-sharing methods with small β exist: Definition 1.8 (competitiveness) states that the aggregate costs charged to demand pairs must be reasonable, while Definition 5.2 insists that the cost allocated to each demand pair is sufficiently large.

5.3 The Steiner Forest Algorithm

Throughout the thesis we have relied upon primal-dual algorithms to supply us with cost shares. This time we turn our attention to the Steiner forest algorithms of Agrawal et al. [1] and Goemans and Williamson [38]. The reader might hope that plugging such an algorithm into the boosted sampling framework and defining the cost shares ξ according to the corresponding dual solution, would be enough to obtain a constant-factor approximation for MRoB. While we have no evidence that this would not work, it is not hard to find examples showing that straightforward implementations of this idea cannot give β-strict cost-sharing methods for any constant β, preventing us from using the machinery of Theorem 5.1. On the other hand, in these families of examples, an $O(1)$-strict cost-sharing method can be defined if a few extra edges are bought (extra edges make Definition 5.2 easier

[3] Remember that a cost-sharing method can be β-strict for one algorithm and not for another, as strictness depends on the distance $d_{G/F}(s,t)$, and the edge set F is algorithm-dependent.

to satisfy, since the shortest-path distance $d_{G/F}$ relative to the bought edges F decreases). Our main technical result is that buying a few extra edges beyond what is advocated by the algorithms of [1, 38] *always suffices* for defining an $O(1)$-uni-strict cost-sharing method, enabling the application of Theorem 5.1. (The question whether it suffices to obtain full β-strictness remains open, although Section 5.5 makes a promising first step in this direction.)

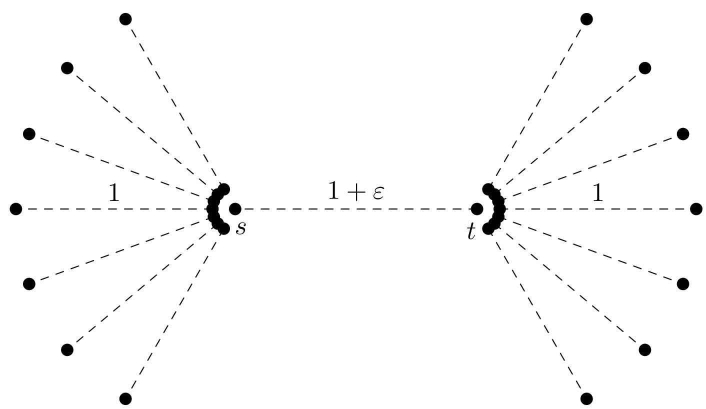

Figure 5.1: The Goemans-Williamson algorithm does not support strict cost sharing. In this figure, there are $2n+1$ source-sink pairs, each source-sink pair is connected by a dotted line. The cost share of the (s,t) pair is $O(1/n+\epsilon)$, but since on the input $\mathcal{D}-(s,t)$, the G-W stops just before building the s-t edge, augmenting to satisfy the (s,t) pair costs us $1+\varepsilon$.

5.3.1 The Algorithm PD and the Cost Shares ξ

In this subsection we show how to extend the algorithms of [1, 38] to "build a few extra edges" while remaining constant-factor approximation algorithms for the Steiner forest problem. We also describe our cost-sharing method.

Recall that we are given a graph $G=(V,E)$ and a set $\mathcal{D}$ of source-sink pairs

$\{(s_k, t_k)\}$. Let D be the set of *demands* — the vertices that are either sources or sinks in $\mathcal{D}$ (without loss of generality, all demands are distinct). It will be convenient to associate a cost share $\xi(\mathcal{D}, j)$ with each demand $j \in D$; the cost share $\xi(G, \mathcal{D}, (s, t))$ is then just $\xi(\mathcal{D}, s) + \xi(\mathcal{D}, t)$. Note that we have also dropped the reference to G; in the sequel, the cost shares are always w.r.t. G.

Before defining our algorithm, we review the LP relaxation and the corresponding LP dual of the Steiner forest problem that was used in [38]:

$$\min \sum_e c_e x_e \tag{SF-LP}$$

$$x(\delta(S)) \geq 1 \quad \forall \text{ valid } S \tag{5.2}$$

$$x_e \geq 0$$

$$\max \sum_{S \subseteq V \text{ valid}} y_S \tag{SF-DP}$$

$$\sum_{S \subseteq V : e \in \delta(S)} y_S \leq c_e \tag{5.3}$$

$$y_S \geq 0,$$

where a set S is *valid* if for some i, it contains precisely one of s_i, t_i. Though y_S is defined only for such sets, we can define y_S for all subsets S of V, as long as these new variables do not contribute to the objective function of the dual.

We now describe a general way to define primal-dual algorithms for the Steiner forest problem. As is standard for the primal-dual approach, the algorithm with maintain a feasible (fractional) dual, initially the all-zero dual, and a primal integral solution (a set of edges), initially the empty set. The algorithm will terminate with a feasible Steiner forest, which will be proved approximately optimal with the dual solution (which is a lower bound on the optimal cost by weak LP duality). The algorithms of [1, 38] arise as a particular instantiation of the following algorithm. Our presentation is closer to [1], where the "reverse delete step" of Goemans and Williamson [38] is implicit; this version of the algorithm is more suitable for our analysis.

Our algorithm has a notion of *time*, initially 0 and increasing at a uniform rate. At any point in time, some demands will be *active* and others *inactive*. All demands are initially active, and eventually become inactive. The vertex set is also partitioned into *clusters*, which can again be either active or inactive. In our algorithm, a cluster will be one or more connected components (w.r.t. the currently built edges). Initially, each vertex is a cluster, and the demands are the active clusters. We will consider different rules by which demands and clusters become active or inactive. To maintain dual feasibility, whenever the constraint (5.3) for some edge e between two clusters S and S' becomes tight (i.e., first holds with equality), the clusters are *merged* and replaced by the cluster $S \cup S'$. We raise dual variables of active clusters until there are no more such clusters.

We have not yet specified how an edge can get built. Toward this end, we define a (time-varying) equivalence relation $\mathcal{R}$ on the demand set. Initially, all demands lie in their own equivalence class; these classes will only merge with time. When two active clusters are merged, we merge the equivalence classes of all active demands in the two clusters. Since inactive demands cannot become active, this rule ensures that all active demands in a cluster are in the same equivalence class.

We build enough edges to maintain the following invariant: the demands in the same equivalence class are connected by built edges. This clearly holds at the beginning, since the equivalence classes are all singletons. When two active clusters meet, the invariant ensures that, in each cluster, all active demands lie in a common connected component. To maintain the invariant, we join these two components by adding a path between them. Building such paths without incurring a large cost is simple but somewhat subtle; Agrawal et al. [1] (and implicitly, Goemans and Williamson [38]) show how to accomplish it. We will not repeat their work here, and instead refer the reader to [1].

Remark 5.3 For the reader more familiar with the exposition of Goemans and Williamson [38], let us give an (informal) alternate description of the network output by the algorithm given above. Specifically, we grow active clusters uniformly, and when *any* two clusters merge, we build an edge between them. At the end, we perform a reverse-delete step—when looking at an edge e, if e lies on the path between some x and y with (x, y) in the final relation $\mathcal{R}$, then we keep the edge, else we delete it. The reader will notice that the network finally output by this new description is *exactly* the same as the one previously given; however, they may not be the same at intermediate points of the execution, and hence we will *not* use this restatement in the rest of the paper. ∎

Specifying the rule by which clusters are deemed active or inactive now gives us two different algorithms:

1. **Algorithm** $\mathsf{GW}(G, \mathcal{D})$: A demand s_i is active if the current cluster containing it does not contain t_i. A cluster is active as long as it contains at least one active demand. This implementation of the algorithm is equivalent to the algorithms of Agrawal et al. [1] and Goemans and Williamson [38].

2. **Algorithm** $\mathsf{Timed}(G, D, T)$: This algorithm takes as an additional input a function $T : V \to \mathbb{R}_{\geq 0}$ which assigns a *stopping time* to each vertex. (We

can also view T as a vector with coordinates indexed by V.) A vertex j is active at time τ if $j \in D$ *and* $\tau \leq T(j)$. (T is defined for vertices not in D for future convenience, but such values are irrelevant.) As before, a cluster is said to be active if at least one demand in it is active.

To get a feeling for $\mathsf{Timed}(G, D, T)$, consider the following procedure: run the algorithm $\mathsf{GW}(G, \mathcal{D})$ and set $T_{\mathcal{D}}(j)$ to be the time at which vertex j becomes inactive during this execution. (If $j \notin D$, then $T_{\mathcal{D}}(j)$ is set to zero.) Since the period for which a vertex stays active in the two algorithms $\mathsf{GW}(G, \mathcal{D})$ and $\mathsf{Timed}(G, D, T_{\mathcal{D}})$ is the same, they clearly have identical outputs.

The Timed algorithm gives us a principled way to essentially force the GW algorithm to build additional edges: run the Timed algorithm with a vector of demand activity times that is larger than what is naturally induced by the GW algorithm.

The Algorithm PD: The central algorithm, **Algorithm** $\mathsf{PD}(G, \mathcal{D})$ is obtained thus: run $\mathsf{GW}(G, \mathcal{D})$ to get the time vector $T_{\mathcal{D}}$; then run $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$—the timed algorithm with the GW-induced time vector scaled up by a parameter $\gamma \geq 1$—to get a forest $F_{\mathcal{D}}$. (We will fix the value of γ later in the analysis.)[4]

The fact that $F_{\mathcal{D}}$ is a feasible Steiner network for $\mathcal{D}$ is easily verified, using the two facts that each demand pair (s_i, t_i) was connected at time $T_{\mathcal{D}}(s_i) = T_{\mathcal{D}}(t_i)$, and that $\gamma \geq 1$. We now define the cost shares ξ.

[4] A technical point is that when $\gamma > 1$, algorithm PD may raise the dual variables of vertex sets that are not valid, and hence do not contribute to the value of the dual objective function. As we will see in Lemma 5.7, however, this will not hinder our analysis.

The Cost Shares ξ: For a demand $j \in D$, the cost share $\xi(\mathcal{D}, j)$ is the length of time during the run $\mathsf{GW}(G, \mathcal{D})$ in which j was the only active vertex in its cluster. Formally, let $a(j, \tau)$ be the indicator variable for the event that j is the only active vertex in its cluster at time τ; then

$$\xi(\mathcal{D}, j) = \int a(j, \tau)\, d\tau, \tag{5.4}$$

where the integral is over the entire execution of the algorithm.

Remark 5.4 Note that the cost shares defined by Equation 5.4 do not account for the full cost of the dual solution y, as the cost of growth of clusters with more than one active demand is not reflected at all. We could fix this e.g. by splitting the cost of each such component equally among active demands in that component (by defining $a(j, \tau) = 1/|C_j|$ if j is active, and $|C_j|$ is the number of active terminals in j's component); however, this change would not improve our approximation ratio.

In the sequel, we will prove our main technical result:

Theorem 5.5 *For any $\gamma \geq 2$,* PD *is a $\alpha = \gamma + 1$-approximation for the Steiner network problem, and ξ is a $\beta = 2\frac{\gamma}{\gamma-1}$-strict cost-sharing method for* PD*.*

We can pick the parameter γ to optimize the overall approximation ratio. Setting $\gamma = 1 + \sqrt{2}$ then gives us a $2 + \sqrt{2}$-approximation algorithm that admits a $2 + \sqrt{2}$-strict cost-sharing method.[5]

[5]The original paper [43] proved Theorem 5.5 with $\alpha = 2\gamma$ and $\beta = 6\gamma/(2\gamma - 3)$. Shortly after publication of the paper, Gupta and Pál (unpublished) simplified the analysis and showed Theorem 5.5 with $\beta = 2\gamma/(\gamma - 1)$. Finally, ideas of Bechetti, Könemann and Leonardi [18] led to an improved $\alpha = \gamma + 1$.

5.4 Proof of Theorem 5.5

We first show that algorithm PD is a $\gamma + 1$-approximation algorithm for Steiner forest. We begin with an intuitive monotonicity lemma, which implies that the set of edges built by algorithm PD is monotone in the parameter γ.

Lemma 5.6 *Let T and T' be two time vectors with $T(j) \leq T'(j)$ for all demands $j \in D$. Let $\mathcal{R}(\tau)$ and $\mathcal{R}'(\tau)$ be the connectedness equivalence relations at time τ in* $\mathsf{Timed}(G, D, T)$ *and* $\mathsf{Timed}(G, D, T')$ *respectively. Then, $\mathcal{R}(\tau) \subseteq \mathcal{R}'(\tau)$, for any τ, that is , whenever a pair of vertices is connected in $\mathcal{R}$, it must also be connected in $\mathcal{R}'$.*

Proof. The lemma can be proved by straightforward, if somewhat tedious induction on the time τ. At time $\tau = 0$ both relations are empty, and all clusters are singleton. By analyzing the two executions, it is atraightforward to verify that the clusters in $\mathsf{Timed}(G, D, T)$ are always subsets of clusters of $\mathsf{Timed}(G, D, T')$, and that $\mathcal{R}(\tau) \subseteq \mathcal{R}'(\tau)$ at all times. ■

We can now prove the claimed approximation ratio.

Lemma 5.7 *The cost of the Steiner forest $F_{\mathcal{D}}$ constructed by algorithm* PD *for instance $(G, \mathcal{D})$ is at most $(\gamma + 1) \sum_S y_S \leq (\gamma + 1)\, c(F^*_{\mathcal{D}})$, where $F^*_{\mathcal{D}}$ is an optimal Steiner forest for $(G, \mathcal{D})$.*

Proof. First, we claim that the cost of $F_{\mathcal{D}}$ is at most $2 \sum_S y'_S$. This follows from the arguments in Agrawal et al. [1] (since our algorithm builds paths between merging clusters as in [1]).

We next relate $\sum_S y'_S$ to the cost of an optimal Steiner forest, via the feasible dual solution $\{y_S\}$. We claim that

$$\sum_S y'_S \leq (\gamma+1)/2 \sum_S y_S, \tag{*}$$

provided $\gamma \geq 1$.

Toward this end, we look at the behavior of active clusters in the execution $\mathsf{Timed}(G, \mathcal{D}, \gamma T_{\mathcal{D}})$ over time, and charge the cost of their growth to clusters in $\mathsf{GW}(G, \mathcal{D})$. At any time τ, we distinguish two types of active clusters in the execution $\mathsf{Timed}(G, \mathcal{D}, \gamma T_{\mathcal{D}})$:

1. **Type 1 cluster** contains at least one demand j that is not satisfied in the execution $\mathsf{GW}(G, \mathcal{D})$ at time τ (i.e. j is not in the same cluster as its pair j' in $\mathsf{GW}(G, \mathcal{D})$).

2. **Type 2 cluster** only contains satisfied demands (i.e. every demand $j \in C$ is connected to its pair j' in $\mathsf{GW}(G, \mathcal{D})$ at time τ).

Note that in the beginning, all clusters are singleton denamds, and hence are of type 1. Note that at any time τ, Lemma 5.6 implies that each type 1 cluster C is a superset of some cluster C' of the execution $\mathsf{GW}(G, \mathcal{D})$, and hence the cost of growing C can be charged to C'.

The only way we may arrive at a type 2 cluster C is when a demand pair (s_i, t_i) gets satisfied in the execution $\mathsf{GW}(G, \mathcal{D})$. This happens by merging two active clusters C'_1 and C'_2 in $\mathsf{GW}(G, \mathcal{D})$, forming an inactive cluster $C' = C'_1 \cup C'_2$. Note that this happens exactly at time $\tau = T_{\mathcal{D}}(s_i, t_i)$. We will charge the cost of growing the type 2 cluster C to the expense of creating the inactive cluster C'. Note that the cluster C will get deactivated by time $\gamma\tau$, an hence has at most

$(\gamma - 1)\tau$ lifetime remaining. On the other hand, each of the clusters C'_1, C'_2 must have spent at least τ money on growing each. Thus if we charge the cost of growing C to C', we overcharge C' by a factor at most $(\gamma - 1)/2$.

Since each cluster of $\mathsf{GW}(G, \mathcal{D})$ is charged at most its cost for type 1 clusters and at most $(\gamma - 1)/2$ for type 2 clusters, the inequality (*) follows. As $\sum_S y_S$ is a lower bound on the optimal cost $c(F^*_{\mathcal{D}})$ (by LP duality), the lemma is proved.

Lemma 5.8 *The function ξ satisfies*

$$\sum_{(s,t)\in\mathcal{D}} \xi(G, \mathcal{D}, (s,t)) = \sum_{j\in D} \xi(\mathcal{D}, j) \leq \sum_S y_S \leq c(F^*_{\mathcal{D}}).$$

Proof. At any point τ in time, $\sum_j a(j,\tau) \leq a(\tau)$, since each active cluster can have at most one demand j with non-zero $a(j,\tau)$. Thus $\int \sum_j a(j,\tau) d\tau \leq \int a(\tau) d\tau = \sum_S y_S$.

5.4.1 Proof of Strictness

We first recall some notation we will use often. The algorithm $\mathsf{PD}(G, \mathcal{D})$ first runs $\mathsf{GW}(G, \mathcal{D})$ to find a time vector $T_{\mathcal{D}}$, and then runs $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$ to build a forest $F_{\mathcal{D}}$. Let (s,t) be a new source-sink pair. Define $\mathcal{D}' = \mathcal{D} \cup \{(s,t)\}$, $D' = D \cup \{s,t\}$ and let $T_{\mathcal{D}'}$ be the time vector obtained by running $\mathsf{GW}(G, \mathcal{D}')$. Our sole remaining hurdle is the following theorem, asserting the strictness of our cost-sharing method ξ for the algorithm PD.

Theorem 5.9 (Strictness) *Let (s,t) be a source-sink pair $\notin \mathcal{D}$, and let $\mathcal{D}' = \mathcal{D} + (s,t)$ denote the demand set obtained by adding this new pair to $\mathcal{D}$. Then the length of the shortest path $d_{G/F_{\mathcal{D}}}(s,t)$ between s and t in $G/F_{\mathcal{D}}$ is at most $\beta(\xi(\mathcal{D}', s) + \xi(\mathcal{D}', t))$, where $\beta = \frac{2\gamma}{\gamma - 1}$.*

Simplifying our goals

The main difficulty in proving Theorem 5.9 is that the addition of the new pair (s,t) may change the behavior of primal-dual algorithms for Steiner forest in fairly unpredictable ways. In particular, it is difficult to argue about the relationship between the two algorithms we care about: (1) the algorithm $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$ that gives us the forest $F_{\mathcal{D}}$, and (2) the algorithm $\mathsf{GW}(G, \mathcal{D}') = \mathsf{Timed}(G, D', T_{\mathcal{D}'})$ that gives us the cost-shares. The difficulty of understanding the sensitivity of primal-dual algorithms to small perturbations of the input is well known, and has been studied in detail in other contexts by Garg [33] and Charikar and Guha [24].

In this section, we apply some transformations to the input data to partially avoid the detailed analyses of [24, 33]. In particular, we will obtain a new graph H from G (with analogous demand sets D_H and D'_H), as well as a new time vector T_H so that it now suffices to relate (1') the algorithm $\mathsf{Timed}(H, D_H, \gamma T_H)$ and (2') the algorithm $\mathsf{Timed}(H, D'_H, T_H)$. In the rest of this section, we will define the new graph and time vector; Section 5.4.1 will show that this transition is kosher (i.e. that the reductions satisfy all the necessary invariants), and then Sections 5.4.1 and 5.4.2 will complete the argument.

A simpler time vector T: To begin, let us note that the time vectors $T_{\mathcal{D}}$ and $T_{\mathcal{D}'}$ may be very different, even though the two are obtained from instances that differ only in the presence of the pair (s,t). However, a monotonicity property *does* hold till time $T_{\mathcal{D}'}(s) = T_{\mathcal{D}'}(t)$, as the following lemma shows:

Lemma 5.10 *The set of tight edges at time $\tau \leq T_{\mathcal{D}'}(s)$ during the run of the algorithm $\mathsf{GW}(G, \mathcal{D})$ is a subset of the set of tight edges at the same time in $\mathsf{GW}(G, \mathcal{D}')$.*

Proof of Lemma 5.10: Since $\tau \leq T_{\mathcal{D}'}(s)$, both s and t are active at time τ in $\mathsf{GW}(G, \mathcal{D}')$. Any cluster that has not merged yet with clusters containing s or t has the same behavior in both runs. A cluster that merges with a cluster containing s or t will continue to grow. So compared with $\mathsf{GW}(G, \mathcal{D})$, only more edges can get tight in $\mathsf{GW}(G, \mathcal{D}')$. ■

Corollary 5.11 *Let T be the vector obtained by truncating $T_{\mathcal{D}'}$ at time $T_{\mathcal{D}'}(s)$, i.e., $T(j) = \min(T_{\mathcal{D}'}(j), T_{\mathcal{D}'}(s))$. Then for all demands $j \in D$, $T(j) \leq T_{\mathcal{D}}(j)$.*

Proof of Corollary 5.11: If $T_{\mathcal{D}'}(s) \leq T_{\mathcal{D}}(j)$, the claim clearly holds. If $T_{\mathcal{D}}(j) < T_{\mathcal{D}'}(s)$, then there is a tight path from j to its partner at time $T_{\mathcal{D}}(j)$. But the monotonicity Lemma 5.10 implies that these edges must be tight in $\mathsf{GW}(G, \mathcal{D}')$ at time $T_{\mathcal{D}}(j)$ as well, and hence $T(j) = T_{\mathcal{D}'}(j) \leq T_{\mathcal{D}}(j)$. ■

The vector T is now a time vector for which we can say something interesting for both runs. Suppose we were to now run the algorithm $\mathsf{Timed}(G, D, \gamma T)$ as the second step of $\mathsf{PD}(G, \mathcal{D})$ (instead of $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$ prescribed by the algorithm). By the monotonicity results in Lemma 5.6 and Corollary 5.11, the edges that are made tight in $\mathsf{Timed}(G, D, \gamma T)$ are a subset of those in $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$. Hence it suffices to show that the distance between s and t in the forest resulting from $\mathsf{Timed}(G, D, \gamma T)$ is small; this is made precise by the following construction.

A simpler graph H: Let us look at the equivalence relation defined by the run of $\mathsf{Timed}(G, D, \gamma T)$ over the demands, which we shall denote by $\mathcal{R}$. (Recall that for $j_1, j_2 \in D$, $(j_1, j_2) \in \mathcal{R}$ if at some time τ during the run, they are both active and the clusters containing them meet. Equivalently, at some time τ, both j_1 and j_2 are active and some path between them becomes tight.) Similarly, let the equivalence relation $\mathcal{R}_D$ be obtained by running $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$. Lemma 5.6

and Corollary 5.11 now imply that the former is a refinement of the latter, i.e., $\mathcal{R} \subseteq \mathcal{R}_D$. (Note that this also means that the equivalence classes of $\mathcal{R}_D$ can be obtained by taking unions of the equivalence classes of $\mathcal{R}$.)

Since $F_{\mathcal{D}}$ connects all the demands that lie in the same equivalence class of $\mathcal{R}_D$, the fact that $\mathcal{R} \subseteq \mathcal{R}_D$ implies that it connects up all demands in the same equivalence class in $\mathcal{R}$ as well. Hence, to show Theorem 5.9 that there is a short s-t path in G/F_D it suffices to show the following result.

Theorem 5.12 (Strictness restated) *Let H be the graph obtained from G by identifying all the vertices that lie in the same equivalence class of $\mathcal{R}$. Then the distance between s and t in H is at most $\beta(\xi(\mathcal{D}', s) + \xi(\mathcal{D}', t))$.*

Relating the runs on G and H

We will need some new (but fairly obvious) notation: note that each vertex in H either corresponds to a single vertex in G, or to a subset of the demands D that formed an equivalence class of $\mathcal{R}$. The vertices of the latter type are naturally called the demands in H, and denoted by D_H. The set D'_H is just $D_H \cup \{s, t\}$. Each demand $j \in D'_H$ has a set $C_j \subseteq D'$ of demands that were identified to form j. A new time-vector T_H is defined by setting $T_H(s) = T_H(t) = T(s) = T(t)$; furthermore, for $j \in D_H$, we set $T_H(j) = \max_{x \in C_j} T(x)$.

Going from $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$ to $\mathsf{Timed}(H, D_H, \gamma T_H)$

Note that H was obtained from G by identifying some demands; the edge sets of G and H are exactly the same. We now show they *behave* identically as well. Let us denote the execution of $\mathsf{Timed}(H, D_H, \gamma T_H)$ by $\mathcal{E}$.

Lemma 5.13 *The set of tight edges in the two runs* $\mathsf{Timed}(G, D, \gamma T)$ *and* $\mathcal{E}$ *at any time* τ *are the same.*

The following monotonicity lemma will be used in the following proofs, and we omit the simple proof here:

Lemma 5.14 *For all* $\gamma \geq 1$*, the set of tight edges in* $\mathsf{Timed}(H, D_H, \gamma T_H)$ *contains the tight edges in* $\mathsf{Timed}(G, D, \gamma T)$*. Similarly, the tight edges of* $\mathsf{Timed}(H, D'_H, \gamma T_H)$ *contain those in* $\mathsf{Timed}(G, D', \gamma T)$*.*

Proof of Lemma 5.13: By Lemma 5.14, we know that the latter edges contain the former; we just have to prove the converse. For a demand $j \in D_H$, we claim that each demand $x \in C_j$ is in some active cluster of $\mathsf{Timed}(G, D, \gamma T)$ till time $\gamma T_H(j)$. Suppose not: let $x \in C_j$ be in an inactive cluster at time $\tau < \gamma T_H(j)$. By the definition of the equivalence relation $\mathcal{R}$, no more demands are added to the equivalence class of x (which is C_j). But then $\max_{y \in C_j} \gamma T(y) \leq \tau < \gamma T_H(j)$, a contradiction. Hence all demands in C_j are in active clusters till time $\gamma T_H(j)$. Thus, in the run $\mathsf{GW}(G, D, \gamma T)$, we had grown regions around these demands till time τ, giving us the same tight edges as in $\mathcal{E}$. ■

Corollary 5.15 *For all* $\gamma \geq 1$*, an active cluster at time* τ *during the run of* $\mathcal{E}$ *contains only one active demand. In particular, two active clusters never merge during the entire run of the algorithm.*

Proof of Corollary 5.15: Suppose $j \neq j' \in D_H$ are two active demands in the same (active) cluster at time τ. Let C_j and $C_{j'}$ be the corresponding sets of demands in D respectively. By Lemma 5.13, the set of edges that become tight at time τ is the same as that in $\mathsf{Timed}(G, D, \gamma T)$, and hence there must be demands $x \in C_j, x' \in C_{j'}$ which lie in the same active cluster of $\mathsf{Timed}(G, D, \gamma T)$ at time τ.

By an argument identical to that in the proof of the previous lemma, there must be active demands $y \in C_j$ and $y' \in C_{j'}$ in the same cluster as well. Hence $C_j = C'_j$ by the definition of the equivalence class, contradicting that $j \neq j'$. ■

Note that this implies that the run $\mathcal{E}$ is very simple: each demand $j \in D_H$ grows a cluster around itself; though this cluster may merge with inactive clusters, it never merges with another active one.

Going from $\mathsf{Timed}(G, D', T_{\mathcal{D}'})$ **to** $\mathsf{Timed}(H, D'_H, T_H)$**:**

Recall that the cost shares $\xi(\mathcal{D}', s)$ and $\xi(\mathcal{D}', t)$ are defined by running $\mathsf{GW}(G, \mathcal{D}') = \mathsf{Timed}(G, D', T'_{\mathcal{D}})$ and using the formula (5.4). The following lemma shows that it suffices to look instead at $\mathsf{Timed}(H, D'_H, T_H)$ in order to define the cost-share. (Let $\mathcal{E}'$ be short-hand for the execution $\mathsf{Timed}(H, D'_H, T_H)$.)

Lemma 5.16 *In $\mathcal{E}'$, if s (respectively, t) is the only active vertex in its cluster at time τ, then $a(s, \tau) = 1$ (resp., $a(t, \tau) = 1$) in the run $\mathsf{Timed}(G, D', T_{\mathcal{D}'})$.*

Proof of Lemma 5.16: Suppose $a(s, \tau) = 0$ in $\mathsf{Timed}(G, D', T_{\mathcal{D}'})$, and some active demand $j \neq s$ lies in s's cluster at time τ. By the definition of T, j and s are both active in the same cluster in $\mathsf{Timed}(G, D', T)$ as well. Now by Lemma 5.14, j and s must lie in the same cluster in $\mathcal{E}'$ too; furthermore, they must both be active (by the definition of T_H). This contradicts the assumption of the theorem.

■

Corollary 5.17 *Let $\mathsf{alone}(s)$ be the total time in the run $\mathcal{E}'$ during which s is the only active vertex in its cluster, and define $\mathsf{alone}(t)$ analogously. Then $\xi(\mathcal{D}', s) + \xi(\mathcal{D}', t) \geq \mathsf{alone}(s) + \mathsf{alone}(t)$.*

A property similar to Corollary 5.15 can also be proved about the run $\mathcal{E}'$.

Lemma 5.18 *Let τ_{st} be the time at which s and t become part of the same cluster in $\mathcal{E}'$. Any active cluster at time $\tau \leq \tau_{st}$ in $\mathcal{E}'$ contains at most one active demand from D_H, and at most one active demand from the set $\{s, t\}$.*

Proof of Lemma 5.18: Let C be an active cluster in $\mathcal{E}'$ at time τ with two active demands $j, j' \in D_H$. If C contains neither s nor t, then C is also a cluster (with two active demands) in $\mathcal{E}$ at the same time τ, which contradicts the implication of Corollary 5.15.

Hence C must contain one of s or t (in addition to j, j'); it cannot contain both since $\tau \leq \tau_{st}$. Suppose it has s; we claim that we can prove that j and j' must lie in the same cluster in $\mathcal{E}$ at time $2\,\tau$. (For this to make sense, we have to assume that $\gamma \geq 2$.) To prove this, consider a tight path between s and j in $\mathcal{E}'$ at time τ—since they both lie in the same cluster, such a path must exist. Since s has been active for time τ, the portion of this path which is tight due to a s's cluster has length at most τ. Now consider the same path in $\mathcal{E}$ at time τ — all but a portion of length at most τ of this path must already be tight. (Note that though we have dropped both s and t, we lose only τ, since no part of the path could have been made tight by t's cluster. If this were to happen, then $\tau_{st} < \tau$, which is assumed not to happen.) Hence the cluster containing j will contain s after another τ units of time. The same argument can be made for j'. But this would imply that j and j' would lie in the same cluster in $\mathcal{E}$ at time 2τ, contradicting Corollary 5.15. ■

The path between s and t

Lemma 5.19 *The vertices s and t lie in the same tree in the Steiner forest output by $\mathcal{E}'$.*

Proof of Lemma 5.19: By the definition of T, s and t lie in the same cluster at time $T(s) = T(t)$. Since $T_H(s) = T_H(t) = T(s)$, applying Lemma 5.14 implies that s and t lie in the same cluster in $\mathcal{E}'$ as well. Since they are both active at the time their clusters merge, they lie in the same connected component of the Steiner forest. ■

This simple lemma gives us a path $\mathbb{P}$ between s and t in H whose length we will argue about. Note that this path is already formed at time τ_{st}, and hence the rest of the argument can (and will) be done truncating the time vector at time τ_{st} instead of at $T_H(s)$.

A useful fact to remember is that all edges in $\mathbb{P}$ must be tight in the run of $\mathcal{E}'$. The proof of the following theorem, along with Corollary 5.17, will complete the proof of Theorem 5.12.

Theorem 5.20 (Strictness restated again) *The length of the path* $\mathbb{P}$ *is bounded by* $\beta(\mathsf{alone}(s) + \mathsf{alone}(t))$.

We will prove the theorem with $\beta = 2\gamma/(\gamma - 1)$. Before we proceed, here is some more syntactic sugar:

Given an execution of an algorithm, a *layer* is a tuple $(C, I = [\tau_1, \tau_2))$, where $C \subseteq V$ is an active cluster between times τ_1 and τ_2. If $I = [\tau_1, \tau_2)$ is a time interval, its *thickness* is $\tau_2 - \tau_1$. A *layering* $\mathcal{L}$ of an execution is a set of layers such that, for every time τ and every active cluster C, there is exactly one layer $(C, I) \in \mathcal{L}$ such that $\tau \in I$.

Given any layering $\mathcal{L}$, another layering $\mathcal{L}'$ can be obtained by *splitting* some layer $(C, I) \in \mathcal{L}$—i.e., replacing it with two layers $(C, [\tau_1, \hat{\tau})), (C, [\hat{\tau}, \tau_2))$ for some $\hat{\tau} \in (\tau_1, \tau_2)$. Conversely, two layers with the same set C and consecutive time intervals can be merged. It is easy to see that given some layering of an execution, all other layerings of the same execution can be obtained by splittings and mergings.

We fix layerings of the two executions $\mathcal{E}$ and $\mathcal{E}'$, which we denote by $\mathcal{L}$ and $\mathcal{L}'$ respectively. The only property we desire of these layerings is this: if $(C, I) \in \mathcal{L}$ and $(C', I') \in \mathcal{L}'$ with $1/\gamma I \cap I' \neq \emptyset$, then $I = \gamma I'$. (I.e., $I = [\gamma\tau_1, \gamma\tau_2)$ and $I' = [\tau_1, \tau_2)$.) It is easy to see that such a condition can be satisfied by making some splittings in $\mathcal{L}$ and $\mathcal{L}'$.

Lemma 5.18 implies that each active layer $(C, I) \in \mathcal{L}'$ in categorized thus:

- **alone:** The only active demand in C is either s and t. *Assign* the layer to that demand.

- **shared:** C contains one active demand from D_H **and** one of s and t. The layer is *assigned* to the active demand from D_H.

- **unshared:** The only active demand in C is from D_H. Again, the layer is *assigned* to that demand.

Note that the total thickness of the lonely layers is a lower bound on $\mathsf{alone}(s) + \mathsf{alone}(t)$. Furthermore, since $\mathbb{P}$ consists only of tight edges in $\mathcal{E}'$, the length of $\mathbb{P}$ is exactly the total thickness of the layers of $\mathcal{L}'$ that $\mathbb{P}$ crosses. If L, S, and U denote the total thickness of the alone, shared and unshared layers that $\mathbb{P}$ crosses, then

$$\mathsf{len}(\mathbb{P}) = L + S + U \tag{5.5}$$

Of course, any layer $(C, I) \in \mathcal{L}'$ that crosses $\mathbb{P}$ must have $I \subseteq [0, \tau_{st}]$, since the path is tight at time τ_{st}. Hence their corresponding layers have time intervals that lie in $[0, \gamma\tau_{st}]$.

Note that the total thickness of the layers of $\mathcal{L}$ that $\mathbb{P}$ crosses is a lower bound on its length. This suggests the following plan for the rest of the proof: for each shared or unshared layer in $\mathcal{L}'$, there is a corresponding distinct layer in $\mathcal{L}$ that is γ times thicker. In an ideal world, each crossing of a layer in $\mathcal{L}'$ would also correspond to a crossing of the corresponding layer in $\mathcal{L}$; in this case, $\gamma(S+U) \leq \mathsf{len}(\mathbb{P})$, and hence $L \geq \frac{\gamma-1}{\gamma}\mathsf{len}(\mathbb{P})$. Sadly, we do not live in an ideal world and the argument is a bit more involved than this, though not by much.

Mapping shared and unshared layers of $\mathcal{L}'$: Each such layer $\ell' = (C', I)$ is assigned to a demand $j \in D_H \cap C'$. Since j is active during the interval γI in $\mathcal{E}$, there must be a layer $\ell = (C, \gamma I) \in \mathcal{L}$ containing j—this is defined to be the layer corresponding to ℓ'. (The properties of the layerings ensure that the two time intervals are just rescalings of each other by a factor γ.) Furthermore, since each layer in $\mathcal{L}$ contains only one active vertex, the mapping is one-one. It remains to show that crossings of $\mathbb{P}$ by layers is preserved (approximately) by this correspondence.

Lemma 5.21 *Each* unshared layer $\ell' = (C', I = [\tau_1, \tau_2))$ *is crossed by* $\mathbb{P}$ *either zero or two times. Furthermore, if its corresponding layer is* $\ell = (C, \gamma I)$*, then* $C' \subseteq C$*.*

Proof. Since $\mathbb{P}$ begins and ends outside ℓ', it must cross ℓ' an even number of times. Furthermore, $\mathbb{P}$ cannot cross ℓ' more than twice—if it does so, there must exist vertices j_1, j_2, and j_3 visited by $\mathbb{P}$ (in order), such that $j_1, j_3 \in C'$, but $j_2 \notin C'$. However, our algorithms ensure that if j_1 and j_3 lie in the same cluster C', then the edges joining them lie within C' as well. This implies that there must be two disjoint paths between j_1 and j_2, contradicting that the algorithms construct a forest.

For the second part, note that if C' is assigned to j and does not contain s or t at time τ_2, then the cluster containing j at time $\gamma\tau_2$, i.e., C must contain C'.

Lemma 5.22 *If $\ell' = (C', I = [\tau_1, \tau_2))$ is a* shared layer *containing s (resp., t) which assigned to j, and $\gamma \geq 2$, then its corresponding layer $\ell = (C, \gamma I)$ contains s (resp., t).*

Proof of Lemma 5.22: The proof of the first claim is similar to that of Lemma 5.18; we just sketch the idea again. Consider a tight path between j and s at time τ_1; at most a τ_1 portion of it can be tight due to s. Hence by time $\gamma\tau_1 \geq 2\tau_1$, the path must be completely tight. ■

5.4.2 Finally, the book-keeping

Let $\ell' = (C', I) \in \mathcal{L}'$ be an unshared layer that $\mathbb{P}$ crosses, and let its corresponding layer be $\ell = (C, \gamma I) \in \mathcal{L}$. If $v \in \mathbb{P} \cap C'$ is a vertex on the path, then $v \in \mathbb{P} \cap C$ by Lemma 5.21. If both s and t are outside C, then P crosses C twice as well, and we get a contribution of 2γ times the thickness of ℓ'. Suppose not, and $s \in C$ or $t \in C$: then we lose γ times the thickness of ℓ for each such infringement. For shared layers ℓ' (which map to $\ell \in \mathcal{L}$), Lemma 5.22 implies that we lose only when s and t both lie inside ℓ. Hence, if W denotes the wastage, the

$$\mathsf{len}(\mathbb{P}) \geq \gamma(S + U) - W. \tag{5.6}$$

We bound the wastage in the following way: suppose that $\ell' = (C', I)$ is a layer that is wasted (more precisely, we should be talking about a wasted crossing of ℓ and $\mathbb{P}$), and let $\ell = (C, \gamma I)$ be its corresponding layer in $\mathcal{L}$. The reason that ℓ' is wasted is that for one of the endpoints of $\mathbb{P}$ (let it be s), $s \in C$, but $s \notin C'$. We claim that at this time, s must have been growing alone. Indeed, suppose that s was in a shared layer with some terminal j in the time interval I. Then s and j must have been growing a layer $\ell'_1 = (C'_1, I)$ in $\mathcal{L}'$ that is distinct from ℓ', since $s \notin \ell'$. But then the layer $\ell_1 = (c_1, \gamma I)$, corresponding to ℓ'_1 (which must be distinct from ℓ, as they contain distinct active terminals) intersects ℓ in $\mathcal{L}$, which is a contradiction. Thus, for each wasted layer there is at least one lonely layer and hence

$$\gamma L \geq W. \tag{5.7}$$

Adding up Inequalities 5.6 and 5.7 and subtracting a γ-multiple of Equation 5.5 we obtain

$$2\gamma L \geq (\gamma - 1)\mathsf{len}(P). \tag{5.8}$$

Rearranging, we get

$$\mathsf{alone}(s) + \mathsf{alone}(t) \geq L \geq \frac{2\gamma}{\gamma - 1}\mathsf{len}(\mathbb{P}), \tag{5.9}$$

proving Theorem 5.20 with $\beta = \frac{2\gamma}{\gamma-1}$. ■

5.5 Group Rent-or-Buy

We can generalize our result a little bit. Instead of associating each user j with two terminals (s_j, t_j), we allow each user j to have a group $g_j = \{t_1, t_2, \ldots, t_k\}$ of terminals ($k \geq 2$). Each user wishes to have her terminals

In the regular Multicommodity Rent-or-Buy setting, each user is j associated with a pair of vertices that she desires to connect by a path. In the Group Rent-or-Buy problem we associate each user j with a set S_j of terminals (instead of just a pair (s_j, t_j)) she wishes to be connected by a Steiner tree. The goal then is to design a collection of Steiner trees, one for each user, to minimize the overall cost. For each edge e we can either buy it at cost $M\, c_e$ (in which case any number of trees can use it at no additional cost), or rent it, in which case we pay the cost c_e multiplied by the number of trees using the edge e.

The corresponding buy-only problem is what we call the Group Steiner Forest[6] problem: buy a subset of edges, so that each set S_j belongs to a single connected component (which, without loss of generality, is a tree). Note that although Group Steiner Forest and regular Steiner Forest are equivalent as far as approximation algrithms are concerned (as each group $S_j = \{t_1, t_2, \ldots, t_k\}$ can be simulated by the terminal pairs (t_1, t_2), (t_2, t_3),. . . ,(t_{k-1}, t_k)), Group Steiner Forest is strictly more general when cost sharing is concerned. In particular, to obtain a uni-strict cost sharing method, we must be able to exhibit an augmentation procedure that can augment for a whole set S_j, not just a terminal pair (s_j, t_j) (in which case, finding the shortest s_j-t_j path was sufficient). We are able to prove the following.

Theorem 5.23 *There is a $1 + \gamma$ approximation algorithm for the Group Steiner Forest problem, that admits a $4\gamma/(\gamma-2)$-strict cost sharing function. Hence (setting $\gamma = 2 + 2\sqrt{2}$) there is a $7 + 4\sqrt{2} \leq 12.6$-approximation algorithm for the* Group Rent-or-Buy *problem.*

Corollary 5.24 *There is a 12.6-approximation algorithm for the* Stoc(Unrooted Steiner Tree) *problem from Section 4.6.3.*

The proof of Theorem 5.23 goes along lines similar to the proof in the single pair case. We make the cost share of a set S_j to pay for the growth of all clusters

[6] This is different from the Group Steiner Tree, where a tree is sought that visits one vertex from each group. We hope this name will not be a source of confusion.

whose only active terminals are those in S_j. Then, we pick the unique minimal tree constructed by the uninflated algorithm that connects the S_j terminals, and argue that this tree can be partially paid for by the cost share. Let $\mathcal{D} = \{S_1, S_2, \ldots, S_k\}$ be the set of demand groups.

The Algorithm PD: As before, we first run the agorithm $\mathsf{GW}(G, \mathcal{D})$, and obtain a time vector $T_\mathcal{D}$, where $T_\mathcal{D}t$ is the time when the terminal $t \in S_j$ got satisfied. Note that the members of a group S_j get satisfied at the same time, when they all join into a single cluster, hence $T_\mathcal{D}(t)$ is the same for all terminals in S_j. Then, we run the timed algorithm $\mathsf{Timed}(G, D, \gamma T_\mathcal{D})$ with the time vector inflated by γ, to obtain a set of dual variables $\{y_S\}$ and a forest $F_\mathcal{D}$.

The Cost Shares ξ: We want the cost share of a group S_j of users to account for the growth of components that grow solely because they contain terminals from S_j. Formally, let $a(j, \tau)$ be the number of active clusters that contain a terminal from S_j but do not contain any active terminals outside S_j, in the execution of $\mathsf{GW}(G, \mathcal{D})$. We define the cost share of S_j to be

$$\xi(\mathcal{D}, j) = \int a(j, \tau)\, d\tau, \tag{5.10}$$

where the integral is over the entire execution of the algorithm.

The Augmentation Algorithm $\mathsf{Aug}_\mathcal{A}$: Previously, when each group S_j consisted only of one terminal pair, it was clear that the best we can do is to find a shortest path between these two terminals in the graph $G/F_\mathcal{D}$. With larger groups S_j, we need to construct a Steiner tree on the set S_j in $G/F_\mathcal{D}$. While we could use any general purpose Steiner tree algorithm available (which is probably what we would recommend in a practical implementation), in the course of our proof we will construct a specific Steiner tree on S_j which we use to show strictness. (Our proof is constructive, and the tree can be constructed in polynomial time.)

5.5.1 Proving strictness

To prepare the stage, we work through the same chain of reductions as before. We take a set of demand groups $\mathcal{D}$, an additional demand group $g \notin \mathcal{D}$, and let $\mathcal{D}' = \mathcal{D} \cup \{g\}$. We run the algorithm $\mathsf{GW}(G, \mathcal{D} + g)$ to compute the cost share $\xi(\mathcal{D} + g, g)$ of the group g, and to obtain a time vector $T_{\mathcal{D}+g}$. We truncate $T_{\mathcal{D}+g}$ at the time when group g got satisfied (call the truncated time vector T), and run $\mathsf{Timed}(G, \mathcal{D} + g, \gamma T)$. We create a new graph H by contracting all pairs of vertices u, v that were together active in the same component. The final comparison takes place between the executions $\mathcal{E}' = \mathsf{Timed}(H, \mathcal{D} + g, T)$ and $\mathcal{E} = \mathsf{Timed}(H, \mathcal{D}, \gamma T)$.

5.5.2 The tree spanning terminals of g

The terminals of g end up in the same cluster in the execution ϵ', and hence are in the same connected component of the forest output by ϵ'. Let $\mathbb{P}$ be the unique minimal subtree of this forest that spans all the terminals in S. Our goal is to prove that the cost of $\mathbb{P}$ is bounded by $\beta\xi(\mathcal{D} + g, g)$.

We label each layer $\ell' = (C, [\tau_1, \tau_2))$ in the execution $\mathcal{E}'$ as lonely, shared, and unshared as follows.

- **lonely**: all active demands inside C belong to S.
- **shared**: there is an active demand from S, as well as from $\mathcal{D}$.
- **unshared:** $S \cap C = \emptyset$.

Note that now a lonely or shared layer may contain more than one terminbal from S, and a shared layer may contain multiple terminals outside S.

Mapping shared and unshared layers of $\mathcal{L}'$ to layers of L: Each shared or unshared layer $\ell' = (C', [\tau_1, \tau_2)) \in \mathcal{L}'$ contains at least one active terminal $t \in C' \cap \mathcal{D}$, and hence there is a unique layer $\ell = (C, [\gamma\tau_1, \gamma\tau_2)) \in \mathcal{L}$ such that $t \in C$. We thus map ℓ' to ℓ. Note that some shared layers ℓ' may contain more than one active terminal $t \in \mathcal{D}$; we choose one of them arbitrarily. This way, each shared or unshared layer from $\mathcal{L}'$ is mapped to a layer in $\mathcal{L}$; note that since every layer in $\mathcal{L}$ contains only one active terminal, two layers from $\mathcal{L}'$ never map to the same layer $\ell \in \mathcal{L}$.

We claim that the image ℓ of every shared layer ℓ' contains at least one terminal $t \in g$. This can be verified along the lines of Lemma 5.22. For every unshared layer $\ell' = (C', I)$, we know that its image $\ell = (C, \gamma I)$ is "larger", i.e. $C' \subseteq C$. Hence, for any shared or unshared layer ℓ' that intersects $\mathbb{P}$ we know that its inflated image ℓ must either also intersect $\mathbb{P}$, or $\mathbb{P}$ must be fully contained in ℓ. We call such layers *wasted*.

Let L, S, and U be the total thickness of lonely, shared and unshared layers that intersect the tree $\mathbb{P}$. Note that this time we count each layer only once in the respective sum, irrespective of how many intersections it has. Let us express the total length of the tree as

$$|\mathbb{P}| = L + S + U + X, \tag{5.11}$$

where X accounts for all the extra intersections of layers that cross $\mathbb{P}$ more than once (for example, if a lonely layer intersects $\mathbb{P}$ three times, it is counted once in L and twice in X).

Now, realizing that each layer in $\mathcal{L}'$ carves out a contiguous part of the tree, we note that at any time τ, the total number of intersections is at most twice the number of distinct layers intersecting $\mathbb{P}$, and hence

$$X \leq L + S + U. \tag{5.12}$$

As before, the γ times the total thickness of shared and unshared layers intersecting $\mathbb{P}$ that are not wasted is a lower bound on the cost of $\mathbb{P}$:

$$\gamma(S + U - W) \leq |\mathbb{P}|. \tag{5.13}$$

Finally, we argue that for every layer wasted, there must be a lonely layer growing at the same time. Indeed, suppose that for a shared or unshared layer $\ell' = (C', [\tau_1, \tau_2))$, its inflated image $\ell = (C, [\gamma\tau_1, \gamma\tau_2))$ contains the whole tree. Consider a terminal t from g that was not a part of the cluster C' at time τ_1. We claim that t must not have been in a shared layer ℓ'_t at time τ_1; if this was the case, the inflated image ℓ_t of ℓ'_t would have intersected $\mathbb{P}$, and hence have a nonempty intersection with ℓ, which is not possible. Hence we conclude that t was growing in a lonely layer in the time interval $[\tau_1, \tau_2)$.

$$W \leq L. \tag{5.14}$$

Combining these inequalities (multiplying 5.11 and 5.12 by γ and 5.13 by 2 and 5.14 by 2γ and adding up) we obtain

$$(\gamma - 2)|\mathbb{P}| \leq 4\gamma L. \tag{5.15}$$

Thus, the augmenting tree costs at most $4\gamma/(\gamma - 2)\, \xi(\mathcal{D} + g, g)$.

BIBLIOGRAPHY

[1] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995. (Preliminary version in *23rd STOC*, 1991).

[2] Matthew Andrews. Hardness of buy-at-bulk network design. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 2004.

[3] Matthew Andrews and Lisa Zhang. Approximation algorithms for access network design. *Algorithmica*, 34(2):197–215, 2002. (Preliminary version in 39th FOCS, 1998.).

[4] Elliot Anshelevich, Anirban Dasgupta, and Éva Tardos. Near-optimal network design with selfish agents. In *Proceedings of the 35th Annual Symposium on Theory of Computing*, 2003.

[5] Aaron Archer, Ranjith Rajagopalan, and David Shmoys. Lagrangian relaxation for the k-median problem: new insights and continuity properties. In *Proceedings of the 11th European Symposium on Algorithms*, 2003.

[6] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation.* Springer-Verlag, Berlin Heidelberg, 1999.

[7] Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proceedings of the 19th Annual Symposium on Theory of Computing*, pages 230 – 240, New York, 1987. ACM.

[8] Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 542–547, 1997.

[9] Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized Steiner problem. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (Atlanta, GA, 1996)*, pages 68–74, New York, 1996. ACM.

[10] V. Bala and S. Goyal. A non-cooperative model of network formation. *Econometrica*, 24:1181–1229, 2000.

[11] M. L. Balinski. On finding integer solutions to linear programs. In *Proc. IBM Scientific Computing Symp. on Combinatorial Problems*, pages 225–248, 1966.

[12] Reuven Bar-Yehuda and Shimon Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Disc. Math.*, 25:27–46, 1985.

[13] Yair Bartal. *Competitive Analysis of Distributed On-line Problems — Distributed Paging.* PhD thesis, Tel-Aviv University, Israel, 1994.

[14] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the 30th Annual Symposium on Theory of Computing*, pages 161–168, 1998.

[15] Yair Bartal, Moses Charikar, and Piotr Indyk. On page migration and other relaxed task systems. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 43–52, 1997.

[16] Yair Bartal, Amos Fiat, and Yuval Rabani. Competitive algorithms for distributed data management. *J. Comput. System Sci.*, 51(3):341–358, 1995. (Preliminary version in 24th STOC, 1992).

[17] E. M. L. Beale. On minimizing a convex function subject to linear inequalities. *J. Roy. Statist. Soc. Ser. B.*, 17:173–184; discussion, 194–203, 1955. (Symposium on linear programming.).

[18] Luca Becchetti, Jochen Könemann, Stefano Leonardi, and Martin Pál. Sharing the cost more efficiently: Improved approximation for multicommodity rent-or-buy. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[19] M. Bern and P. Plassman. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176, 1989.

[20] John R. Birge and François Louveaux. *Introduction to stochastic programming.* Springer Series in Operations Research. Springer-Verlag, New York, 1997.

[21] O. N. Bondareva. Some applications of linear programming to cooperative games. *Problemy Kibernetiki*, 10:119–139, 1963.

[22] O. Borůvka. O jistém problému minimálním. *Práce Moravské Přírodovědecké Společnosti*, 3:37–53, 1926.

[23] P. Chardaire. On the core of facility location games. Unpublished manuscript.

[24] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 378–388, 1999.

[25] Chandra Chekuri, Sanjeev Khanna, and Joseph Naor. A deterministic algorithm for the cost-distance problem. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 232–233, 2001.

[26] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* MIT Press, 2001.

[27] George B. Dantzig. Linear programming under uncertainty. *Management Sci.*, 1:197–206, 1955.

[28] Nikhil Devanur, Milena Mihail, and Vijay V. Vazirani. Strategyproof cost-sharing mechanisms for set cover and facility location games. In *ACM Conference on Electronic Commerce*, 2003.

[29] Shane Dye, Leen Stougie, and Asgeir Tomasgard. The stochastic single resource service provision problem. *Naval Research Logistics*, 50(8):869–887, 2003.

[30] Alex Fabrikant, Ankur Luthra, Elitza Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *Proceedings of 22nd ACM Symposium on Principles of Distributed Computing (PODC)*, 2003.

[31] U. Faigle, S.P. Fekete, W. Hochstättler, and W. Kern. On approximately fair cost allocation for the euclidean traveling salesman problem. *OR Spektrum*, 20:29–37, 1998.

[32] Abraham Flaxman, Alan Frieze, and Michael Krivelevich. On the random 2-stage minimum spanning tree. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[33] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 302–309, 1996.

[34] Naveen Garg, Rohit Khandekar, Goran Konjevod, R. Ravi, F. S. Salman, and Amitabh Sinha. On the integrality gap of a natural formulation of the single-source buy-at-bulk network design problem. In *Proceedings of the 8th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2001.

[35] Ashish Goel and Deborah Estrin. Simultaneous optimization for concave costs: Single sink aggregation or single source buy-at-bulk. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 499–505, 2003.

[36] Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *40th Annual Symposium on Foundations of Computer Science (New York, 1999)*, pages 579–586. IEEE Computer Soc., Los Alamitos, CA, 1999.

[37] Michel X. Goemans and Martin Skutella. Cooperative facility location games. In *Proceedings of the 11th Symposium on Discrete Algorithms*, 2000.

[38] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. (Preliminary version in *5th SODA*, 1994).

[39] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. Hierarchical placement and network design problems. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 2000.

[40] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation for the single source edge installation problem. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 383–388, 2001.

[41] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. Improved algorithms for fault tolerant facility location. In *Symposium on Discrete Algorithms*, pages 636–641, 2001.

[42] Anupam Gupta, Amit Kumar, Jon Kleinberg, Rajeev Rastogi, and Bülent Yener. Provisioning a Virtual Private Network: A network design problem for multicommodity flow. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 389–398, 2001.

[43] Anupam Gupta, Amit Kumar, Martin Pál, and Tim Roughgarden. Approximation via cost sharing: A simple approximation algorithm for the multicommodity rent or buy problem. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 606–615, 2003.

[44] Anupam Gupta, Amit Kumar, and Tim Roughgarden. Simpler and better approximation algorithms for network design. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 365–372, 2003.

[45] Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. Boosted sampling: Approximation algorithms for stochastic optimization. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, 2004.

[46] Anupam Gupta, R. Ravi, and Amitabh Sinha. An edge in time saves nine: Lp rounding approximation algorithms. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 2004.

[47] Anupam Gupta, Aravind Srinivasan, and Éva Tardos. Cost-sharing mechanisms. Unpublished manuscript, 2003.

[48] H.Heller and S. Sarangi. Nash networks with heterogeneous agents. Technical Report E-2001-1, Virginia Tech, 2001.

[49] Nicole Immorlica, David Karger, Maria Minkoff, and Vahab Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[50] Nicole Immorlica, Mohammad Mahdian, and Vahab Mirrokni. Limitations of cross-monotonic cost sharing schemes. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[51] Kamal Jain and Vijay V. Vazirani. Applications of approximation algorithms to cooperative games. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 364–372, 2001.

[52] Kamal Jain and Vijay V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. *Journal of the ACM*, 48:274–296, 2001.

[53] Kamal Jain and Vijay V. Vazirani. Equitable cost allocation via primal-dual type algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 313–321. ACM Press, 2002.

[54] Peter Kall and Stein W. Wallace. *Stochastic programming*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons Ltd., Chichester, 1994.

[55] David R. Karger and Maria Minkoff. Building steiner trees with incomplete global knowledge. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 613–623, 2000.

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

[57] K. Kent and D. Skorin-Kapov. Population monotonic cost allocation on mst's. In *Operational Research Proceedings KOI*, pages 43–48, 1996.

[58] Samir Khuller and An Zhu. The general Steiner tree-star problem. *Information Processing Letters*, 84:215–220, 2002.

[59] Tae Ung Kim, Timothy J. Lowe, Arie Tamir, and James E. Ward. On the location of a tree-shaped facility. *Networks*, 28(3):167–175, 1996.

[60] Willem K. Klein Haneveld and Maarten H. van der Vlerk. Stochastic integer programming: general models and algorithms. *Ann. Oper. Res.*, 85:39–57, 1999. Stochastic programming. State of the art, 1998 (Vancouver, BC).

[61] Willem K. Klein Haneveld and Maarten H. van der Vlerk. *Stochastic Programming*. Department of Econometrics and OR, University of Groningen, Netherlands, 2003.

[62] Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM J. Comput.*, 30(1):191–217 (electronic), 2000.

[63] Jochen Könemann, Stefano Leonardi, and Guido Schäffer. A group-strategyproof mechanism for steiner forests. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[64] J.B. Kruskal. On the shortest spanning tree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[65] Amit Kumar, Anupam Gupta, and Tim Roughgarden. A constant factor approximation algorithm for the multicommodity rent-or-buy problem. In *Proceedings of the 43rd Annual Symposium on Foundations of Computer Science*, 2002.

[66] Martine Labbé, Gilbert Laporte, Inmaculada Rodrígues Martín, and Juan José Salazar González. The median cycle problem. Technical Report 2001/12, Department of Operations Research and Multicriteria Decision Aid at Université Libre de Bruxelles, 2001.

[67] Youngho Lee, Yuping Chiu, and Jennifer Ryan. A branch and cut algorithm for a Steiner tree-star problem. *INFORMS Journal on Computing*, 8(3):194–201, 1996.

[68] Stefano Leonardi and Guido Schäfer. Cross-monotonic cost sharing methods for connected facility location games. Unpublished manuscript, 2004.

[69] Mohammad Mahdian, Yinye Ye, and Jiawei Zhang. Improved approximation algorithms for metric facility location problems. In *APPROX*, 2002.

[70] Ramgopal R. Mettu and C. Greg Plaxton. The online median problem. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 339–348. IEEE Comput. Soc. Press, 2000.

[71] Adam Meyerson, Kamesh Munagala, and Serge Plotkin. Cost-distance: Two metric network design. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 624–630, 2000.

[72] Adam Meyerson, Kamesh Munagala, and Serge Plotkin. Designing networks incrementally. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 406–415, 2001.

[73] Rolf H. Möhring, Andreas S. Schulz, and Marc Uetz. Approximation in stochastic scheduling: the power of lp-based priority policies. *Journal of the ACM (JACM)*, 46(6):924–942, 1999.

[74] Hervé Moulin. *Cooperative Microeconomics: A Game-Theoretic Introduction.* Princeton University Press, 1995.

[75] Hervé Moulin and Scott Shenker. Strategyproof sharing of submodular costs: budget balance versus efficiency. *Economic Theory*, 18(3):511–533, 2001.

[76] Martin Pál. Cost sharing for metric facility location. Final project for the class OR&IE 629 Game Theory, 2002.

[77] Martin Pál and Éva Tardos. Group strategyproof mechanisms via primal-dual algorithms. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 584–593, 2003.

[78] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 1995.

[79] Robert C. Prim. Shortest interconnection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[80] R. Ravi and F.S. Salman. Approximation algorithms for the traveling purchaser problem and its variants in network design. In *Proceedings of the 7th European Symposium on Algorithms (ESA '99–Prague)*, volume 1643 of *Lecture Notes in Comput. Sci.*, pages 29–40, Berlin, 1999. Springer.

[81] R. Ravi and Amitabh Sinha. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. In *Proceedings of the 10th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2004. *GSIA Working Paper 2003-E68*.

[82] Gabriel Robins and Alexander Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779, 2000.

[83] F. Sibel Salman, Joseph Cheriyan, R. Ravi, and Sairam Subramanian. Approximating the single-sink link-installation problem in network design. *SIAM Journal on Optimization*, 11(3):595–610, 2000.

[84] R. Schultz, L. Stougie, and M. H. van der Vlerk. Two-stage stochastic integer programming: a survey. *Statist. Nederlandica*, 50(3):404–416, 1996.

[85] L. S. Shapley. On balanced sets and cores. *Naval Research Logistics Quaterly*, 14:453–460, 1957.

[86] David Shmoys and Chaitanya Swamy. Stochastic optimization is (almost) as easy as deterministic optimization. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 2004.

[87] Martin Skutella and Marc Uetz. Scheduling precedence-constrained jobs with stochastic processing times on parallel machines. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 589–590. Society for Industrial and Applied Mathematics, 2001.

[88] Chaitanya Swamy and Amit Kumar. Primal-dual algorithms for connected facility location problems. In *APPROX*, 2002.

[89] Kunal Talwar. Single-sink buy-at-bulk LP has constant integrality gap. In *Proceedings of the 9th Integer Programming and Combinatorial Optimization Conference*, volume 2337 of *Lecture Notes in Computer Science*, pages 475–486, 2002.

[90] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, New York, 2003.

[91] Dachuan Xu. A multilevel facility loaction game. Unpublished manuscript, 2004.

[92] H. P. Young. Cost allocation. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory*, volume 2, chapter 34, pages 1193–1235. North-Holland, 1994.