Approximation Algorithms for Combinatorial Optimization Under Uncertainty

by

Maria Minkoff

S.B., Mathematics with Computer Science
MIT, 1998
S.M., Electrical Engineering and Computer Science
MIT, 2000

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

Author .....
Department of Electrical Engineering and Computer Science
August 18, 2003

Certified by .....
David R. Karger
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Approximation Algorithms for Combinatorial Optimization Under Uncertainty

by

Maria Minkoff

Submitted to the Department of Electrical Engineering and Computer Science
on August 18, 2003, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Combinatorial optimization problems arise in many fields of industry and technology, where they are frequently used in production planning, transportation, and communication network design. Whereas in the context of classical discrete optimization it is usually assumed that the problem inputs are known, in many real-world applications some of the data may be subject to an uncertainty, often because it represents information about the future. In the field of stochastic optimization uncertain parameters are usually represented as random variables that have known probability distributions.

In this thesis we study a number of different scenarios of planning under uncertainty motivated by applications from robotics, communication network design and other areas. We develop approximation algorithms for several $\mathcal{NP}$-hard stochastic combinatorial optimization problems in which the input is uncertain - modeled by probability distribution - and the goal is to design a solution in advance so as to minimize *expected* future costs or maximize *expected* future profits. We develop techniques for dealing with certain probabilistic cost functions making it possible to derive combinatorial properties of an optimum solution. This enables us to make connections with already well-studied combinatorial optimization problems and apply some of the tools developed for them.

The first problem we consider is motivated by an application from AI, in which a mobile robot delivers packages to various locations. The goal is to design a route for robot to follow so as to maximize the value of packages successfully delivered subject to an uncertainty in the robot's lifetime. We model this problem as an extension of the well-studied Prize-Collecting Traveling Salesman problem, and develop a constant factor approximation algorithm for it, solving an open question along the way.

Next we examine several classical combinatorial optimization problems such as bin-packing, vertex cover, and shortest path in the context of a "preplanning" framework, in which one can "plan ahead" based on limited information about the problem input, or "wait and see" until the entire input becomes known, albeit incurring additional expense. We study this time-information tradeoff, and show how to approximately optimize the choice of what to purchase in advance and what to defer.

The last problem studied, called *maybecast* is concerned with designing a routing network under a probabilistic distribution of clients using locally availabel information. This problem can be modeled as a stochastic version of the Steiner tree problem. However probabilistic objective function turns it into an instance of a challenging optimization problem with concave costs.

Thesis Supervisor: David R. Karger
Title: Associate Professor

Acknowledgments

I am very grateful to many people for helping me to obtain my Ph.D. First and foremost I would like to thank my advisor, David Karger, who has given me a great deal of his time throughout my years as a graduate student, providing direction and encouragement, stimulating new ideas and helping me to work through some setbacks. I am also grateful to David for his generous feedback on the multiple drafts of this thesis. He has been instrumental in helping me to achieve a greater clarity and unity of presentation, as well as in recovering the missing articles from my writing.

Next I would like to thank my thesis committee members Piotr Indyk and Santosh Vempala for their time and input, and my co-authors Avrim Blum, Shuchi Chawla, Nicole Immorlica, Terran Lane, Adam Meyerson, and Vahab Mirrokni for collaborating with me on various research problems that have led to the results presented in this thesis.

I am extremely fortunate to have been supported by an AT&T Labs Fellowship throughout my graduate studies. In addition to providing generous financial support, this program has given me an opportunity to meet and work with some great minds at AT&T Labs. I am eternally grateful to David Johnson, my mentor in the program, who had gotten me started on my first serious research project which has subsequently led to a paper as well as my Master's thesis. As my mentor, David was always quick to offer encouragement and helpful advice whenever I turned to him. My gratitude also goes to David Applegate, Howard Karloff, S. Muthukrishnan, Steven Phillips, Peter Shor, and many others who made me feel welcome at the Labs during my three very memorable summers there.

My experience at MIT's Lab for Computer Science would not have been as enjoyable if not for many great students here. Student seminars in the Theory group provided a wonderful opportunity to share results and learn from others (as well as to determine the number of theory students it takes to change a light bulb in a projector). I am particularly thankful to Yevgeniy Dodis, Venkatesan Guruswami, Eric Lehman, Anna Lysyanskaya, Sofya Raskhodnikova, Matthias Ruhl for their friendship and helpful advice on research, academic career, and life in general.

Most of all I must thank my close friends and family. Dejan Mircevski for his tremendous support and encouragement that have helped to carry me through many ups and downs of my graduate career; Richard Hahnloser for his patience and calming presence that helped me to maintain my sanity through the most stressful moments of the thesis preparation. I am eternally grateful to my parents for instilling in me the appreciation of reason and logic, and for teaching me to never give up (and resign) a game of chess (or any other endeavor in life) until checkmated. Finally, I would like to dedicate this thesis to my grandfather, Lazar Shteingart, who has been tirelessly saving for me all the news articles on modern supercomputers, and whose own dreams of earning a Ph.D. in Electrical Engineering were first put on hold by World War II, and later shattered by the antisemitic policies of the Soviet regime.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Combinatorial optimization problems arise in many fields of industry and technology, where they are frequently used in production planning, transportation, and communication network design. The area of the theoretical computer science, in particular the analysis of algorithms subfield, has developed a large number of classical discrete optimization problems, in which one seeks to optimize a linear cost function over a (finite) set of combinatorial objects, and a body of techniques to solve them. Within this framework it is usually assumed that one has perfect information about the input when solving the problem. However, for many real-world applications some of the data may be subject to uncertainty, either due to measurement errors, or, more fundamentally, because it is not yet determined. In practice it is frequently necessary to purchase or allocate resources for needs that are not precisely known in advance and might evolve over time. For example, telecommunication companies must install sufficient bandwidth in their networks before the demand for it is realized. Manufacturers have to decide where to build new facilities (e.g. stores or warehouses) and how much capacity to allocate in the presence of fluctuations in needs of potential customers. Electrical utility companies must plan how much power to produce in advance without knowing future demand which frequently depends on such an uncertain factor as weather [46].

There are many ways of dealing with uncertain inputs that evolve over time when solving an optimization problem. In contrast to the framework of on-line algorithms, in which one assumes nothing (or next to nothing) about future inputs, we are interested in studying scenarios in which the future is known to some degree. In particular we employ *stochastic* models in which uncertain parameters are represented as random variables that have known probability distributions. Such a distribution might be available from statistical data, or may be just an educated guess. This approach connects us with the framework of stochastic programming, an extension of mathematical programming in which some of the parameters incorporated into the objective function and/or constraints are random variables. Such stochastic modeling of the future is a particularly attractive approach for planning under uncertainty because it takes into account the effects of all potential outcomes, but allows us at the same time to seek a solution which is "good" on average.

We consider three potential scenarios which arise in the context of planning under uncertainty. In the first scenario, a solution must be constructed before any information about unknown parameters is revealed. Thus, the same solution is used for all possible outcomes, although its value is a function of the random input. This means that the cost function itself becomes a random variable that inherits its probability distribution from uncertain inputs.

Our objective is to optimize the solution value in expectation over the random values of uncertain parameters. Of course, this framework is applicable only if there exists at least one solution that is feasible for all possible outcomes of the random variables.

An alternative approach to solving problems involving uncertainties is to allow one to postpone making a final decision (e.g. constructing a full solution) until after learning the values of inputs. In fact, this might be the only way to deal with scenarios in which it is impossible to make a sensible decision without observing at least some of the inputs first. Under the stochastic modeling of the future, we are given a probability distribution on all the possible outcomes. Thus, we could specify in advance an individual solution to be used for each particular outcome. In practice, however it might be desirable to construct at least some part of the solution in advance as allocating resources at the last minute may be considerably more expensive than reserving them in advance. In such a setting there is a tradeoff between having perfect information (which becomes available with time) and minimizing cost (which increases with time).

There are also scenarios in which although technically we are not required to specify a fixed solution before observing random inputs, it might be difficult to take full advantage of the additional information when it becomes available. This is frequently the case in the distributed network setting, in which obtaining global information might be too expensive (or might take too long), so one has to act based just on local information. In such a scenario, we are interested planning a set of distributed solutions that depend only on the values of local parameters. This locality requirement prevents us from having an optimal solution for each possible outcome (as that would require using global information), so once again we aim to minimize the expected value of the solution.

In this thesis we study several stochastic combinatorial optimization problems representing these different scenarios for planning under uncertainty. Although a number of our problems can be modeled and solved using stochastic linear (or integer) programming tools, we are interested in tackling these problems by exploiting their underlying combinatorial structure and directly applying techniques developed for combinatorial optimization. One of our approaches to dealing with the stochastic aspects of the problems is to analyze combinatorial properties of an optimal solution for very small and very large probability values. In many cases reconciling the two allows us to derive "threshold" properties of the cost function which makes it possible to decompose the problem into independent parts. A related approach is to aggregate probability mass so as to obtain essentially deterministic subproblems. This enables us to make connections with already well-studied discrete optimization problems and apply some of the techniques developed for them. Alternatively, new exciting variants of old classic problems may arise along the way, and new methods have to be developed to deal with them. Some of these novel problems become challenging theoretical questions all by themselves.

The first problem we consider is motivated by an application from the field of artificial intelligence, in which a mobile robot delivers packages to various locations. The goal is to design a route for robot to follow so as to maximize the value of packages successfully delivered before the robot's battery runs out or it breaks down. If we know in advance precisely how long the robot is going to last, then this problem can be modeled as an instance of the *Orienteering* problem, a variant of the well-studied Prize-Collecting Traveling Salesman problem (PC-TSP). In practice however, the robot's lifetime tends to be random, and becomes known too late to be useful in planning a delivery route. Our approach to dealing with such an uncertainty is to aim for a solution that optimizes (over the random duration of a route) the expected value of packages delivered. Modeling robot's lifetime

as a random variable with an exponential distribution leads us to consider the *Discounted-Reward Traveling Salesman Problem*. This problem can be thought of as an extension of the PC-TSP, but with an exponential instead of a linear dependence of the objective function on the length of the tour.

Next we consider a "preplanning" framework for studying stochastic versions of several classical combinatorial optimization problems. In this framework, one may purchase a partial solution early and cheaply based on limited information about the problem input. Once the entire input becomes known, the rest of the solution has to be constructed, albeit at a greater cost. We study the tradeoff between planning ahead and deferring decisions about resource allocation until having the complete information for a number of classical discrete optimization problems: bin packing, vertex cover, min-cost flow, shortest path, and the Steiner tree problem.

Finally we turn our attention to the so-called *maybecast* problem which is concerned with designing a routing network to connect up a set of potential clients to a central server so as to minimize total routing cost. The server might contain some data or documents that can be requested by any number of clients. We consider a scenario in which the exact set of requesters is not known in advance – each client makes his own decision independently of others on whether he wants a particular document. Our goal is to plan a local solution to be used – each requester specifies his own path to the server without any knowledge of who else in the network wants the same data item – that minimizes the total routing cost in expectation over a random set of requesters. This problem is essentially a stochastic version of the (rooted) Steiner tree problem: given a probability with which a node needs to be connected to the root r, specify a path for it to use to communicate with the root so as to minimize the expected network usage. The probabilistic objective function turns the classical Steiner tree problem into an instance of a challenging optimization problem with concave costs.

In the remainder of this chapter we separately introduce each of the problems considered, briefly describe technical challenges associated with solving each, and provide an overview of related work.

1.1 Planning package delivery in advance

Imagine a new building housing MIT's Artificial Intelligence laboratory in which a mobile robot delivers packages to various offices. Each package has a value of importance associated with it. We would like the robot to drop off packages in a timely manner, while attempting to respect the relative priority of packages. This can be modeled by associating with each package a reward r_i (related to its priority value) to be collected upon a successful delivery. The goal is to plan a delivery route so as to maximize aggregated reward over time. Unfortunately, there is a certain degree of unreliability in the robot's behavior: for example, it may break down or its battery may get discharged before all the packages are delivered and the route is completed.

Such uncertainty about the future arises in many real-world planning scenarios. Within the rich framework of Markov Decision Processes (MDPs), the area which motivated this problem, such uncertainty is often handled by *discounting* the present value of rewards that will be received in the distant future [36, 37]. A common approach is to use an *exponential discount factor* γ, and assign to a reward r that will be collected at time t a discounted *present value* of $r \cdot \gamma^t$. This guides a robot to get as much reward as possible as early as

possible, thus providing some protection against early failure. One can motivate the use of an exponential discount by imagining that at each time step there is a fixed probability of robot losing power (or some other catastrophic failure occurring), so the probability that the robot is still alive at time t is exponentially distributed. Hence exponentially discounted reward reflects the value of the reward collected in expectation.

It is worth noting that when γ is very close to 1, e.g. $\gamma = 1 - \epsilon$ for $\epsilon \to 0$, then the discount is roughly a linear function of time $\gamma^t \to 1 - \epsilon t$. Thus, our objective of maximizing the total discounted reward becomes equivalent to minimizing the weighted sum of arrival times (latencies) at the reward locations. The latter problem (with unit rewards) is known as the traveling repairman [1] or the minimum latency problem (MLP) [11, 23].

A different approach to maximizing reward collected by the robot before it becomes disabled is to estimate its expected lifetime, and use that as hard deadline on the duration of the robot's tour. Now we are interested in a delivery route that collects the maximum possible amount of reward before the deadline. In this scenario, the robot gets an (undiscounted) reward r_i for package i only if the delivery is made before a deadline D. This is known as the *Orienteering* (or Bank Robber) problem [25, 6]. In other words, the value of a reward goes to zero after the deadline. Note that the discounted reward approach can be viewed as a relaxation of the hard-deadline approach, as the value of a reward as a function of time decreases more smoothly in the former. We will show that these two approaches are in fact intimately related, reducing one to the other and providing the first constant-factor approximation algorithms for both types of problems.

1.1.1 Related work

The minimum latency problem is known to be $\mathcal{NP}$-hard even for the weighted trees [47]. This immediately implies that our Discounted-Reward TSP is also $\mathcal{NP}$-hard. There are several constant-factor approximation algorithms known for the MLP in general graphs [11, 23, 15], with the best having a guarantee of 3.59 [15]. Although the special case of the Discounted-Reward TSP (with γ close to 1) is equivalent to the minimum latency problem, the approximation guarantee from the latter is not preserved, since the problems have complimentary maximize-minimize objectives.

The motivation behind our Discounted-Reward problem is to find a path that collects as much reward as possible as early as possible. This makes it somewhat akin to the well-studied Prize-Collecting Traveling Salesman problem (PC-TSP) [8, 25, 24]. In the PC-TSP, a salesman has two goals: to maximize the amount of reward and to minimize the distance traveled. There are several ways of optimizing the tradeoff between these conflicting objectives. In the PC-TSP itself, one seeks to minimize the linear combination of the length of the tour and the prize left uncollected. An alternative approach is to fix one of the objectives and optimize the other one. Thus, in k-TSP the salesman is required to obtain a reward of value at least k while minimizing the length of the tour. In contrast, in the Orienteering problem defined above, one instead fixes a deadline D (a maximum distance that can be traveled) and aims to *maximize* the total reward collected. All three problems are $\mathcal{NP}$-hard by reduction from a classical Traveling Salesman Problem.

The PC-TSP can be approximated to within a factor of $2 - \frac{1}{n-1}$ (where n is the number of nodes) using a classic Primal-Dual algorithm for the Prize Collecting Steiner Tree problem, due to Goemans and Williamson [24]. The latter problem is identical to PC-TSP, except that we are required to construct a tree (as opposed to a tour). There are several approximations known for the k-TSP problem [5, 21, 13, 6], the best being a $(2+\epsilon)$-approximation

due to Arora and Karakostas [5]. Most of these approximations are based on Goemans-Williamson's algorithm mentioned above. These algorithms for k-TSP extend easily to the *unrooted* version of the Orienteering problem in which we do not fix the starting location [6]. Arkin et. al [4] give a constant-factor approximation to the rooted Orienteering problem for the special case of points in the plane. However, there is no previously known $O(1)$ approximation algorithm for the rooted Orienteering Problem or Discounted-Reward TSP in general graphs.

1.1.2 Our approach

Although the exponential discount is relatively rare in combinatorial optimization, we establish a property of an optimal solution that connects our problem with the standard framework of "prize-collecting" problems in which objective and/or constraints have a linear dependency on the path length. In particular we show that an optimal path must obtain a constant fraction of its reward early on when it is not discounted too much. Thus, by finding a path of fixed small length that collects a lot of (undiscounted) prize, we can approximate the optimal discounted reward. This is exactly the Orienteering problem. But, as mentioned above, the Orienteering problem previously had no constant-factor approximation for the rooted version, which is needed in order to solve our discounted reward collection problem.

Although there exist several approximation algorithms for the complimentary k-TSP problem, which can also produce paths of minimum length collecting a lot of prize, they are not of much help here. Solutions produced by these algorithms for k-TSP meet the fixed requirement on the total prize collected by allowing constant-factor approximation in the total length of the tour. As length feeds into the exponent of our discount factor, a constant-factor approximation in length turns into a polynomial factor in discounted reward, yielding terrible approximation to that quantity.

In order to solve both the Discounted-Reward and Orienteering problems, we devise a *min-excess* approximation algorithm for Prize Collecting TSP that approximates to within a constant factor the optimum *difference* between the length of a prize-collecting path and the length of the shortest path between its endpoints. Note that this is a strictly better guarantee than what can be obtained by using an algorithm for k-TSP. The latter would return a path that has length at most a constant multiple times the *total* optimal length from s to t. This could be much larger than what we can accept for our algorithms.

We give a $(2+\epsilon)$-approximation for the min-excess path problem. We then use this algorithm as a subroutine to obtain a 6.75-approximation for the Discounted-Reward TSP and a 4-approximation for the Orienteering problem. Although we use an approximation algorithm for the k-path problem (a variant of the k-TSP problem with a path instead of a tour) as a subroutine, our algorithms are largely combinatorial and exploit the structure of the problems.

This portion of the thesis is based on joint work with Avrim Blum, Shuchi Chawla, David Karger, Terran Lane, and Adam Meyerson. The preliminary version of the results will appear in FOCS 2003 [12].

1.2 Planning in stages with growing costs

In the robot navigation scenario considered above the main uncertainty arises from not knowing how long a robot will last ahead of time. Since this information becomes available

too late to be useful, we have to specify in advance a fixed solution that optimizes the desired objective function in expectation.

However, for some planning scenarios with uncertain inputs it is possible to "wait and see", deferring decisions about allocating resources and specifying strategies until after more information becomes known. There is often a tradeoff involved, in that decisions made later may be more expensive. For example, consider a scenario in which one needs to provision a communication network with only a limited knowledge of the demands of the future users of this network. It is usually preferable to reserve capacity ahead of time as one can get a cheap rate from a wholesale market. However over-provisioning can be quite unprofitable for the network provider. Instead, one can choose to use a conservative estimate of the client's demand and purchase additional capacity "on-demand", albeit at a more expensive rate.

We study this time-information tradeoff for combinatorial optimization problems with uncertainty in the inputs in the context of the stochastic programming framework. Formally, we postulate a probability distribution $\Pr[I]$ over problem instances I of a given combinatorial optimization problem. A feasible solution to each instance is a collection of elements from a ground set that satisfies a certain set of constraints. We are allowed to purchase a solution element e in advance at cost $c(e)$, or pay $\lambda c(e)$ ($\lambda \geq 1$) for it once all the "active" constraints that a solution has to satisfy are revealed. The goal is to select a set of elements to be purchased in advance so as to minimize the cost of a feasible solution in expectation over the choice of a random instance.

We examine a number of (generally $\mathcal{NP}$-hard) combinatorial optimization problems in which it makes sense postulate some probability distribution over input instances and to specify a portion of the solution in advance. We develop algorithms to find approximately optimal pre- and post-sampling parts of a feasible solution. In particular, we study stochastic versions of the following problems:

Min Cost Flow. Given a source and sink and a probability distribution on demand, buy some edges in advance and some after sampling (at greater cost) such that the given amount of demand can be routed from source to sink. We show that this problem can be solved via linear programming for a discrete demand distribution.

Bin packing. A collection of items is given, each of which will need to be packed into a bin with some probability. Bins can be purchased in advance at cost 1; after the determination of which items need to be packed, additional bins can be purchased at cost $\lambda > 1$. How many bins should be purchased in advance to minimize the expected total cost? We give a fully polynomial-time approximation scheme for bounded values of λ, i.e. we can obtain a solution of cost $(1+\epsilon)$ times the optimum in time polynomial in the length of the input, λ and $1/\epsilon$.

Vertex Cover. A graph is given, along with a probability distribution over sets of edges that may need to be covered. Vertices can be purchased in advance at cost 1; after determination of which edges need to be covered, additional vertices can be purchased at cost λ. Which vertices should be purchased in advance? We give a 4-approximation for the case where the probability distribution involves only polynomially many distinct edge sets, and a different 4-approximation algorithm for the case when each edge has to be covered with fixed probability p.

Shortest Path. We are given a graph and told that a randomly selected pair of vertices (or one fixed vertex and one random vertex) will need to be connected by a path. We

can purchase edge e at cost c_e before the pair is known or at cost λc_e after and wish to minimize the expected total edge cost. We show that this problem is equivalent to the multicommodity rent-or-buy problem, a known $\mathcal{NP}$-hard network design problem with a constant factor approximation algorithm.

Steiner Tree. A graph is given, along with a probability distribution over sets of terminals that need to be connected up. Edge e can be purchased at cost c_e in advance or at cost λc_e after the exact set of terminals is known. We give a constant factor approximation for the case where the expected number of terminals is constant (generalizing the Cheap Path result). We give a constant factor approximation for the case where edges form an ultrametric and an $O(\log n)$ approximation for general edge costs.

1.2.1 Related work

Our preplanning approach is a special case of two-stage stochastic programming with recourse, one of the most widely used stochastic programming models [42]. In this model, a planner makes some decisions in the first stage, after which a random event occurs, for example some uncertain parameters become known. In the second stage, a planner can take a recourse action to respond to that event, e.g. improve a solution by using additional information that has become available. The goal is to minimize the cost of the first-stage decision plus the expected cost of the second-stage recourse action.

Stochastic programs are generally tackled via a combination of mathematical programming and advanced probabilistic techniques. A key difficulty in solving these problems efficiently is dealing with an uncertainty space that is huge and frequently leads to computationally intractable problems. In fact, several optimization problems that are polynomial-time solvable with deterministic inputs become $\mathcal{NP}$-hard in a stochastic setting [2]. When the random data is discretely distributed, one may duplicate constraints for each potential setting of uncertain parameters and write out the expectation as a finite sum of solution costs over all outcomes weighted by the corresponding probability. The resulting formulation is a deterministic linear (mathematical) program, but its size is proportional to the total number of outcomes, which might be exponential in the size of the original problem. Even in those cases in which it is possible to use only polynomially many linear constraints to describe the set of feasible solutions (for an exponential number of potential outcomes), the expected cost function frequently doesn't have the nice properties (linearity and convexity) of the original cost function.

1.2.2 Our approach

We start by illustrating how a standard merger of linear programs approach described above can be employed to solve stochastic versions of some combinatorial optimization problems, using the min-cost flow and the vertex cover (with polynomially many distinct edge sets) problems as examples. This method is feasible for both problems since in each only a limited number of instances can arise, so a combined linear (integer) program that represents the preplanning version of each problem has bounded size. Hence, we can find an optimum solution for the preplanning version of the min-cost flow problem by applying any polynomial-time algorithm for linear programming. In the case of the vertex cover problem we show how to obtain a constant factor approximation by rounding the fractional solution to the linear programming relaxation of the corresponding integer programming formulation.

However, our main approach involves exploiting the underlying combinatorial structure of the problems and making connections with already well-studied combinatorial optimization problems with known approximation algorithms. One of our main tools is the "threshold property", which is a local-optimality condition that specifies that an element is worth purchasing in advance if and only if it is likely to be used in a solution of a random instance. By aggregating probability mass (with only a slight increase in the solution cost) we can ensure that certain solution elements get to be used with high probability. This in turn allows us to determine the set of elements to buy in advance. For example, in the case of the preplanning vertex cover problem in which edge have to be covered independently with probability p, we can identify a subset of vertices of high degree, each of which has a very high probability of being used in a nearly optimal vertex cover of a random instance. We show that buying all of them in advance gives a solution of cost not much higher than optimal.

Another important tool that we repeatedly apply involves reducing (modulo a small constant factor) the preplanning version of an $\mathcal{NP}$-hard problem (such as the Steiner tree problem) to a preplanning instance of another optimization problem (such as the minimum spanning tree problem) that has a polynomial-time algorithm. The latter problem might have some nice combinatorial properties that we can exploit in constructing our solution.

This portion of the thesis is based on joint work with Nicole Immorlica, David Karger, and Vahab Mirrokni.

1.3 Planning routing paths with incomplete global knowledge

In contrast to our preplanning framework, in which we have assumed that one eventually learns the values of all uncertain parameters, there are many scenarios in which one does not get to observe the full set of random inputs even after they are fully determined. This is frequently the case in the distributed network setting, in which obtaining global information might be too expensive (or might take too long), so one has to act based just on local information. In such a case we are restricted to planning solutions that have simple local specifications.

Consider a networking problem of distributing a single data item to multiple requesters while minimizing network usage. If caches are placed at nodes in the network then as soon as a requester's path to the root encounters a cached copy of the item, the item can be returned without traversing the rest of the path. Thus every edge on the set of paths is traversed at most once by the item, so the total number (cost) of edges on the paths connecting root to requesters reflects the total network bandwidth (transmission cost) allocated to the distribution of the item.

If the the full set of clients who would want an item is known in advance, we can solve this problem by modeling it as a Steiner tree problem: connect all requesters to the root using a minimum cost set of edges. Consider now a scenario in which the set of clients requesting an item is random. In this case, there might be a separate optimal tree for each possible set of requesters. However, constructing one requires learning the set of actual requesters, information which often becomes available too late to be useful. Instead we aim for a solution designed to make use only of local information – each client's decision on whether to make a request. Thus, we would like to specify for each client a fixed path to the central server to be used in case he decides to get an item. Alternatively, we might

require a "routing tree" solution, in which each node simply specifies a parent to which it forwards any request it receives, and the path to the server is determined by following parent pointers. Note that with both approaches, the path via which a client makes a request and receives the data item of interest does not depend in any way on who else wants the same item. We will show that these two approaches are in fact equivalent.

Such a philosophy has led us to study the following *maybecast* problem. We are given a network with edge costs, a *root* node, and a collection of N *clients* at nodes in the network. Each client i will choose to contact the root independently from others with some probability p_i. In advance of the requests we must choose, for each client in the network, a path from the client's node to the root. If a client chooses to contact the root, the edges on this path become *active*. Our goal is to minimize, over the random choices of the clients, the expected cost of active network edges. We can think of this as a stochastic version of the Steiner tree problem.

The maybecast problem is $\mathcal{NP}$-complete, as the Steiner tree problem is a special case with terminals having probability 1 of becoming active. The problem remains $\mathcal{NP}$-complete even in the uniform case of one client per network node and unit costs on edges. Thus, we focus our attention on approximation algorithms.

1.3.1 Our approach

Given a particular set of paths for clients to use, the expected cost of an edge e is equal to its length times the probability of it being used, e.g. the probability that at least one of the clients whose assigned path contains e becomes active. Note that this probability is a concave function of the number of clients assigned to use the edge. Thus, our problem can be thought of as single-sink min-cost flow problem with concave costs. This implies that an optimal solution to the maybecast problem is invariably a tree. However, the obvious first choice of a shortest path tree can cost a factor as large as $\Theta(n^{1/2})$ times the optimum in an n-node graph.

To find a better tree, we analyze structural properties of an optimum solution. In particular, we note that the optimum tree consists of a central "hub" area within which all edges are basically certain to be used, together with a fringe of "spokes" in which multiple clients can be considered to be contributing *independently* (and linearly) to the cost of the solution. In other words, the optimum solution first aggregates probability mass, i.e. clusters clients into "super-clients" whose probability of being active is very close to 1. The resulting problem reduces to the deterministic case, e.g. an instance of the regular min-cost Steiner tree problem.

To cluster clients, we introduce a new version of the facility location problem, which we call the r-gathering problem. In this version, also known as a load balanced facility location problem [26], every open facility is required to have some minimum amount of demand assigned to it. We obtain a simple bi-criterion approximation for this problem, one which is loose in both assignment cost and minimum demand. Our solution uses this algorithm with $r = 1$ to build hubs which are then connected up to the root using an approximation algorithm for the Steiner tree problem to yield a constant factor approximation to the maybecast problem.

This portion of the thesis is based on joint work with David Karger. A preliminary version of the results appeared in FOCS 2000 [33].

1.3.2 Related Work

The maybecast problem can be represented as a kind of min-cost flow problem with infinite capacities and a *concave* cost function. We can think of a client i as having "demand" for a flow of capacity p_i to the root. The cost of routing along an edge exhibits an *economy of scale*: the more paths use an edge, the cheaper it is per path. By approximating our cost function by a piece-wise linear function, we can establish a connection with the *buy-at-bulk network design* problem (originally introduced by Salman et al. [49]), another problem which exhibits an economy of scale. As an example, consider the problem of wiring a telephone network in some metric space such that every pair of clients is connected by a path of capacity one (devoted to just that pair). Our approximate cost function would correspond to the situation in which two types of wires are available: low cost wires of unit capacity, and "infinite" capacity wires of large cost.

In the general buy-at-bulk network design problem, one is given an undirected graph, and a set of source-sink pairs with non-negative demands. The goal is to install enough capacity on the edges in order to route all the demands while minimizing the total cost, which is a concave function of the capacity bought. The problem is known to be $\mathcal{NP}$-hard [49].

Prior work. Awerbuch and Azar [7] provided a randomized $O(\log^2 n)$ approximation algorithm for this problem, where n is the number of nodes in the network. Recent improvements in tree embedding guarantees (employed by the latter algorithm) yield a somewhat better approximation ratio of $O(\log n)$ [18]. Subsequent work has tended to focus on the single-sink version of buy-at-bulk network design and its variations. Andrews and Zhang [3] gave an $O(K^2)$-approximation algorithm for the access network design problem, a special case of the sigle-sink buy-at-bulk network design, which was subsequently improved to a constant factor (independent of K) by Guha et al. [26]. However, restrictions placed on the cost function used in the access network design problem preclude application to our maybecast problem.

Subsequent results. Since the initial (conference) publication of our results [33] there followed a number of developments in network design with economies of scale. In particular, considerable improvements have been made in performance guarantees for single-sink buy-at-bulk network design with arbitrary concave, piece-wise linear cost function [22, 27, 54], with the best being a 73-approximation due to Gupta et. al [30]. Although this result is applicable to the maybecast problem, we achieve a better guarantee of 37 by using our customized algorithm for the problem.

Finally, we note that our maybecast problem can be modeled (modulo a small constant factor) as an instance of the Connected Facility Location (CFL) problem formulated by Gupta et al. [28] after the initial publication of our work. In the latter problem one seeks to open a subset of facilities, assign demands to them, and finally connect up all the opened facilities via a min-cost Steiner tree. The current state-of-art algorithm for the CFL provides an approximation guarantee of 3.55 [30] which translates into roughly 6-approximation for the maybecast problem.

Chapter 2

Profit-making Under Uncertainty

Algorithms for traveling salesmen and mobile robots

In this chapter we consider a scenario of planning under uncertainty motivated by an application from the field of artificial intelligence, in which a mobile robot delivers packages to various locations. The goal is to design a route for robot to follow so as to maximize the value of packages successfully delivered before the robot's battery runs out or it breaks down. If one can estimate in advance how long the robot is going to last, then this problem can be modeled as an instance of the Orienteering problem, a variant of the well-studied Prize-Collecting Traveling Salesman problem (PC-TSP). Alternatively, taking the robot's lifetime to be a random variable with an exponential distribution and aiming for a route that maximizes the expected reward collected leads us to consider the Discounted-Reward Traveling Salesman Problem. We give constant factor approximation algorithms for both problems. Although the unrooted orienteering problem, where there is no fixed start node, has been known to be approximable using algorithms for related PC-TSP problems, ours is the first to approximate the rooted variant, solving an open problem [6, 4].

2.1 Introduction

Consider a robot delivering packages to various locations in a building. Each package has a value or importance associated with it. The robot should deliver packages in a timely manner, with the more important packages getting priority over less important ones. Unfortunately, there is a certain degree of unreliability in the robot's behavior: for example, it may break down or its battery may get discharged before all the packages are delivered and the route is completed. The objective then becomes to maximize the value of packages that can be delivered before the robot becomes disabled.

This type of problem with uncertainty about the future can be modeled in many ways. In the artificial intelligence community, motivated by ideas from Markov decision processes (MDPs) [52, 43, 36, 37], it is traditional to consider a "discounted reward model" in which the value associated with reaching a given state decreases the longer it takes to reach that state. The theory community, on the other hand, has modeled this scenario as kinds of Prize-Collecting Traveling Salesman Problems [25, 8, 24, 6], in which the goal is to obtain a required amount of prize as quickly as possible.

2.1.1 Markov Decision Processes

A Markov Decision Process (MDP) consists of a state space S, a set of actions A, a probabilistic transition function T, and a reward function R. At any given time step, an agent acting in an MDP will be located at some state $s \in S$, where he can choose an action $a \in A$. The agent is subsequently relocated to a new state s' determined by the transition probability distribution $T(s'|s,a)$. The probabilistic nature of the transition function allows one to model unreliability in the robot's behavior or external forces that might do something unpredictable to the robot's state. At each state s, an agent collects reward $R(s)$ (or, sometimes, rewards are put on state-action pairs). For example, a package-delivery robot might get a reward every time it correctly delivers a package. Note that each action defines a probability distribution of the next state; if actions were pre-determined, then we would get just a Markov chain.

In order to encourage the robot to perform the tasks that we want, and to do so in a timely manner, a standard objective considered in MDPs is to maximize *discounted reward*. Specifically, for a given *discount factor* $\gamma \in (0,1)$, the value of reward collected at time t is discounted by a factor γ^t. Thus the total discounted reward, which we aim to maximize, is $R_{tot} = \sum_{t=0} R(s_t)\gamma^t$. This guides the robot to get as much reward as possible as early as possible.Exponential discounting is a good model for scenarios in which at each time step, there is some fixed probability the tour will end (the robot loses power, a catastrophic failure occurs, etc.) In that case, the probability that the robot is alive at time t is exponentially distributed. Thus, for a given path, the total discounted reward is equal to the amount of reward collected on that path in expectation. Exponential discounting also has the nice mathematical property that it is *time-independent*, meaning that an optimal strategy can be described just by a *policy*, a mapping from states to actions. The goal of planning in an MDP is to determine the optimal policy: the mapping of states to actions that maximizes expected discounted reward $E[R_{tot}]$.

There are well-known algorithms for solving MDPs in time polynomial in the state space [9, 43, 52]. However, one drawback of using the MDP model in for our application is that the robot receives the reward for a state every time that state is visited (or every time the robot performs that action from that state if rewards are on state-action pairs). Thus, in order to model a package-delivery or search-and-rescue robot, one would need a state representing not only the current location of the robot, but also a record of all locations it has already visited (or alternatively, which packages remain to be delivered). This causes an exponential increase in the size of the state space. Thus, it would be preferable to directly model the case of rewards that are given only the *first* time a state is visited [36, 37].

As a first step towards tackling this general problem, we restrict our consideration to deterministic, reversible actions. This leads us to study the *Discounted-Reward Traveling Salesman Problem*, in which we assume we have an undirected weighted graph (edge weights represent the time to traverse a given edge), with a prize (reward) value π_v on each vertex v, and our goal is to find a path visiting each vertex v at time t_v so as to maximize $\sum \pi_v \gamma^{t_v}$.

2.1.2 PC-TSP and Orienteering problems

A different approach to our prize-distance tradeoff is to estimate the robot's lifetime and use it as a hard deadline on the time the robot may spend delivering its packages. The robot gets a reward equal to the value (prize) of the package on a delivery, but only if the delivery is made before a deadline D. If the deadline is exceeded, he gets no reward. This

problem has been studied previously as the Orienteering [25] or Bank Robber's [6] Problem.

Orienteering belongs to the family of the Prize-Collecting Traveling Salesman problems (PC-TSP). Given a set of cities with non-negative prize values associated with them and a table of pairwise distances, a salesman needs to pick a subset of the cities to visit so as to minimize the total distance traveled while maximizing the total amount of prize collected. Note that there is a tradeoff between the cost of a tour and how much prize it spans. The original version of the PC-TSP introduced by Balas [8] deals with these two conflicting objectives by combining them: one seeks a tour that minimizes the *sum* of the total distance traveled and the penalties (prizes) on cities skipped, while collecting at least a given quota amount of prize. Goemans and Williamson subsequently focused on a special case of this problem in which the quota requirement is dropped, and provided a primal-dual 2-approximation algorithm for it [24].

An alternative approach to the bicriterion optimization is instead to optimize just one of the objectives while enforcing a fixed bound on the other. For example, in a quota-driven version of the PC-TSP, called k-TSP, every node has a prize of one unit and the goal is to minimize the total length of the tour, while visiting at least k nodes. Similarly, Orienteering can be viewed as a budget-driven version of the PC-TSP, since we are maximizing total amount of prize collected, while keeping the distance traveled below a certain threshold.[1]

There are several constant-factor approximations known for the k-TSP problem [5, 21, 13, 6], the best being a $(2+\epsilon)$-approximation due to Arora and Karakostas [5]. Most of these results are based on a classic primal-dual algorithm for PC-TSP due to Goemans and Williamson [24] (mentioned above). The basic idea behind these algorithms is to run Goemans-Williamson's primal-dual algorithm for PC-TSP on a series of instances derived from the k-TSP problem with different (uniform) prize values assigned to nodes. By scaling prizes up or down, we can control how many vertices get picked by the algorithm for a tour. Chudak et al. show that this prize-scaling technique can be interpreted as a Lagrangean relaxation approach [16]. They start with a linear program for the k-MST problem that is closely modeled after the linear programming formulation of PC-TSP used by Goemans and Williamson in their primal-dual algorithm. Taking the quota constraint (which requires a tour to span at least k vertices) and applying Lagrangean relaxation to it, one obtains a linear program for PC-TSP with the Lagrangean factor playing the role of node prizes.

The algorithms for k-TSP extend easily to the *unrooted* version of the Orienteering problem in which we do not fix the starting location [6]. In particular, given a tour (cycle) of value Π whose length is cD for some $c > 1$, we can just break the cycle into c pieces of length at most D, and then take the best one, whose total value will be at least Π/c. Noting that an optimal cycle of length $2D$ must span at least as much prize as an optimal path of length D (since could just traverse the path forward and then back), we get a $2c$-approximation guarantee on the amount of prize contained in a segment we pick. However, this doesn't work for the rooted problem because the "best piece" in the above reduction might be far from the start. Arkin et. al [4] give a constant-factor approximation to the rooted Orienteering problem for the special case of points in the plane. However, there is no previously known $O(1)$ approximation algorithm for the rooted Orienteering Problem in general graphs. It should be noted that the Lagrangean relaxation technique used to solve k-MST is not quite applicable to the Orienteering problem because of a mismatch in the

[1]Strictly speaking, a budget-driven version of the PC-TSP would require a tour, e.g. a path that ends at the start node, whereas the Orienteering problem is content with a path that ends at an arbitrary node. We consider both versions of the problem.

objective functions: PC-TSP minimizes the sum of the total edge cost and prize foregone, whereas Orienteering maximizes the total prize collected. Applying Lagrangean relaxation to the budget constraint on the length of a tour results in an objective function that maximizes the total prize collected minus distance traveled. Although from the viewpoint of optimization this "net-worth" version of the PC-TSP is equivalent to the traditional (minimization) version, the same is not true when it comes to approximation. In particular, the net-worth objective is $\mathcal{NP}$-hard to approximate to within any constant factor [19].

2.1.3 Our Contribution

In this chapter, we give constant factor approximation algorithms for both the Discounted-Reward TSP and Orienteering problems. To do this, we devise a *min-excess* approximation algorithm that, given two endpoints s and t, approximates to within a constant factor the optimum *difference* between the length of a prize-collecting s-t path and the length of the shortest path between the two endpoints. Note that this is a strictly better guarantee than what can be obtained by using an approximation algorithm for k-TSP, which would return a path that has length at most a constant multiple times the *total* optimal length from s to t.

Using an approximation algorithm with a guarantee of α_{CP} for the min-cost s-t path problem as a subroutine, we get an $\alpha_{EP} = \frac{3}{2}\alpha_{CP} - \frac{1}{2}$ approximation for the min-excess (s,t)-path problem, a $1 + \lceil \alpha_{EP} \rceil$ approximation for Orienteering, and a roughly $e(\alpha_{EP}+1)$ approximation for Discounted-Reward TSP. Using results of Chaudhuri et. al [15], we get constants of $2+\epsilon$, 4, and $6.75+\epsilon$ for these problems respectively.

2.1.4 Organization

The rest of this chapter is organized as follows. We begin with some definitions in section 2.2. In section 2.3 we describe an algorithm for the min-cost path problem based on the results from the paper by Chaudhuri et. al [15]. We then give an algorithm for Min-Excess path in section 2.4, followed by algorithms for Discounted PC-TSP and Orienteering in sections 2.5 and 2.6 respectively. In section 2.7 we extend some of the algorithms to tree and multiple-path versions of the problems. We conclude in section 2.8.

2.2 Formal Problem Definitions

Our work encompasses a variety of problems. In this section we introduce the notation to be used throughout the chapter, provide formal problem statements and describe a uniform naming scheme for them.

Let $G = (V, E)$ be an undirected graph, with a distance function on edges, $d : E \to \mathbb{R}^+$, and a *prize* or *reward* function on nodes, $\pi : V \to \mathbb{R}^+$. Let $\pi_v = \pi(v)$ be the reward on node v. Let $s \in V$ denote a special node called the *start* or *root*.

For a path P visiting u before v, let $d^P(u,v)$ denote the length along P from u to v. Let $d(u,v)$ denote the length of the *shortest* path from node u to node v. For ease of notation, let $d_v = d(s,v)$ and $d^P(v) = d^P(s,v)$. For a set of nodes $V' \subseteq V$, let $\Pi(V') = \sum_{v \in V'} \pi_v$. For a set of edges $E' \subseteq E$, let $d(E') = \sum_{e \in E'} d(e)$.

Our problems aim to construct a certain subgraph—a path, tree, or cycle, possibly with additional constraints. Most of the problems attempt a trade-off between two objective functions: the *cost* (length) of the path (or tree, or cycle), and *total prize* spanned by it.

From the point of view of exact algorithms, we simply need to specify the cost we are willing to tolerate and the prize we wish to span. Most variants of this problem, however, are $\mathcal{NP}$-hard, so we focus on approximation algorithms. We must then specify our willingness to approximate the two distinct objectives. We refer to a *min-cost* problem when our goal is to *approximately* minimize the cost of our objective subject to a fixed lower bound on prize (thus, prize is a feasibility constraint while our approximated objective is cost). Conversely, we refer to a *max-prize* problem when our goal is to *approximately* maximize the prize collected subject to a fixed upper bound on cost (thus, cost is a feasibility constraint while our approximated objective is prize). For example, the min-cost tree problem is the traditional k-MST: it requires spanning k prize and aims to minimize the cost of doing so. Both the rooted and unrooted min-cost tree problems have constant-factor approximations [13, 6, 21, 5, 32]. The max-prize path problem, which aims to find a path of length at most D from the start node s that visits a maximum amount of prize, is the Orienteering problem.

The main subroutine in our algorithms requires that we introduce a variation on approximate cost. Define the *excess* of a path P from s to t to be $d^P(s,t) - d(s,t)$, that is, the difference between that path's length and the distance between s and t in the graph. Obviously, the minimum-*excess* path that spans total prize $\Pi(P)$ is also the minimum-*cost* path spanning prize $\Pi(P)$; however, a path of a constant factor times minimum cost may have more than a constant-factor times the minimum excess. We therefore consider separately the *minimum excess path* problem. Note that an (s,t) path approximating the optimum excess ϵ by a factor α will have length $d(s,t) + \alpha\epsilon \leq \alpha(d(s,t) + \epsilon)$ and therefore approximates the minimum cost path by a factor α as well. Achieving a good approximation to this min-excess path problem will turn out to be a key ingredient in our approximation algorithms.

Finally, as discussed earlier, we consider a different means of combining length and cost motivated by applications of Markov decision processes. We introduce a *discount factor* $\gamma < 1$. Given a path P rooted at s, let the *discounted reward* collected at node v by path P be defined as $\rho_v^P = \pi_v \gamma^{d^P(s,v)}$. That is, the prize gets discounted exponentially by the amount of time it takes for the path to reach node v. The *max-discounted-reward* problem is to find a path P rooted at s, that maximizes $\rho^P = \sum_{v \in P} \rho_v^P$. We call this the *discounted-reward TSP*. Note that the length of the path is not specifically bounded in this problem, though of course shorter paths produce less discounting.

2.2.1 Results

We present a constant-factor approximation algorithm for the max-prize path (rooted Orienteering) problem, solving an open problem of [6, 4], as well as the discounted-reward TSP. Central to our results is a constant-factor approximation for the *min-excess path* problem defined above, which uses an algorithm for the min-cost s-t path problem as a subroutine. We also give constant-factor approximations to several related problems, including the max-prize tree problem—the "dual" to the k-MST (min-cost tree) problem—and max-prize cycle. Specific constants are given in Table 2.1.

Our approximation algorithms reflect a series of reductions from one approximation problem to another. Improvements in the approximations for various problems will propagate through. We state approximation factors in the form α_{XY} where XY denotes the problem being approximated; the first letter denotes the objective (cost, prize, excess, or discounted prize denoted by C, P, E, and D respectively), and the second the structure

Problem	Current approx.	Source/Reduction
min-cost s-t path (α_{CP})	$2+\epsilon$	[15]
min-excess path (α_{EP})	$2.5+\epsilon$	$\frac{3}{2}(\alpha_{CP})-\frac{1}{2}$
	$2+\epsilon$	using α_{CP} from [15]
max discounted-prize path (α_{DP})	$6.75+\epsilon$	$(1+\alpha_{EP})(1+1/\alpha_{EP})^{\alpha_{EP}}$
max-prize path (α_{PP})	4	$1+\lceil\alpha_{EP}\rceil$
max-prize tree (α_{PT})	8	$2\alpha_{PP}$
max-prize cycle (α_{PC})	8	$2\alpha_{PP}$
max-prize multiple-path (α_{kPP})	4.52	$(1-e^{1/\alpha_{PP}})^{-1}$
max discounted multiple-path (α_{kDP})	$7.26+\epsilon$	$(1-e^{1/\alpha_{DP}})^{-1}$

Table 2.1: Approximation factors and reductions for our problems.

(path, cycle, or tree denoted by P, C, or T respectively).

Note that for the Min-Excess problem, we assert two approximation factors of $2.5+\epsilon$ and an improved factor of $2+\epsilon$. The first one is obtained using an algorithm for the min-cost s-t path as a "black box", so it applies to any such algorithm. The latter result is based explicitly on the the min-cost s-t path algorithm of [15] and employs a tight analysis of its approximation guarantee. To be more precise, we obtain $\alpha_{EP}=\max(\alpha_{CP},\frac{3}{2}\alpha_{CP}-1)$ using properties of the latter algorithm.

2.2.2 Preliminaries

Our algorithms for the max-prize variants guess the value Π of the optimum solution (or some closely related value) via binary search[2] and then apply our min-excess path algorithm with the target prize value Π. The latter algorithm uses a subroutine that finds a path of small length that spans a given amount of prize. Moreover, this subroutine is invoked as part of a dynamic program for all possible prize values in the range from 0 up to the target value Π. In order for our algorithms to run in polynomial time, the target prize value Π has to be polynomially bounded (in the number of vertices n). We can ensure this by multiplying all vertex prizes by n^k/Π (for some large constant k), thus setting the target value to n^k. If we now round every prize down to the nearest integer, we lose at most n units of prize overall, which is a negligible multiplicative factor. This negligible factor does mean that our approximation algorithms for the max-prize problems with guarantee c on polynomially bounded inputs have (weaker) guarantee "arbitrarily close to c" on arbitrary inputs. In the following discussion of the min-cost or min-excess variants, we will assume that the given prize value Π is polynomially bounded.

2.3 Min-Cost path algorithm

In the following sections we use as a subroutine an approximation algorithm for the min-cost path problem, in which given a target prize value k, the goal is to find a path between two pre-specified nodes of (approximately) minimum length that collects prize at least k. Note

[2]Technically we will be finding the highest value Π such that our algorithm comes within its claimed approximation ratio.

that the k-TSP problem is a special case of the min-cost path problem (with start and end nodes equal).

Let P^* be an optimum min-cost s-t path that spans at least k units of prize. We can find a path with prize value k that achives approximation guarantee $\alpha_{CP} = \alpha_{CC} + 1$ by the following algorithm that we will call MCP. MCP begins by merging s and t to a vertex r, and solving k-TSP with root r. The original path solution has become a (feasible) cycle, so the optimum cycle length is at most $d(P^*)$, meaning we find an approximate solution of length $\alpha_{CC}d(P^*)$. On the original graph, this solution may be a path from s to t, in which case we are done. Alternately, it is either a cycle ending at s, or two disjoint cycles: one at s and one at t. In these latter cases, we simply add a shortest s-t path (which is clearly no longer than $d(P^*)$), increasing the approximation ratio by at most 1. Substituting the best currently known approximation factor for k-TSP $\alpha_{CC} = 2 + \epsilon$ [5], we obtain an approximation guarantee of $3 + \epsilon$ for the min-cost path problem.

However, an even better approximation guarantee can be obtained by using results from the recent paper by Chaudhuri et. al [15]. This paper provides an approximation algorithm for the minimum latency problem by repeatedly computing a solution to the k-MST (min-cost tree) problem whose cost is bounded in terms of the cost of the min-cost s-t *path* containing k vertices (such a path provides a lower bound on the latency of the k_{th} vertex in a minimum latency tour). In particular, the subroutine of Chaudhuri et. al returns a tree containing s and t that spans at least k vertices and has cost at most $(1 + \delta)$ times the cost of the shortest s-t path with k vertices, for any fixed constant δ. Note that the algorithm can be used to obtain a tree spanning a given target *prize value* via the following tansformation of the original graph: each node v with a prize π_v gets $\pi_v - 1$ leaves attached to it. (Complete details of such a transformation can be found [39].)

To construct an s-t path from the tree obtained by the subroutine of Chaudhuri et. al, we can double all the edges, except those along the tree path from s to t. This gives us a partial "Euler tour" of the tree that starts at s and ends at t. Clearly, the cost of such a path is at most $(2+\delta)$ times the cost of the shortest s-t path spanning prize k, for any fixed constant δ.

2.4 Min-Excess Path

We now proceed to give our approximation algorithm for the Min-Excess Path (MEP) problem. Let P^* be the shortest path from s to t with $\Pi(P^*) \geq k$. Let $\epsilon(P^*) = d(P^*) - d(s,t)$. Our algorithm returns a path P with $\Pi(P) \geq k$ and length $d(P) = d(s,t) + \alpha_{EP}\epsilon(P^*)$, where $\alpha_{EP} = \frac{3}{2}\alpha_{CP} - \frac{1}{2}$. Thus we obtain a $(2.5+\delta)$-approximation to min-excess path using an algorithm of Chaudhuri et. al [15] for min-cost s-t path with $\alpha_{CP} = 2 + \delta$.

The idea for our algorithm for min-excess path is as follows. Suppose that the optimum solution path encounters all its vertices in increasing order of distance from s. We call such a path *monotonic*. We can find this optimum monotonic path via a simple dynamic program: for each possible prize value p and for each vertex i in increasing order of distance from s, we compute the minimum excess path that starts at vertex s, ends at i, and collects prize at least p.

We will solve the general case by breaking the optimum path into continuous *segments* that are either monotonic (so can be found optimally as just described) or "wiggly" (generating a large amount of excess). We will show that the total length of the wiggly portions is comparable to the excess of the optimum path; our solution uses the optimum monotonic

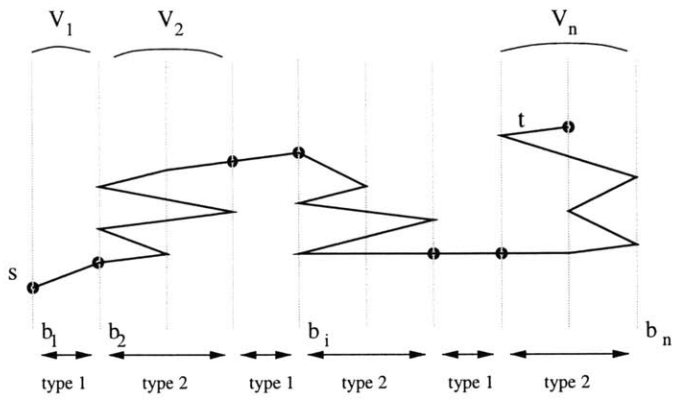

Figure 2-1: Segment partition of a path in graph G

paths and approximates the length of the wiggly portions by a constant factor, yielding an overall increase proportional to the excess.

Consider the optimal path P^* from s to t. We divide it into segments in the following manner. For any real d, define $f(d)$ as the number of edges on P^* with one endpoint at distance less than d from s and the other endpoint at distance at least d from s. Note that $f(d) \geq 1$ for all $0 \leq t \leq d_t$ (it may also be nonzero for some $d \geq d_t$). Note also that f is piecewise constant, changing only at distances equal to vertex distances. We break the real line into intervals according to f: the *type-1* intervals are the maximal intervals on which $f(d) = 1$; the *type-2* intervals are the maximal intervals on which $f(d) \geq 2$. These intervals partition the real line (out to the maximum distance reached by the optimum solution) and alternate between types 1 and 2. Let the interval boundaries be labeled $0 = b_1 < b_2 \cdots b_m$, where b_m is the maximum distance of any vertex on the path, so that the i^{th} interval is $[b_i, b_{i+1}]$. Note that each b_i is the distance label for some vertex. Let V_i be the set of vertices whose distance from s falls in the i^{th} interval. Note that the optimum path traverses each set V_i exactly once—once it leaves some V_i it does not return. For any pair of adjacent intervals, one must be of type-1; if the path left this interval and returned to it then $f(d)$ would exceed 1 within the interval. Thus, the vertices of P^* in set V_i form a contiguous *segment* of the optimum path that we label as $S_i = P^* \cap V_i$. A segment partition is shown in Figure 2-1.

Note that for each i, there may be (at most) 1 edge crossing from V_i to V_{i+1}. To simplify the next two lemmas, let us split that edge into two with a vertex at distance b_i from s, so that every edge is completely contained in one of the segments (this can be done since one endpoint of the edge has distance exceeding b_i and the other endpoint has distance less than b_i). Placing a vertex at each interval boundary ensures that the length of a segment is *equal to* the integral of $f(d)$ over its interval.

Lemma 2.1. *A segment S_i of type-1 has length at least $b_{i+1} - b_i$. A segment S_i of type-2 has length at least $3(b_{i+1} - b_i)$, unless it is the segment containing t in which case it has length at least $3(d_t - b_i)$.*

Proof. The length of segment S_i is lower bounded by the integral of $f(d)$ over the i^{th} interval. In a type-1 interval the result is immediate. For a type-2 interval, note that $f(d) \geq 1$ actually implies that $f(d) \geq 3$ by a parity argument—if the path crosses distance

d twice only, it must end up at distance less than d. □

Corollary 2.2. *The total length of type-2 segments is at most* $3\epsilon/2$.

Proof. Let ℓ_i denote the length of segment i. We know that the length of P^* is $d_t + \epsilon = \sum \ell_i$. At the same time, we can write

$$d_t \leq b_m = \sum_{i=1}^{m-1} (b_{i+1} - b_i) \leq \sum_{i \text{ type-1}} \ell_i + \sum_{i \text{ type-2}} \ell_i/3$$

It follows that

$$\epsilon = \sum \ell_i - d_t \geq \sum_{i \text{ type-2}} 2\ell_i/3$$

Multiplying both sides by 3/2 completes the proof. □

Having completed this analysis, we note that the corollary remains true even if we do not introduce extra vertices on edges crossing interval boundaries. The crossing edges are no longer counted as parts of segments, but this only decreases the total length of type-2 segments.

2.4.1 A Dynamic Program

Our algorithm computes, for each interval that might be an interval of the optimum solution, a segment corresponding to the optimum solution in that interval. It then uses a dynamic program to paste these fragments together using (and paying for) edges that cross between segments. The segments we compute are defined by 4 vertices: the closest-to-s and farthest-from-s vertices, c and f, in the interval (which define the start- and end-points of the interval: our computation is limited to vertices within that interval), and the first and last vertices, x and y, on the segment within that interval. They are also defined by the amount p of prize we are required to collect within the segment. There are therefore $O(\Pi n^4)$ distinct segment to compute, where Π is the total prize in the graph. For each segment we find an optimum solution for a type-1 and a type-2 interval. For a type-1 interval the optimum path is monotonic; we can therefore compute (in linear time) an optimum (shortest) monotonic path from x to y that collects prize p. If the interval is of type-2, the optimum path need not be monotonic. Instead, we use the min-cost path algorithm from Section 2.3 to approximate to within a constant factor the minimum length of a path that starts at x, finishes at y, stays within the boundaries of the interval defined by c and f, and collects prize at least p.

Given the optimum type-1 and near-optimum type-2 segment determined for each set of 4 vertices and prize value, we can find the optimal way to paste some subset of them together monotonically using a dynamic program. Note that the segments corresponding to the optimum path are considered in this dynamic program, so our solution will be at least as good as the one we get by using the segments corresponding to the ones on the optimum path (i.e., using the optimum type-1 segments and using the approximately optimum type-2 segments). We need only show that this solution is good.

We focus on the segments corresponding to the optimum path P^*. Consider the segments S_i of length ℓ_i on the optimum path. If S_i is of type-1, our algorithm will find a (monotonic) segment with the same endpoints collecting the same amount of prize of no greater length.

If S_i is of type-2, our algorithm (through its use of subroutine MCP) will find a path with the same endpoints collecting the same prize over length at most $\alpha_{CP}\ell_i$. Let L_1 denote the total length of the optimum type-1 segments, together with the lengths of the edges used to connect between segments. Let L_2 denote the total length of the optimum type-2 segments. Recall that $L_1 + L_2 = d_t + \epsilon$ and that (by Corollary 2.2) $L_2 \leq 3\epsilon/2$. By concatenating the optimum type-1 segments and the approximately optimum type-2 segments, the dynamic program can (and therefore will) find a path collecting the same total prize as P^* of total length

$$\begin{aligned} L_1 + \alpha_{CP} L_2 &= L_1 + L_2 + (\alpha_{CP} - 1)L_2 \\ &\leq d_t + \epsilon + (\alpha_{CP} - 1)(3\epsilon/2) \\ &= d_t + \left(\frac{3}{2}\alpha_{CP} - \frac{1}{2}\right)\epsilon. \end{aligned}$$

In other words, we approximate the minimum excess to within a factor of $\frac{3}{2}\alpha_{CP} - \frac{1}{2}$.

2.4.2 An Improved Approximation for Min-Excess

Our approximation guarantee for min-excess path α_{EP} is based on the approximation guarantee α_{CP} for the min-cost path problem. However, we use the algorithm for the latter problem as a "black-box" subroutine. In this section we show how to slightly improve our approximation guarantee for min-excess path problem by exploiting the details of the min-cost path algorithm derived from the work of Chaudhuri et. al [15].

Let us consider a segment of type-2 of an optimum min-excess path with endpoints u and v and length $\ell = d(u, v) + \epsilon$. If we apply the algorithm of Chaudhuri et. al with roots u and v, we get a tree containing u and v of total edge cost arbitrarily close to ℓ. Our min-cost path algorithm then converts this tree into a path from u to v by doubling all edges, except for the ones on the tree path from u to v. Noting that the total cost of "non-doubled" edges is at most $d(u, v) = \ell - \epsilon$, we get a path from u to v of length at most $(1 + \delta)\ell + \epsilon$, for any fixed small constant δ.

Next, applying Corollary 2.2 just to the path between u and v (e.g. taking u in place of s and v in place of t), we must have that $\ell \leq 3\epsilon/2$. Thus, the length of the segment of type-2 between u and v found in the process of running our dynamic program is at most

$$\ell + (1 + \frac{3}{2}\delta)\epsilon = d(u, v) + (2 + \frac{3}{2}\delta)\epsilon.$$

In other words, we can approximate excess of a wiggly segment of type-2 to within a factor arbitrarly close to 2. Now recall that our dynamic program finds optimal monotone segments of type 1. Thus, summing over all segments, we get an approximation ratio of $2 + \delta'$ for the min-excess path problem, for any small constant δ'.

2.5 Maximum Discounted-Prize Path

In this section we present an approximation algorithm for the Discounted-Reward TSP problem which builds upon our min-excess path algorithm. Recall that our goal is to find a path P that maximizes total discounted reward $\rho(P) = \sum \gamma^{d_v^P} \pi_v$. Assume without loss of generality that the discount factor is $\gamma = 1/2$—we simply rescale each length ℓ to ℓ' such

ALGORITHM FOR DISCOUNTED PC-TSP

1. Re-scale all edge lengths so that $\gamma = 1/2$.
2. Replace the prize value of each node with the prize discounted by the shortest path to that node: $\pi'_v = \gamma^{d_v}\pi_v$. Call this modified graph G'.
3. Guess t—the last node on optimal path P^* with excess less than ϵ.
4. Guess k—the value of $\Pi'(P_t^*)$.
5. Apply our min-excess path approximation algorithm to find a path P collecting scaled prize k with small excess.
6. Return this path as the solution.

Figure 2-2: Approximation algorithm for Maximum Discounted-Prize Path

that $\gamma^\ell = (\frac{1}{2})^{\ell'}$, i.e. $\ell' = \ell \log_2(1/\gamma)$.

We first establish a property of an optimal solution that we make use of in our algorithm. Define the *scaled prize* π' of a node v to be the (discounted) reward that a path gets at node v if it follows a shortest path from the root to v. That is, $\pi'_v = \pi_v \gamma^{d_v}$. Let $\Pi'(P) = \sum_{v \in P} \pi'_v$. Note that for any path P, the discounted reward obtained by P is at most $\Pi'(P)$.

Now consider an optimal solution P^*. Fix a parameter ϵ that we will set later. Let t be the last node on the path P^* for which $d_t^{P^*} - d_t \leq \epsilon$—i.e., the excess of path P^* at t is at most ϵ. Consider the portion of P^* from root s to t. Call this path P_t^*.

Lemma 2.3. *Let P_t^* be the part of P^* from s to t. Then, $\rho(P_t^*) \geq \rho(P^*)(1 - \frac{1}{2^\epsilon})$.*

Proof. Assume otherwise. Suppose we shortcut P^* by taking a shortest path from s to the next node visited by P^* after t. This new path collects (discounted) rewards from the vertices of $P^* - P_t^*$, which form at least $\frac{1}{2^\epsilon}$ of the total by assumption. The shortcutting procedure decreases the distance on each of these vertices by at least ϵ, meaning these rewards are "undiscounted" by a factor of at least 2^ϵ over what they would be in path P^*. Thus, the total reward on this path exceeds the optimum, a contradiction. □

It follows that we can approximate $\rho(P^*)$ by approximating $\rho(P_t^*)$. Based on the above observation, we give the algorithm of Figure 2-2 for finding an approximately optimal solution. "Guess t" is implemented by going through all possible vertices, and "guess k" is done by performing a binary search in the range from 0 to the total (rescaled) prize value of the graph.

Our analysis below proceeds in terms of $\alpha = \alpha_{EP}$, the approximation factor for our min-excess path algorithm.

Lemma 2.4. *Our approximation algorithm finds a path P that collects discounted reward $\rho(P) \geq \Pi'(P)/2^{\alpha\epsilon}$.*

Proof. The prefix P_t^* of the optimum path shows that it is possible to collect scaled prize $k = \Pi'(P_t^*)$ on a path with excess ϵ. Thus, our approximation algorithm finds a path collecting the same scaled prize with excess at most $\alpha\epsilon$. In particular, the excess of any

vertex v in P is at most $\alpha\epsilon$. Thus, the discounted reward collected at v is at least

$$\rho(v) \geq \pi_v \left(\frac{1}{2}\right)^{d_v+\alpha\epsilon} = \pi_v \left(\frac{1}{2}\right)^{d_v} \left(\frac{1}{2}\right)^{\alpha\epsilon} = \pi'_v \left(\frac{1}{2}\right)^{\alpha\epsilon}$$

Summing over all $v \in P$ completes the proof. □

Combining Lemma 2.4 and Lemma 2.3, we get the following:

Theorem 2.5. *The solution returned by the above algorithm has* $\rho(P) \geq (1-\frac{1}{2^\epsilon})\rho(P^*)/2^{\alpha\epsilon}$.

Proof.

$$\begin{aligned}
\rho(P) &\geq \Pi'(P)/2^{\alpha\epsilon} && \text{by Lemma 2.4} \\
&\geq \Pi'(P_t^*)/2^{\alpha\epsilon} && \text{by choice of } P \\
&\geq \rho(P_t^*)/2^{\alpha\epsilon} && \text{by definition of } \pi' \\
&\geq \left(1-\frac{1}{2^\epsilon}\right)\rho(P^*)/2^{\alpha\epsilon} && \text{by Lemma 2.3}
\end{aligned}$$

□

We can now set ϵ as we like. Writing $x = 2^{-\epsilon}$ we optimize our approximation factor by maximizing $(1-x)x^\alpha$ to deduce $x = \alpha/(\alpha+1)$. Plugging in this x yields an approximation ratio of $(1+\alpha_{EP})(1+1/\alpha_{EP})^{\alpha_{EP}}$.

2.6 Orienteering

In this section we present an algorithm for computing an approximately maximum-prize path of length at most D that starts at a specified vertex s. We solve this problem by invoking the algorithm for min-excess path given in section 2.4 as a subroutine. Our algorithm for the max-Prize problem is given in Figure 2-3. As before "guess t" is implemented by exhausting all polynomially many possibilities, and "guess k" is done by performing a binary search. We analyze this algorithm by showing that any optimum orienteering solution contains a low-excess path which, in turn, is an approximately optimum orienteering solution.

More precisely, we prove that for some vertex v, there exists a path from s to v with excess at most $\frac{D-d_v}{\alpha_{EP}}$ that collects prize at least $\frac{\Pi^*}{\alpha_{PP}}$ where α_{EP} is the approximation ratio for min-excess path, α_{PP} is the desired approximation ratio for Max-Prize Path, and Π^* is the prize of the optimum Max-Prize Path. Assuming this path exists, our min-excess path computation on this vertex v will find a path with total length at most $d_v + \alpha_{EP}\frac{D-d_v}{\alpha_{EP}} = D$ and prize at least $\frac{\Pi^*}{\alpha_{PP}}$, providing an α_{PP}-approximation for orienteering.

Let t be the vertex on the optimum orienteering path that has maximum distance from s. We first consider the case where the optimum orienteering path ends at t (as opposed to continuing past t back towards s).

Lemma 2.6. *Suppose there exists a path P from s to t of length at most D that collects prize Π, such that t is the furthest point from s along this path. Then for any integer $r \geq 1$ there exists a path from s to some node v with excess at most $\frac{D-d_v}{r}$ and prize at least $\frac{\Pi}{r}$.*

ALGORITHM FOR THE ORIENTEERING PROBLEM

1. Guess k, the amount of prize collected by an optimum orienteering solution.
2. For each vertex v, compute a min-excess path from s to v
3. Find a vertex v such that the min-cxcess path from s to v obtained in previous step has length at most D.
4. Return the corresponding path from s to v.

Figure 2-3: Approximation algorithm for Max-Prize Path

Proof. For each point u along the original path P, let $\epsilon_u = d_u^P - d_u$; in other words, ϵ_u is the excess in the length of the path to u over the shortest-path distance. Since the total length of path P is at most D, we must have $\epsilon_t \leq D - d_t$.

Consider mapping the points on the path to a line from 0 to ϵ_t according to their excess (note that excess only increases as we traverse path P). Divide this line into r intervals with length $\frac{\epsilon_t}{r}$. Some such interval must contain at least $\frac{\Pi}{r}$ prize, since otherwise the entire interval from 0 to ϵ_t would not be able to collect prize Π. Suppose such an interval starts at node u and ends at node v. We consider a path from s to v that takes the shortest s-u path, then follows path P from u to v. This new path collects all of the prize in the interval from u to v of the original path, which is at least $\frac{\Pi}{r}$ as desired. The total length of this new path is

$$d_u + d^P(u,v) = d_u + d_v^P - d_u^P = d_v + \epsilon_v - \epsilon_u \leq d_v + \frac{\epsilon_t}{r}.$$

Thus, this path has excess at most $\frac{\epsilon_t}{r} = \frac{D-d_t}{r} \leq \frac{D-d_v}{r}$ as desired. □

Of course, in general the optimum orienteering path might have some intermediate node that is farther from s than the terminal node t. We will generalize the above lemma to account for this case.

Lemma 2.7. *If there exists a path from s to t of length at most D that collects prize Π, then for any integer $r \geq 1$ there exists a path from s to some node v with excess at most $\frac{D-d_v}{r}$ that spans at least $\frac{\Pi}{r+1}$ prize.*

Proof. Let f be the furthest point from s along the given path P. We are interested in the case where $f \neq t$. We can break path P into two pieces: a path segment from s to f and then a segment from f to t. Using the symmetry of our metric, we can produce a second path from s to f by using the shortest path from s to t and then following the portion of our original path from f to t in reverse. We now have two paths from s to f, each of which has length at most D. The total length of these paths is bounded by $D + d_t$. We will call our paths A and B, and let their lengths be $d_f + \epsilon_A$ and $d_f + \epsilon_B$ respectively.

Next we map path A to the interval from 0 to ϵ_A according to the excess at each point, much as in Lemma 2.6. We then divide this interval into pieces of length $\frac{\epsilon_A+\epsilon_B}{r}$ (the last sub-interval may have shorter length if ϵ_A does not divide evenly). We perform the same process on path B. This creates a total of $r+1$ sub-intervals, since the total length of the two intervals combined is $\epsilon_A + \epsilon_B$ and all sub-intervals save for two are of length $\frac{\epsilon_A+\epsilon_B}{r}$.

We conclude that some such sub-interval contains at least a $\frac{1}{r+1}$ fraction of the total prize collected by path P. Suppose without loss of generality that this interval spans a

portion of path A from u to v. As in Lemma 2.6 we can shortcut A by taking the shortest path from s to u and then follow path A from u to v . The length of this new path is bounded by $d_v + \frac{\epsilon_A + \epsilon_B}{r}$ resulting in an excess of at most $\frac{D-d_f}{r} \leq \frac{D-d_v}{r}$ as desired. □

Thus, combining our approximation guarantee for the min-excess path problem from section 2.4 together with the result of Lemma 2.7, we can show that the algorithm given in Figure 2-3 finds a solution for the orienteering problem within a factor of 4 from optimal.

Theorem 2.8. *Our algorithm is an* $(\lceil \alpha_{EP} \rceil + 1)$*-approximation for the max-prize path (orienteering) problem, where* α_{EP} *is the approximation factor for min-excess path.*

Proof. Lemma 2.7 implies that there exists a path from s to some v with excess $\frac{D-d_v}{\alpha_{EP}}$ that spans $\frac{1}{\lceil \alpha_{EP} \rceil + 1}$ fraction of the optimum prize Π^*. Such a path has length $d_v + \frac{D-d_v}{\alpha_{EP}}$, implying that our approximation algorithm for min-excess will find a path from s to v of length at most $d_v + (D - d_v) = D$ and at least the same prize. □

2.7 Extensions

2.7.1 Budget Prize Collecting Steiner Tree

In this section, we consider the tree variant of the Orienteering problem, called Max-Prize Tree in our notation. Namely, given a graph G with root r, prize function π and lengths d, we are required to output a tree T rooted at r with $d(T) \leq D$ and maximum possible reward $\Pi(T)$. This problem is also called the Budget Prize-Collecting Steiner Tree problem [32]. Although the unrooted version of the problem can be approximated to within a factor of $5 + \epsilon$ via a 3-approximation for k-MST [32, 39], the version of the problem in which a tree is required to contain a specified vertex has remained open until recently.

Let the optimal solution for this problem be a tree T^*. Double the edges of this tree to obtain an Euler tour of length at most $2D$. Now, divide this tour into two paths, each starting from the root r and having length at most D. Among them, let P' be the path that has greater reward. Now consider the Max-Prize Path problem on the same graph with distance limit D. Clearly the optimal solution P^* to this problem has $\Pi(P^*) \geq \Pi(P') \geq \frac{\Pi(T^*)}{2}$. Thus, we can use the α_{PP}-approximation for Orienteering to get a $2\alpha_{PP}$-approximation to T^*.

2.7.2 Multiple-Path Orienteering and Discounted-Reward TSP

In this section we consider a variant of the Orienteering and Discounted-Reward TSP in which we are allowed to construct up to k paths. For the Orienteering problem, each path must have length at most D. For the Discounted-Reward problem, the k robots move simultaneously, so the discounting is done based on the individual lengths of the paths.

For both problems, we apply the single-path algorithms described in sections 2.5 and 2.6 respectively to successively construct the k paths. At the i-th step, we set the prizes of all nodes visited by the first $i - 1$ paths to 0, and construct the i-th path on the new graph, using the previously described algorithms. Using a set-cover like argument, we get the following approximation guarantees.

Theorem 2.9. *If all the paths are required to have a common start node, the above algorithm gives a* $(1 - e^{-1/\alpha_{PP}})^{-1}$ *(*$(1 - e^{-1/\alpha_{DP}})^{-1}$*) approximation for the Multiple-Path Orienteering (resp. Discounted-Reward TSP).*

If the paths are allowed to have different start nodes, the above algorithm gives a $\alpha_{PP}+1$ approximation for the Multiple-Path Orienteering.

Proof. First let us consider the variant of the Multiple-Path Orienteering problem when all the paths must have a common source. Consider the prize collected by the optimal solution, but not collected by our solution by stage i. At least one of the paths in the optimal solution spans at least a $\frac{1}{k}$ fraction of this prize. Then, using the approximation guarantee of the algorithm for orienteering, our solution collects at least a $\frac{1}{k\alpha_{PP}}$ fraction of this prize. Hence, by the end of k rounds, the total prize collected by the optimal solution, but not collected by us, is at most $(1-\frac{1}{k\alpha_{PP}})^k \leq e^{-\alpha_{PP}}$, and the result follows.

To prove this result for the Discounted-Reward TSP, it suffices to literally substitute "discounted reward" for "prize" and α_{DP} for α_{PP}.

Next consider the case when the paths have different sources. Let O_i be the set of nodes visited by the i-th path in the optimal solution, and let A_i be the corresponding set of nodes visited by our algorithm. Let Δ_i be the set of nodes that are visited by the i-th path in the optimal solution and some other path in our solution. Let $O = \cup_i O_i$ and $A = \cup_i A_i$ denote the set of all nodes visited by the paths of the optimal and respectively our solution, and set $\Delta = \cup_i \Delta_i$ to be the set of nodes spanned by both solutions. Note that $\Pi(O)$ then denotes the prize collected by the optimum solution, whereas $\Pi(A)$ gives the total prize value obtained by our algorithm.

At the beginning of the i-th step of our algorithm, the i-th path of the optimal solution would collect prizes from all nodes it visits except for those that have been already covered by one of the previous $i-1$ paths constructed by our solution; at the very least it gets the prize from all the nodes in $O_i \setminus \Delta_i$. Hence, there exists a path that obtains at least $\Pi(O_i) - \Pi(\Delta_i)$ prize. Since our algorithm finds a path that spans at least a $1/\alpha_{PP}$ fraction of the maximum possible prize available at step i, we must have $\Pi(A_i) \geq \frac{1}{\alpha_{PP}}(\Pi(O_i) - \Pi(\Delta_i))$. Summing over all steps, we get $\alpha_{PP}\Pi(A) \geq (\Pi(O) - \Pi(\Delta))$. Noting that the total prize value of the nodes visited by both the optimal solution and our algorithm can be at most the total prize value obtained by our algorithm, i.e. $\Pi(\Delta) \leq \Pi(A)$, we get $\Pi(A) \geq \frac{1}{\alpha_{PP}+1}\Pi(O)$.

□

2.8 Open Problems

In this chapter we give constant factor algorithms for the Orienteering problem, Discounted-Reward TSP, and some of their variants. An interesting open problem is to consider non-uniform discount factors, e.g. each vertex would get its own γ_v. Similarly we could look at the version of the Orienteering (max-prize) problem in which each vertex has its own deadline. Thus, the reward collected at vertex v at time t could be given by $\rho_v = \pi_v$ if $t < D_v$ and 0 otherwise. We can also introduce "release dates" for rewards, so that one collects the reward at vertex v only after time R_v (and before the deadline D_v).

Note that having an approximation algorithm for the max-prize problem with arbitrary deadlines and release dates would mean that we could approximately solve Discounted-Reward TSP for any type of a discount function. This could be accomplished by discretizing time into polynomially many small intervals and attaching to each node v (with a positive prize value) by a zero cost edge a leaf node v_i ($i = 1, \ldots$) for each time interval $(t_{i-1}, t_i]$. The original node v gets undiscounted prize value π_v with the deadline set to 0 and a negative release date. Each artificial node v_i is assigned a prize equal to the discounted reward available at v at time t_i, with the release date and the deadline for collecting that prize set

to t_{i-1} and t_i respectively. Assuming that each time interval is sufficiently small (and the discount function is reasonably smooth), the discounted reward value doesn't drop off too rapidly inside each interval. Thus the prize value collected by a given path in the modified graph provides a good approximation on the discounted reward value of the corresponding path in the original graph.

Another interesting open problem is to consider the directed versions of the problems, although we believe that it may be hard to approximate these to within constant or even logarithmic factors. Note though that the only portion of our algorithms which relies on the symmetry of arc costs (i.e. undirectedness) is the subroutine for the min-cost s-t path problem. Recall that the latter subroutine obtains a path by taking a tour of a min-cost tree. The approximation guarantee on the length of such path wouldn't hold in the directed case. However, if we could construct a *directed* s-t path of small length that collects a given prize value, our algorithms would produce provably good solutions for the directed versions of the problems.

Even more ambitiously, returning to the Markov Decision Process motivation for this work, one would like to generalize these results to probabilistic transition functions. However, this has the additional complication that the optimum solution may not even have a short description (it is no longer just a path). Still, perhaps some sort of non-trivial approximation bound, or a result holding in important special cases, can be found.

Chapter 3

Preplanning framework for stochastic combinatorial optimization problems

So far we have considered a scenario of planning under uncertainty in which the decision has to be taken in advance of any information about unknown inputs (other than their probability distribution). This advance planning is necessary because information becomes available too late to be useful for optimizing one's reward. However, there are a number of other scenarios in which it is possible to respond to information that becomes available after the initial planning stage. In this chapter we study the tradeoffs involved in making some purchase/allocation decisions early to reduce cost while deferring others at greater expense to take advantage of additional information. We consider a number of combinatorial optimization problems (primarily in network design) in which the input is uncertain—modeled by a probability distribution—and in which elements can be purchased cheaply now or at greater expense after the distribution is sampled. We show how to approximately optimize the choice of what to purchase in advance and what to defer.

3.1 Introduction

Combinatorial optimization is often used to "plan ahead," purchasing and allocating resources for needs that are not precisely known at the time of solution. For example, a network designer might have to make their best guess about the future demands in a network and purchase capacity accordingly. Moreover, there are many scenarios, such as the robot navigation problem considered in the previous chapter, in which unknown information becomes available too late to be useful – it might be impossible to modify a solution or improve its value once the actual inputs are revealed.

At other times, however, there is a possibility to "wait and see," deferring decisions about resource allocation until demands or constraints become clear. This allows one to plan an optimal solution for each potential outcome. There is often a tradeoff involved, in that allocations made later may be more expensive. For example, the network designer may be able to arrange cheap long-term contracts for capacity purchased ahead of time, but may need to purchase extra capacity at the last minute on a more expensive "spot market."

Beyond the basic optimization problem, then, there is the problem of deciding which part of the solution should be set early and cheaply based on limited information about the

problem input, and which part should be deferred and solved more expensively with the arrival of additional information about the input.

In this chapter, we study a particular framework derived from stochastic programming, for dealing with this time-information tradeoff in the presence of uncertainty. Formally, we postulate a probability distribution $\Pr[\mathcal{I}]$ over problem instances $\mathcal{I}$. We consider a collection of variables x_i and y_j that describe candidate solutions to the problem, where different settings of the variables are feasible for different inputs $\mathcal{I}$. We are required to set the variables x_i, then sample a problem instance $\mathcal{I}$ from the distribution, and finally, with knowledge of the instance, to set the variables y_j so that (x, y) is feasible for $\mathcal{I}$. Given a cost function $c(x, y)$ on solutions, our goal is to minimize the expected cost $E[c(x, y)]$ subject to the feasibility constraints.

In the standard two-stage stochastic programming with recourse model [10], the problem instances are polytopes over x_i and y_j (representing linear or integer linear programs), and the cost function is linear. Traditionally, the second-stage variables y_j are interpreted as a recourse action against any bad effects such as solution infeasibility due to a particular realization of uncertainty [48]. Under a discrete probability distribution over random instances $\{\mathcal{I}\}$, the problem can be formulated as large scale linear (integer) program and solved using standard methods from mathematical programming.

We are interested in exploring this framework in the context of problems with more combinatorial structure, so that we can take advantage of additional techniques that exploit this structure. Specifically, we study stochastic versions of the following problems: min-cost flow, bin packing, vertex cover, shortest path, and the Steiner tree problem. In each, a ground set of elements $e \in E$ is specified (vertices in the vertex cover problem, edges in the Steiner problems, bins in bin packing). A (randomly selected) problem instance $\mathcal{I}$ defines a set of feasible solutions, each corresponding to a subset $\mathcal{F}_{\mathcal{I}} \subseteq 2^E$. We can buy certain elements "in advance" at cost c_e, then sample a problem instance, and must then buy other elements at "last-minute" costs λc_e so as to produce a feasible set for our problem instance. In this case, variables x_e are set to 1 for the set of elements purchased in advance, and variables y_e are subsequently set to 1 for those elements added after sampling the problem instance. Our goal is to minimize the expected total cost.

It is noteworthy that all of our problems are "monotone," in that any superset of a feasible solution is feasible. This is convenient because it means that purchasing elements in advance never "invalidates" a potentially feasible solution—the advance purchases may be wasted, but at worst they can be ignored and any desired feasible solution constructed from the post-sampling purchases. Thus, we can focus all of our attention on optimizing cost without worrying about making feasibility-missteps.

3.1.1 Our Contribution

We examine a number of (generally $\mathcal{NP}$-hard) combinatorial optimization problems in which it makes sense postulate some probability distribution over input instances and to specify a portion of the solution in advance. We develop algorithms to find approximately optimal pre- and post-sampling parts of a feasible solution. In particular, we study the following problems:

Min Cost Flow. Given a source and sink and a probability distribution on demand, buy some edges in advance and some after sampling (at greater cost) such that the given amount of demand can be routed from source to sink. We show that this problem can be solved via linear programming for a discrete demand distribution.

Bin packing. A collection of items is given, each of which will need to be packed into a bin with some probability. Bins can be purchased in advance at cost 1; after the determination of which items need to be packed, additional bins can be purchased at cost $\lambda > 1$. How many bins should be purchased in advance to minimize the expected total cost? We show that this problem can be efficiently approximated arbitrarily close to optimal.

Vertex Cover. A graph is given, along with a probability distribution over sets of edges that may need to be covered. Vertices can be purchased in advance at cost 1; after determination of which edges need to be covered, additional vertices can be purchased at cost λ. Which vertices should be purchased in advance? We give a 4-approximation for the case where the probability distribution involves only polynomially many distinct edge sets and a different 4-approximation algorithm for the case when each edge has to be covered with fixed probability p.

Cheap Path. We are given a graph and told that a randomly selected pair of vertices (or one fixed vertex and one random vertex) will need to be connected by a path. We can purchase edge e at cost c_e before the pair is known or at cost λc_e after and wish to minimize the expected total edge cost. We show that this problem is equivalent to the multicommodity rent-or-buy problem, so that corresponding approximation algorithms apply.

Steiner Tree. A graph is given, along with a probability distribution over sets of terminals that need to be connected by a Steiner tree. Edge e can be purchased at cost c_e in advance or at cost λc_e after the set of terminals is known. We give a constant factor approximation for the case where the expected number of terminals is constant (generalizing the Cheap Path result). We give a constant factor approximation for the case where edges form an ultrametric and an $O(\log n)$ approximation for general edge costs.

3.1.2 Related work

Stochastic programming is a tremendous field with a vast literature [55]. It is applicable whenever probability distributions of inputs are known or can be estimated. One of the most widely used models in stochastic programming is the two-stage recourse model mentioned earlier. It involves an initial deterministic decision, an opportunity for observing additional information, and then a recourse action in response to each random outcome. The two-stage model can be naturally generalizedby adding additional recourse stages, each consisting of an observation and a decision responding to it. Stochastic linear programs are generally tackled via a combination of mathematical programming and advanced probabilistic techniques. A key difficulty in solving these problems is dealing with a very large uncertainty space, as one gets a separate set of constraints for each potential outcome of the future. Some standard large-scale optimization techniques utilized include decomposition methods (in which smaller subproblems involving only a subset of constraints are repeatedly solved), and various Lagrangian relaxation approaches. In addition, a number of computational methods have been developed that use specific stochastic program structure; they include extreme-point and interior-point methods employing sparse factorization, column splitting, and others. [10].

The stochastic multicommodity flow problem in which various attributes of the network such as arc costs, arc capacities, and demands between pairs of terminals, are random variables has been studied particularly extensively (for a survey of various problems and formulations see [51]). A variant of the problem in which demands and capacities are fixed, and only per-unit flow arc costs are stochastic (but independent of particular flow values) can be modeled as a linear program with deterministic arc costs set to the corresponding expected values [51]. A probabilistic arc capacity is usually modeled by a stochastic arc cost that depends on the corresponding flow value, becoming infinite when the capacity is exceeded. The latter approach results in a non-linear (but still) convex objective function, so standard convex programming methods can be applied. The most difficult version of the problem seems to arise when one considers probabilistic demands. Although the problem can be solved for some special cases of probability distribution functions (such as independent Poisson PDF), we are not aware of any general-purpose methods for the case of more than one commodity.

Nevertheless, the stochastic multicommodity flow problem has been used extensively as a model for a variety of real-world applications. Recently, Mitra and Wang [40] have derived a flow-based framework for stochastic traffic engineering in which the objective is to maximize revenue from serving demands which are specified by probability distributions. Their model is somewhat similar to ours, in that it uses a two-tier cost function, and explores the tradeoff between deterministic allocations and probabilistic provisioning. They present conditions under which the problem can be reduced to an instance of convex programming.

A number of other classical discrete optimization problems have been considered in the stochastic setting. Andreatta and Romeo have studied a version of the shortest path problem, in which a subset of the edges of a network may fail with some probability [2]. The goal is to choose a path before knowing the state of the network so as to minimize the expected length of the resulting traversal. Andreatta and Romeo consider the problem in the two-stage recourse model, so that the planner has to pick in addition to the "preferred" path, an alternative path (detour) to follow from each node in case the outgoing edge of the original path fails. The authors assume that there always exists such a detour from every node. For the case when one is given a polynomial number of possible sets of failed edges, the authors show how to solve the problem using stochastic dynamic programming techniques.

A rather different stochastic optimization framework assumes random instances but requires only that constraints be satisfied *with certain probability.* This framework is sometimes known as "chance constrained programs." For example, in [34] Kleinberg, Rabani and Tardos consider stochastic knapsack, bin-packing and load-balancing problems. In the stochastic knapsack variant, one is concerned with choosing from among a set of items, each with a value and a probability distribution on its size, a subset of maximum value that is *unlikely* to exceed a given capacity threshold. Similarly, in the stochastic bin-packing, given a probability distribution on the sizes of items, the goal is to determine the minimum number of bins so that the probability of any one overflowing is small. The objective of stochastic load-balancing is to minimize the expected maximum weight in any bin, given a fixed number of bins. Kleinberg et.al provide a constant-factor approximation for the latter problem, and somewhat weaker guarantees that are function of $\log p^{-1}$ (where p is the probability with which constraints are allowed to be violated) for the stochastic knapsack and bin-packing.

3.2 Preliminaries

In this section we give a formal definition of the preplanning framework for stochastic combinatorial optimization problems, discuss a number of basic properties shared by the problems in this framework, and present an overview of our approaches and the results we obtain.

3.2.1 Formal Problem Statement

Formally, a preplanning version of a combinatorial optimization problem is specified in our framework by a ground set of elements $e \in E$, a probability distribution on instances $\{I\}$ a cost function $c : E \to \mathbb{R}$, and a penalty factor $\lambda \geq 1$. Each instance $\mathcal{I}$ has a corresponding set of feasible solutions $\mathcal{F}_{\mathcal{I}} \subseteq 2^E$ associated with it. Suppose a set of elements $A \subseteq E$ is purchased before sampling the probability distribution. Let c_A denote the *posterior cost function*, i.e.

$$c_A(e) = \begin{cases} 0 & \text{if } e \in A \\ \lambda c(e) & \text{otherwise} \end{cases}$$

The objective of a preplanning combinatorial optimization problem is to choose a subset of elements A to be purchased in advance so as to minimize the total expected cost of a feasible solution

$$c(A) + \mathrm{E}\left[\min_{S \in \mathcal{F}_{\mathcal{I}}} c_A(S)\right]$$

over a random choice of an instance $\mathcal{I}$.

In solving a preplanning problem, one option is to postpone purchasing any elements until after sampling a problem instance. This approach readily yields a λ-approximation as it allows us to buy an "optimal" solution for a given problem instance (assuming have access to an exact algorithm or oracle; using a c-approximation algorithm yields λc-approximation). However a large value of λ might make such approach prohibitively expensive. At the other extreme, one can buy in advance a minimum-cost solution that is feasible for all possible problem instances (for example, the set of all elements). However, such a solution might be far more expensive than an optimal solution for any given instance from the distribution. We are interested in studying how much preplanning one should do given only a probability distribution on the inputs.

3.2.2 The Threshold Property

Our first observation is that optimal preplanning solutions exhibit a natural local-optimality property.

Consider a solution that purchases some set of elements $A \subseteq E$ in advance and then, on sampling problem instance $\mathcal{I}$, buys additional elements $L_{\mathcal{I}}$. Note that $A \cup L_{\mathcal{I}}$ is a feasible solution for the instance $\mathcal{I}$. This suggests the following way of looking at the preplanning framework: we have to specify (in advance) for each instance $\mathcal{I}$ a complete feasible solution $F_{\mathcal{I}} \subseteq E$ to be used in case $\mathcal{I}$ is sampled. The objective is to minimize the expected cost of elements used in a solution for a randomly chosen instance $\mathcal{I}$. Given the set of solutions $\{F_{\mathcal{I}}\}$, we can obtain a probability that an element $e \in E$ is used in a solution for a random instance $\mathcal{I}$: $\Pr[e \text{ used}] = \sum_{\mathcal{I}: e \in F_{\mathcal{I}}} \Pr[\mathcal{I}]$. The expected cost to the objective function

incurred by element e is $\lambda c_e \Pr[e \text{ used}]$ if it is bought in the future, or c_e if it is bought ahead of time. The objective is minimized when this cost is equal to the minimum of the two, i.e. if $\Pr[e \text{ used}] \leq 1/\lambda$, then we should pay $\lambda c_e \Pr[e \text{ used}]$ for the element e, and c_e otherwise. This immediately implies the following *Threshold Property*:

Theorem 3.1. *An element should be purchased in advance if and only if the probability it is used in a solution for a randomly chosen instance exceeds* $1/\lambda$.

We note that if $\Pr[e \text{ used}] = 1/\lambda$, then it doesn't really matter whether e is purchased in advance, or added to a solution later on if needed. In either case, its expected contribution to the objective function is 1.

3.2.3 Using Approximation Algorithms as Subroutines

The construction of a solution for a preplanning problem proceeds in two stages in our framework. In stage 1, one picks a set A of elements to be purchased in advance. In stage 2, after sampling the problem instance, one needs to find a solution satisfying the set of constraints of that instance while minimizing the cost of additional elements to be purchased. Note that stage 2 reduces to solving the regular version of the optimization problem with the cost function c_A.

Most of the problems we are interested in putting into the preplanning framework are $\mathcal{NP}$-hard to start from. This immediately implies that the preplanning versions of these problems are also $\mathcal{NP}$-hard. Additionally, it complicates our task since we are unable to find an optimal solution in stage 2. Instead we have to resort to finding an approximately optimal solution. But this could lead to us picking a sub-optimal set of elements to buy in advance. Fortunately, as we illustrate below, an algorithm that produces an α-approximation for the original combinatorial optimization problem could be used in place of an optimal algorithm both in planning an advance purchase set and in stage 2, with loss of a factor of α in the quality of the solution.

Consider an optimization problem with cost function c. Let $c^{\mathsf{OPT}}(\mathcal{I})$ denote the value of a min-cost feasible solution for an instance $\mathcal{I}$. Suppose we are also given an approximation algorithm ALG that for any instance $\mathcal{I}$ finds a feasible solution of cost at most $c^{\mathsf{ALG}}(\mathcal{I}) \leq \alpha c^{\mathsf{OPT}}(\mathcal{I})$.

Theorem 3.2. *Given an α-approximation algorithm ALG, let A_0 be a subset of elements that minimizes the expected cost of a solution obtained with ALG over a random choice of a instance $\mathcal{I}$, i.e. $c(A) + \mathrm{E}\left[c_A^{\mathsf{ALG}}(\mathcal{I})\right]$. Then the cost of a preplanning solution that purchases elements in A_0 in advance is at most α times the minimum possible cost whether one uses an exact or an approximation algorithm in stage 2.*

Proof. Let us consider an optimal solution to the preplanning problem. Let A^* denote a subset of elements that this optimal solution purchases in advance, and let OPT denote the expected cost of an optimal preplanning solution. Since for any instance $\mathcal{I}$ we have $c_{A^*}^{\mathsf{ALG}}(\mathcal{I}) \leq \alpha c_{A^*}^{\mathsf{OPT}}(\mathcal{I})$, we get

$$\begin{aligned} c(A^*) + \mathrm{E}\left[c_{A^*}^{\mathsf{ALG}}(\mathcal{I})\right] &\leq c(A^*) + \alpha \mathrm{E}\left[c_{A^*}^{\mathsf{OPT}}(\mathcal{I})\right] \\ &\leq \alpha\left(c(A^*) + \mathrm{E}\left[c_{A^*}^{\mathsf{OPT}}(\mathcal{I})\right]\right) \\ &= \alpha \cdot \mathsf{OPT}, \end{aligned}$$

e.g. the expected cost of a solution constructed by ALG using set A^* costs at most αOPT. At the same time it must be at least as large as the expected cost of a solution constructed by ALG using set A_0, e.g.

$$c(A_0) + \mathrm{E}\left[c_{A_0}^{\mathsf{ALG}}(\mathcal{I})\right] \leq c(A^*) + \mathrm{E}\left[c_{A^*}^{\mathsf{ALG}}(\mathcal{I})\right].$$

Thus, we have

$$c(A_0) + \mathrm{E}\left[c_{A_0}^{\mathsf{OPT}}(\mathcal{I})\right] \leq c(A_0) + \mathrm{E}\left[c_{A_0}^{\mathsf{ALG}}(\mathcal{I})\right] \leq \alpha \cdot \mathsf{OPT}$$

Note that we have shown that the expected cost of a solution that buys A_0 in advance and then uses *either* an optimum solution or an α-approximation for a random instance is within a factor of α from the optimum solution. □

The above theorem implies that in some instances we can reduce the preplanning version of an $\mathcal{NP}$-hard problem to solving a preplanning instance of another optimization problem that has a polynomial-time algorithm. For example, since in a metric space an MST over a subset of nodes S provides a 2-approximation for min-cost Steiner tree on S, a preplanning version of a Steiner tree problem can be reduced to a preplanning version of a minimum spanning tree (MST) problem by adding an edge between each pair of terminals of cost equal to the shortest path distance in the original graph (thus creating a metric).

3.2.4 Overview of Results

We begin our study of stochastic combinatorial optimization problems in the preplanning framework by illustrating the use of the standard stochastic programming technique of merging together linear programs for individual instances.This solution technique applies when the probability distribution over problem instances has bounded support, that is only a limited number of instances can arise. In the case when each individual instance can be solved via a linear program, it is straightforward to create a "combined" linear program that represents the preplanning version of the problem; one simply shares the prepurchase variables among the different linear programs and adds the objective function values of them all, weighed by their probability of occurrence.

The min-cost flow problem with discrete stochastic demand introduced in Section 3.3 is a good candidate for the application of this technique, since the number instances is restricted to the number of possible demand values (which are part of the input), and the deterministic version of the problem has a linear programming formulation of polynomial size. However the problem is still interesting in that it highlights some challenging issues in preplanning and stochastic combinatorial optimization. For example, whereas in the case of the regular min-cost flow problem with integral capacities there always exists an integral optimal solution (and one can use combinatorial algorithms to find it), it is not immediately clear whether the same is true of the preplanning version of min-cost flow.

Next we proceed to analyze in Section 3.4 the bin packing problem with preplanning. Using a fully polynomial approximation scheme (FPAS) for the regular bin-packing problem, we obtain an algorithm that efficiently computes a solution for the preplanning version of the problem that is arbitrarily close to optimal. It appears that the bin-packing problem with preplanning is easier than other problems we consider. This can be explained by the fact that the elements of the ground set in this problem (i.e. bins) are indistinguishable,

resulting in a polynomially bounded space of possible solutions (i.e. the number of bins to buy in advance). Nonetheless, this problem is interesting and educational as it allows us to demonstrate the application of the basic properties derived above.

In Section 3.5 we consider two versions of the vertex cover problem with preplanning. In the first version, the number of instances, that is the number of possible subsets of edges that need to be covered, is polynomially bounded. We extend the "merger of linear programs" technique (used to solve min-cost flow with preplanning) to this problem to obtain a 4-approximation. In the second version of the vertex cover problem with preplanning, each edge needs to be covered independently with some probability, so the number of instances is exponential in the number of edges. We exploit combinatorial properties of the problem and apply the Threshold Property to obtain an approximation algorithm with a guarantee of 4.

Finally, in Section 3.6 we move on to the preplanning version of the Steiner tree problem, perhaps the most difficult of all the problems we study here. As a special (easier) case we consider the cheap path problem, in which we are restricted to random sets of terminals of cardinality 2. Although this problem can be also thought of as a stochastic version of the shortest path problem (which can be solved in polynomial time), we show that the cheap path problem in the preplanning framework is equivalent to a known $\mathcal{NP}$-hard network problem that has several approximation algorithms with constant factor guarantees. Generalizing the cheap path result, we give a constant factor approximation for the case where the expected number of terminals is constant. In the general case with an arbitrary number of random terminals, we reduce the preplanning Steiner tree problem to an instance of the MST preplanning problem. We then cluster the probability mass and apply our threshold property to obtain a 5-approximation for the special case where edges form an ultrametric. We also provide an $O(\log n)$ approximation algorithm (where n is the number of nodes in the graph) for general edge costs using a tree metric embedding.

3.3 Network Predesign for Min-Cost Flow

Consider a min-cost flow problem in which one wishes to send a certain amount of demand from source to sink. Each edge has (non-negative) cost and capacity values associated with it. The goal is to find a flow of minimum cost that obeys capacity constraints on edges. This problem also models a slightly different network design scenario in which the goal is to purchase sufficient amount of capacity in order to ship the demand at minimum cost. Notice that all the costs can be charged directly to installed capacity, as the amount of capacity purchased on an edge is exactly equal to the flow sent on that edge in any optimal solution.

We can extend our framework of preplanning to the min-cost flow problem described above. Suppose that instead of knowing the exact value of flow that needs to be shipped from source to sink, we are given a probability distribution on how much demand is going to be requested. We have an option of pre-installing some amount of capacity in advance at a regular cost per unit. However, as it might be too costly to purchase enough capacity to satisfy the maximum possible demand (which is unlikely to arise), we are allowed to rent additional capacity once the demands become known, but at a much higher cost. The sum of capacity installed in advance and capacity rented must also satisfy a given upper bound (total capacity) for each edge. The goal is to minimize (over the given probability distribution on demands) the expected cost of installing sufficient capacity in the network so as to be able to satisfy all of the demands.

Formal problem definition In a min-cost flow network predesign problem we are given a directed graph $G = (V, E)$ with source s and sink t. Each edge $(i, j) \in E$ has two parameters associated with it: c_{ij}, the cost of purchasing unit capacity on (i, j) (in advance), and $\hat{u}_{ij}$, the upper bound on capacity that can be installed on (i, j) (possibly infinite). We are also given a discrete stochastic demand $D \in \{0, 1 \ldots D\}$ with a probability distribution $\Pr[D = l] = p(l)$ that needs to be shipped from s to t. The objective is to minimize the expected cost of capacity required to satisfy demand D, provided that we can pre-install capacity in advance, or rent it after the actual value of D is revealed at a cost $\lambda \geq 1$ times the original price.[1] That, is we seek to minimize the cost of capacity purchased in advance plus λ times the cost of required additional capacity in expectation over the demand distribution.

A solution to a min-cost flow network predesign instance is specified by a set of values $\{a_{ij}\}$ for each edge $(i, j) \in E$ corresponding to the amount of capacity pre-installed in the network in advance. The capacity values have to be feasible with respect to the upper bounds, that is $a_{ij} \leq \hat{u}_{ij}$ for each edge $(i, j) \in E$. To minimize the cost of installing additional capacity once the actual demand becomes known, we can solve an instance of min-cost flow problem on the *preplanned* network $G' = (V, E')$. In the new network, each edge $e = (i, j)$ with non-zero value a_{ij} gets split into two parallel edges: e' with capacity a_{ij} and cost 0, and $e'' = (i, j)$ with capacity $\hat{u}_{ij} - a_{ij}$ and cost λc_{ij}. This reflects the fact, that routing up to a_{ij} units of flow on edge (i, j) is free (as that capacity has been already paid for), but more flow can be pushed on that edge by purchasing additional capacity.

The min-cost flow network predesign problem is polynomial-time solvable (as we will show below) using the standard "merger of linear programs" approach from stochastic programming described above. However, the problem is interesting in that it highlights some interesting issue in preplanning. We show that the amount of capacity that needs to

[1] Using LP approach we can actually solve the problem with non-uniform penalties $\lambda_{ij} \geq 1$.

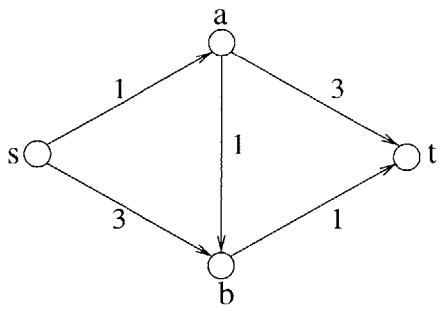

Figure 3-1: Example demonstrating anomalies in min-cost flow network predesign.

be pre-installed is not simply a function of expected demand. We also provide examples when an optimal solution doesn't use a shortest path to send any flow, and when edges on which capacity is purchased in advance do not constitute s-t paths.

Provisioning for expected demand. Contrary to one's intuition, provisioning for the expected amount of demand, e.g. buying capacity sufficient to route expected demand in the network, can be quite suboptimal. Consider a network consisting of a single path between terminal nodes s and t. Without loss of generality let the length of this path be 1. The demand D that has to be sent from s to t is equal to d with probability p, and is zero otherwise. Consider a solution that installs capacity pd (equal to expected demand value) in advance, and buys the rest of it later if the actual demand turns out to be d. Then the expected cost of this solution

$$\mathrm{E}\left[\text{cost}\right] = pd + \lambda p(1-p)d = (1+\lambda)pd - \lambda p^2 d$$

Let $p = \frac{1+\lambda}{2\lambda}$. Since $\lambda p \geq 1$, the optimal solution would buy capacity d in advance, incurring overall (expected) cost d. The ratio between these two solutions is equal to

$$(1+\lambda)p - \lambda p^2 = \frac{(1+\lambda)^2}{2\lambda} - \lambda\frac{(1+\lambda)^2}{4\lambda^2} = \frac{(1+\lambda)^2}{4\lambda}$$

Since the ratio grows as $\Omega(\lambda)$, it follows that a solution that installs in advance the amount of capacity equal to the expected demand can perform arbitrarily worse than an optimal solution.

Prepurchasing flow on non-shortest paths. Consider the graph G on 4 nodes shown in Figure 3-1. Suppose that each edge has maximum allowed capacity 1, and the amount of demand that needs to be shipped from s to t is 0, 1 or 2. Let $\lambda = 2$, $\Pr[D=0] = 1/4$, $\Pr[D=1] = 1/4$, and $\Pr[D=2] = 1/2$. Note that the only way to route 2 units of demand is by saturating the edges $(s,a),(a,t),(s,b),(b,t)$. Thus, the probability of using each of these edges is at least $1/2 \geq 1/\lambda$, so by the Threshold Property introduced in Section 3.2.2 any optimal solution has to buy 1 unit of capacity on each of those arcs in advance. However, once they are pre-installed, it is unnecessary to use arc (a,b). Hence, the shortest s-a-b-t path is never used to route flow, even if only 1 unit of demand needs to be sent.

Prepurchases need not form paths. Using the same graph G in Figure 3-1 with unit capacities on all edges, let us take $\lambda = 2, \Pr[d = 1] = 1/4, \Pr[d = 2] = 1/3$. In this case an optimal solution ends up pre-installing 1 unit capacity only on arcs (s, a) and (b, t), yielding expected cost of 6.5. To see this, note that either edge (s, a) or edge (b, t) (or both) has to be used in solutions for instances $D = 1$ and $D = 2$. Thus, the probability of using edge (s, a) (or edge (b, t)) is $1/4+1/3 > 1/\lambda$. Thus, by the Threshold Property some capacity has to be purchased in advance on at least one of these edges. Buying capacity in advance on the path s-a-t (or s-b-t) results in the expected cost of at least 20/3, whereas pre-installing 1 unit of capacity only along the shortest path s-a-b-t costs 7 in expectation. Thus, in the preplanned network, there are no paths of zero cost.

3.3.1 Linear Programming formulation

We can model the preplanning version of the min-cost flow problem with a linear program. We will use a variable a_{ij} to denote capacity purchased in advance on edge (i, j). Recall that once the exact value of demand to be shipped through the network is known, the cheapest way to install additional capacity can be computed by solving min-cost flow problem in a modified network. Thus, we can have a set of flow variables for each possible demand value. Let f_{ij}^l denote the flow routed on edge (i, j) in the event that the demand takes on value l. Although not needed, to simplify the LP, we will also use a variable u_{ij}^l to denote the additional capacity that has to be purchased on edge (i, j) later on to route the flow f_{ij}^l. The cost of installing capacity in advance is $\sum_{(i,j)\in E} c_{ij}a_{ij}$, while the expected cost of purchasing additional capacity is $\sum_{l=0}^{D} p(l) \sum_{(i,j)\in E} \lambda c_{ij} u_{ij}^l$, where $p(l)$ denotes probability $\Pr[D = l]$.

Writing down conservation of flow and capacity constraints for each demand value l, we obtain the following LP:

$$\text{Min} \quad \sum_{(i,j)\in E} c_{ij}a_{ij} + \sum_{l=0}^{D} p(l) \sum_{(i,j)\in E} \lambda c_{ij} u_{ij}^l$$

subject to:

for $l \in \{0, 1 \ldots D\}$

$$\sum_{j:(i,j)\in E} f_{ij}^l - \sum_{k:(k,i)\in E} f_{ki}^l = \begin{cases} l & \text{if } i = s \\ -l & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \qquad \text{for all } i \in V$$

$$f_{ij}^l \le a_{ij} + u_{ij}^l \qquad \text{for all } (i, j) \in E$$

$$a_{ij} + u_{ij}^l \le \hat{u}_{ij} \qquad \text{for all } (i, j) \in E$$

$$f_{ij}^l, a_{ij}, u_{ij}^l \ge 0$$

Thus, applying any polynomial-time algorithm for linear programming, we can obtain an optimal solution for min-cost flow network predesign problem.

Note that the above linear programming formulation can be easily extended to the version of the problem in which there is a constant number of commodities (each with its own demand distribution) which have to be shipped between their respective terminal nodes. In this case, we would get a separate set of flow and capacity variables for each possible combination of demand values. Since the number of such combinations grows

exponentially with the number of commodities, this approach does not extend to the general multicommodity flow problem with preplanning.

Ideally, we would like to obtain a combinatorial algorithm for min-cost flow network predesign. However, it is not entirely clear to us if the above LP is integral, that is if there always exists an optimal solution that purchases only integral amounts of capacity in advance.

3.4 Bin packing with preplanning

In the classical bin-packing problem, one is given a set of n items, with item i having size s_i ($0 \leq s_i \leq 1$). The goal is to pack all items into unit-size bins, so as to minimize the cost of a packing, which is equal to the total number of bins used. Now suppose that each item i is present (active) with probability p_i independently of the others. Thus we might be able to use considerably fewer bins than are necessary to pack the entire set once we know the set of active items. However we have to pay a higher price for bins added in the future. Our objective is to minimize the cost of bins required for packing in expectation over a random selection of a set of items.

Formal Problem Statement Let $X_1, X_2, \ldots X_n$ be mutually independent random variables representing items to be packed. We are given $\Pr[X_i = s_i] = p_i$, where $0 < s_i \leq 1$. Let B be a discrete random variable denoting the minimum number of unit-sized bins required to pack items $X_1, X_2, \ldots X_n$. Let a denote the number of bins allocated in advance, before one learns the values of random variables $\{X_i\}$. Then the cost of packing with bin pre-allocation a is equal to $a + \lambda \max(B - a, 0)$, where $\lambda \geq 1$ is the cost of bins to be added in the future. Our objective is to find the optimum number of bins a to buy in advance so as to minimize the expected cost of packing $a + \lambda \mathrm{E}[\max(B - a, 0)]$ (over a random set of active items).

Since the classical bin-packing problem ($\mathcal{NP}$-hard by itself [20]) is a special case of our preplanning version with deterministic items and $\lambda > 1$, the best we can hope is to match the approximation bound for bin-packing. Several heuristics for one-dimensional bin-packing (First Fit Decreasing and Best Fit Decreasing) achieve an *absolute* worst-case ratio guarantee of 3/2 (i.e. for any instance they use at most 1.5 times more bins than the optimum does). We should also note that the one-dimensional bin-packing admits a fully polynomial *asymptotic* approximation scheme (FPAS) (i.e. we can get a solution arbitrarily close to the optimum one for instances in which the optimum uses sufficiently many bins).

Contrary to a plausible intuition, a solution that pre-allocates in advance the minimum number of bins sufficient to pack deterministic items of size $\mathrm{E}[X_i]$ is suboptimal in general. To see this, consider an instance with just one item of size s that is active with probability p. Clearly, we need exactly one bin to pack a deterministic item of size ps. However, if $\lambda p \ll 1$, then it is much cheaper (in expectation) to see if the item becomes active and purchase the bin then, rather than to buy it in advance.

Using Theorem (3.2), we can reduce the preplanning bin-packing problem to that of computing the optimum bin pre-allocation that minimizes the expected cost of a packing obtained with any one of the approximation algorithm for bin packing (losing at most the corresponding performance guarantee in approximation). Let B_h be a discrete random variable denoting the number of bins used by a chosen bin-packing heuristic. Our problem is to find a value of a that minimizes $c(a) = a + \lambda \mathrm{E}[\max(B_h - a, 0)]$. The following result follows from the Threshold Property introduced in Section 3.2.2.

Lemma 3.3. *$c(a)$ is minimized for $a^* = \max\{a : \Pr[B_h \geq a] > 1/\lambda\}$.*

Proof. We can think of the bins as elements of the ground set to be purchased. Let us number the bins $b_1, b_2, \ldots b_N$, so that bin b_i gets used only after bins $b_1 \ldots b_{i-1}$ are picked. By the threshold property (Theorem 3.1), bin b_i should be bought in advance if and only if it is used with probability over $1/\lambda$. Thus, we get our claim.

□

Thus, we have reduced our problem to computing probabilities $\Pr[B_h \geq a]$ for each value of $a = 1, \ldots N$, where N is the number of bins required (by our heuristic) to pack all items. This is quite difficult in most cases (unless the random variables X_i are of some special form). Fortunately for us, it turns out that it is sufficient to approximate these probabilities in order to compute an approximately optimal pre-allocation a.

Lemma 3.4. *Suppose we are given a non-increasing function $\rho : \{0, \ldots n\} \mapsto [0, 1]$ that approximates the probabilities $\Pr[B_h \geq i]$ within a relative error ϵ, e.g.*

$$(1 - \epsilon) \Pr[B_h \geq i] \leq \rho(i) \leq (1 + \epsilon) \Pr[B_h \geq i].$$

Let $a_\rho = \max\{a : \rho(a) > 1/\lambda\}$. Then $c(a_\rho) \leq \frac{1+\epsilon}{1-\epsilon} c(a^*)$, i.e. pre-allocating a_ρ bins in advance yields a solution arbitrarily close the optimum.

Proof. Let's assume that $\rho(0) = 1$ (otherwise we can just set it to 1 without violating any of its properties). Note that $\rho(i)$ is a feasible probability distribution function in our context, since $0 \leq \rho(i) \leq 1$ and $\rho(i) \geq \rho(i + 1)$ for $i = 0 \ldots N$.

Recall that $c(a) = a + \lambda \mathrm{E}[\max(B_h - a, 0)]$. Writing expectation in terms of probabilities, we get

$$c(a) = a + \lambda \sum_{k=1}^{N-a} k \cdot \Pr[B_h = a + k] = a + \lambda \sum_{i=a+1}^{N} \Pr[B_h \geq i].$$

Let $c_\rho(a) = a + \lambda \sum_{s=a+1}^{N} \rho(i)$. Since $\rho(i)$ approximates $\Pr[B_h \geq i]$, we must have that

$$(1 - \epsilon)c(a) \leq c_\rho(a) \leq (1 + \epsilon)c(a).$$

By Lemma 3.3, $a^* = \max\{a : \Pr[B_h \geq a] > 1/\lambda\}$ minimizes the cost function $c(a) = a + \lambda \sum_{s=a+1}^{N} \Pr[B_h \geq i]$. Thus, a_ρ must minimize the value of $c_\rho(a)$. Hence, $c_\rho(a^*) \geq c_\rho(a_\rho)$. Combining this with the previous inequality, we get

$$(1 - \epsilon)c(a_\rho) \leq c_\rho(a_\rho) \leq c_\rho(a^*) \leq (1 + \epsilon)c(a^*),$$

yielding our claim. □

3.4.1 Probability estimation

Approximating the probabilities can easily be done through Monte Carlo sampling: perform many experiments in which a set of active items is generated according to the probability distribution, and keep track of how often a given number of bins is needed. In particular, to estimate $\Pr[B_h \geq i]$, we note the number of times m_i a packing required at least i bins, and return m_i divided by the total number of trials as our estimate. A polynomial number of trials suffices to give a sufficiently accurate estimate for the desired threshold number of bins [41]. We provide the full argument for the sake of completeness.

First we note that in order to compute a nearly-optimal pre-allocation, we don't need to estimate $\Pr[B_h \geq i]$ for all values of i. All we need is to find the largest value of i for which the estimated value of $\Pr[B_h \geq i]$ is at least $1/\lambda$. Thus, if our estimates are accurate to within a relative error ϵ, it suffices to estimate probabilities only for values of i for which $\Pr[B_h \geq i] > 1/\lambda(1 + \epsilon)$. Next we bound the number of samples required in order to obtain accurate estimates of the probabilities for such values of i.

Estimating $\Pr[B_h \geq i]$ is equivalent to performing a sequence of *Bernoulli* trials, each having probability of success $p = \Pr[B_h \geq i]$. The following lemma establishes how many samples we need in order to estimate $\Pr[B_h \geq i]$ with relative error ϵ.

Lemma 3.5. *Let Y be a Bernoulli random variable with probability of success p. If we take $k = \frac{4}{p\epsilon^2} \ln \frac{2}{\delta}$ samples of Y, then with probability at least $1 - \delta$*

$$(1 - \epsilon)p \leq s/k \leq (1 + \epsilon)p,$$

where s is the observed number of successes.

Proof. If we take k samples of Y, then the expected number of successes is kp. For $\epsilon < 1$ (which is the relative error we desire), the Chernoff bound tells us that

$$\Pr[|s - kp| > \epsilon kp] < 2e^{-\epsilon^2 kp/4}$$

which is at most δ for $k = \frac{4}{p\epsilon^2} \ln \frac{2}{\delta}$. Thus, with probability $1 - \delta$, s/k doesn't deviate from p by more than ϵp. □

Finally let us note that we can compute probability estimates for various values of i in parallel: it suffices to sample item sizes k times to obtain k values of random variable B_h, which yield k Bernoulli trials for each value of i. Since we are looking for the largest value of i such that $\Pr[B_h \geq i] > 1/\lambda(1+\epsilon)$ taking the number of samples sufficient for obtaining an accurate estimate for the smallest probability $1/\lambda(1 + \epsilon)$ is good enough for estimating the rest of probabilities. Combining Lemmas 3.4 and 3.5, we obtain the following result:

Theorem 3.6. *Taking $(1 + \epsilon)\lambda \frac{4}{\epsilon^2} \ln \frac{2}{\delta}$ samples of item sizes is sufficient to obtain a pre-allocation that with probability $1 - \delta$ yields a solution of cost at most $(1 + \epsilon)/(1 - \epsilon)$ times the cost of a solution that uses an optimal pre-allocation for a given bin-packing heuristic).*

The above result combined with the Theorem 3.2 implies that we can obtain an approximation guarantee for the preplanning version of the bin-packing problem that comes arbitraily close to any performance guarantee for regular bin-packing. Assuming that λ is polynomially bounded in the number of items, we get a polynomial-time approximation algorithm for one-dimensional bin-packing with preplanning that achieves an absolute approximation guarantee of $3/2 + \epsilon$. Moreover, using an FPAS for the regular version of the problem, we attain a polynomial (asymptotic) approximation scheme.

Remark 1. Notice that our approach to bin-packing did not use explicitly neither the distribution of random variables $X_1, \cdots X_n$ denoting items, nor their independence. In fact, it could have worked with any probability distribution, as long as we could sample from it. Thus, the above approach solves the problem (approximately) for stochastic item sizes, provided that the corresponding distributions can be sampled efficiently.

3.5 Vertex Cover with preplanning

In the (unweighted) vertex cover problem, given an undirected graph, the goal is to find a subset of vertices of minimum cardinality, such that for every edge of the graph, at least one of its endpoints is in the subset. Now suppose that only a subset of edges needs to be covered, but we don't know in advance exactly which ones. Given a probability distribution on the sets of edges to be covered, the goal of the preplanning vertex cover problem is to determine an optimum set of vertices to buy in advance (at cost 1), so as to minimize the expected cost of a vertex cover for a random subset of edges, provided that additional vertices can be added at cost λ each.

Since the regular vertex cover ($\mathcal{NP}$-hard by itself [20]) is a special case of our stochastic preplanning version (with a deterministic set of active edges, $\lambda > 1$), the best we can hope for is to get an approximation algorithm. In this section we show how to obtain constant-factor approximations for two different types of probability distributions. We achieve performance guarantees of 4 for both cases using completely different methods.

First we consider the case when the probability distribution over problem instances has bounded support, i.e. the number of possible subsets of edges to be covered is polynomially bounded. We make use of a standard technique from stochastic programming of combining linear programs for individual problem instances that we have used to solve the preplanning version of the min-cost flow problem. We show how this technique can be extended to combinatorial optimization problems that are solved by rounding fractional solutions to linear programming relaxations, and apply it to the vertex cover problem.

Next we consider the version of the problem in which edges have to covered independently with uniform probability p. Notice that the number of possible instances of the vertex cover problem is exponential in this case, so we cannot apply the stochastic programming technique. Instead, we exploit combinatorial properties of the problem and reduce it to that of finding a cover for a set of edges in a k-matching, a type of matching in which each vertex is allowed to have up to k incident edges.

3.5.1 Polynomial number of edge sets

Suppose we are given a graph $G = (V, E)$ and a probability distribution on the polynomial number of "active" edge sets from $\mathcal{F} \subseteq 2^E$ that might to be covered by the vertices. In other words, for each $F \in \mathcal{F}$, we have access to p^F, the probability that we have to construct a vertex cover for exactly the edges in F. For $F \notin \mathcal{F}$, $p^F = 0$.

We can model the preplanning version of the vertex cover problem with an integer program. For each vertex i, let $x_i = 1$ if i is bought in advance, and let $y_i^F = 1$ if i is added to the vertex cover once it is revealed that F is the set of active edges that have to be covered. For edge $(i, j) \in F$, either i or j has to be in a vertex cover for edge set F, e.g. at least one of the variables x_i, x_j, y_i^F, y_j^F has to be 1. Writing this constraint for all edges in each of the potential edge sets, obtain the following integer program:

$$\begin{array}{rll} \text{Min} & \sum_{i \in V} x_i + \lambda \sum_{i \in V} p^F y_i^F & \\ \text{subject to} & x_i + x_j + y_i^F + y_j^F \geq 1 & \forall\, F \in \mathcal{F}, \forall\, (i,j) \in F \\ & x_i \in \{0,1\} & \forall\, i \in V \\ & y_i^F \in \{0,1\} & \forall\, F \in \mathcal{F},\ \forall\, i \in V \end{array}$$

Since the number of edge sets $F \in \mathcal{F}$ is polynomial, the corresponding linear programming

relaxation can be solved efficiently. The value of an optimal solution to this LP relaxation provides a lower bound on the optimal integer solution.

Next we construct a solution to the vertex cover preplanning problem by rounding an optimal solution to the LP. Let us buy vertices i such that $x_i > 1/4$ (i.e. we round such x_i to 1). Once the set F of edges to be covered is revealed, we purchase the additional vertices i such that $y_i^F > 1/4$ (i.e. we round such y_i^F to 1). Let $\hat{x}, \hat{y}$ be the corresponding integral solution. To see that this solution is a feasible vertex cover for F, notice that for any edge $(i,j) \in F$, an averaging argument applied to the corresponding constraint implies that one of x_i, x_j, y_i^F, y_i^F must be at least $1/4$. Hence, at least one of $\hat{x}_i, \hat{x}_j, \hat{y}_i^F, \hat{y}_j^F$ must be 1 (i.e., at least one of the endpoints of (i,j) must have been bought at some stage). Furthermore, since we multiply each fractional variable x_i, y_i^F by a factor of at most 4 when we round them to $\hat{x}_i, \hat{y}_i^F$, we must have that

$$\sum_i x_i + \lambda \sum_i p^F y_i^F \leq 4 \left(\sum_i \hat{x}_i + \lambda \sum_i p^F \hat{y}_i^F \right),$$

and so our solution is a 4-approximation.

3.5.2 Independent edge set

In this section we consider the version of the problem in which each edge is active (e.g. has to be covered) independently with probability p. As before, given a graph $G = (V, E)$, we would like to determine an optimum set of vertices $A \subset V$ to buy in advance (at cost 1), so as to minimize the expected cost of a vertex cover for a random subset of edges $F \subseteq E$, provided that additional vertices can be added at cost λ each. Note that once vertices in the set A are specified, extending it to a cover of the edge set F is equivalent to finding a vertex cover in the $V \setminus A$-induced subgraph of $G_F = (V, F)$.

Let us assume that $\lambda \geq 3$. In the event $\lambda < 3$, we can obtain a trivial 3-approximation algorithm by not purchasing any vertices in advance.

Let A^* denote the set of vertices to be purchased in advance so as to minimize the expected cost of the minimum vertex cover. Then the threshold property from Section 3.2 implies that all vertices in A^* must be used in an optimal vertex cover of a random subset of edges at least with probability $1/\lambda$. In particular this means that any vertex bought in advance must have sufficiently large degree.

Lemma 3.7. *Let d_v denote the degree of vertex v. Then for all $v \in A^*$, $d_v \geq \log_{1-p}(1 - \frac{1}{\lambda})$.*

Proof. Suppose for the sake of contradiction that for some $v \in A^*$, $d_v < \log_{1-p}(1 - \frac{1}{\lambda})$. Note that the probability of using v in a vertex cover for a random set of edges is upper-bounded by the probability of v being adjacent to at least one edge in the set. The latter probability is given by $1 - (1-p)^{d_v} < 1 - (1 - 1/\lambda) = 1/\lambda$. But by the threshold property, this probability must be at least $1/\lambda$, and so we get a contradiction. □

Note that if the degree of a vertex $v \in A^*$ is exactly $\log_{1-p}(1 - \frac{1}{\lambda})$, then the solution cost does not increase if we postpone purchasing v until after a random set of edges to be covered is sampled. Consequently, we get the following result:

Corollary 3.8. *If for all $v \in V$, $d_v \leq \log_{1-p}(1 - \frac{1}{\lambda})$, then buying no vertices in advance is optimal.*

The idea of our algorithm is to select a subset of high-degree vertices that are going to be used in vertex covers of a large fraction of subsets of edges.

Definition 3.9. Given a graph $G = (V, E)$, define a *k-matching* to be a subset of edges that induces degree at most k on every vertex. Call a vertex $v \in V$ *tight* if its degree is exactly k, that is exactly k edges of the matching are incident on v.

Note that we can easily construct a maximal k-matching with a greedy algorithm: go through the edge list in some order, adding an edge to the matching unless one (or both) of its endpoints is already incident to k edges previously picked.

Set $k \left\lceil \frac{1}{1-p}(1 - \frac{1}{\lambda}) \right\rceil$. Let A_t denote the set of tight vertices of some maximal k-matching. The cost of the solution that purchases in advance vertices in A_t consists of the preplanning cost $|A_t|$, and the "wait and see" cost, which is equal to the expected cost of a vertex cover in the $V \setminus A_t$-induced subgraph. We will bound these two costs separately in terms of the cost of an optimum solution to the vertex cover preplanning problem, which we denote by OPT.

Lemma 3.10. $|A_t| \leq 3\mathsf{OPT}$.

Proof. Let us consider an instance of the preplanning problem in which only edges of the k-matching M_k have to covered (e.g. the edge set of the graph from which some edges are sampled consists just of the edges of the k-matching). Clearly, the cost of an optimal solution to this instance is no more than OPT.

Let $v \in A_t$ be one of the tight vertices. Suppose that an incident edge of v, say $e = (u, v)$, is active (i.e. has to be covered). Conditioned on this fact, the probability that u, the other endpoint of e has no other active incident edges is

$$(1-p)^{d_u - 1} \geq (1-p)^k \geq 1 - \frac{1}{\lambda} \geq \frac{2}{3},$$

since $\lambda \geq 3$. The probability that at least one incident edge of v is active is $1 - (1-p)^k \geq \frac{1}{\lambda}$. Combining, the probability that there is an active edge incident on v with no other active edge incident on the other endpoint of that edge is at least $\frac{2}{3\lambda}$.

Note that when an edge adjacent to (u, v) is active, then either v or u has to be included in the vertex cover. If the optimum purchases v in advance, then v contributes to the expected cost of our solution just as much as it contributes to the optimum. In case v is not purchased in advance, as shown above, with probability at least $\frac{2}{3\lambda}$, we get an instance such that there is an active edge adjacent to v whose other endpoint does not "cover" any additional edges. We can expect v to be added to the vertex cover at least half the time in this event, resulting in expected cost of $\lambda \cdot \frac{1}{3\lambda}$. Hence, by buying v in advance, we overpay by at most a factor 3. Thus, the cost purchasing all the tight vertices in advance is at most $3 \cdot \mathsf{OPT}$. □

Next we bound the expected cost of a vertex cover of a random edge subset in the $V \setminus A_t$-induced subgraph. Let us think of having paid $|A_t|$ to reduce our preplanning problem in the graph $G = (V, E)$ to the preplanning problem in the $V \setminus A_t$-induced subgraph of G. Note that the cost of an optimal preplanning solution for the latter instance of the problem is at most the optimum cost of the former. In the following lemma we show that an optimum algorithm for this new instance of the preplanning problem buys nothing in advance. Thus, its expected cost (which can be at most OPT) is exactly equal to the expected cost of a vertex cover in the $V \setminus A_t$-induced subgraph.

Lemma 3.11. *It is optimal to buy nothing in advance for an instance of the preplanning problem in the $V \setminus A_t$-induced subgraph of G.*

Proof. First note that in the original graph, every edge not in the k-matching must have at least one tight endpoint (otherwise it could have been added to the k-matching, making it non-maximal). Thus, vertices in A_t cover all edges not in the k-matching. This means that the $V \setminus A_t$-induced subgraph contains only edges from the k-matching. Since the edges of the k-matching induce degree at most k on every vertex, and A_t contains all vertices with degree k, the maximum degree of a vertex in the $V \setminus A_t$-induced subgraph is at most $k - 1$. By Corollary 3.8, it is optimal to buy no vertices in advance. □

Corollary 3.12. *The expected cost of a vertex cover of a random edge subset in the $V \setminus A_t$-induced subgraph is at most* OPT

The above corollary provides an upper bound on the "wait and see" cost of a solution that purchases A_t, the set of tight vertices induced by a maximal k-matching. Combining this result with Lemma 3.10, which bounds the cost of purchasing vertices in A_t in advance, we obtain the following theorem:

Theorem 3.13. *Purchasing in advance a set of all tight vertices induced by a maximal k-matching, where $k = \lceil \log_{1-p}(1 - \frac{1}{\lambda}) \rceil$ yields a solution of cost at most 4 times the optimum.*

Note that we can optimize the approximation guarantee by setting $\lambda = 2 + \sqrt{3}$ to be threshold value below which we do not purchase any part of a vertex cover in advance. This results in a slightly better approximation guarantee of 3.73.

Remark 2. Note that the above theorem implicitly assumes that the solution that purchases the set of tight vertices induced by a maximal k-matching is able to obtain an *optimal* vertex cover for a random subset of edges. Since the vertex cover problem is $\mathcal{NP}$-hard, in practice we might have to use an approximation algorithm instead. A trivial 2-approximation can be obtained by constructing a (regular) maximal matching, and including both endpoints of each edge of the matching in the cover. Using this algorithm in the second stage (in place of an exact one) we get a solution whose expected cost is within a factor of 5 of optimal.

3.6 Steiner network predesign

In the network Steiner tree problem we are given an edge-weighted graph $G = (V, E)$ and a subset of nodes $S \subset V$ that need to be connected. The goal is to find a minimum cost tree spanning S. We consider preplanning versions of this problem, in which S is the set of active clients drawn from some distribution. Most Steiner network predesign problems are $\mathcal{NP}$-hard since they contain the Steiner tree problem as a special case (with $\lambda > 1$, and a deterministic active set).

Formal problem statement Let $G = (V, E)$ be an undirected edge-weighted graph. Let $c_e \geq 0$ denote the cost of an edge $e \in E$. We call a subset S of nodes (clients) active if all of the nodes (clients) in it wish to be connected. Given a probability distribution over potential active sets $\{S\}$, the objective is to minimize the expected cost of connecting up clients in the active set. To this end a subset of edges $A \subset E$ can be purchased in advance at cost c_e, whereas the rest can be added later on at cost λc_e, where $\lambda \geq 1$. Once a set S of active clients is revealed, the cheapest way to connect up all the clients in S is to build a min-cost Steiner tree over the vertices of S using the edge cost function c_A, where $c_A(e) = 0$ if $e \in A$, and $c_A(e) = \lambda c_e$ otherwise. Let $T_{\mathsf{ST}}(S)$ be an optimum Steiner tree over S for the edge cost function c_A. The objective of the network predesign problem is to choose A so as to minimize the cost of the solution $c(A) + c_A(T_{\mathsf{ST}}(S))$ in expectation over a random set of active clients S.

A number of interesting special cases of this problem can be shown equivalent to previously-studied combinatorial optimization problems; we therefore inherit constant factor approximations from those problems.

Cheap path to root. In this problem, a root is specified in advance and it a single, randomly chosen node wishes to be connected by a path to that root. We show that this problem is equivalent to the connected facility location problem, with probability (scaled by λ) replacing demand, and hence can be approximated to within a constant [30, 53].

Cheap path. A randomly chosen pair of nodes wishes to be connected. This problem can be shown to be equivalent to the multicommodity rent-or-buy problem, with probability (scaled by λ) replacing demand, which has several constant factor approximation algorithms [35, 29].

However, our main focus is on problem instances in which each client chooses to become active with probability p_i *independently* of all other nodes. Thus, the number of possible active nodes can be large. We assume that there exists a special root vertex r to which all active clients have to connected (e.g. the root is a client that is active with probability 1). We give a constant factor approximation for the case where the expected number of active clients is constant (generalizing the Cheap Path result). For the case with an arbitrary number of active clients we obtain a 5-approximation for ultrametric edge costs and an $O(\log n)$ approximation for general edge costs.

3.6.1 Cheap path

In this section we consider a special case of the Steiner network predesign in which each active set consists of a pair of vertices (s_k, t_k) which have to be connected by a path with

probability p_k. We show that this problem is equivalent to the Multicommodity Rent-or-Buy (MROB) network design problem with probabilities acting as demands.

An instance of MROB design is specified by a graph $G = (V, E)$, an edge cost $c_e \geq 0$ for each $e \in E$, a set of source-sink pairs $\{(s_k, t_k)\}$ with corresponding demand values $\{d_k\}$, and a parameter $M > 1$. The goal is to assign each demand pair (s_k, t_k) to a path in G and install enough capacity in the network to allow simultaneous shipping demands between each pair. We are given a choice to rent capacity on edge e at unit cost c_e or pay $c_e M$ for unlimited usage of capacity on the edge paying. Thus the cost of the solution is given by $\sum_{e \in E} c_e \min\{a_e, M\}$, where a_e is the amount of capacity rented on an edge e. Kumar et al. gave the first constant factor approximation algorithm for this $\mathcal{NP}$-hard problem with a performance guarantee of 73 [35], which was recently improved by Gupta et. al to 12 [29].

Notice that in our cheap path predesign problem, once a set of edges A to be installed in advance is specified, we can assign a specific path to be used for each pair (s_k, t_k) (namely any shortest path under the cost function c_A). Furthermore, for each edge $e \in A$ it must be true that $\Pr[e \text{ used}] \geq 1/\lambda$, since otherwise it doesn't make sense to buy this edge in advance. Thus, we can reformulate our problem as that of assigning pairs (s_k, t_k) to paths so as to minimize $\sum_{e \in E} c_e \min\{\lambda \Pr[e \text{ used}], 1\}$. Observe that $\Pr[e \text{ used}]$ is just a sum of p_k over all pairs k that are assigned to use edge e. To reduce this problem to an instance of MROB, take arbitrary $M > 1$ and set demand $d_k = p_k \lambda M$.

Similarly, to reduce an instance of the MROB problem to an instance of cheap path predesign, let $D = \sum_k d_k$ denote the total demand that needs to be shipped. We can assume that $D \geq M$, since otherwise the MROB instance can be solved trivially by renting sufficient capacity on shortest paths between pairs of terminals (as there is no advantage in buying unlimited capacity). To complete the transformation, we set $p_k = d_k/D$ and take $\lambda = D/M$.

Cheap path to root. A special case of the cheap path predesign problem in which all random pairs are of the form (s_k, r) (or (r, t_k) merits a special consideration. This problem is equivalent to preplanning a *cheap path to the root* for a randomly chosen active client. Given a set of edges A bought in advance, an active client needs to just find the shortest path to the root-containing component of A. In the context of MROB design the corresponding instance of the problem simply reduces to an instance of the Connected Facility Location (CFL) problem (with zero facility-opening costs), a variant of the facility location problem in which one seeks to minimize the cost of assigning demands to facilities plus the cost of building a Steiner tree on all opened facilities. The CFL problem is also $\mathcal{NP}$-hard, but can be directly approximated to within a factor of $2 + \alpha_{ST}$ using a randomized algorithm by Gupta et al. [30] or within a factor of $3 + \alpha_{ST}$ via a deterministic primal-dual algorithm of Swamy and Kumar [53], where α_{ST} is an approximation factor for the Steiner tree problem (with the best current value being 1.55 [44]).

3.6.2 Small probabilities

The remaining discussion in this chapter is concerned with rooted Steiner network predesign. In this problem variant we are given a special root node r to which all active clients have to connected up. A client i becomes active with probability p_i independently of others. We first consider a special case of the problem in which the expected number of active clients is constant. Unsurprisingly, this problem is quite similar to the cheap path-to-root problem, in which the expected number of active clients is 1.

Let us first examine the case when the expected number of active clients is at most 1, i.e. $\sum_i p_i \leq 1$. Then, by the Markov inequality, the probability of having 2 or more active clients is at most $1/2$. Thus, at least half of the time we are in the situation in which at most one client is active. In that case, it makes sense to use a solution for the cheap path-to-root problem in which exactly one client is *allowed* to become active under the same probability distribution $\{p_i\}$ (in case $\sum_i p_i < 1$, we can create a phantom client attached by a zero-cost edge to the root r and assign it probability $1 - \sum_i p_i$). In fact, we can show that the optimum values for the two problems are closely related.

Theorem 3.14. *If the expected number of active clients is at most 1, an optimal solution for cheap path-to-root is a 2-approximation to the (rooted) Steiner network predesign problem.*

Proof. Let OPT and OPT_1 denote respectively the (expected) cost of an optimal solution to the original problem (each client becomes active independently with probability p_i) and the cheap path-to-root problem (a single active client is chosen according to the probability distribution $\{p_i\}$). We will fist show that $\mathsf{OPT} \geq 1/2 \cdot \mathsf{OPT}_1$.

Let X be a random variable denoting the number of active clients. As we have already shown, $\Pr[X \leq 1] \geq 1/2$. If we were given an option of constructing separate solutions for the case when $X = 1$ and $X \geq 2$ and then picking which one to use depending on the value that X takes on, we would do at least as well as OPT. Thus, we get the desired lower bound $\mathsf{OPT} \geq 1/2 \cdot \mathsf{OPT}_1$.

Next let us analyze the cost of an optimal solution to the cheap path-to-root problem (in which exactly one client becomes active). Let A denote the set of edges that an optimal solution purchases in advance. Now, when client i becomes active, it just needs to find $\mathcal{P}(i)$, the shortest path to the root r under the edge cost function c_A. Let U_e denote the set of clients that use the edge e in their shortest path to the root, that is $U_e = \{i : e \in \mathcal{P}(i)\}$. Thus,

$$\mathsf{OPT}_1 = c(A) + \sum_{i \in V} p_i \sum_{e \in \mathcal{P}(i)} c_A(e) = c(A) + \sum_{e \in E} c_A(e) \sum_{i \in U_e} p_i$$

Now let us use the set A for solving the original Steiner predesign problem (in which more than one client can be active). Although, given a set of active clients S the cheapest way to connect clients is by a min-cost Steiner tree over S (under the edge cost function c_A), we will the shortest path tree $T(S)$ instead. Clearly the expected cost of such a solution must be at least OPT. More precisely, it is given by

$$c(A) + \sum_{S \subseteq V} c_A(T(S)) \Pr[S] = c(A) + \sum_{e \in E} c_A(e) \sum_{S : e \in T(S)} \Pr[S] \tag{3.1}$$

Next we would like to show that this solution costs at most OPT_1. We do this by bounding $\sum_{S : e \in T(S)} \Pr[S]$, the probability that a given edge e is in the shortest path tree of a random subset S of clients. Note that the shortest path tree over the set S is a union of the shortest paths to the root of individual clients in S, i.e. $T(S) = \cup_{i \in S} \mathcal{P}(i)$. Recalling that U_e is the set of all clients that use e in their shortest path to the root, we obtain $\sum_{S : e \in T(S)} \Pr[S] \leq \sum_{i \in U_e} p_i$. Substituting this into the expression for the expected cost of

our solution (Equation 3.1, we get

$$\mathsf{OPT} \leq c(A) + \sum_{e \in E} c_A(e) \sum_{i \in U_e} p_i = \mathsf{OPT}_1.$$

But $\mathsf{OPT}_1 \leq 2\mathsf{OPT}$. Thus, using the set A for solving our problem we get a 2-approximation. □

Since the cheap path-to-root problem can be approximated to within a constant factor 3.55 [30] we have the following result:

Corollary 3.15. *There exists an approximation algorithm with a performance guarantee of 7.1 for the rooted Steiner network predesign problem in the special case when expected number of active clients is at most one.*

Let us now consider a more general case, in which the expected number of clients is equal to some constant $k > 1$. Let us divide up all clients into $2k$ groups, such that the sum of probabilities (expected number) of clients in each group is at most 1. (To see that it is always possible to do so, note that in any grouping of clients such that each group contains at most 1 active client in expectation, the probabilities sum to over $1/2$ in at most $2k - 1$ groups, so the rest of the groups can be combined into a single one.) Now, let us construct a solution for each group separately and independently – each group buys its own set of edges, and the set of active clients within each group build their own min-cost Steiner tree containing the root. Clearly, the expected cost of an optimal solution for each group is at most the expected cost of an optimal optimal solution for the entire graph. Moreover, putting all the individual solutions together, we get a solution for the entire graph of cost at most the sum of individual costs. In particular, this solution cost at most $2k$ times the most expensive individual group solution, whose cost is in turn upper-bounded by OPT. Thus, if we could find an optimal solution for each group, we would get a $2k$-approximation for the entire graph. Since each group has at most 1 active client in expectation, we can obtain a constant-factor approximation for it by Corollary 3.15. By the above argument, this yields a constant-factor approximation for the entire graph.

Theorem 3.16. *There exists an $O(1)$-approximation algorithm for rooted Steiner network predesign problem in the special case when the expected number of active clients is constant.*

3.6.3 Large node probabilities

The opposite of the few-clients case is when every client is active with some large probability. In this section we show that if all the probabilities are lower-bounded by $1/\lambda$, then the optimum is 2-approximated in this case simply by purchasing a minimum spanning of the entire node set. This is of limited interest but is an important component of our general solution.

Since the minimum Steiner tree is $\mathcal{NP}$-hard in general graphs [20], we can apply Theorem 3.2 and use any approximation algorithm instead of an exact one to come up with a prepurchase set. In particular a minimum spanning tree (MST) over all terminal nodes (using shortest path distances between pairs of nodes that are not connected by edges) provides a 2-approximation [45]. Although there exist better approximation algorithms for the Steiner tree problem, MST is particularly convenient to work with in the context of solving our problem.

Let OPT_M denote the cost of an optimum solution to the rooted Steiner network predesign problem, assuming we are restricted to connecting up the set of active clients by MSTs in place of min-cost Steiner trees. By Theorem 3.2 $\mathsf{OPT}_M \leq 2 \cdot \mathsf{OPT}$. Naturally, the best we can do for a given active set S is to buy in advance all edges of $T_{\mathsf{MST}}(S)$, a minimum spanning tree over nodes in S. Thus, $\mathrm{E}\left[c(T_{\mathsf{MST}}(S))\right]$ is a lower bound on OPT_M. Next we show that the expected cost of a minimum spanning tree over an active set is on the order of the cost of an MST over all clients when all probabilities are sufficiently large.

Lemma 3.17. *Let p be the smallest activation probability of a potential client. The expected cost of an MST over a set S of active clients is lower-bounded by p times the cost of MST on all the potential clients.*

$$\mathrm{E}\left[c(T_{\mathsf{MST}}(S))\right] \geq p \cdot c(T_{\mathsf{MST}}(V_c))$$

Before we proceed with the proof of this lemma, let us introduce the notion of ultrametrics.

Definition 3.18. An ultrametric is a metric which satisfies the following strengthened version of the triangle inequality:

$$d(x, y) \leq \max(d(x, z), d(y, z))$$

for all x, y, z.

Given a graph $G = (V, E)$ with edge weights c_{uv}, we can extract an ultrametric distance metric $[\tilde{c}_{uv}]$ by considering any minimum spanning tree of G. In particular, given a fixed MST for each pair of nodes $u, v \in V$ take $\tilde{c}_{uv}$ to be equal to the heaviest edge on the path from u to v in the MST. Note that $\tilde{c}_{uv} \leq c_{uv}$ for any $(u, v) \in E$. Clearly, $\tilde{c}_{uv} = c_{uv}$ for each MST edge (u, v). If (u, v) is not part of the MST, then by the optimality of the MST (u, v) must cost at least as much as the heaviest edge on the path from u to v in the MST, and so $\tilde{c}_{uv} \leq c_{uv}$. However, replacing the original edge costs by the MST-induced ultrametric maintains the MST optimality condition and hence preserves the MST itself.

Proof Lemma 3.17: To simplify the following discussion, we assume that our original graph has a client with a positive activation probability on every node. (We can always achieve that without changing MST costs by adding an edge between each pair of potential clients of cost equal to the shortest path length between them, and deleting all nodes whose activation probability is equal to zero.)

As a first step, let us replace the edge costs c_{uv} in the given graph G by an MST-induced ultrametric $\tilde{c}_{uv}$ as described above. Note that this does not increase the cost an MST over active clients while keeping the cost of the MST over the whole graph unchanged. Thus, it suffices to prove the theorem statement just for these MST-induced ultrametric edge costs.

Let T be an MST of the graph G produced by the Prim's Algorithm. Let us think of T as being rooted at r and let $\pi(v)$ be the parent of v in the T. Recall that each client decides to become active independently of others. Thus, instead of having the set of active clients determined at once, we can think of it as being revealed sequentially: going through the clients in an arbitrary order, at time step i the i-th client decides to become active with probability p_i. Now, consider running Prim's algorithm to find MST over a set of active clients as a similar random process: at the beginning, r is the only visited vertex. At each step, we select the next vertex v which minimizes its distance to the current set of visited

vertices. This vertex becomes active with probability p_v. If it becomes active, we add the minimum edge from v to the set of visited vertices and mark v as visited. Note that the cost of the resulting MST depends only on the set of active clients, since restricting the algorithm to just the set of active clients would have produced exactly the same ordering of nodes.

To prove our statement we will show that the cost of the edge from v to its parent in this new MST over a set of active clients is at least the cost of the edge from v to $\pi(v)$, its parent in T. Let $T(v)$ be the subtree of T rooted at v. Consider some node $w \notin T(v)$. Recall that $\tilde{c}_{vw} = \max\{c_e : e \in \mathcal{P}(v, w)\}$. Then for any $u \in T(v)$, we must have that $\tilde{c}_{vw} \leq \tilde{c}_{uw}$ since the path from u to w in T must pass through the vertex v, and hence contains all of the edges on the path from v to w in T.

Next we prove that our random process picks v first among the nodes in $T(v)$. Fix a step of the random process before it has considered any nodes in $T(v)$ and suppose S is the set of already visited vertices. Note that the random process considers the node $u \notin S$ that minimizes $\tilde{c}(u, S) = \min_{s \in S} \tilde{c}_{us}$. Since $T(v) \cap S = \emptyset$, we must have that $\tilde{c}(v, S) \leq \tilde{c}(u, S)$ for any $u \in T(v)$. Thus the algorithm has to pick the node v before any other nodes in $T(v)$.

Thus, at each step of the algorithm, the cost of the edge being added is at least the cost of the MST edge from a newly visited node v to its parent $\pi(v)$ in T. Hence, the expected added cost is at least $\tilde{c}_{v\pi(v)} p_v + 0 \cdot (1 - p_v) = p_v \tilde{c}_{v\pi(v)}$. Thus, the total expected cost of an MST on all active clients is at least $p \cdot c(T_{\mathsf{MST}}(V))$ by linearity of expectation. □

Corollary 3.19. *If each client becomes active with probability at least $1/f$ where $f \geq 1$) we obtain a $2f$-approximation to the Steiner network predesign problem by purchasing in advance edges of an MST on all potentially active clients.*

We can further lower the probability threshold for buying an MST (without losing the approximation guarantee) by performing a more careful analysis of the overall expected cost of a solution.

Corollary 3.20. *If for all clients i, $p_i \geq \frac{1}{f\lambda}$ (where $f \geq 1$), then buying in advance a minimum spanning tree over all potentially active clients is a $2f$-approximation.*

Proof. We will assume as before that each node in the graph has a client with a positive activation probability associated with it. Let A be the set of edges that an optimum solution to MST preplanning buys in advance and let OPT_M be its expected cost. Recall that $\mathsf{OPT}_M = c(A) + \mathrm{E}_S[c_A(T_M(S))]$, where $T_M(S)$ denotes an optimal MST on a set S with respect to cost function c_A. From Lemma 3.17 we have

$$\mathrm{E}_S[c_A(T_M(S))] \geq \frac{1}{f\lambda} c_A(T_M(V)) = \frac{1}{f} \sum_{e \in T_M(V) \setminus A} c_e$$

Thus, the cost of the optimum solution

$$\mathsf{OPT}_M \geq \sum_{e \in A} c_e + \frac{1}{f} \sum_{e \in T_M(V) \setminus A} c_e \geq \frac{1}{f} \sum_{e T_M(V)} c_e$$

Since $T_M(V)$ costs (with respect to the original edge costs c_e) at least as much as an MST on all nodes, we must have that OPT_M is at least $\frac{1}{f}$ times the cost of the latter. Recalling that OPT_M is within a factor of 2 from the optimum for min-cost rooted Steiner network predesign, we deduce that an MST on all vertices of G is a $2f$-approximation. □

3.6.4 Algorithm for Ultrametric Case

We now give a constant factor approximation algorithm for the Steiner network predesign problem for the case when the underlying graph $G = (V, E)$ forms an *ultrametric* — an assignment of edge weights such that $\tilde{c}_{uv} \leq \max(\tilde{c}_{uw}, \tilde{c}_{wv})$. This ultrametric property implies that the shortest-path distance between any two vertices in the graph is equal to the weight of the heaviest edge on that path.

The basic idea of our algorithm is to cluster nodes into components C_i of bounded size and then build a minimum spanning tree over representative nodes, one from each component. Intuitively, each component can be thought of as a super-client that is active whenever at least one of its clients is active. We would like for each component to be large enough so that the probability that it becomes active is roughly $1/\lambda$:

$$\Pr[C] \equiv 1 - \prod_{v \in C} (1 - p_v) \approx \frac{1}{\lambda}$$

. Lemma 3.17 lets us conclude that since the probability mass in each client is large, an approximately optimum approach to solving a new instance of the predesign problem on the super-clients is to purchase in advance an MST spanning the super-clients. At the same time, since the probability mass within each super-client is not *too* large, we can show that there is no real benefit in buying any edges *inside* any super-client. We need to have both an upper bound b and a lower bound a on the activation probabilities $\Pr[C]$ of clusters in order to apply both sides of the argument. Formalizing this intuition, we have the following algorithm for rooted Steiner network predesign.

ALGORITHM FOR STEINER NETWORK PREDESIGN

1. Compute a minimum spanning tree T over all clients.
2. Run CLUSTER$(T, \frac{1}{2\lambda}, \frac{1}{\lambda})$ to obtain a set of clusters together with their representatives $\{C, rep(C)\}$.
3. Purchase in advance edges of a minimum spanning tree T_C on the representatives $\{rep(C)\}$.

Figure 3-2: Approximation algorithm for Steiner network predesign with ultrametric edge costs.

Our algorithm uses a subroutine for clustering nodes provided in Figure 3-3. The basic idea of our clustering procedure is to take a (minimum) spanning tree and break it up into connected components by removing its edges in the order of decreasing weight until we get components of just the right size. Note that deleting an edge might result in one of the two resulting components being too small, so we need to deal with such cases separately. In particular, unless the larger (probability-wise) component consists of just one node, we merge the "too small" component into the larger component by shrinking the edge attaching it to the larger component. In that special case, we output a cluster containing both the large one-node component and the small component, possibly resulting in a cluster with the activation probability over $1/\lambda$. (Note that this is the only case when this is possible.)

Our clustering procedure outputs each cluster together with its representative. Except in the case of a large one-node component mentioned above, a cluster representative is an

```
CLUSTER(T, a, b)
  e ← arg max{c_e : e ∈ T}
  Suppose T_0 and T_1 are the connected components of T − {e}.
  if Pr[T_0] ≥ a and Pr[T_1] ≥ a
       if Pr[T_0] ≤ b then output (T_0, rep(T_0))
         else CLUSTER(T_0)
       if Pr[T_1] ≤ b then output (T_1, rep(T_1))
         else CLUSTER(T_1)
  if Pr[T_{1−i}] < a and |T_i| > 1 where i = 0 or i = 1
       then  contract e = (u_0, u_1) into node u_i to create tree T|_e
             assign p_{u_i} ← 1 − (1 − p_{u_i})(1 − Pr[T_{1−i}])
             CLUSTER(T|_e)
  if Pr[T_{1−i}] < a and |T_i| = 1 where i = 0 or i = 1
       then  output (T, v) where v = T_i
```

Figure 3-3: Clustering procedure.

arbitrary vertex in the cluster present at the time it is formed. Thus, nodes that are part of contracted small components are not eligible to become cluster representatives. This guarantees us the following property:

Lemma 3.21. *The cost of a minimum spanning tree on cluster representatives is equal to the net cost of the edges of the original MST deleted (but not shrunk) by the clustering procedure.*

Proof. Let us consider two adjacent clusters which were originally connected by an MST edge e deleted by the algorithm. Let u and v be their representatives. We will show below that $\tilde{c}_{uv} = \tilde{c}_e$. But that means that we can replace the edge e in the original MST by (u, v), obtaining yet another minimum spanning tree of the full graph. Repeating this procedure for every pair of adjacent clusters, we obtain an MST over all nodes of the graph that contains as a subtree a minimum spanning tree on cluster representatives. The cost of this subtree is equal to the cost of the edges deleted by our clustering procedure, yielding our claim.

Now let us show that $\tilde{c}_{uv} = \tilde{c}_e$ for given representatives u, v of two clusters originally connected by a deleted MST edge e. By the ultrametric property $\tilde{c}_{uv}$ is equal to the weight of the heaviest edge on the path from u to v in the MST of the full graph. If neither cluster contains a merged small component, then e was the first and only MST edge on the path from u to v considered by the algorithm, and so it must be the heaviest. If one of the components does contain a shrunk edge, say e_s, then either e_s does not lie on the path from u to v in the original MST, or this edge must have been considered after e — if it were the other way around, then e_s could have been deleted resulting in a different cluster split. In all cases, the edge e turns out to be the heaviest edge on the path from u to v, and so $\tilde{c}_{uv} = \tilde{c}_e$ as desired. □

We will analyze the cost of a solution produced by the algorithm in two stages. Let OPT denote the cost of an optimal solution to the rooted Steiner network predesign problem. First we show that the cost of the minimum spanning tree T_C constructed in Step 3 of our

algorithm given in Figure 3.6.4 is within a constant factor of OPT. Next we show that once the edges of tree T_C are purchased in advance, it is suboptimal to buy any additional edges in advance.

Lemma 3.22. *The cost of the minimum spanning tree T_C on the representatives of all clusters is at most $4 \cdot \mathsf{OPT}$.*

Proof. Let us run our algorithm on a graph G to obtain a clustering of the vertices $V(G)$. Consider a new instance of the problem. In this instance (let's call it G_C), there is one node corresponding to each cluster and the cost of an edge between two nodes is equal to the cost of the shortest edge between two vertices of the corresponding clusters.

Consider an optimum solution to the original instance of the problem for a given active set of clients. Clearly this solution is a feasible solution for the new instance G_C. Since we ignore the cost of edges inside the clusters (e.g. charge 0 for them), the cost of this solution for the instance G_C is at most OPT. Since this is true for any active set of clients, the cost of the optimum solution of the new instance G_C is at most the cost of an optimum solution to the original instance.

From Corollary 3.20, a minimum spanning tree on all nodes of G_C is a 4-approximation for the optimum solution in G_C. Therefore, its cost is at most 4OPT. We can show (analogously to Lemma 3.21) that the cost of the minimum spanning tree on nodes of G_C is equal to the net cost of the MST edges deleted by our clustering procedure. (Instead of considering representatives of two adjacent clusters, we can consider the endpoints of the shortest edge between the two clusters.) By Lemma 3.21 the latter cost is equal to the cost of the minimum spanning tree T_C on the cluster representatives, and so we must have that the cost of T_C is at most 4OPT. □

Let A denote the set of edges purchased in advance by our algorithm. Starting from the original instance G, let us construct yet another instance of the problem, called G_A, by contracting all the edges in A into the root node (the root node is spanned by edges of A). If we think of cluster assignments obtained by our algorithm, then contracting edges of A is equivalent to merging all cluster representatives into one big super-root. We claim that an optimum solution to this new instance of the Steiner network predesign problem on graph G_A buys no edges in advance.

Lemma 3.23. *It is optimal to buy no edges in advance for an instance of the problem in the graph G_A .*

To prove Lemma 3.23, we first need to establish the following claim:

Lemma 3.24. *Let C and C' be two distinct clusters obtained by our algorithm. Then the length of an edge between any two nodes $u \in C$ and $v \in C'$ (in the new graph G_A) is at least the distance from u to the super-root r, i.e. $d_{ur} \leq \tilde{c}_{uv}$.*

Proof. Let s be the representative of the cluster C. Since s was merged into the root, $d_{ur} = d_{us}$. Naturally, if $u = s$ the result holds trivially.

Since u and v are in different clusters, it means that our algorithm has removed an MST edge e between C and C' at some time, which we will denote t. By optimality of MST, $\tilde{c}_e \leq \tilde{c}_{uv}$. We need to consider two following cases. Case 1: u is part of a "too small" component contracted into some node of C. Case 2: u is a regular node inside cluster C (and not part of any small subcomponent).

We begin by considering Case 2. By the ultrametric property, the shortest-path distance from u to s in G is equal to the length of the heaviest edge on the MST path $u \rightsquigarrow s$. Let e_1 be that edge, so $d_{us} = \tilde{c}_{e_1}$. Since u is a regular node of C, the algorithm must have considered edge e before it got to edge e_1 (otherwise, e_1 would have been removed or shrunk). Hence, $\tilde{c}_{e_1} \leq \tilde{c}_e$ as edges are considered in the decreasing weight order. Thus, we have

$$d_{us} = \tilde{c}_{e_1} \leq \tilde{c}_e \leq \tilde{c}_{uv}$$

as desired.

Now consider Case 1. Let edge e_2 be the edge that was shrunk when u's small component was formed. In the case when $\tilde{c}_{e_2} \leq \tilde{c}_e$ (i.e. it was considered after the edge e), the same argument as for Case 1 applies. Otherwise, e_2 must be the heaviest edge on the path $u \rightsquigarrow s$. If there exists e_3 that is heavier than e_2, the algorithm would have considered e_3 first and would have either shrunk it (creating a different small component) or deleted it, creating a different cluster. Thus, by the ultrametric property $d_{us} = \tilde{c}_{e_2}$. But $\tilde{c}_{e_2} \leq \tilde{c}_{uv}$, since otherwise we could have deleted e_2 from the MST and inserted (u, v) in its place while decreasing the cost. Hence, we have $d_{us} \leq \tilde{c}_{uv}$.

□

Proof of Lemma 3.23: We would like to show that it is optimal to buy no edges in advance for the problem instance in a graph G_A, in which all edges of a minimum spanning tree on clusters constructed by our algorithm have been shrunk into the root node.

Let us first consider edges connecting nodes from different clusters. From Lemma 3.24 we know that there is never a need to buy an edge (u, v) between two clusters, since we can instead hook up node u (v) to the super-root by purchasing edges (path) to its respective cluster representative for less. Thus, for any random instance of the problem, the only edges used by an optimal solution are inside the clusters.

Next recall that except in the case of a cluster consisting of a large one-node component attached by an edge to a small component, the probability of a cluster becoming active (i.e. having at least one active client in inside of it) is bounded by $1/\lambda$. Thus the probability of using any given edge inside a cluster is at most $1/\lambda$. This also holds for the edges inside irregular clusters mentioned above, since the node making up the larger component is designated to be that cluster's representative, and the rest of the nodes have cumulative activation probability of less than $1/2\lambda$. Hence, by the threshold property (Theorem 3.1) none of the internal edges should be bought in advance. □

Theorem 3.25. *Our algorithm produces a solution which is a 5-approximation to the rooted Steiner network predesign problem.*

Proof. Recall that our algorithm clusters all clients into groups with activation probabilities close to $1/\lambda$, and then purchases the set of edges A forming a minimum spanning tree on cluster representatives. The cost of our solution is $c(A) + \mathrm{E}_S[c_A(T_{\mathsf{ST}}(S))]$, where $T_{\mathsf{ST}}(S)$ denotes a min-cost Steiner tree on the set of active clients S. We will analyze this cost by bounding the advance purchase cost $c(A)$ (e.g. the cost of the MST on clusters) and the expected "buy-later" cost $c_A(T_{\mathsf{ST}}(S))$ (e.g. the cost of additional edges necessary to connect up all the active clients to the root) separately.

From Lemma 3.22 we have that the cost of the minimum spanning tree on cluster representatives produced by our algorithm is at most $4 \cdot \mathsf{OPT}$.

Next note that the cost of an optimum solution to the Steiner predesign problem in the contracted graph G_A is at most OPT since any optimal solution for the original problem is feasible for the new problem (with edges in A automatically having cost zero). By Lemma 3.23 there exists an optimal solution for the new instance which buys no edges in advance. Thus, the optimum cost of the new instance is exactly equal to the expected cost of a min-cost Steiner tree in the contracted graph G_A, i.e. $\mathrm{E}[c_A(T_{\mathsf{ST}}(S))]$. Hence, the expected buy-later cost of a solution obtained by our algorithm is at most OPT. Combining the two costs we get the stated claim. □

Remark 3. It should be noted that the above argument on bounding the expected "buy-later" cost assumes that we have access to an oracle that outputs a minimum cost Steiner tree on the set of active clients. In practice, one would probably have to use an approximately optimal Steiner tree instead. Using 1.55-approximation algorithm of Robins and Zelikovsky [45], we would obtain a solution with expected cost within a factor of 5.55 of optimum.

3.6.5 General Case

Finally, we remark that if edge costs form a tree metric, we can solve the problem optimally in polynomial time. In a tree metric minimum cost Steiner tree on any subset of nodes is simply a spanning tree on that subset. Thus, for each edge we can compute the exact probability of using that edge in the Steiner tree over a random set of active clients. In particular for a given edge e, we have $\Pr[e \text{ used}] = 1 - \prod i \in U_e(1 - p_i)$, where U_e is the set of clients whose tree path to the root contains e. The edge should be purchased in advance if and only if the probability of it being used is at least $1/\lambda$.

Given a graph with arbitrary edge weights, we can induce a metric on this graph by replacing all edge costs with shortest path distances between their endpoints. Clearly doing so does not increase the cost of an optimal solution. Since any metric can be embedded into a tree metric with distances approximated by a factor of $O(\log(n))$ in expectation [18], we obtain the following result:

Theorem 3.26. *There is an $O(\log(n))$ approximation for the metric Steiner tree network predesign problem.*

3.7 Conclusions and Open Problems

In this chapter we presented a novel "preplanning" framework that allows to study the time-information tradeoff in solving problems with uncertainty in the inputs. We have examined a number of (generally $\mathcal{NP}$-hard) combinatorial optimization problems in which it makes to postulate a probability distribution over possible instances and to specify a portion of the solution in advance, and developed algorithms for computing approximately optimal pre- and post-sampling parts of a solution.

We leave open a number of questions concerning the problems we have considered. Although the Steiner network predesign problem appears to be quite challenging, we believe that there exists a constant factor approximation for general edge costs. Another interesting open question is the integrality of the linear program for the min-cost flow with preplanning. If it could be in fact shown that there exists an optimal solution that pre-installs only integral amounts of capacity, then the next natural question to ask is whether we can solve the problem using a purely combinatorial algorithm.

We could also extend our framework to other combinatorial optimization problems. One natural problem to consider in the preplanning framework is facility location. The latter problem is often used to model planning scenarios in which manufacturers have to decide where to build stores and warehouses and how much capacity to allocate them. However, this model assumes that all of the customer demands and costs are well-known in advance. In practice, a lot of these parameters are subject to an uncertainty. In particular, client demands are frequently estimated based on preliminary data, but are subject to changes in the future once the facilities are built and start operating. In this case, the manufacturer might want to save some money by building a minimum number of facilities in advance, and opening more as demands go up. Naturally there might be a penalty for postponing construction of facilities, as some demand might go unserved. Thus, in the preplanning version of the facility location problem, given a probability distribution on the demand of each client, we would like to determine which facilities should be opened in advance (and which postponed) so as to minimize the overall expected assignment cost plus the facility opening cost.

We can also easily formulate a number of stochastic scheduling problems in the context of our framework. Taking job duration times to be probabilistically distributed, we may ask how many machines should be reserved in advance in order to complete all jobs by some deadline, or how much processing time to reserve in advance (with an option of extending it later) given a fixed number of machines. The simplest version of the former problem with no ordering or release time constraints is basically captured by the bin-packing problem. However, incorporating these additional constraints might make the problem quite interesting and non-trivial.

Chapter 4

Provisioning routing networks with incomplete global knowledge

In this chapter we consider a networking problem of planning under uncertainty in a distributed setting in which obtaining global information might be infeasible, so one has to act based on local information. This scenario combines some features of the two scenarios considered earlier in the thesis. While we assume that, just like in the preplanning framework, one does not have to provide a fixed a solution in advance of observing random inputs, we are restricted to getting only a local, partial picture of active constraints. Thus, we must plan a solution that has a simple local specification, which prevents us from coping with every possible outcome in an optimal manner. In fact, we show that our locality requirement leads to a fixed "meta-solution" whose cost is a function of a random outcome. This connects us with the robot navigation scenario in which we specified the path for the robot to follow, but the amount of reward collected depended on how long the robot's batteries lasted.

4.1 Introduction

A networking problem of present-day interest is that of distributing a single data item to multiple requesters while minimizing network usage. In the presence of caches, it is never necessary for an item to traverse a network link more than once. Thus, this problem can be modeled as a Steiner tree problem: connect all requesters to the root using a minimum cost set of edges. However, the full set of clients who want an item might not be known in advance.In many real-world scenarios, the set of requesters is drawn from a larger set of clients which can be determined in advance. Furthermore, it is reasonable to assume that each client decides independently of others on whether he wants a particular document or not. Since learning the full set of requesters might be too expensive, or simply take too long, each client interested in obtaining the data item needs to make a *local* decision by specifying his own path to the server without knowing who else in the network is making a request.

This leads us to define the following *maybecast* problem. We are given a network (possibly with edge costs), a *root* node, and a collection of N *clients* at nodes in the network. Each client i will choose to contact the root independently from others with some probability p_i. In advance of the requests we must choose, for each client in the network, a path from the client's node to the root. If a client chooses to contact the root, the edges on this

path become *active*. Our goal is to minimize, over the random choices of the clients, the expected number (or cost) of active network edges. We can think of this as a probabilistic version of the rooted Steiner tree problem, in which one desires to connect a set of terminals with the root while minimizing total edge cost.

Our maybecast scenario is not unlike the preplanning framework considered in the previous chapter. Just like in the preplanning setting, in the maybecast problem we are given a probability distribution over problem instances $\{I\}$ (e.g. the set of active clients). We get to plan a solution to be used for each instance in advance. The actual cost of a solution depends on a random outcome in the second stage. However, in the maybecast problem, in the second stage each node is restricted to act only on local information. Thus, our preplanned solution must have a local specification, and so a solution to each instance consists of independent pieces determined in advance.

Our maybecast problem is of course $\mathcal{NP}$-complete, since the Steiner tree problem is a special case (with all terminals active with probability 1). In fact, our problem remains $\mathcal{NP}$-complete even in the uniform case of one client per network node and unit costs on edges. We give a constant-factor approximation algorithm. Our solution relies on some structural analysis of the optimum solution which leads to an application of facility location and Steiner tree algorithms to solve the problem.

4.1.1 Formal Problem Statement

Input: We consider an undirected graph $G = (V, E)$ with a non-negative edge weight function $l : E \to \Re_+$ and a root vertex $r \in V$. A set of N clients is assigned to a subset of vertices $S \subset V$. Client i becomes active independently with probability $p_i > 0$, in which case it communicates with the root r along some to-be-specified path. Every edge on the path from an active client to the root becomes *active*.
Output: Construct a set of paths connecting each client to the root; this is the path that will be used if the client becomes active. The goal is to minimize the *expected* total weight of *active* edges.

In order to study this problem, we think of l_e as the length of edge e and define a per-unit-length edge cost function c_e reflecting the probability that an edge will be used. Given a solution, let us denote by U_e the set of clients using edge e to communicate with the root. Then $c_e = \Pr[e \text{ is active}] = 1 - \prod_{i \in U_e}(1 - p_i)$. Using linearity of expectation, we can express the objective function as the sum over all edges of probabilities that an edge is active weighted by its length

$$E[\text{weight of active edges}] = \sum_{e \in E} \left(1 - \prod_{i \in U_e} (1 - p_i)\right) l_e$$

4.1.2 Our Contribution

In this chapter we design a constant factor approximation algorithm for the maybecast problem. We begin with a study of the optimum solution. We show that the optimum solution is invariably a tree. However, the obvious first choice of a shortest path tree can cost a factor as large as $\Theta(n^{1/2})$ times the optimum in an n-node graph. To find a better tree, we note that the optimum tree consists of a central "hub" area within which all edges are basically certain to be used, together with a fringe of "spokes" in which multiple clients can be considered to be contributing *independently* (and linearly) to the cost of the solution.

We use a facility location algorithm to identify a good set of "hubs" to which we route clients at independent (linear) costs, and then use a Steiner tree algorithm to connect the hubs to the root. Our approximation algorithm achieves a performance guarantee of 37.

To identify a good set of hubs, we introduce a new version of the facility location problem: one in which every open facility is required to have some minimum amount of demand assigned to it. This problem can also be phrased as a clustering problem where we wish to minimize the average radius of clusters without letting any cluster be too small. We present a simple bicriterion approximation for this problem, one which is loose in both assignment cost and minimum demand. This suffices for our application. We leave open the question of finding an algorithm that produces a truly feasible approximate solution.

4.1.3 Related Work

The maybecast problem can be represented as a kind of min-cost flow problem with infinite capacities and a *concave* cost function. We can think of a client i as having "demand" for a flow of capacity p_i to the root. The cost of routing along an edge exhibits an *economy of scale*: the more paths use an edge, the cheaper it is per path. By approximating our cost function c_e by a piece-wise linear function $\min\left(\sum_{i \in U_e} p_i, 1\right)$, we can establish a connection with the *buy-at-bulk network design* problem (originally introduced by Salman et al. [49]), another problem which exhibits an economy of scale. As an example, consider the problem of wiring a telephone network in some metric space such that every pair of clients is connected by a path of capacity one (devoted to just that pair). Our approximate cost function would correspond to the situation in which two types of wires are available: low cost wires of unit capacity, and "infinite" capacity wires of large cost.

In the general buy-at-bulk network design problem, one is given an undirected graph, and a set of source-sink pairs with non-negative demands. The goal is to install enough capacity on the edges in order to route all the demands while minimizing the total cost, which is a concave function of the capacity bought. The problem is known to be $\mathcal{NP}$-hard [49]. Awerbuch and Azar [7] provide a randomized $O(\log^2 n)$ approximation algorithm for this problem, where n is the number of nodes in the network. Their approach relies on the tree metric embedding of Bartal; subsequent improvements in tree embedding yield a somewhat better approximation ratio of $O(\log n)$ [18].

Subsequent work has tended to focus on the single-sink version of buy-at-bulk network design and its variations. One special case considered is the *access network design* problem, a variant of the problem in which one needs to construct a network using a set of K trunk-types with a fixed overhead charge and a service charge proportional to capacity used. The cost structure obeys economy of scale, however restrictions placed on the relation between the fixed and proportional components of the cost preclude its application to our problem. Andrews and Zhang [3] gave an $O(K^2)$-approximation algorithm for the access network design problem which was subsequently improved to a constant factor (independent of K) by Guha et al. [26].

After the initial publication of our results there followed a number of developments in network design with economies of scale. In particular, considerable improvements have been made in approximation guarantees for single-sink buy-at-bulk network design with arbitrary concave, piece-wise linear cost function. Garg et al. [22] gave an $O(K)$-approximation algorithm for the problem based on an LP rounding technique, while Guha et al. [27] independently came up with a constant-factor combinatorial approximation algorithm. The performance guarantee of the latter algorithm has been estimated to be roughly 2000 [54]. It

has been subsequently improved by orders of magnitude, first by Talwar [54] who reduced it roughly by a factor of 10, and more recently by Gupta et. al [30] who gave an approximation algorithm with a guarantee of 73. While this algorithm solves our problem as a special case, we obtain a better constant with our algorithm for the maybecast problem.

Recently Kumar et al. [35] introduced the multicommodity *rent-or-buy* (MROB) problem, a type of multicommodity buy-at-bulk network design with a two-piece linear cost function identical to our approximate cost function. This problem models a scenario in which one can rent capacity at some small cost per unit, or pay a large fixed cost for unlimited (e.g. infinite) amount of capacity. The first constant-factor approximation algorithm for this problem provided by Kumar et al. [35] was quite complex and had a very large performance guarantee (though not explicitly stated). Gupta et al. [29] has recently introduced a much simpler 12-approximation algorithm. It is worth noting that although our maybecast problem (with the approximate two-piece linear cost function) is a special case of the MROB problem (with just one commodity), it doesn't have a natural multicommodity formulation by itself, as the cost of transmission of different data items over the same link cannot be shared the same way that the capacity is shared in MROB. In particular, a given link might have to be traversed multiple times if there is more than one type of item (e.g. commodity) being sent over it.

Finally, we note that our maybecast problem can be modeled (modulo the loss of a small constant factor) as an instance of the Connected Facility Location (CFL) problem formulated by Gupta et al. [28] in the context of provisioning Virtual Private Networks after the initial publication of our work. This problem is a variant of the traditional facility location problem, as it seeks to open a subset of facilities, assign demands to them, and additionally connect up all the opened facilities via a min-cost Steiner tree. The current state-of-art algorithm for the CFL provides an approximation guarantee of 3.55 [30] which translates into roughly 6-approximation for the maybecast problem.

4.2 Preliminaries

In this section we provide some preliminary results regarding the structure of the optimum solution. We show that the paths of the optimum solution must define a tree, but that the obvious shortest paths tree can be a terrible approximation.

4.2.1 Solution Structure

We start by showing that the optimum solution to any maybecast problem is a tree.

Given a set of paths connecting each client with a root, let us think of the solution as a flow along those paths. A client i contributes $d_i = -\ln(1-p_i)$ units of flow to every edge on its path to the root. Thus, a given edge e carries $f_e = \sum_{i \in U_e} -\ln(1-p_i)$ units of flow. The cost of the flow on that edge is given by

$$\begin{aligned}\left(1 - \prod_{i \in U_e}(1-p_i)\right) l_e &= \left(1 - \prod_{i \in U_e} e^{-d_i}\right) l_e \\ &= (1 - e^{-f_e}) l_e\end{aligned}$$

This cost function is concave in f_e, the flow on that edge. To see that note that the second derivative of the cost function with respect to to the flow is negative since $\frac{d^2}{dx^2}(1-e^{-x}) =$

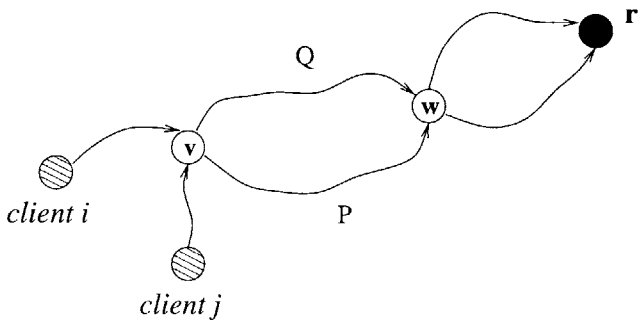

Figure 4-1: Intersecting paths

$-e^{-x} < 0$.

If we have two distinct paths between the same pair of nodes with non-zero flows f_1 and f_2 on each, by properties of concave functions, $cost(f_1 + f_2) \leq cost(f_1) + cost(f_2)$. This suggests that an optimal solution never sends flow from a given node to the root along 2 distinct paths. We deduce that it has to be a tree.

We can actually prove this in a general setting.

Theorem 4.1. *Consider any single-sink min-cost flow problem where capacities are infinite and every edge e has a nondecreasing concave cost function c_e of the total flow actually carried by that edge. Then there is an optimal (min-cost) solution to the flow problem that is a tree (that is, every vertex has at most one outgoing edge carrying flow).*

Proof. For simplicity we provide a proof only for the case where all cost functions are increasing and *strictly* convex, i.e. $f(\lambda x + (1-\lambda)y) > \lambda f(x) + (1-\lambda)f(y)$.

Consider an optimum solution. Let us decompose the flow into paths carrying flow to the root and cycles carrying flow in circles. Note that if there is a flow-carrying cycle, we can decrease flow on every edge of the cycle, decreasing the solution cost, a contradiction. So the optimum flow is decomposed into paths to the root.

We will now show that this solution must be a tree. For the sake of contradiction, assume the solution is not a tree. This can only happen if there exist two clients i and j with paths to the root that intersect at a non-root vertex and then separate, i.e. use 2 different edges out of that vertex as shown in Figure 4-1 (as a special case we might have $i = j$). Let v be a vertex where the separation happens and let w be the next intersection of the two paths. Let P denote the segment of the first path between v and w and Q the corresponding segment of the other path. Let u_P denote the amount of flow on path P, and u_Q the amount of flow on path Q. Notice that we can substitute segment P for Q to obtain a different valid path to the root for client i, and vice versa for client j. We argue that at least one of these substitutions yields an improved solution, violating our assumption of optimality.

To see this, we evaluate the cost of our flow in 3 pieces: the u_P units of flow on P, the u_Q units of flow on Q, and everything else. If we leave out the flow on P and Q, the remaining flow (which is not feasible, since it violates conservation at v and w) has some total amount of flow u_e on each edge e for some total fixed cost C.

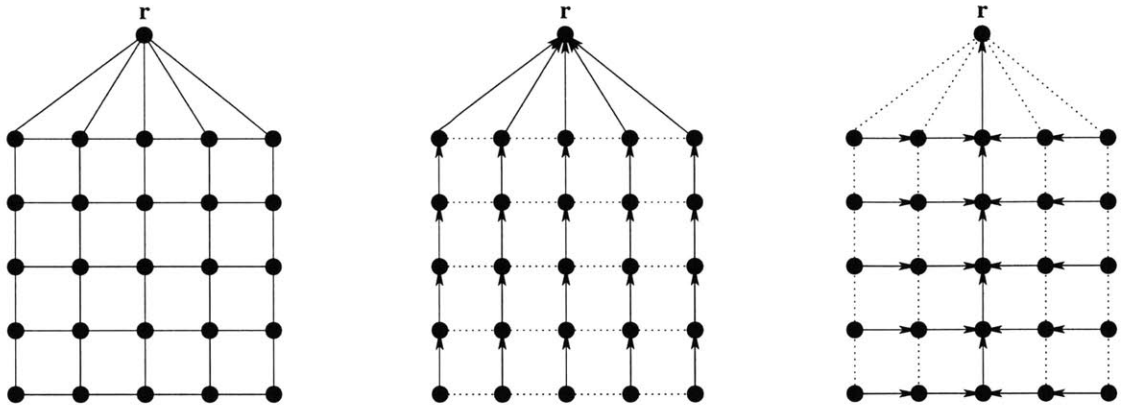

Figure 4-2: The shortest path example. a) $m \times m$ grid graph with a root. b) Shortest path tree solution. c) Alternative stem path solution.

To make this flow feasible we must add $u_{PQ} = u_P + u_Q$ units of flow from P to Q, but we can distribute this flow any way we like between P and Q. Let us consider the incremental cost over C of placing u'_P units of flow on P and u'_Q units on Q such that $u'_P + u'_Q = u_{PQ}$. The incremental cost on edge $e \in P$ of sending an additional u'_P units is $c_e(u_e + u'_P)$, which (since c_e is strictly concave) is a concave function of u'_P (offsets of concave functions are strictly concave). The overall incremental cost $f_P(u'_P)$ of sending u'_P units on P is thus a sum of strictly concave functions and thus strictly concave. Similarly, the overall incremental cost $f_Q(u'_Q)$ of sending u'_Q units of flow on Q is also strictly concave. Paths P and Q are disjoint, so our overall solution cost is simply $C + f_P(u'_P) + f_Q(u'_Q)$. We wish to minimize this subject to $u'_P + u'_Q = u_{PQ}$, and both positive which means minimizing $f_P(\lambda u_{PQ}) + f_Q((1-\lambda)u'_P)$ over $0 \leq \lambda \leq 1$. But since f_P and f_Q are strictly concave functions, this sum is a strictly concave function of λ, so is optimized at one of the "endpoints" $\lambda = 0$ or 1, i.e. $u'_P = 0$ or $u'_P = u_{PQ}$.

This shows that we can find a solution better than the (presumed optimal) one which sent flow on both P and Q, a contradiction. □

4.2.2 The shortest path tree heuristic

Now that we know that an optimal solution is a tree, a shortest path tree is a logical candidate for an approximate solution. Unfortunately, the shortest path tree can have polynomial approximation gap, as the following example illustrates.

Consider an unweighted $m \times m$ grid graph with a root vertex attached to the top m vertices by single edges as shown in Figure 1a. All edge lengths are 1.

Consider an instance in which only the m vertices on the bottom have clients, each having the same activation probability p. Let $n = m^2 + 1$ be the total number of vertices. The number of clients in this instance is then given by $N = m = \sqrt{n-1}$.

In a shortest path tree solution, every client will use a path that goes up along vertical

edges, until it merges into the root (see Figure 1b). Let us compute the cost of such a solution. Consider a vertical path of length m. Each edge of this path is used with probability p, so the cost of this path is $p \cdot m$. Summing over all m vertical paths, the total cost of the shortest path solution is pm^2.

Now consider an alternative solution, in which all clients in the bottom row go to the middle vertex of the row, and then use 1 central vertical path to get to the root (illustrated in Figure 1c). The central vertical path costs at most m since the maximum cost of an edge is 1 and there are m edges in it. Similarly, the cost of the paths converging in the center of the bottom row is at most $m - 1$. Thus, the total cost of this solution is at most $2m$. The ratio of costs of the shortest path solution and the central path solution is at least $pm^2/2m = \Omega(p\sqrt{n-1}) = \Omega(n^{1/2})$ when p is constant. Thus, a shortest path tree can give a polynomially-bad approximate solution.

4.3 A price function

To develop our approximation algorithm, we convert our concave cost function into one that is piecewise linear but closely approximates our original function.

Given a solution, let us define the *unit cost* of an edge e in the solution as the probability that e becomes active, $c_e = 1 - \prod_{i \in U_e}(1 - p_i)$, where U_e is the set of clients whose paths to the root contain e. We can assume that $|U_e| > 0$, since we can always throw out edges that are not used by any of the clients. We can upper bound c_e by the sum of activation probabilities of the clients using e, that is $c_e \leq \sum_{i \in U_e} p_i$. Notice that when this sum is small, c_e is very close to it, whereas when the sum becomes large (greater than 1), c_e rapidly approaches 1. Based on this observation, let us introduce a *unit price* function

$$\hat{c}_e = \min\left\{\sum_{i \in U_e} p_i, 1\right\}$$

We have already argued that our unit price function is an upper bound on the actual unit cost. We show that it is a tight upper bound: that $\hat{c}_e$ is $\Theta(c_e)$, independent of probabilities p_i.

Lemma 4.2.

$$c_e \leq \hat{c}_e \leq \frac{1}{1 - 1/e} c_e.$$

Proof. We have already argued the first inequality. For the second inequality, let us fix $s \equiv \sum_{i \in U_e} p_i$. Observe first that since $1 - x \leq e^{-x}$,

$$\prod_{i \in U_e}(1 - p_i) \leq \prod_{i \in U_e} e^{-p_i} = e^{-s}.$$

So we must have that $c_e \geq 1 - e^{-s}$. Now we show that

$$\hat{c}_e \leq \frac{1}{1 - 1/e}(1 - e^{-s})$$

which, combined with the previous observation, proves the lemma.

Case 1: $s \geq 1$, **so** $\hat{c}_e = 1$. The desired inequality holds trivially

$$\frac{\hat{c}_e}{1 - e^{-s}} \leq \frac{1}{1 - 1/e}$$

Case 2: $s < 1$, **so** $\hat{c}_e = s$. Consider the quantity $\frac{s}{1-e^{-s}}$. This is an increasing function of s on the interval $(0, 1]$, so the maximum is attained at $s = 1$. So once again,

$$\frac{\hat{c}_e}{1 - e^{-s}} \leq \frac{1}{1 - 1/e}$$

□

In a problem with edge lengths, the actual contribution of an edge to the objective function is its length multiplied by its unit cost, $l_e \cdot c_e$. Let us refer to this quantity as the weighted cost of an edge, or simply its cost. We can similarly define the (weighted) price of an edge to be $l_e \cdot \hat{c}_e$. Because of the linear relationship to unit price and cost, the weighted price of an edge is an upper bound on its actual weighted cost and their ratio is bounded by the same constant $1 - 1/e$.

Since costs and prices are related within a constant, we see that the optimum *cost* maybecast tree has cost within a constant factor of the optimum *price* maybecast tree. It follows immediately that any algorithm that approximates the minimum price tree to within a constant will also approximate the minimum cost tree to within a the same constant times $e/(e-1)$.

It is worth noting some similarities between our edge price function and the two-tier cost structure of the preplanning framework. Recall that, as shown in Section 3.2, the expected cost to the objective function incurred by an edge e in the Steiner network predesign problem is equal to $l_e \cdot \min\{\lambda \Pr[e \text{ used}], 1\}$. This suggests that the same kinds of techniques involving aggregation of probability mass via clustering of clients might be applicable to the maybecast problem. Below we demonstrate that this is indeed the case.

4.4 A hub and spoke model

What the grid graph example in Section 4.2.2 demonstrated was that sometimes instead of using a shortest path to the root, it pays for clients to cluster together at some node and then share one path to the root. To get a better understanding of why this is the case, let us go back to our flow analogy. Suppose each client i has to send p_i units of flow to the root via some path. Once the total amount of flow from clients using the same path reaches a certain value, adding more flow (from one more client) onto the path doesn't increase the real unit cost (the probability of the edges on the path being active) by much. This is captured by our unit price function, which uniformly charges 1 for every edge once $\sum_{i \in U_e} p_i$, the amount of flow on that edge, reaches 1.

Consider any tree solution T to the maybecast problem. Every client has a unique path to the root. We will say that a client i *passes through* node v if node v lies on i's path to the root. As paths of different clients converge towards the root, they might start sharing the same edges and passing through the same nodes. Define a *hub* to be a node that has at least 1 unit of flow from clients passing through it. The unit price of every edge on the path from a hub to the root is 1. Thus, once a client gets to a hub, the rest of its path to the root

is already paid for. Let $hub(i)$ be the first hub on the path of a client i to the root. We will say that client i is *assigned* to $hub(i)$. In building a solution, if we were given a set of hubs and an assignment of the clients to the hubs, to obtain the rest of the solution we would need to connect all of the hubs up to the root at the minimum edge cost. But edges on the paths from hubs to the root have price one, so the optimum "hub tree" is just a Steiner tree on the hubs (which we can approximate via a Steiner tree approximation algorithm).

Let us now analyze the cost of assigning a client to a hub. Any edge e on the path of i to $hub(i)$ carries no more than 1 unit of flow; otherwise we would have a hub sitting earlier on that path. This means that unit price of edge e is $\hat{c}_e = \sum_{i \in U_e} p_i$, where U_e is the set of clients using that edge. Thus, client i contributes p_i towards the unit price of every such edge. Let $h(i)$ denote the length of the path (distance) from client i to $hub(i)$. Then over all the edges on its path to a hub, client i contributes $p_i \cdot h(i)$ to the overall solution.

We can therefore decompose our problem into two parts: first, clustering clients at hubs, and second, connecting hubs up at minimum cost. Of course, we do not know where the hubs are, rather we have to create them. In this respect, the problem is similar to the facility location problem in which, given costs for opening facilities and assigning locations to them, one wants to open a set of facilities and connect each location to an open facility while minimizing net cost [31]. There is no direct cost associated with opening a hub, however the Steiner tree parallel applies only if there is a large number of clients assigned to every hub. So how can we find a set of hubs, such that there is a hub not too far from each client but every hub receives a substantial amount of demand?

4.5 A gathering problem

In the uncapacitated facility location problem [14, 17, 31, 50], we are given a set of facilities F, a set of locations C, a cost f_i for opening each facility $i \in F$, and for each pair $i \in F, j \in C$, the cost c_{ij} of connecting location j to (opened) facility i. The goal is to select a subset of facilities to open, and assign each location to an opened facility so as to minimize the total sum of the assignment costs and the costs of opening facilities. There are a number of constant-factor approximation algorithms for solving the uncapacitated facility location problem when the assignment costs satisfy the triangle inequality [14, 17, 31, 50]. The currently best approximation guarantee, due to Mahdian et. al [38], is 1.52. In a demand-weighted version of the problem, we are also given a non-negative demand d_j for each location j, so that the cost of assigning j to a facility i is $d_j c_{ij}$.

Intuitively, facility costs are supposed to encourage us to use facility resources frugally and assign many clients to the same facility. However, it is still possible to have an optimal solution in which a facility serves only very few clients. If we wanted to prevent opening a facility that serves too few clients, instead of having a fixed cost associated with each facility, we could require that a facility i can be opened only if at least a certain number of clients get assigned to i, or in the demand-weighted case, facility i gets to serve at least some minimum required demand.

Let us define an r-**gathering** problem to be a special type of demand-weighted facility location problem. The goal is to open a subset of facilities and assign demands to them, such that each opened facility has at least r units of demand assigned to it, and the total assignment cost is minimized. In our particular application, the cost of opening a facility is zero if that facility is feasible. However, the problem and our solution generalize to include facility-opening costs, and also to allow different lower bounds at different facilities.

We can formulate our problem of finding a set of hubs close to all the clients as an instance of the r-gathering problem. Take the set of all nodes of the graph to be the set of potential facilities, and the set of clients to be the set of locations. Set assignment costs to be the shortest path distances between nodes. Let the demand of a client j be p_j. If we solve the r-gathering problem on this instance with $r = 1$, the opened facilities can be thought of as hubs. Next we will show that there exists a 1-gathering solution of cost comparable to the cost of an optimum solution to our original problem.

The r-gathering problem also makes sense in the context of clustering problems: an r-gathering solution attempts to divide the clients into clusters (one per open facility) so as to minimize the average radius (distance from the cluster center) while forbidding any cluster from having too few items.

4.5.1 Solving the r-gathering problem

We now describe an approximation algorithm for the r-gathering problem. For simplicity, we first consider the case without facility-opening costs, and assume that all lower bounds are the same value r. This is the problem we need to solve in our maybecast application. This simplified version can be described as follows: given a set of potential facilities (indexed by i), a set of of clients (indexed by j), each with a demand d_j, and a cost matrix c_{ij} satisfying the triangle inequality, we wish to open a set of facilities, and assign each client to an open facility, such that

1. At least r units of demand are assigned to each open facility
2. The total distance traveled by all the demand to its assigned facility is minimized.

More formally, we wish to set assignment variables x_{ij}, where $x_{ij} = 1$ if j is assigned to i and 0 otherwise, such that

1. For every i, $\sum_j x_{ij} \geq r$ (facility i is open) *or* $\sum_j x_{ij} = 0$ (facility i is not open).
2. We minimize $C = \sum_{i,j} x_{ij} c_{ij} d_j$.

We present a *bicriterion approximation* to this problem.

Theorem 4.3. *Let C^* be the optimum cost of a feasible solution to an r-gathering problem. In polynomial time, we can find a solution to the problem, of cost C, such that*

1. *For every i, $\sum_j x_{ij} \geq r/2$ (facility i is open)* or $\sum_j x_{ij} = 0$ *(facility i is not open).*
2. $C = O(C^*)$.

That is, we can find a solution that gathers at least half the required demand at every open facility, and does so at cost within a constant factor of the optimum gathering cost. There is a typical tradeoff: we can in fact come within $(1-\epsilon)$ of the required demand (nearer feasibility) at every open facility if we are willing to worsen the approximation ratio.

To prove this theorem, given our gathering problem, we define a related facility location problem and show that its solution meets our requirements. To define the facility location problem, we assign a *cost* f_i to each facility i. The cost of i is defined as twice the minimum cost of moving r units of demand from the clients to facility i.

More formally, let $j_1, j_2, \ldots, j_n$ be the clients, ordered in increasing distance from facility i (that is, $c_{ij_1} \leq c_{ij_2} \leq \cdots$). Let k be minimum such that $d_{j_1} + d_{j_2} + \cdots d_{j_k} \geq r$. For

simplicity let us assume this sum is exactly equal to r—if not, just split client k into two smaller demands. Then the cost

$$f_i = 2(c_{ij_1}d_{j_1} + \cdots + c_{ij_k}d_{j_k}).$$

Note that for any assignment that assigns at least r units of demand to i, we must have $\sum_j c_{ij}x_{ij}d_j \geq f_i/2$.

Let $\mathcal{F}$ denote the facility location problem with costs c_{ij} inherited from the gathering problem $\mathcal{G}$ and with facility costs f_i.

Lemma 4.4. *The cost of an optimum solution to $\mathcal{F}$ is at most thrice that of $\mathcal{G}$.*

Proof. Consider any solution to $\mathcal{G}$. It opens certain facilities and makes assignments x_{ij}. The same solution is clearly feasible for $\mathcal{F}$; let us analyze its cost under $\mathcal{F}$. The assignment cost $\sum x_{ij}c_{ij}d_j$ is the same for both problems; we need only measure the added cost of opening the facilities in $\mathcal{F}$.

Consider some facility i that was opened in $\mathcal{G}$. By feasibility of the solution for $\mathcal{G}$, there are at least r units of demand incident on i. It follows that the total cost of shipping this demand to i, $\sum_j x_{ij}c_{ij}d_j \geq f_i/2$. Thus, summing over all open i, we have

$$\sum_{i \text{ open}} f_i \leq 2 \sum_{i \text{ open}} \sum_j x_{ij}c_{ij}d_j,$$

that is, the total cost of opening facilities is at most twice the total assignment cost. But that total assignment cost is the entire cost of $\mathcal{G}$, so the cost of $\mathcal{F}$ is at most 3 times the cost of $\mathcal{G}$. □

Lemma 4.5. *Any solution to $\mathcal{F}$ can be converted into a solution of no greater cost that assigns at least $r/2$ units of demand to every open facility.*

Proof. Suppose we have a solution to $\mathcal{F}$ that does not satisfy the gathering requirement. We give a procedure that modifies the solution to a cheaper one by closing a facility. We repeat this procedure until the gathering requirement is satisfied. Clearly this will happen within n repetitions.

As a first step, convert the solution to one that is *locally optimal:* assign every client to the nearest open facility. Clearly this can only improve the cost.

Now suppose there is an open facility i with less than $r/2$ incident demand. Consider the nearest clients $j_1, \ldots, j_k$ that are used to define f_i. By definition, these clients in total have $d_{j_1} + d_{j_2} + \cdots + d_{j_k} = r$ units of demand, and send at most $r/2$ units to i, so at least $r/2$ of this demand must be assigned to other facilities. Let

$$c = \frac{1}{r}(d_{j_1}c_{ij_1} + d_{j_2}c_{ij_2} + \cdots + d_{j_k}c_{ij_k}) = f_i/2r$$

be the *average distance* of these r units of nearest demand. By Markov's inequality, less than half of this nearby demand is at distance exceeding $2c$ from i. But by assumption less than $r/2$ of these nearby demand units are assigned to i. Thus, some client j' *not* assigned to i is at distance less than $2c$ from i. But by local optimality, this j' must be assigned to a facility i' at distance less than $2c$ from j'. By the triangle inequality, this facility is at distance less than $4c$ from i.

So suppose that we close facility i, take all the demand assigned to i, and assign it to i' instead. By the triangle inequality, this adds at most $4c$ to the assignment distance of those units of demand; thus the total increase in assignment cost is the amount of demand at i (less than $r/2$) times the added distance (at most $4c$) for a total of less than $2rc$. But now note that $c = f_i/2r$, meaning that the change in assignment cost is less than f_i.

Thus, by closing a facility, we have saved f_i in facility cost, and paid less than f_i in assignment cost. We can repeat this process until no facility remains with less than $r/2$ units of incident demand. □

The conversion outlined in this lemma is clearly algorithmic. This leads to the following result:

Corollary 4.6. *Given a ρ-approximation algorithm for the facility location problem, and given an r-gathering problem, we can find a solution of cost at most 3ρ times the optimum gathering cost that gathers at least $r/2$ units of demand at every open facility.*

Proof. Given the gathering problem $\mathcal{G}$ with cost G, define the related facility location problem as above, which has cost at most $3\mathcal{G}$. Run the ρ-approximation algorithm to find a solution of cost at most $3\rho G$. To that solution apply the conversion that ensures that every facility has at least $r/2$ incident demand at no greater cost. □

Applying current approximation bounds for facility location yields our initial theorem:

Corollary 4.7. *There is a bicriterion approximation algorithm for the r-gathering problem that gives an $r/2$-gather costing at most 4.56 times the optimum r-gathering.*

Proof. Use the 1.52-approximation algorithm for facility location [38]. □

There is a tradeoff between the approximation factor and the factor by which the lower bound on demand gets relaxed. By scaling our special facility cost f_i appropriately we can demonstrate the following result.

Corollary 4.8. *There is a bicriterion approximation algorithm for the r-gathering problem that gives an $r\epsilon$-gather costing at most $\frac{1+\epsilon}{1-\epsilon} \cdot 1.52$ times the optimum r-gathering.*

4.5.2 Generalizations

Our approximation algorithm straightforwardly handles two generalizations (which are not needed for our application):

- We can include an actual facility opening cost f_i'.
- We can allow the lower bound of demand needed to open each facility to be a distinct value r_i.

To handle nonzero facility costs, we simply add our special facility cost f_i (minimum cost to ship r units of demand to i) to the original facility opening cost f_i' to get the facility opening cost $f_i' + f_i$ in our derived facility location problem $\mathcal{F}$. Our analysis still shows that $\mathcal{F}$ has cost within a constant factor of $\mathcal{G}$. Furthermore, as in the simpler problem, the added cost f_i is enough to pay for closing any facility with less than $r/2$ demand and rerouting to a different open facility. Note that in fact, it suffices to set the derived problem's facility cost to be $\max(f_i, f_i')$; this might improve performance in practice.

To handle distinct lower bounds, we note that our analysis is essentially local to every vertex. If we define f_i to be the minimum cost of shipping r_i units of demand to facility i, the entire analysis goes through unchanged. We end up with a constant factor approximation that places at least $r_i/2$ units of demand on facility i if it is open.

4.6 Gathering for maybecast

In this section, we use our gathering algorithm as a black box for solving the maybecast problem. We apply the following algorithm:

MAYBECAST ALGORITHM

1. Let $F = V$, $C = S_c$. For any pair $i \in F, j \in C$, let c_{ij} be the shortest path distance between i and j. For any client $j \in S_c$, let $d_j = p_j$. Set $r = 1$. Run our bicriterion approximation algorithm on this instance of the r-gathering problem to obtain a feasible solution to an $r/2$-gathering problem.
2. Let $H \subset F$ be the set of facilities opened in Step 1. Build an approximately minimum cost Steiner tree T_S connecting nodes in H to the root r.
3. For a client j, its path P_j to the root consists of its path to the facility i to which it was assigned in Step 1 plus the path from i to the root in the tree T_S.
4. If there are cycles in P_j, remove them.

Figure 4-3: The Maybecast Algorithm

Our algorithm clusters clients into large enough groups and then connects up all the groups with the root. For clustering purposes, we will think of a client i as having p_i units of demand and take assignment costs to be equal to shortest path distances. We solve an r-gathering problem on this instance using our bicriterion approximation algorithm and then build a Steiner tree on all of the facilities opened by the r-gathering algorithm, thus connecting each client with the root via its gathering point. In the last step we simplify the paths of all clients, clearly without increasing the cost of solution.

We analyze this algorithm's performance in two steps. First, we show that the derived gathering problem has an optimum cost close to that of the maybecast optimum. Then we show that, given a solution to the derived gathering problem, we can convert it to a maybecast solution of similar cost. Combining these two arguments with our approximation algorithm for gathering shows that the maybecast algorithm finds a good solution.

4.6.1 Cost of the derived gathering problem

In this section we analyze the cost of the derived gathering problem.

Theorem 4.9. *Given an instance $\mathcal{I}$ of the maybecast problem with an optimal solution of cost* OPT, *there exists a solution to the derived r-gathering problem ($r = 1$) of cost $O(\mathsf{OPT})$.*

To prove this theorem, we will take an optimal maybecast tree for instance $\mathcal{I}$ and construct an r-gathering solution using the structure of the tree. In the optimal maybecast

solution, the path from each client eventually reaches a hub. By the definition of a hub, an ancestor of a hub is also a hub, which means that the hubs form a subtree T_H of the maybecast tree (containing the root) which we will call the *hub tree*. Demand from any one client moves on non-hub edges until it reaches a hub, and then moves on the hub tree to the root.

Consider the price $\hat{C}^* = O(\mathsf{OPT})$ of the optimum (with respect to the original cost function) maybecast solution. Let us decompose the price paid for the overall solution into the price of moving demand from each client to its first hub in the hub tree (along non-hub edges) and the price of moving demand from the hubs to the root. Along non-hub edges, the price function is linear: client i sending demand p_i along non-hub-edge e contributes $l_e p_i$ to the price function $\hat{c}_e$. So if we define $h(i)$ to be the distance (under l_e) from client i to the first hub on its path to the root, then client i contributes $p_i h(i)$ in total to the price on non-hub edges. On the other hand, by definition edges in the hub tree are carrying demand exceeding 1. Thus the price of each edge is just the length of that edge. So $\hat{C}^*_H$, the price of edges in T_H, is just the total length of hub-tree edges. We have thus argued that

$$\hat{C}^* = \sum_{\text{clients } i} p_i h(i) + \hat{C}^*_H.$$

We use this two-part decomposition to construct a solution to the gathering problem. In the first step of the solution, we move the demand from each client to its first hub, exactly as in the optimum solution. This costs us exactly the first term, $\sum p_i h(i)$. As a hub might get fewer than r units of demand, we proceed to gather all this shifted demand from the hubs into facilities, each holding at least r demand in total, at cost $\hat{C}^*_H$. This provides a solution to the gathering problem whose cost is the sum of the two parts, namely $\sum p_i h(i) + \hat{C}^*_H$. This is precisely the price of the maybecast solution, which in turn is within a constant factor of $(1 - 1/e)^{-1}$ times the true cost of the maybecast solution.

That the first part costs $\sum p_i h(i)$ is immediate from the definition of the (linear) gathering problem objective function. It remains to prove that we can gather the demand from where it arrives at the hubs at cost $\hat{C}^*_H$.

Lemma 4.10. *Given any tree with edge costs $c_e \geq 0$, demands $d_i \geq 0$ on nodes, and total demand at least r, there exists a solution to the demand-weighted r-gathering problem of cost no more than $r \cdot C$, where C is the total edge cost of T.*

Proof. By induction on the size of the tree. Suppose first that there is a leaf (degree-one node) with less than r units of demand. Move this demand from the leaf to its parent along edge e. Since we are moving less than r units this costs at most rc_e. Then we can delete edge e and the leaf. This leaves a tree with total edge cost $C - c_e$ and demand which, by induction, can be gathered at cost $r(C - c_e)$. Thus, the total gathering cost is at most $rc_e + r(C - c_e) \leq rC$ as desired.

Now suppose every leaf has at least r units of demand. Take one leaf and open a facility there. Assign all demand on the leaf to the facility. This has no cost. Then delete the leaf and its demand. Since all (unrooted) trees have at least two leaves, the remaining tree still has demand greater than r so, again by induction, it can be gathered at cost rC. □

Theorem 4.9 implies that the cost of an optimal r-gathering solution is $O(\mathsf{OPT})$. The $r/2$-gathering solution obtained with our bicriterion approximation algorithm must then

cost $O(\mathsf{OPT})$ as well. Note that in general, if we run our bicriterion approximation algorithm with a fixed demand relaxation factor of ϵ, the resulting $r\epsilon$-gathering solution is also within a constant of OPT.

Corollary 4.11. *The $r\epsilon$-gathering solution $(0 < \epsilon < 1)$ produced by our bicriterion approximation algorithm costs at most*

$$\frac{1.52}{1-1/e} \cdot \frac{1+\epsilon}{1-\epsilon} \cdot \mathsf{OPT}$$

Proof. In proving Theorem 4.9 we have shown that there exists a solution to the r-gathering problem whose cost is bounded by the price of the optimum maybecast tree, which in turn is always within a factor of $(1-1/e)^{-1}$ of the true cost of the solution. Hence, an optimal r-gathering must cost at most $\mathsf{OPT}/(1-1/e)$. By Corollary 4.8, the cost of an $r\epsilon$-gathering solution produced by our bicriterion algorithm is within a factor of $\frac{1+\epsilon}{1-\epsilon} \cdot 1.52$ from the optimum r-gathering solution. Combining this with the previous bound on the cost of the optimum r-gathering solution, we get the desired result. □

4.6.2 Cost of the Steiner Tree

The previous section showed that the cost of our gathering solution is $O(\mathsf{OPT})$. In this section, we show that the cost of the Steiner tree we build on the gathering points is also $O(\mathsf{OPT})$.

Theorem 4.12. *The cost of the minimum Steiner tree on the gathering points is $O(\mathsf{OPT})$.*

Proof. Consider the following new maybecast instance, based on our prior solution to the $r/2$-gathering problem with $r = 1$. Let us go back to our flow-demand analogy that we used in defining the hub-and-spoke model. Send each client's demand (flow) to the gathering point to which it was assigned in the 1/2-gathering solution. Since each gathering point has at least 1/2 unit of demand (flow), we will refer to it as a *half-hub*.

Let us construct a solution to this new maybecast instance with all the demands (clients) gathered at half-hubs. We will first send all the demand back to its original nodes (via shortest paths), and from there route them to the root using an optimal solution to the original maybecast problem at cost OPT. Notice that the price (which is an upper bound on the cost) of sending demand back to its original nodes is equal to the cost of gathering it, which is precisely the cost of the $r/2$-gathering returned by our bicriterion algorithm. Let us denote by $\mathsf{GAT}_{r/2}$ the latter cost and by OPT' the cost of an optimum solution to the modified maybecast instance. Thus,

$$\mathsf{OPT}' \leq \mathsf{GAT}_{r/2} + \mathsf{OPT} = O(\mathsf{OPT})$$

As has been already demonstrated in Theorem 4.1, it is actually suboptimal to send clients from the same node on different paths to the root. In an optimal solution (to the modified instance) all of the clients from a given node will use the same path to the root. Because we gathered at least 1/2 unit of demand at each gathering point, each path will have at least 1/2 unit of demand, so every edge on any path will carry at least 1/2 unit of demand. So the unit cost of each edge on the path from a half-hub to the root in this optimal solution is at least

$$1 - \prod_{i \in h}(1-p_i) \geq 1 - e^{-\sum_{i \in h} p_i} \geq 1 - e^{-1/2}.$$

It follows that the cost of every edge we use is at least $1 - e^{-1/2}$ times its length, and that the cost of the optimum solution is at least $1 - e^{-1/2}$ times the sum of the lengths of edges used.

A Steiner tree T_S on the gathering points connects all the half-hubs to the root while minimizing the total edge weight (length). Let $\mathsf{ST} = \sum_{e \in T_S} c_e$ denote the weight of an optimal Steiner tree on half-hubs. Then $(1 - e^{-1/2}) \cdot \mathsf{ST}$ is the lower bound on the OPT'. Thus we have

$$\mathsf{ST} \leq \frac{\mathsf{OPT}'}{1 - e^{-1/2}} = O(\mathsf{OPT})$$

□

We can use a known 1.55-approximation algorithm [44] to build a Steiner tree on the set of facilities opened by our bicriterion gathering algorithm. By the above theorem it is guaranteed to cost $O(\mathsf{OPT})$. Note that if we use the $r\epsilon$-gathering solution to open hubs, we can derive a similar guarantee on the cost of the resulting Steiner tree. It suffices to substitute ϵ for $1/2$ in the above argument.

Corollary 4.13. *The Steiner tree on the $r\epsilon$-gathering points obtained with our bicriterion approximation algorithm costs at most*

$$\frac{\mathsf{GAT}_{r\epsilon} + \mathsf{OPT}}{1 - e^{-\epsilon}} \cdot 1.55$$

Notice that we can decrease the cost of the Steiner tree by picking an appropriate demand relaxation factor ϵ.

4.6.3 Performance of the approximation algorithm

In this section, we show that our algorithm finds a constant factor approximation to the optimum maybecast solution.

First let us verify that we don't increase the cost of the solution by performing Step 4 of our Maybecast Algorithm given in Figure 4-3. In that step we remove any cycles after merging clients' paths to the hubs (opened by our gathering algorithm) and the Steiner tree on hubs. Consider some edge e and the set of clients U_e using it to get to the root before Step 4. Notice than when we simplify the clients' paths in Step 4, we don't add any clients to e, but we might remove some. This means that the cost of e can only decrease after Step 4. Thus, the cost of solution T obtained before we simplified the paths is an upper bound on the cost of the final solution.

To evaluate the performance of the algorithm, we measure the price of the solution obtained. The price paid can be decomposed into two components: (i) the price of moving demand from each client to its gathering point, and (ii) the price of moving demand from all the gathering points to the root. The first price component (i) is precisely the cost of the gathering solution, which we already proved $O(\mathsf{OPT})$. The second one is equal to the total price of flow on Steiner tree edges, which is upper bounded by the total length of the Steiner tree edges, i.e. the weight of the Steiner tree. We already showed this is $O(\mathsf{OPT})$. Thus, our total solution cost is $O(\mathsf{OPT}) + O(\mathsf{OPT}) = O(\mathsf{OPT})$ as claimed.

We have thus shown that the price of the solution T is at most $O(\mathsf{OPT})$. Since the price function is an upper bound on the true cost, we have the following result.

Theorem 4.14. *The* MAYBECAST ALGORITHM *yields an $O(1)$-approximation solution to the maybecast problem.*

We can rebalance the tradeoff between the cost of $r\epsilon$-gathering solution and the cost of the Steiner tree built on the corresponding gathering points. In particular, our approximation guarantee is a function of the demand relaxation factor ϵ used by the gathering bicriterion algorithm and is given by

$$\frac{1.52}{1-1/e}\cdot\frac{1+\epsilon}{1-\epsilon}+\frac{1.55}{1-e^{-\epsilon}}\cdot\left(\frac{1.52}{1-1/e}\cdot\frac{1+\epsilon}{1-\epsilon}+1\right)$$

By setting $\epsilon = 0.361$, we can obtain a solution to the maybecast problem which is within a factor of 36.5 of optimal.

Corollary 4.15. *There is an approximation algorithm for the maybecast problem that produces a solution of cost at most 37 times the optimum cost.*

4.6.4 A local optimization

In our algorithm above, we set the path of every client to go via the hub to which it was gathered in the derived gathering problem. This can result in a solution which is not a tree (paths can cross). We can fix this problem, without making the solution worse, if after finding the Steiner tree on gather points, we simply route each client, via a shortest path, to the closest point in the Steiner tree. Equivalently, we can imagine contracting the Steiner tree into the root, and building a shortest path tree on what remains. This will clearly result in a tree solution to the maybecast problem.

To see that the cost of our modified solution is no worse than before, note that our analysis above assumes that every edge in the Steiner tree is saturated, with unit price (and cost) 1. Thus, no matter how we reroute the client paths, we will never pay more than we thought on the Steiner tree edges. As for the remaining edges, we bounded their unit cost by the total demand ($\sum p_i$) of paths through the edge, thus bounding the total cost on those edges by the sum of path lengths from each client to the Steiner tree. It follows that minimizing the sum of path lengths to the Steiner tree can only improve our solution, and that is precisely what taking shortest paths to the Steiner tree does.

4.7 Conclusion

We have studied a particular routing problem in which certain global information is hard to gather both due to uncertainty and the distributed nature of the network, and developed a local-decision approximation algorithm that achieves results within constant of optimum. Our problem belongs to a class of challenging "concave cost" network design problems. Perhaps the techniques used could be applied as well to more general cost functions or the design of more complex networks.

We might also ask whether our solution can get by with less global information. At present we require global knowledge of the set of clients and activation probability p. Is it possible to define a path-based scheme which works *regardless* of p? Under such a model, all we are requiring is that every equally sized subset of clients be equally likely to activate—a plausible assumption for non-geographically-dependent requests. Even better would be to define a scheme in which our path solution is always competitive against the best possible

solution *for the set of active terminals.* But we suspect there are strong lower bounds for this variant.

4.8 Open problems

We can extend our local decision-making framework, in which only part of the solution can be preplanned, while the remainder must be generated quickly and locally, to a number of other stochastic combinatorial optimization problems. For example, when dealing with the stochastic vertex cover problem (in which each edge is "active" and thus needs to be covered with probability p) we can require that each edge picks a vertex that is going to cover it in advance. Our goal is then to pick a subset of vertices in advance that cover all edges, so that the expected number of "active" vertices (e.g. vertices adjacent to at least one active edge) is minimized. Notice that similar to maybecast, we can approximate the expected cost of each vertex v chosen to be in a cover by $\min\{pu_v, 1\}$, where u_v is the number of adjacent edges that v is assigned to cover. Thus, once a vertex of degree at least $1/p$ is included in a solution set, it makes sense for all of its adjacent edges to designate it as their cover. Observing the similarity with the threshold property from the preplanning framework, we believe that our algorithm for the vertex cover with preplanning problem that picked high-degree vertices induced by a maximal k-matching can be used to solve the local-specification version of the problem as well.

Another classical combinatorial optimization problem that has a natural local specification variant is bin-packing. Imagine a scenario in which each item needs to be assigned to a particular bin ahead of time. After observing which items actually need to be packed, we purchase only those bins that have at least one "active" item assigned to them. The goal is to assign items in such a way so as to minimize the total number of bins that need to be purchased. Note that the probability of a bin being active depends on the number of items in the bin (and their respective probabilities) but not on the sizes of items. Intuitively, we would want to gather many small items together to be assigned to the same bin(s), and have the rest of the bins filled with only a few large items per bin. This way, there are only a small number of bins that would need to be purchased for packing a random selection of items. However, this goes against the standard approach of using small items to fill gaps in mostly-packed bins. The latter approach can result in nearly every bin having a good chance of becoming active.

Bibliography

[1] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the traveling repairman problem. *Theoretical Informatics and Applications*, 20(1):79–87, 1986.

[2] G. Andreatta and L. Romeo. Stochastic shortest paths with recourse. *Networks*, 18:193–204, 1988.

[3] Matthew Andrews and Lisa Zhang. Approximation algorithms for access network design problem. *Algorithmica*, 34(2):197–215, 2002.

[4] E. M. Arkin, J. S. B. Mitchell, and G. Narasimhan. Resource-constrained geometric network optimization. In *Symposium on Computational Geometry*, pages 307–316, 1998.

[5] S. Arora and G. Karakostas. A $2+\epsilon$ approximation algorithm for the k -MST problem. In *Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 754–759, 2000.

[6] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *Siam J. Computing*, 28(1):254–262, 1999. Preliminary version in 27th Annual ACM Symposium on Theory of Computing, 1995.

[7] Baruch Awerbuch and Yair Azar. Buy-at-bulk network design. In *Proc. 38th Symposium on Foundations of Computer Science*, pages 542–547, 1997.

[8] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[9] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 1995.

[10] John R. Birge. Stochastic programming and applications. *INFORMS Journal on Computing*, 9(2):111–133, spring 1997.

[11] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proc. of the 26th Annual ACM Symposium on Theory of Computing*, pages 163–171, 1994.

[12] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. In *Proc. of the 44th Annual Symposium on Foundations of Computer Science*, Cambridge, Massachusetts, 2003.

[13] A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k-MST problem. *JCSS*, 58:101–108, 1999. Preliminary version in 28th Annual ACM Symposium on Theory of Computing, 1996.

[14] Moses Charikar and Sudipto Guha. Improved approximation algorithms for facility location and k-median problems. In *Proc. 40th Symposium on Foundations of Computer Science*, pages 378–388, 1999.

[15] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proc. of the 44th Annual Symposium on Foundations of Computer Science*, Cambridge, Massachusetts, 2003.

[16] F.A. Chudak, T. Roughgarden, and D.P. Williamson. Approximate k-MSTs and k-steiner trees via the primal-dual method and lagrangean relaxation. In *Proc. of the 8th International IPCO Conference*, volume 2081 of *Lecture Notes in Computer Science*, pages 60–70, Utrecht, the Netherlands, June 2001. Springer-Verlag.

[17] Fabian Chudak and David Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. 1998.

[18] F., S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proc. ACM Symposium on Theory of Computing*, 2003.

[19] J. Feigenbaum, C. Papapdimitriou, and S. Shenker. Sharing the cost of multicast transmissions. In *Proc. of the 32nd Annual ACM Symposium on Theory of Computing*, 2000.

[20] M.R. Garey and D.S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness.* W. H. Freeman and Company, 1979.

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

[22] 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-sink buy-at-bulk network design problem. In *Proc. 8th International IPCO Conference*, volume 2081 of *Lecture Notes in Computer Science*, pages 170–184, Utrecht, the Netherlands, June 2001. Springer-Verlag.

[23] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proc. of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 152–158, 1996.

[24] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24:296–317, 1995. Preliminary version in 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, 1992.

[25] B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

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

[27] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation for the single sink edge installation problems. In *Proc. 33d Annual ACM Symposium on Theory of Computing*, pages 383–388, 2001.

[28] A. Gupta, J.M. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. Provisioning a Virtual Private Network: a network design problem for multicommodity flow. In *Proc. ACM Symposium on Theory of Computing*, pages 389–398, 2001.

[29] A. Gupta, A. Kumar, M. Pal, and T. Roughgarden. Approximation via cost-sharing: a simple approximation algorithm for the multicommodity rent-or-buy problem. In *Proc. 44th Annual Symposium on Foundations of Computer Science*, Cambridge, MA, 2003.

[30] A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. In *Proc. ACM Symposium on Theory of Computing*, pages 365–372, 2003.

[31] Kamal Jain and Vijay V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proc. 40th Symposium on Foundations of Computer Science*, pages 2–13, 1999.

[32] D. Johnson, M. Minkoff, and S. Phillips. The prize collecting steiner tree problem: Theory and practice. In *Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000.

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

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

[35] A. Kumar, A. Gupta, and T. Roughgarden. A constant factor approximation algorithm for the multicommodity rent-or-buy problem. In *Proc. 43d Annual Symposium on Foundations of Computer Science*, pages 333–342, 2002.

[36] T. Lane and L. P. Kaelbling. Approaches to macro decompositions of large markov decision process planning problems. In *Proc. of the 2001 SPIE Conference on Mobile Robotics*, Newton, MA, 2001. SPIE.

[37] T. Lane and L. P. Kaelbling. Nearly deterministic abstractions of markov decision processes. In *Proc. of the Eighteenth National Conference on Artificial Intelligence*, Edmonton, 2002.

[38] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithmsfor metric facility location problems. In *Proc. 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 229–242, 2002.

[39] Maria Minkoff. The prize collecting steiner tree problem. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, May 2000.

[40] Debasis Mitra and Qiong Wang. Stochastic traffic engineering, with applications to network revenue management. In *Proceedings of IEEE INFOCOM 2003*, 2003.

[41] R. Motwani and P. Raghavan. *Ramdomized Algorithms.* Cambridge University Press, 1995.

[42] Andy Philpott. Official cosp stochastic programming introduction. World Wide Web, `http://stoprog.org/index.html?spintroduction.html`.

[43] M. L. Puterman. *Markov Decision Processes.* Wiley, 1994.

[44] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779, 2000.

[45] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779, 2000.

[46] R.T. Rockafellar. Optimization under uncertainty. Lecture Notes, University of Washington.

[47] R.Sitters. The minimum latency problem is NP-hard for weighted trees. In *Proc. of the 9th International IPCO Conference*, volume 2337 of *Lecture Notes in Computer Science*, pages 230–239, Cambridge, Massachusetts, USA, May 2002. Springer-Verlag.

[48] N.V. Sahinidis. Optimization under uncertainty: state-of-the-art and opportunities. University of Illinois,Urbana, February 2003.

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

[50] David B. Shmoys, Eva Tardos, and Karen I. Aardal. Approximation algorithms for facility location problems. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[51] Hossein Soroush and Pitu B. Mirchandani. The stochastic multicommodity flow problem. *Networks*, 20:121–155, 1990.

[52] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[53] C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. In *Proc. 5th APPROX*, volume 2462 of *LNCS*, pages 256–269, 2002.

[54] K. Talwar. Single-sink buy-at-bulk lp has constant integrality gap. In *Proc. 9th International IPCO Conference*, volume 2337 of *Lecture Notes in Computer Science*, pages 475–486, Cambridge, Massachusetts, USA, May 2002. Springer-Verlag.

[55] Maarten H. van der Vlerk. Stochastic programming bibliography. World Wide Web, `http://mally.eco.rug.nl/spbib.html`, 1996-2003.