Approximation Techniques for Stochastic Combinatorial Optimization Problems

Ravishankar Krishnaswamy

CMU-CS-12-120

May 2012

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant numbers CCF-0448095, CCF-0729022, and CCF-0964474. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Approximation Algorithms, Stochastic Optimization, Network Design, Scheduling

Abstract

The focus of this thesis is on the design and analysis of algorithms for basic problems in Stochastic Optimization, specifically a class of fundamental combinatorial optimization problems where there is some form of uncertainty in the input. Since many interesting optimization problems are computationally intractable (NP-Hard), we resort to designing approximation algorithms which provably output good solutions. However, a common assumption in traditional algorithms is that the exact input is known in advance. What if this is not the case? What if there is uncertainty in the input?

With the growing size of input data and their typically distributed nature (e.g., cloud computing), it has become imperative for algorithms to handle varying forms of input uncertainty. Current techniques, however, are not robust enough to deal with many of these problems, thus necessitating the need for new algorithmic tools. Answering such questions, and more generally identifying the tools for solving such problems, is the focus of this thesis. The exact problems we study in this thesis are the following: (a) the *Survivable Network Design* problem where the collection of (source,sink) pairs is drawn randomly from a known distribution, (b) the *Stochastic Knapsack* problem with random sizes/rewards for jobs, (c) the *Multi-Armed Bandits* problem, where the individual Markov Chains make random transitions, and finally (d) the *Stochastic Orienteering* problem, where the random tasks/jobs are located at different vertices on a metric. We explore different techniques for solving these problems and present algorithms for all the above problems with near-optimal approximation guarantees. We also believe that the techniques are fairly general and have wider applicability than the context in which they are used in this thesis.

Acknowledgments

First and foremost, I would like to thank my advisor Anupam Gupta, for being all that I sought in an advisor and more. He has been a constant source of encouragement and patience (needless to mention the technical ideas) throughout my PhD. I am deeply indebted for all the time and effort he has devoted (often going out of his way) towards my research and graduate life here at Carnegie Mellon University.

I have also had the great fortune of having worked with some amazing collaborators, from whom I have learned a lot. Many thanks to R. Ravi, Kirk Pruhs, Nikhil Bansal, Viswanath Nagarajan, Sungjin Im, Benjamin Moseley, Kunal Talwar, Udi Wieder, Aditya Bhaskara, Marco Molinaro, Maxim Sviridenko, Barna Saha, Danny Segev and Amit Kumar. I'm also grateful to all members of the theory group at CMU for all good times — courses, theory lunches, and just the positive vibe in general.

The last five years have been truly memorable, and for that I am thankful to all my friends (and you too, Divya). Last but not least, I would like to thank Amma and Appa for their endless support, encouragement, and prayers throughout this journey.

Contents

List of Figures

Chapter 1

Introduction

The main focus of this thesis is on the design and analysis of algorithms for stochastic combinatorial optimization problems, especially those arising in network design and scheduling. Since many interesting problems in these domains turn out to be computationally intractable (NP-Hard), we resort to designing efficient *approximation algorithms*, i.e., those that compute solutions which are provably near-optimal.

1.1 Traditional Approximation Algorithms

Owing to the intractability of most naturally occurring optimization problems, we have witnessed rapid strides made in the development of techniques for the design and analysis of approximation algorithms over the last two decades. However, the models that have predominantly been studied thus far have been somewhat stylized and restrictive in their assumptions. and somewhat restrictive. Indeed, an important modeling assumption that is typically made by most earlier works is that the *exact input* is known in advance by the algorithm. For example, consider the problem of scheduling jobs to machines in order to minimize the makespan (i.e., the maximum load on any machine). Over the years, we have developed very good approximation algorithms for this problem, including a PTAS (polynomial-time approximation scheme) via dynamic programming, and a 2-approximation using LP rounding techniques [73, 88]. However, all “classical algorithms” require knowing the exact sizes of the jobs in advance. What if the algorithm does not know the exact sizes in advance? What if it is revealed these values only as and when the jobs are being processed?

1.2 Online Optimization and its Drawbacks

A common way to overcome this deficiency of traditional algorithms is by using models of *online optimization*: here, the algorithm does not know the entire input up front, which is only revealed over time. The algorithm must (typically) make irrevocable decisions in order to satisfy the input that has been revealed until now. For example, in the online Steiner tree problem, the vertices which require connectivity to the root are revealed online, and the algorithm must always maintain a feasible solution for the currently revealed demands.

Clearly the main advantage of this model is that the algorithm can handle varying inputs, and does not need to have the entire input up front. In other words, this forces the algorithm to adapt its solution online depending on the input that is being revealed. While this tackles the issue traditional algorithms face in terms of handling input uncertainty, it has its own drawbacks in terms of both modeling and measurement of an algorithm's performance.

Indeed, the measure of performance often used in online analysis, i.e., *the competitive ratio*, is typically too pessimistic an estimator, especially in real-world problems. This is because it compares the worst case (over all possible input sequences) ratio of the cost of the online algorithm to that of an optimal offline algorithm (one that knows the exact input values in advance). For many problems, such a comparison presents an inherently large information-theoretic gap between an online algorithm's solution and that of an optimal offline solution. For example, many scheduling problems (e.g. broadcast scheduling on one server [8]) admits very good algorithms in the offline setting, but there are very strong lower bounds on the competitive ratio of any online algorithm [8]. Intuitively, since the competitive ratio is the worst case over all inputs, and the optimal solution knows the input, the competitive ratio can be thought to be the cost of not *knowing anything up front* about the final input.

However, it is often the case in practice that the algorithm is already cognizant about the input, although not entirely. For example, we typically have some distributional information about the jobs in a scheduling problem, which the algorithm could put to use. This modeling advantage is not available for online algorithms. Furthermore, what if we don't compare our algorithm's performance against the best-in-hindsight algorithm? What if we compare its performance against an optimal algorithm, *which also has to face the same uncertainty*? The model which captures both these phenomena is that of *stochastic optimization*. The following section will detail the aspects relevant for this thesis.

1.3 Stochastic Optimization

In this model we assume that the algorithm is given a *distribution* over possible actual inputs to the optimization problem up front (e.g. the distribution of possible processing times of jobs). The actual input is, however, known only at a later stage. The goal is to design algorithms that handle such uncertainty while also ensuring a good approximation guarantee (w.r.t the optimal stochastic algorithm). Depending on the particular problem considered, we have different models on how the randomness is revealed. Furthermore, the solution space of the algorithm varies based on the model being considered (and this is typically very different than the solution space of the underlying optimization problem). Finally, there are also various metrics for quantifying the performance of an algorithm in a given model. In this thesis, we restrict our attention to two stochastic models, the ones most relevant to the two kinds of problems we study — namely network design and scheduling. In the coming sections, we will highlight the different aspects (modeling, solution space, and performance measurement) of these two models.

1.4 Model I: Two-Stochastic Optimization with Recourse

In this model, often useful in capturing long-term decision-making problems, all the input uncertainty is revealed in one shot. That is, the initial input contains some uncertainty in the data. Then the stochastic algorithm has to take some decisions, after which the exact input is revealed. The algorithm can then take recourse actions. The final cost of the solution is then a function of the initial action and the random recourse action (which depends on the how the input randomness was realized, and may additionally depend on any inherent randomness in the algorithm). We now present more details.

1.4.1 Model Definition

The stochastic algorithm works in two stages. In the first stage, it is given the distributional information about the uncertain parts of the input. The parts of the input which are certain are completely specified exactly. At this point, the algorithm commits to an initial solution, called the first-stage solution, *just by looking at the distributions.* Subsequently, the actual input is revealed (i.e., the uncertainty is resolved) in the second stage. Now, the algorithm must take *recourse action* in order to augment the first-stage solution and ensure that the augmented solution is feasible for the exact input that has been revealed. The objective function in this scenario is some function of the first-stage solution and the recourse action taken, with the overall goal of designing a good first-stage solution (and subsequent recourse actions) to minimize the expected objective function, where the expectation is taken over the different possible actual inputs which can be revealed in the second stage.

To better motivate the model, consider the following practical long-term decision-making problem in the domain of infrastructure building. Indeed, consider a network design problem where an ISP (internet service provider) would like to build a network before beginning service. Moreover, it is reasonable to expect that, by conducting some surveys and demographic research, the ISP has some distributional information about the bandwidth requirement of each demand vertex in the network. Therefore, in the *first stage*, the ISP would build a base network, by just looking at these distributions. Subsequently, when the service is launched, the ISP gets to see the exact subscriber base, and he may have to augment the network in the *second stage.* Of course, the augmentation process suffers inflated costs, due to the fact that it has to be done on short notice, in rapid reaction to the observed demand. The goal, then is to carefully build the first stage network in order to minimize the total expected cost of the final network (first stage cost plus expected (inflated) second stage cost).

Intuitively, the main algorithmic challenge is to carefully build a (common) first-stage solution on which the typical cost of recourse is small, for a random realization of the input. We present this formally, in the definition below.

Definition 1.4.1 (Two-Stage Stochastic Optimization) *We are given an underlying optimization problem. The input to the 2-stage stochastic version of the problem consists of a probability distribution over possible realizations of the input data called scenarios, and the goal is to construct a feasible solution in two stages. In the first stage, the algorithm may take some decisions to construct an anticipatory part of the solution, x, incurring a cost of $c(x)$. Subsequently* *a scenario A is realized according to the distribution, and in the second stage, the algorithm may augment the initial decisions by taking recourse actions y_A (in order to make the combined solution $x \cup y_A$ feasible for the scenario A), incurring a certain (typically inflated) augmentation cost $f(x, y_A)$. The goal is then to choose the initial decisions so as to minimize the expected total cost, $c(x) + E_A [f(x, y_A)]$, where the expectation is taken over all scenarios A according to the given probability distribution.*

Typically, the possible actions in the two stages belong to the same set, but the actions are more expensive in the second stage than in the first. By having the second-stage actions be more expensive, the algorithm faces a trade-off between committing to actions in the first stage while having only imprecise information (but at lower costs), and deferring decisions to the second stage, when we have no uncertainty about the input, but the actions are more expensive. As mentioned above, there are many practical applications of this model, and much of the textbook of Birge and Louveaux [20] is devoted to problems in this model. The interested reader may refer to this book for additional examples of the two-stage stochastic optimization model.

1.4.2 Example: Stochastic Steiner Tree

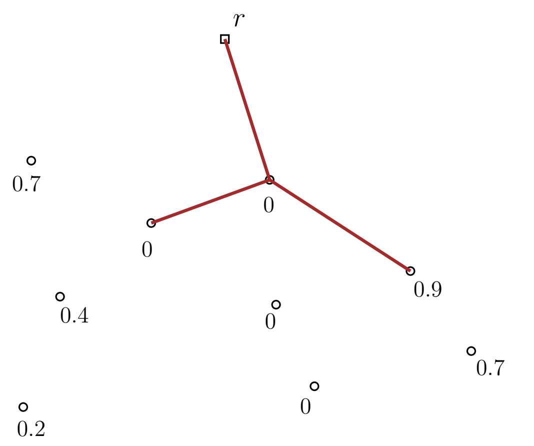
In this problem, we are given a graph $G = (V, E)$ with a non-negative cost function $c : E \to \mathbb{R}^+$ on the edges, and a designated root vertex r. Each non-root vertex $v \in V \setminus \{r\}$ has a probability p_v of requiring connectivity to r in the final input. We are also given an inflation parameter $\sigma > 0$ which specifies how much more expensive edges are in the second stage. These probabilities are what captures the uncertainty in the input. Indeed, if $p_v \in \{0, 1\}$, then this is just the deterministic (minimum) Steiner tree problem. The algorithm, in the first stage is allowed to build an anticipatory subgraph $H \subseteq E$, which should intuitively be expandable (at low cost) to cater to a typical demand realization.

Then, in the second stage, the actual set of terminals is sampled from the product distribution, i.e., any vertex v requires connectivity with probability p_v, *independent of the other vertices*. Now, let D be the set of vertices which are sampled to require connectivity. Then, the algorithm must augment H with a subgraph $H_D \subseteq E$ such that $H \cup H_D$ contains a feasible Steiner tree connecting the vertices in $D \cup \{r\}$. The objective function is then to minimize the total expected cost of the first stage solution H and the recourse solution H_D, i.e., $\min_H c(H) + \mathbb{E}_D [\sigma \cdot c(H_D)]$.

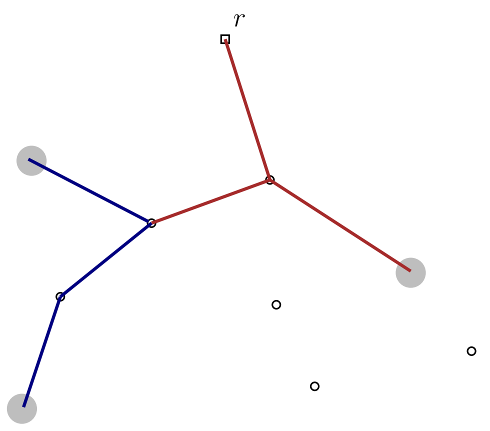
Notice how the choice of H has to carefully trade-off (i) a high cost of anticipatory solution, and (ii) a high recourse cost. Indeed, not building any anticipatory solution would lead to very high second-stage costs, while on the other hand, the algorithm cannot also build an all-encompassing first-stage solution as that would incur a disproportionately large cost in the first stage. An example of a first stage solution is given in Figure 1.1(a), and the corresponding recourse action for a particular demand realization is given in Figure 1.1(b).

1.4.3 Related Work

Two-stage stochastic optimization problems have been widely studied in both the computer science and OR literature, starting with the works of Dantzig [39], and Beale [12] for *stochastic linear programming*. Due to the abundance of literature dealing with the efficient computability



(a) Original input distribution. Brown subgraph is Stage-I solution



(b) Gray blobs denote instantiated demands. Blue subgraph is Stage-II recourse action

of two-stage stochastic linear programming, we refer the interested reader to the book by Birge and Louveaux [20] for a comprehensive discussion of this subject. On the other hand, the design and analysis of algorithms for two-stage stochastic optimization problems, from the point of view of obtaining integral solutions, is relatively less well-understood. The first result in this regard appears to be that of Dye et al. [43] who give a constant-factor approximation algorithm for a resource-allocation problem with uncertainty. Subsequently, a series of papers [64, 76, 94, 95] appeared on this topic in the CS literature, and showed that one can obtain guarantees for a variety of stochastic combinatorial optimization problems, by adapting the techniques developed for the deterministic analogs. In particular Gupta et al. [64] consider the black-box model and show an explicit connection between the approximability of the stochastic problem and the underlying deterministic combinatorial optimization problem. We will crucially use these results in our network design algorithms from Chapter 2.

1.5 Model II: Adaptive Stochastic Optimization

While the model of two-stage optimization is relevant and applicable for decision-making problems over a long time-frame (such as that of network design and layout), we need different models to handle more immediate-term decision-making problems such as scheduling (stochastic) jobs on a processor. Indeed, in such scheduling problems where the input uncertainty is often manifested in the processing times of the jobs, it makes much more sense for the randomness to be revealed in more than one stage (e.g., as and when the jobs are being executed). The model of adaptive stochastic optimization precisely captures such settings. While there is no formal definition of the model we consider (the details differ based on the problem being considered), we will introduce the model with a particular problem in mind. It will be easy to generalize it subsequently for the other problems we consider in this thesis.

1.5.1 Model Definition, for the Stochastic Knapsack problem

In the deterministic Knapsack problem, we are given a collection $\mathcal{J}$ of n items, each with a size (processing time) p_i and reward r_i, and there is a Knapsack of capacity B (which is the total processing budget we have). The goal is to choose a subset of items which can be processed within the budget, such that their total reward is maximized. An alternate view of the same problem that is more amenable to our framework is that of scheduling the best subset of a collection of jobs within a time/makespan budget of B.

The above problem is a classical one in the fields of optimization and approximation algorithms, and is perhaps the poster-child of the class of algorithms which admit an FPTAS *(a fully polynomial time approximation scheme).*

Now, in the basic version of the Stochastic Knapsack problem, the jobs' processing times are random variables, whose exact values are unbeknownst to the algorithm, until the job is actually processed. However, the algorithm is aware of the *distribution* over possible processing times of each item (and these distributions are independent across jobs). An algorithm can therefore schedule jobs in the following *adaptive* manner:

(i) at some time, a job may complete at a certain size, and the algorithm may choose a new job to start, or

(ii) the Knapsack becomes overfilled, at which point the algorithm stops, and the job being processed does not fetch any reward. Notice that in our model, the algorithm does not know the size of a job until the job completes.

The objective is to maximize the total expected reward obtained from all completed items. Notice that in the most basic model stated above, we do not allow the algorithm to cancel an item before it completes, i.e., its decisions are irrevocable. One of the contributions in this thesis is in relaxing such restrictions and solving a very general Stochastic Knapsack problem.

A word on Adaptivity. Indeed, the crucial step in adaptive solutions is step (i) above. In deterministic Knapsack instances, the sizes of jobs are fixed, and therefore the set of jobs to run can be specified as a just a sequence or ordering up front. However, in the Stochastic Knapsack problem, since the sizes are random variables, an algorithm may choose to run *different jobs* based on how long the prior jobs took to complete. This makes the structure of the solution space a lot more complex — in fact, a natural representation of an optimal adaptive strategy is in the form of a *decision tree*, where the different branches correspond to the different random outcomes of the job currently being processed. It can therefore be *exponentially large*[1] to describe completely. This in fact begets the natural goal of investigating whether compact strategies can capture these exponential-sized ones, in an approximate manner with respect to the objective function. The following simple example better illustrates this point.

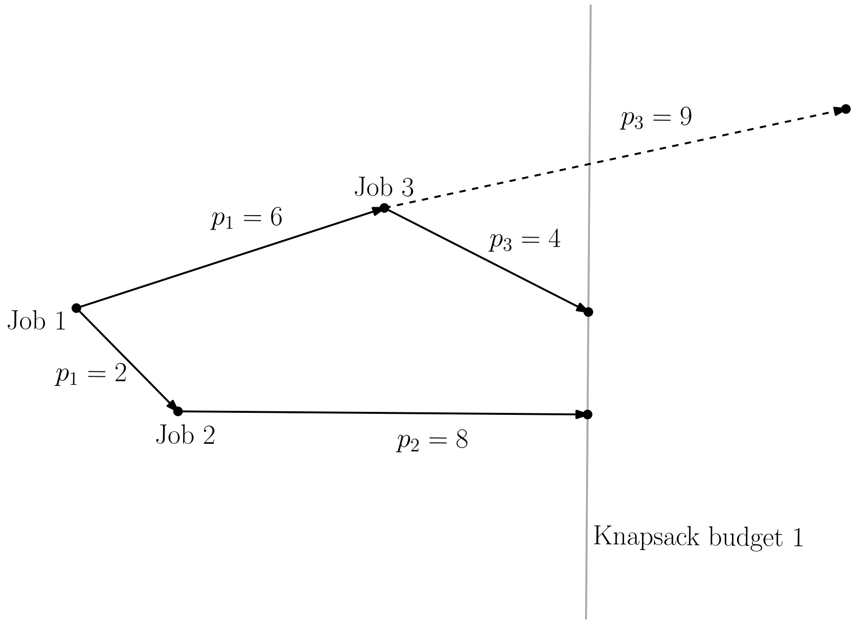
[1]With respect to the input size

1.5.2 A Simple Example

Consider the following instance of Stochastic Knapsack. There are 3 jobs: job 1 takes on a size of $p_1 := 2$ with probability $1/2$, and $p_1 := 6$ with probability $1/2$; job 2 has a deterministic size of $p_2 := 8$, and job 3 has a size of $p_3 := 4$ with probability $1/2$ and $p_3 := 9$ with probability $1/2$. The Knapsack budget $B = 10$, and all the rewards are 1.

Adaptive Optimal Strategy. In this case, an optimal adaptive) schedule is given in Figure 1.1(c), and has the following structure: it first runs job 1. If it finished after 2 timeunits, it runs job 2 (since it has a residual budget of 8, and job 2 can deterministically run in 8 timeunits). After job 2 runs, the Knapsack has become full and it does not collect any subsequent reward. On the other, if job 1 ran for 6 timeunits, there is no point in starting job 2 as it would not finish before the budget runs out. Therefore, the algorithm runs job 3, which finishes within the Knapsack budget with probability $1/2$ (if its size came out to 4. Again, if job 3 runs for longer, the budget is exceeded and the algorithm doesn't collect reward from job 3 also.

A simple expectation calculation shows that the expected total reward of the above strategy is 1.75.



(c) The optimal schedule for the 3 job example. Not to scale beyond $t = 10$

Non-Adaptive Solutions. We now highlight the *power of adaptivity* by estimating the expected reward of the optimal non-adaptive solution. Non-adaptive solutions are those which *do not* process different jobs based on the outcomes of the prior jobs' processing times. As a consequence, these solutions can be expressed as just a sequence of jobs $\langle j_1, j_2, \ldots, j_n \rangle$ to run, just like solutions to the deterministic Knapsack problem.

To this end, consider the non-adaptive solution $\langle j_1, j_2, j_3 \rangle$ for the toy example. It is easy to see that j_2 will be processed completely if and only if j_1 takes on size 2, which happens with probability $1/2$. In either scenario, the Knapsack budget is fully used up (or exceeded) after j_2 and hence the expected reward is 1.5. Moreover, it is easy to see that this is the best possible expected reward which we can achieve using non-adaptive solutions, thereby establishing a $\Omega(1)$ *adaptivity gap*. A common theme in this model is in bounding this adaptivity gap, i.e., can simple non-adaptive solutions approximate sophisticated adaptive solutions well? We will analyze this issue for the problems we consider in this thesis as well.

1.5.3 Related Work

Approximation algorithms have been studied for adaptive versions of a number of combinatorial optimization problems, e.g. in the areas of machine scheduling [89], knapsack [42], budgeted learning [59], matchings [10], stochastic queuing [78], bandit problems [16], etc.

Of all of the above, the basic Stochastic Knapsack problem [42] is the one most related to the results of this thesis. In their seminal paper, Dean et al. [42] show constant-factor approximation algorithms (and also constant-factor upper bounds on the adaptivity gap) for the problem where the jobs can have random sizes (which are independent across different jobs). They left open the various generalizations where, e.g., the reward of a job could be correlated with its size, or the algorithm can preempt jobs, or even prematurely cancel jobs that are taking too long. We address all these problems, and also substantially generalize these results to the case where each job behaves in a Markovian way (making random transitions with every timestep), and the goal is to maximize the expected reward of all the states we visit across the different jobs.

1.6 Results in this Thesis

As mentioned earlier, we study the approximability of some basic problems in the fields of network design and scheduling, from the point of view of handling input uncertainty. We now present a more rigorous description of the problems we study, as well as a few high-level technical ideas and contributions of this thesis. Each of the problems listed below will form the subsequent chapters of this thesis.

1.6.1 Two-Stage Stochastic Survivable Network Design

In this two-stage *Stochastic Survivable Network Design* problem, we are given a graph $G = (V, E)$ with associated edge costs $c : E \to \mathbb{R}^+$. This is the deterministic part of the input. In the uncertain part of the input, we are given, for each pair of vertices $\{s, t\}$, a probability p_{st} of this pair actually needing connectivity (to an extent of r_{st} which is also given as input and is not part of the uncertain data). There is also an inflation parameter of σ, specified in the input.

The goal of the algorithm is to build a subgraph H in the first stage (at a cost $c(H)$). Then a random set A of terminal pairs is sampled according to the given distribution, i.e., $\{s, t\} \in \mathcal{D}$ with probability p_{st} independent of other pairs. Now, in the second stage the algorithm must augment H with a subgraph $H_{\mathcal{D}}$ such that $H \cup H_{\mathcal{D}}$ satisfies the connectivity requirement of the actual demand $\mathcal{D}$, i.e., for all $\{s, t\} \in \mathcal{D}$, $H \cup H_{\mathcal{D}}$ must support r_{st} edge-disjoint paths between s and t. Furthermore, the edges in the recourse stage are each more expensive by a factor of σ. The objective function is then to minimize the total expected cost of the algorithm, i.e., $c(H) + \mathbb{E}_{\mathcal{D}}[\sigma \cdot c(H_{\mathcal{D}})]$.

Main Results

We show the first non-trivial approximation algorithms for the two-stage Stochastic Survivable Network Problem, with poly-logarithmic approximation ratios of $\widetilde{O}(r_{\max} \log^3 n)$, where $r_{\max}$ is the largest connectivity requirement of any demand pair, and n is the number of vertices of the graph. Along the way, we also obtain the first non-trivial *online algorithms* (with similar competitive ratios) for network design problems with connectivity requirements greater than 1. Finally, for the case when the edge costs satisfy the metric property, we can show $O(1)$-approximations for the stochastic problem. These results are presented in Chapter 2.

Techniques

At a high level, our algorithm attempts to leverage techniques which are used to convert algorithms for the deterministic instances to those which work the stochastic instance. Indeed, the seminal work of Gupta et al. [64] shows the following result, stated informally. Suppose an algorithm for the deterministic instance can *apportion* its total solution cost (i.e., assign the so-called *cost shares*) among the different vertices of the demand set $\mathcal{D}$, such that it is always possible, for any $v \in \mathcal{D}$, to augment the algorithm's solution on input $\mathcal{D} \setminus \{v\}$ using a cheap solution of cost at most the "cost share of v". Then, they provide a black-box procedure to use such an algorithm for handling uncertain input demands. Intuitively, such a scheme assigns, to each demand, the fraction of the total cost of the algorithm's solution, that is devoted to satisfying that particular demand.

Such good cost-sharing algorithms are known [64] for some simple network design problems such as the Steiner Tree problem and the Steiner Forest problem. However, we don't know of such schemes for network design problems with *higher connectivity* requirements. Indeed, our main contribution in this area is in developing a new algorithm for the deterministic survivable network problem which yields good cost-sharing schemes. In fact, as mentioned earlier, our techniques for the general setting in fact yield the first online algorithms for general network design problems with higher connectivity. Moreover, our guarantees in the online setting are identical to those we get for the stochastic setting. Admittedly, this goes against our original motivation for studying stochastic algorithms. However, we can show much better results in case the edge costs satisfy the metric property — here, we can indeed obtain a tangible improvement in the stochastic setting, over the guarantees of any online algorithm.

1.6.2 General Stochastic Knapsack

We next study some scheduling problems, from the point of view of adaptive stochastic optimization. Our first results pertain to the basic problem of scheduling a Knapsack: we are given a Knapsack of total budget B and a collection of n stochastic items/jobs. For any item $i \in [1, n]$, we are given a probability distribution over $(\mathsf{size}, \mathsf{reward})$ pairs (refer to Section 1.7.1 for how this distribution is specified in the input). These random $(\mathsf{size}, \mathsf{reward})$ pairs for two different items are still independent of each other.

In general, an adaptive algorithm can take the following actions at the end of each timestep;

(i) an item may complete at a certain size (giving us the corresponding correlated reward), and the algorithm may choose a new item to start, or

(ii) the algorithm may decide to *cancel* the current item; in this case, no reward is obtained from this item and the item is said to be incomplete[2], or

(iii) the Knapsack budget of B is reached, at which point the algorithm stops, and the item being processed does not fetch any reward.

The objective is to maximize the total expected reward obtained from all completed items. Recall that the algorithm can adaptively decide to run/cancel jobs based on how the prior randomness has instantiated.

Main Results

Our main result is to show a constant-factor *non-adaptive* approximation algorithm for the general Stochastic Knapsack problem. A by-product of this result is the constant-factor bound on the *adaptivity gap* of this problem. We also show constant-factor algorithms for the different cases where preemptions are/aren't allowed, and when job cancellations are/aren't allowed. Prior algorithms [19, 41] only give algorithms for the setting when there are no cancellations, preemptions, and most importantly, no correlations between the rewards and sizes for jobs. There are very simple (and practical) examples where such assumptions are not valid. These results are presented in Chapter 3.

Techniques

Our high-level approach is to derive a linear programming formulation for the Knapsack problem (using the expected values of the jobs), and show that (a) an optimal adaptive solution for the Stochastic Knapsack problem can be "embedded" into the LP, i.e., there is a feasible LP solution with value at least that of the optimal solution for the stochastic instance, and (b) any fractional LP solution can be "rounded" into an integral solution (which can subsequently be interpreted as a non-adaptive schedule) with only a constant-factor loss in terms of expected total reward in both steps. A typical manner in which step (a) can be established is by showing that the marginal probabilities of OPT placing the items are a feasible fractional solution for the LP formulation.

The above framework is in fact a general theme of all our results on the adaptive stochastic optimization setting (in Chapters 3- 5). Since the adaptive stochastic optimization problems have a very large solution space, we come up with a *surrogate optimization problem* (from which we can infer solutions which have more structure), and perform the two steps (a) and (b) mentioned above: (a) is an embedding step that shows that even the more restrictive surrogate problem has a solution of value comparable to the optimal adaptive solution for the stochastic instance, and (b) is a decoding step from which we recover a simple solution to the stochastic instance with a large objective value.

[2]Formally, in this model, if a item is canceled it cannot be resumed at a later time, i.e., we are not allowing preemptions. However, we can also extend our analysis to allow preemptions.

However, standard relaxations for the Knapsack problem (and even those for some more general problems such as MAB) fail to handle correlations between rewards and sizes of items. We therefore introduce a new *time-indexed LP formulation*, which includes a family of partial knapsack constraints for all prefixes of $\{1, 2, \ldots, B\}$. While these constraints become redundant for the deterministic Knapsack problem and even the uncorrelated Stochastic Knapsack problem, they play a crucial role in the correlated version.

1.6.3 Multi-Armed Bandits

We next study a substantial generalization of the Stochastic Knapsack problem, where each item/job is characterized my a more complex state space which we denote by an *arm*. In this Multi-Armed Bandits problem (denoted by MAB), there are n arms: arm i has a collection of states denoted by $\mathcal{S}_i$, a starting state $\rho_i \in \mathcal{S}_i$; Without loss of generality, we assume that $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for $i \neq j$. Each arm also has a *transition graph* T_i, which is given as a polynomial-size (weighted) directed graph rooted at ρ_i. If there is an edge $u \to v$ in T_i, then the edge weight $p_{u,v}$ denotes the probability of making a transition from u to v if we "play" arm i when its current state is node u; hence $\sum_{v:(u,v)\in T_i} p_{u,v} = 1$. We can hence view the transition graph as a Markov Chain.

Each of the arms starts at the start state $\rho_i \in \mathcal{S}_i$. At any time instant, the algorithm observes the states of each arm, and must choose one arm to play. If it plays an arm when it is at a state u, the arm changes its state by making a random transition as dictated by its outward edges (and their probabilities), and the algorithm gets the reward of the (random) state that was reached. The goal is to maximize the total expected reward, while making at most B plays across all arms. We note that while our main focus will be on the additive form of the problem, our general framework can handle other problems (like the explore/exploit kind) as well.

It is easy to see that the Stochastic Knapsack problem considered in the previous section is a special case where each item i corresponds to an arm with the following Markov transition graph. The figure below illustrates such a reduction for a job which has the following distribution: it has (size 1, reward r_1) w.p $1/2$, (size 3, reward r_3) w.p $1/4$, and (size 4, reward r_4) w.p $1/4$.

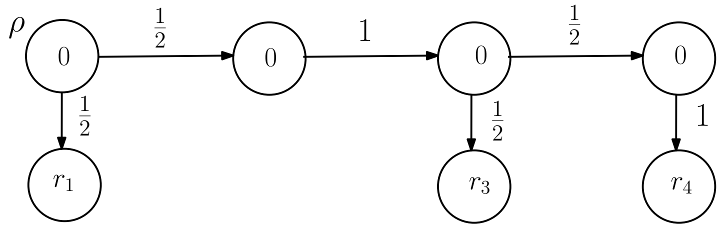


Figure 1.1: Reducing Correlated Stochastic Knapsack to MAB

Main Results

We show a constant-factor approximation algorithm for the general MAB problem. Earlier results [55, 60] show constant-factor algorithms under a strong *Martingaleness* assumption that the rewards of each Markov chain behaves like a Martingale process. That is, the reward of a state is equal to the expected reward of the (random) state after a single pull of the arm. These results are presented in Chapter 4.

Techniques

Our techniques build on those we developed for handling the correlated Stochastic Knapsack problem. As mentioned in Section 1.6.2, our high level idea is to come up with a surrogate problem using which we can decode a solution of better structure for the original problem. Moreover, our surrogate problem is again a suitable time-indexed LP formulation for which we show (a) that LP formulation has a feasible fractional solution of value comparable to the optimal adaptive solution for the stochastic instance, and (b) is a decoding step from which we recover a simple solution to the stochastic instance with a large objective value, by a randomized rounding algorithm.

The crucial difference, though, is that our rounding algorithm for the general MAB problem is *adaptive* which is in contrast to that for the Stochastic Knapsack problem, and also most earlier algorithms for the bandit problems [55, 60]. On the flip side, we show that there can be instances of the MAB problem with arbitrarily large gaps between adaptive and non-adaptive solutions, in essence showing that adaptivity is unavoidable for the general problem.

1.6.4 Stochastic Orienteering

The final problem we consider is another generalization of the Stochastic Knapsack problem, although along a different direction. In the *Stochastic Orienteering* problem, we are given an underlying metric space (V, d) with ground set $|V| = n$ and symmetric integer distances $d : V \times V \to \mathbb{Z}^+$ (satisfying the triangle inequality) that represent travel times between the vertices. Each vertex $v \in V$ is associated with a unique stochastic job, which we also call v. Each job v now has a fixed reward $r_v \in \mathbb{Z}_{\geq 0}$ and a *random processing time* (a.k.a. size) size_v, which is distributed according to a known but arbitrary probability distribution $\pi_v : \mathbb{R}^+ \to [0, 1]$. We are also given a starting "root" vertex ρ, and a budget B on the total time available.

The only actions allowed to an algorithm are to travel to a vertex v and begin processing the job there: when the job finishes after its random length size_v of time, we get the reward r_v (so long as the total time elapsed, i.e., travel time plus processing time, is at most B), and we can then move to the next job. The objective is to find a tour which maximizes the total expected reward of jobs which are completely processed within the budget of B (i.e. traveling times plus processing times must be within B).

Main Results

Our main result is to show an $O(\log \log B)$-approximation algorithm to the optimal adaptive policy[3]. In fact, we show this by existentially showing an upper bound of $O(\log \log B)$ on the adaptivity gap, and then subsequently designing a constant-factor approximation algorithm for the best non-adaptive optimal solution. Our algorithms also extend to the setting where cancellations are permitted, wherein the algorithm can prematurely abort a job. We do not, however, tackle the case of *preemptions* — where an algorithm may stop a job, visit a different job, and then re-visit the first job and resume processing. These results are presented in Chapter 5.

[3]We assume that all metric distances and job sizes are integer multiples of 1.

Techniques

Unlike the Stochastic Knapsack problem, we cannot immediately proceed via an LP-based approach to embed an adaptive optimal solution and subsequently round it, because we are not aware of any good LP-relaxation for even the deterministic orienteering problem. Our main idea to circumvent this, is to create a non LP-based surrogate deterministic problem (with appropriately chosen "average" sizes and rewards for items) such that (i) it has a feasible solution with reward at least OPT, and (ii) we can convert any feasible solution for the surrogate problem into a non-adaptive tour for the Stochastic Orienteering problem with good expected profit.

The advantage LP surrogates provide is that we can show (i) in a fairly procedural manner, by showing that the marginal probabilities of an optimal stochastic solution (which can be viewed as a stochastic process) form a feasible fractional solution to the LP. However, such a proof technique does not carry over to our case, since we need to exhibit a *feasible integral solution* to our deterministic surrogate problem. Indeed, this is the technical heart of our contribution, for which we use a martingale-based analysis on a carefully chosen sequence of random variables pertaining to the optimal stochastic process.

1.7 Some Notes on the Models Studied

1.7.1 Specifying the Uncertainty

An important aspect which we have not yet explained is how the distribution is given as input to the algorithm. Typical models for this include listing the possible scenarios along with their probabilities (which is a viable option only if the number of them is polynomially bounded), or assuming that there is a black-box oracle from which the algorithm can draw samples. Yet another model model is to assume that each vertex has some individual probability of requiring connectivity, and these are independent across vertices, i.e., the probability distribution can be expressed as a product distribution.

We will assume that the input distribution is a *product distribution* over (most of) the elements of uncertainty. This assumption not only makes the input description easy, but also helps significantly with designing the algorithms. The following two paragraphs delve into some details for the two kinds of problems we study in this thesis.

Stochastic Network Design. In the problems on stochastic network design, we will assume that the demands in a network design problem are a probability distribution over vertices. That is, for every vertex we have a probability p_v of it requiring connectivity to a given root. The connectivity requirements are specified up-front, and we assume they are not a part of the uncertain component of the input.

Stochastic Scheduling. For these problems, we will assume that the uncertainty is independent across different jobs, but that the parameters *may be correlated within a job.* For example, a job may have its random variables for the size and profit as being correlated in an arbitrary fashion. Or each job can have a more complex state space (say a Markov Chain), and for each unit of processing it receives, it makes a random transition according to its Markovian process.

Since we allow *arbitrary distributions* over job sizes and rewards, it might be unreasonable to expect the entire distribution to be specified in the input. Indeed, for the simple scheduling problems, our algorithms would only require $\mathrm{poly}(\log B, n)$-attributes about the job distributions. These attributes are of the form $\mathbb{E}_{s_j \sim \pi_j}[\min(s_j, T)]$ for $T \in \mathcal{T}$, where $\mathcal{T}$ is a fixed set of polynomially many values. Here π_j denotes the distribution of job j, and s_j is the random size it assumes. Our algorithms require such attributes for the size and reward of each job.

One of the main objectives of this thesis is in designing techniques to handle *correlations* in the uncertain part of the input data. We believe that our results are a step in this direction, and can potentially lead to more general algorithmic design techniques in achieving this goal.

1.7.2 Computational Intractability

All the problems we study are computationally *at least as hard* as their deterministic counterparts. This is because we allow arbitrary distributions which easily capture deterministic instances. Indeed, if the distribution over demand vertices in the network design problems are $p_v \in \{0, 1\}$, then it captures the underlying deterministic network design problem. Likewise, if the job-size distribution in a stochastic scheduling problem has support at a single value, then it captures the underlying deterministic problem. Combining this observation with the fact that the underlying deterministic problems we study are all NP-hard (and mostly APX-hard as well), motivates our goal of designing *approximation algorithms* for the stochastic problems. Notice that, from a modeling perspective, this is different from *average-case analysis* or *smoothed analysis*, where the randomness assumptions are made to make the problem easier than its deterministic counterpart. The difference here is that in our models, we allow arbitrary distributions on the uncertain input, whereas in the average-case (or smoothed) analysis, these distributions obey some nice properties.

1.7.3 Quality of Solutions

In both the two-stage and the adaptive stochastic optimization models, we compare the (expected) performance of our algorithm to that of the optimal stochastic algorithm, which also has to discover the uncertainty in the random variables only over time (e.g., as and when it completes processing the different jobs, in the scheduling example). Once again, this makes our comparison purely one of computational complexity, as both algorithms (ours and the optimal) have the same information, or entropy, about the input. This is in contrast to the online optimization setting where the algorithm's solution is compared to the best-in-hindsight solution.

1.7.4 Comparison to Competitive Ratio

While our problems are computationally at least as hard as the deterministic counterparts, they are often "easier" (at an intuitive level) than their online versions. That is, the approximation guarantees possible for the stochastic versions are typically better than the optimal competitive ratio of any algorithm for the corresponding online version of the problem. At a high level, this is because the performance guarantee of our algorithms are measured as the ratio of the expected cost that our algorithm obtains to the expected cost that an optimal algorithm obtains, where the corresponding expectations are taken over the randomness associated with the input and the random coin tosses of the respective algorithm. This is in contrast to the competitive ratio of the algorithm, which compares the cost of our algorithm with that of the best minimum cost solution for the instance which was revealed.

1.8 Roadmap

We first consider the two-stage stochastic optimization setting, and present our results for the Stochastic Survivable Network Design problem in Chapter 2. We next move on to the adaptive stochastic optimization model, and focus on the Stochastic Knapsack problem in Chapter 3. Then in Chapter 4, we move on to the more general MAB problem of scheduling bandits in a finite-horizon setting. Finally, we consider the Stochastic Orienteering problem in Chapter 5. We then present a few concluding remarks and mention some interesting avenues of future work in Chapter 6.

1.9 Acknowledgments

The results in Chapter 2 are based on joint work with Anupam Gupta and R. Ravi and appear in [66] (STOC 2009, SICOMP special issue). The results in Chapters 3 and 4 are based on joint work with Anupam Gupta, Marco Molinaro, and R. Ravi and appear in [69] (FOCS 2011). Finally, the results on the orienteering problem in Chapter 5 are based on joint work with Anupam Gupta, Viswanath Nagarajan, and R. Ravi, and appear in [72] (SODA 2012). I would like to thank the aforementioned co-authors for graciously allowing me to include the results in this thesis.

Chapter 2

Stochastic Survivable Network Design

In this chapter, we consider the edge-connectivity version of the *survivable network design* problem (SNDP). Let us first recap the basic SNDP problem in the deterministic setting: we are given a graph $G = (V, E)$ with non-negative edge-costs $c : E \to \mathbb{R}^+$, and edge-connectivity requirements $r_{st} \in \mathbb{Z}_{\geq 0}$ for every pair of vertices $s, t \in V$. The goal is to find a subgraph $H = (V, E')$ with minimum cost $c(E')$ such that H contains r_{st} *edge-disjoint paths* between s and t. Here, the cost of subgraph H is $c(H) := \sum_{e \in E'} c(e)$.

The problem is of much interest in the network design community, since it seeks to build graphs which are resilient to edge failures. Moreover, it has also received significant attention from the theoretical computer science community as well. Since the problem contains the Steiner tree problem as a special case (when $r_{ij} = 1$ for every pair of terminal vertices), the general problem is NP-hard (and also APX-hard), and therefore it has been widely studied through the approximation algorithms lens. The interested reader may refer to [63] for a survey of results and techniques.

Owing to the appealing nature of the problem, both from the theoretical and practical perspectives, the problem has served as a test-bed for several new algorithmic techniques. For instance, some of the earliest applications of the primal-dual method in this area have been to the SNDP problem. There was a sequence of papers applying this technique, eventually leading to the development of an $O(\log r_{\max})$-approximation algorithm [57]. Subsequently, in his breakthrough work, Jain applied the technique of *iterative LP rounding* to this problem to obtain a substantially improved 2-approximation algorithm effectively settling the approximability of the problem (up to constant factors) [77]. However, all these techniques appeal to and crucially use the structural properties that global graph connectivity provides, e.g., the sub-modularity of the cut function. The work presented in this chapter deals with algorithm design for network connectivity, in the presence of uncertain connectivity requirements.

More formally, we extend the study of Survivable Network Design problems in two different directions — the online setting, and the two-stage stochastic optimization setting.

Online SNDP. First, we study this problem in the *online* setting: we are given a graph with edge costs, and an upper bound $r_{\max}$ on the connectivity demand. Now a sequence of vertex pairs

$\{s,t\} \in V \times V$ is presented to us over time, each with some edge-connectivity demand r_{st}—at this point we may need to buy some edges to ensure that all the edges bought by the algorithm provide an edge-connectivity of r_{st} between vertices s and t. The goal is to remain competitive with the optimal offline solution of the current demand set, i.e., minimize the worst-case ratio of the cost of the online algorithm to that of the optimal solution (with the demand sequence known), where the worst-case is taken over all possible input demand sequences.

Stochastic SNDP. Secondly, we extend the online algorithm to the 2-stage stochastic version of the problem with independent demand arrivals, i.e., each pair of vertices $\{s,t\} \in V \times V$ has an associated probability p_{st} of requiring r_{st} edge-connectivity (i.e., the final solution must have r_{st} edge-disjoint paths between s and t), where p_{st} and r_{st} are both given as input to our algorithm. As mentioned earlier, the algorithm can buy a subgraph H in the first stage, after which the exact set $\mathcal{D}$ of terminal pairs requiring connectivity is sampled. Then in the second stage, the algorithm must augment H with a subgraph $H_{\mathcal{D}}$ such that $H \cup H_{\mathcal{D}}$ has r_{st} edge-disjoint paths for all $\{s,t\} \in \mathcal{D}$. The goal is to minimize $c(H) + \sigma \mathbb{E}_{\mathcal{D}}\left[c(H_{\mathcal{D}})\right]$

2.1 Our Results

Our first result is for the online version of the problem.

Theorem 2.1.1 *For the edge-connected Survivable Network Design problem, there is an $\alpha = O(r_{\max} \log^3 n)$-competitive randomized online algorithm against oblivious adversaries. The running time of this algorithm is $O(n^{O(r_{\max})})$, which is polynomially bounded for constant values of $r_{\max}$.*

By combining this with the technique of *Boosted Sampling* [64], we can immediately get approximation algorithms with a similar ratio for the 2-stage stochastic version of the problem as well. This gives us the following result.

Theorem 2.1.2 *For the two-stage stochastic version of the edge-connected Survivable Network Design problem (with independent demand arrivals), there is an $O(r_{\max} \log^3 n)$-approximation algorithm.*

As mentioned earlier, the above result (which has the same performance guarantee as the online algorithm in Theorem 2.1.1) does not do justice to our motivation for studying these problems from a stochastic setting. Our final result addresses this concern for the special case when the input graph is complete, and the edge-costs satisfy the triangle inequality, where we give a constant-factor approximation algorithm for the stochastic version of the problem. Notice that this is in stark contrast with the $\Omega(\log n)$-lower bounds on the competitive ratio of any online algorithm for even the basic Steiner tree problem [74].

Theorem 2.1.3 *For the two-stage stochastic version of the edge-connected Survivable Network Design problem (with independent demand arrivals), there is a constant-factor approximation algorithm if the input graph is complete and the edge costs $c(\cdot)$ satisfy the triangle inequality.*

2.2 Related Work

Steiner network problems have received considerable attention in approximation algorithms: Agrawal et al. [3] and Goemans and Williamson [56] used primal-dual methods to design approximation algorithms for Steiner forests and other 1-connectivity problems (also some higher connectivity problems where multiple copies of edges could be used). Klein and Ravi [80] gave an algorithm for the 2-connectivity problem, which was extended by Williamson et al. [98] and Goemans et al. [57] to higher connectivity problems, yielding $O(\log k)$-approximation algorithms for k-connectivity, all using primal-dual methods. Jain [77] gives an iterative rounding technique to obtain a 2-approximation algorithm for the most general problem of SNDP. These techniques have recently been employed to obtain tight results (assuming $P \neq NP$) for network design with degree constraints [9, 86, 87]. Vertex connectivity problems are less well-understood: [36, 45, 83] consider problems of *spanning* k-connectivity, and provide approximation algorithms with varying guarantees depending on k. Fleischer et al. [48] give a 2-approximation for vertex connectivity when all $r_{ij} \in \{0, 1, 2\}$. Recently, improved approximation algorithms have been given for the problem of *single source* k-vertex connectivity [25, 28], culminating in a simple greedy $O(k \log n)$ algorithm [37]. In fact, the papers [28, 37] also implicitly give $O(k)$-strict cost-shares for the single-source vertex-connectivity problem. As far as we can see, their techniques do not apply in the case of general Survivable Network Design where vertex pairs do not share a common root, nor do they imply online algorithms with adversarial inputs. When the edges have metric costs, there are, quite expectedly, better approximation algorithms for vertex connectivity. Khuller and Raghavachari [79] gave $O(1)$-approximations for k-vertex-connected spanning subgraphs. Cheriyan and Vetta [35] later gave $O(1)$-approximations for the single source k-connected problem and a $O(\log r_{max})$-approximation for metric vertex-connected SNDP. Recently, Chan et al. [26] give constant factor approximations for several degree bounded problems on metric graphs. As for the inapproximability, Kortsarz et al. [84] give $2^{\log^{1-\varepsilon} n}$ hardness results for the vertex-connected Survivable Network Design problem.

Imase and Waxman [74] first considered the online Steiner tree problem and gave a tight $\Theta(\log |\mathcal{D}|)$-competitive algorithm. Awerbuch, Azar and Bartal [6] generalized these results for the online Steiner forest problem, and subsequently Berman and Coulston [13] gave the same $\Theta(\log |\mathcal{D}|)$ guarantee. However, we do not see how to use these ideas for the general problem with higher connectivity. In fact, to the best of our knowledge, no online algorithms were previously known for this problem even for the online rooted 2-connectivity problem (i.e., for the case where all the vertex pairs share a root vertex r and the connectivity requirement is 2 for all pairs)—in fact, we also show a lower bound of $\Omega(\min\{|\mathcal{D}|, \log n\})$ on the competitive ratio for this special case, where $\mathcal{D}$ is the set of terminal pairs given to the algorithm. This is in contrast to the case of online 1-connectivity (i.e., online Steiner forest) where the best online algorithm is $\Theta(\log |\mathcal{D}|)$-competitive [13].

We use the results of Alon et al. [4] for the online (weighted) set cover problem as a subroutine in our online algorithm; the ideas used in that paper have subsequently also been extended by Alon et al. [5] and Buchbinder and Naor [22] to get online primal-dual based algorithms for *fractional* generalized network design. We remark that while we can solve the fractional version of the online k-connectivity problems using these techniques, we do not know how to round this fractional solution online. Please see Section 2.4.2 for a more detailed discussion on this topic.

As for the stochastic version of the problem, the only previous results known for these versions of higher-connectivity problems were $O(1)$-strict *cost-shares* implicitly given by Chuzhoy and Khanna [37], and independently (but explicitly) by Chekuri et al. [32] for the special case of *rooted* connectivity, where all pairs seek k-connectivity to a single source r (and hence to each other). The use of strict cost-shares to get algorithms for rent-or-buy network design appears in [65]. Approximation algorithms for two-stage stochastic problems were studied in [75, 93], and some general techniques were given by [64, 96]; in particular, using strict cost-shares to obtain approximation algorithms for stochastic optimization problems appears in [64].

2.3 Chapter Roadmap

We begin with our discussion on online algorithms for SNDP: we first motivate our algorithm by discussing various approaches and why they fail in Section 2.4.2. We then describe our embedding-based primitive in Section 2.4.3, followed by a quick sketch of why it helps for the case of $k = 2$ connectivity in Section 2.4.4. We then move on to the details of our actual algorithm in Sections 2.4.5-2.4.7. Next we show how these imply good algorithms for the stochastic version of the problem as well in 2.5. Finally, in Sections 2.6 and 2.7, we consider the special case when the graph is complete, and the edge-costs satisfy the triangle inequality to present algorithms with improved guarantees.

2.4 Online SNDP on General Graphs

In this section, we focus our attention on designing good online algorithms for the SNDP problem. For the remainder of the section, we assume that all requirements $r_{st} \in \{0, k\}$ for some constant value k. This is done just for simplicity of the proofs, and our algorithm carries through unchanged for general values of r_{st} as well. Before we presenting our algorithm, we begin with some notation, and then motivate our algorithm by first presenting a few bad examples and failed algorithmic attempts.

2.4.1 Preliminaries

The kECND Problem

As mentioned above, we will present our results in the form of the *k-edge-connected network design problem* (kECND), which is Survivable Network Design where $r_{st} \in \{0, k\}$.

Notation

Consider the kECND problem, and let $\mathcal{D} \subseteq \binom{V}{2}$ denote the set of demand pairs that require k-connectivity (for the online problem, this will be the final set of demand pairs which arrive over time). For the rest of the paper, we shall refer to demand pairs by using curly brackets, e.g., $\{s, t\}$, and use the regular brackets to denote edges following standard convention, like (u, v). Finally, we will use s_i-t_i to denote the i^{th} demand pair in the set $\mathcal{D}$. These can be ordered arbitrarily for the deterministic or stochastic problems, and are ordered according to their arrival in the online version.

2.4.2 Our Techniques and their Motivation

To motivate our algorithms, let us first survey the known techniques for solving the online Steiner tree problem (the special case when $k = 1$). Indeed, there are three broad categories of algorithmic techniques which we can use.

Greedy Algorithms

For the online Steiner tree problem, there is a very elegant and easy to state greedy algorithm: when a new terminal arrives, it simply buys the shortest path to the currently built Steiner tree solution. One of the classical results in online algorithms literature is that this simple algorithm is $\Theta(\log n)$-competitive for the online Steiner tree problem. Moreover, this is the best competitive ratio possible, since there is a matching lower bound of $\Omega(\log n)$ on the competitive ratio of any online algorithm. See [74] for more details on these constructions. One could wonder how such a greedy algorithm would perform for the case of higher connectivity? In the following example, we show that it can be pretty bad, owing to the fact that 2-edge-connectivity is a more global property than 1-edge-connectivity. Indeed, the high-level intuition is that, if there are multiple ways of choosing the second path from a terminal to the root, it is not always optimal to choose the shortest one.

Consider the following *cycle graph* on n vertices, numbered $v_1, v_2, \ldots, v_n$ in clockwise fashion. Each of the cycle edges $e_i := (v_i, v_{(i+1) \mod n})$ has a cost of 1, for all $1 \leq i \leq n$. Additionally, there are private edges $f_i := (v_i, v_1)$ of cost $i-2-\varepsilon$ for $2 \leq i \leq n$. In the online instance, suppose the terminals all require 2-edge-connectivity to the root v_1, and arrive in the order $v_2, v_3, \ldots, v_n$. We now argue that the greedy algorithm will include all the private edges in its solution, whereas the optimal solution is to include just the base cycle! It is easy to see that the cost of the former solution is $\Omega(n^2)$ whereas the cost of the latter solution is n.

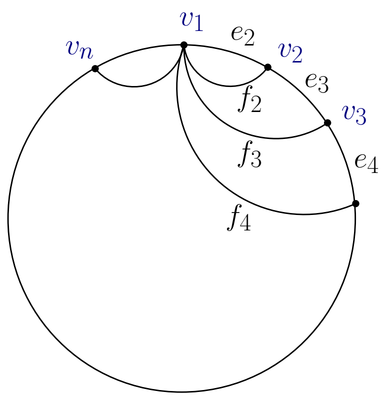


Figure 2.1: Bad example for Greedy algorithm

Indeed, when terminal v_2 arrives, among all choices the greedy algorithm has to 2-edge-connect v_2 to v_1, the cheapest way would be to choose the edges e_2 and f_2. Now, when terminal

v_3 arrives, it is easy to see that *by collapsing* v_2 *and* v_1, the instance is identical to the original one (but with one fewer vertex), and so the greedy algorithm chooses the edges e_3 and f_3 as its augmentation edges, and so on.

The main difficulty with greedy algorithms is that, in 2-edge-connectivity, there is much more global structure to solutions than there was for 1-connectivity. Indeed, it seems difficult to come up with a greedy policy that would make the first terminal v_2 purchase the entire base cycle, instead of the short private cycle. E.g., trying to maximize the number of other vertices covered is no-good as well, because those vertices may never appear as terminals subsequently!

Lower Bound for Online Algorithms. In fact, we show that, if $\mathcal{D}$ is the set of demands that have arrived till some point, then there are instances where the competitive ratio of any online algorithm is $\Omega(|\mathcal{D}|)$ for $|\mathcal{D}| = O(\log n)$, even for the rooted 2-edge connectivity problem. Notice that this in sharp contrast with the case of single-connectivity for which the greedy algorithm is $O(\log |\mathcal{D}|)$-competitive.

Consider the graph given in Figure 2.2. There is a binary tree of depth L, and all the leaves are connected to the root with distinct *"back" edges.* All edges in this graph have unit cost. For ease of exposition, we will assume that the edges bought when seeing any demand are a minimal set of edges to achieve the connectivity requirement for that demand; any edges not in the minimal set are considered to be bought at the first time they are actually used.

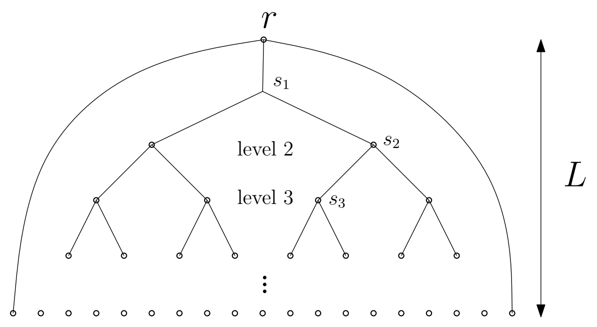


Figure 2.2: A lower bound of $\Omega(|\mathcal{D}|)$

All the requests will be vertices that need 2-connectivity to the root r. The first request is level-1 vertex s_1; one feasible solution is to buy the edge s_1-r, and the second path is some path from s_1 to a leaf and back to r using a "back" edge. However, this is not the only minimal solution possible: perhaps the algorithm can buy two disjoint paths from s_1 to two leaves which use their back edges to connect to r. In any case, there will be at least one vertex on level 3 that is not yet connected to the root. We then give any such vertex on level 3 as the next request. The third request will be some vertex on level 5 that is a descendant of the second request, and which is not yet connected to the root; in general, the next request is always chosen to be a descendant of the previous demands. This ensures that there is always a feasible solution of cost $L + 1$ for all the demands seen thus far, whereas the online algorithm pays at least $\Omega(L)$ for the first $\Omega(L)$ requests, giving us the claimed lower bound.

Note that this construction also works against oblivious adversaries if we choose a random descendant at level $(2i - 1)$ as the i^{th} request.

LP Rounding

Another approach which works for the online Steiner tree problem is that of (i) solving the fractional LP relaxation online, and (ii) rounding the solution online. Indeed, the work of [5] follows precisely this approach for a range of single-connectivity problems. For higher-connectivity though (even, say, $k = 2$), while step (i) of solving the LP relaxation online can be done while maintaining good competitive ratio, we do not know of ways to round the LP solution *online*. This is because typical online rounding strategies are based on the principle of *independent randomized rounding*. For our problems, however, we don't know of good independent rounding strategies, for much of the same reasons why greedy-style algorithms don't work: there is a lot more global correlation in picking edges than there is for the case of single-connectivity.

Tree Embeddings

Yet another strategy which works for single-connectivity is that of using *random tree embeddings*. In this procedure, we first embed the graph onto a arefully chosen tree (e.g., one which preserves distances between all pairs of vertices in expectation). Then, the problem becomes much easier to solve on trees — in fact, the Steiner tree problem is trivial as there is now a unique path between any terminal and the root. The embedding property of the tree ensures that we can move back and forth between solutions on the original graph and solutions on the tree. However, we again run into difficulty when applying this for higher-connectivity problems. Indeed, we can't simply solve the problem on a tree as trees are not 2-edge-connected objects!

What we do, however, is use the embedding to *alter the edge-costs* on the original graph and get much more structural control on it. To this end, imagine we want to convert the connectivity augmentation problem into a hitting set problem: we are given a subgraph H of G that has l-edge-connected a demand pair s_i-t_i (where $l < k$), and we want to $(l + 1)$-edge-connect them. If we think of the s_i-t_i cuts as sets, then we would like to "hit" all these s_i-t_i cuts with edges. This is clearly doomed, since there are $M = 2^{n-1}$ cuts, and an $O(\log M)$-approximation for hitting set will be useless.

We could do better by noting that each minimal s_i-t_i cut in H is given by only l edges. While this bounds the number of cuts in H by $M = \binom{m}{l}$, the subgraph H might contain only a small fraction of G, and there may be many more cuts in G corresponding to the same cut in H—even an exponential number, and we are back to square one. Alternately, we could try to overcome this by hitting the cuts by *paths* connecting two nodes in H (instead of hitting the cuts by edges in G), but there could be exponentially many such paths, and this seems like another bad idea.

What the results in Section 2.4 show is that this is *not* a bad idea at all if we are slightly careful. Loosely speaking, if we take a random distance-preserving spanning subtree $T \subseteq G$, then we show that we can augment the connectivity using only the fundamental cycles (the cycle formed by any non-tree edge (u, v) along with the tree path between u and v) with respect to this spanning tree T. Interestingly, the (random) distance-preserving property allows us to control the cost of these connectivity augmentations. Furthermore, there are only at most m such fundamental cycles, and this enables us to get a compact hitting set instance. Of course, this high-level view oversimplifies things a bit: read on for the complete details. In Sections 2.4.3-2.4.5 we show how we can hit cuts by a small number of cycles/paths, and then Sections 2.4.6 and 2.4.7 use these ideas to develop our algorithms.

2.4.3 Embedding into Backboned Graphs

One of the major advantages of network design problems which only sought 1-edge-connectivity is that one can embed the underlying metric space into random trees [1, 11, 44, 46], where the problems are easier to (approximately) solve. Such a reduction seems impossible even for 2-edge-connectivity as the problem is trivially infeasible on a tree. However, the simple but crucial observation is to not ignore these ideas, as we show below.

Given a graph $G = (V, E)$ with edge lengths/costs $c(e)$, probabilistically embed it into a *spanning* subtree (which we call the *base tree*) using the results of Elkin et al. and Abraham et al. [1, 44]. Formally, this gives a random spanning tree $T = (V, E_T \subseteq E)$ of G with edge lengths $\widehat{c}_T$, such that for all $x, y \in V$:

1. $\widehat{c}_T(e) = c(e)$ for all edges $e \in E_T$, and hence $d_T(x, y) \geq d_G(x, y)$; and
2. $\mathbb{E}[d_T(x, y)] \leq \widetilde{O}(\log n) \cdot d_G(x, y)$, where d_G is the graph metric according to the edge lengths $c(e)$.

The distance d_T is defined in the obvious way: if $P_T(u, v)$ is the unique u-v path in T, then $d_T(u, v) = \sum_{e \in P_T(u,v)} \widehat{c}_T(e)$.

Now *instead of throwing away non-tree edges,* imagine each non-tree edge $e = (u, v) \in E \setminus E_T$ being given a new weight $\widehat{c}_T(e) = \max\{c(e), d_T(u, v)\}$. This suggests the following definition.

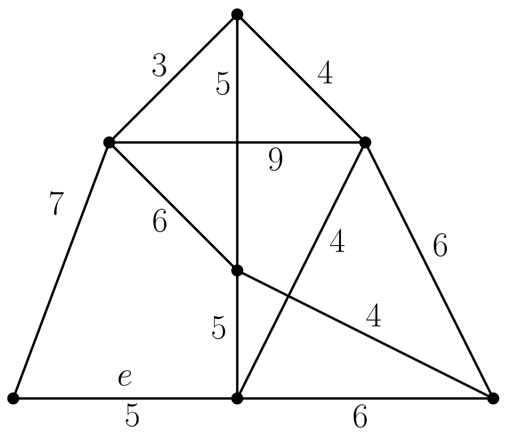
Definition 2.4.1 *A graph $G = (V, E)$ with edge-costs $c : E \to \mathbb{R}$ is called a backboned graph if there exists a spanning tree $T = (V, E_T)$ with $E_T \subseteq E$ such that all edges $e = (u, v) \notin E_T$ have the property that $c(e) \geq d_T(u, v)$. In this case, T is called the base tree of G.*

The figure below illustrates an example of a particular backbone tree.

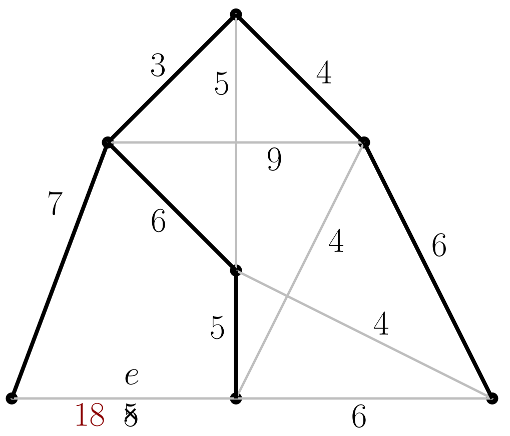
Note that the embeddings of [1, 44] probabilistically embed graphs into a distribution $\mathcal{T}$ over backboned graphs (indexed by their base trees) with small expected stretch, i.e., $\mathbb{E}_{T \sim \mathcal{T}}[\widehat{c}_T(e)] \leq \widetilde{O}(\log n)c(e)$ for all $e \in G$. This show that for any subgraph H, the expected cost $\mathbb{E}_{T \sim \mathcal{T}}[\widehat{c}_T(H)] \leq \widetilde{O}(\log n)c(H)$. Finally, since $\widehat{c}_T(e) \geq c(e)$ for all $e \in E, T \in \mathcal{T}$, we can bound the cost of any subgraph H' w.r.t edge costs c by the corresponding cost $\widehat{c}_T(H')$. We then get the following theorem.

Theorem 2.4.2 *A β-competitive online algorithm for* kECND *on backboned graphs implies a randomized $\beta \times \widetilde{O}(\log n)$-competitive algorithm for* kECND *on general graphs (against oblivious adversaries). Also, β-approximation algorithms for* kECND *on backboned graphs imply randomized $\beta \times \widetilde{O}(\log n)$ approximation algorithms on general graphs.*

Hence, for the subsequent sections (except those for the metric instances) we will assume that the input graph is a backboned graph, and will use its properties to design online and "cost-sharing" approximation algorithms.



(a) Original edge costs



(b) New cost $\widehat{c}(e)$ defined according to tree

Figure 2.3: Illustration of Backbone tree and changed edge costs

2.4.4 Online 2-Edge-Connectivity

As a warm-up, consider the special case of kECND on a backboned graph $G = (V, E)$, when the connectivity requirement is $k = 2$ for all demand pairs, and furthermore, the problem instance is rooted, i.e., all demand pairs are of the form $\{r, t_i\}$ for some fixed root $r \in V$. A natural approach for this problem would be to first 1-edge-connect the root with the terminal which has arrived, and then augment connectivity in the next phase. Because the graph is backboned, it is easy to see that the optimal offline subgraph which just 1-edge-connects a set of terminals $T = \{t_1, t_2, \dots, t_l\}$ with root r is the collection of base tree paths $\cup_{i=1}^{l} P_T(r, t_l)$. Therefore, the online 1-connectivity problem becomes trivial on backboned graphs—when a new terminal t_i arrives, we simply buy the base tree path $P_T(r, t_i)$.

We now see how we can augment edges to 2-connect the terminals with the root, in an online fashion. Consider the stage in the online algorithm when a terminal t_i has arrived. As a first step, like mentioned above, we 1-edge-connect t_i to the root r by buying the path $P_T(r, t_i)$. Now if t_i is not 2-edge-connected to r in the current subgraph, then there must exist a cut-edge $e = (x, y)$ on the path $P_T(r, t_i)$. Removing the edge e also cuts the base tree T into 2 components—call the one containing the root as C_r, and the other containing the terminal t_i as C_{t_i}. Since e is the only tree-edge crossing this cut, there must exist a non-tree edge $f = (u, v)$ in OPT (an optimal offline solution which 2-edge-connects the current set of terminals with r) such that f crosses the cut (C_r, C_{t_i}) (and as a consequence, observe that the edge e would be contained in the base tree path $P_T(u, v)$).

The crucial observation now is the following: if we were to include the *entire cycle* $O_{(u,v)} = P_T(u, v) \cup (u, v)$ to our current subgraph, then e would no longer be a cut-edge separating r from t_i (because there is now an alternate path from x to y in $O_{(u,v)}$). Furthermore, we can use the backbone property of G and in fact charge the cost of the entire cycle to the single edge f that OPT bought.

At a high level, this motivates modeling the augmentation problem as the following online set cover instance, where **(i)** the elements are the tree-edges, **(ii)** the sets S_{uv} correspond to the cycles $O_{(u,v)}$, and **(iii)** an element/tree-edge e is "covered" by a set S_{uv} if and only if $e \in O_{(u,v)}$. By the preceding arguments, we can see that the cost of an optimal offline solution to cover *all* the cut-edges on the paths $P_T(r, t_i)$ is $2\mathsf{OPT}$. Hence, by the polylogarithmic competitiveness of the online set cover algorithm of Alon et al. [4], we would get an online 2-edge-connectivity algorithm with polylogarithmic guarantees. In what follows, we formalize this intuition, and generalize it to the setting of kECND.

2.4.5 A Small Collection of Covering Cycles

In this section, we show how we can augment connectivity (from, say, l to $l+1$) for a demand pair $\{s_i, t_i\}$ by showing that all its minimal cuts can be *covered* by a small collection of fundamental cycles (w.r.t the base tree T) of low cost. For the case when $l = 1$, this is just the set cover instance outlined in the previous section. Before we state our Cut Cover Theorem, we begin with some notation that will be useful for the rest of this section.

Notation: *Base Cycles.* Let G be a backboned graph that is an instance of the kECND problem with demand set $\mathcal{D}$, and let T be the base tree in G. For any edge $e = (u, v) \notin E_T$, define the *base cycle* O_e to be the fundamental cycle $\{e\} \cup P_T(u, v)$ of e with respect to T.

Now, let H be a subgraph which l-edge-connects (for some $l < k$) the vertices s_i and t_i for some demand pair $\{s_i, t_i\} \in \mathcal{D}$, and suppose H also contains the base tree path $P_T(s_i, t_i)$. The l-edge-connectivity assumption implies there are l edge-disjoint paths from s_i to t_i in H: denote this set of edge-disjoint paths by $\mathcal{P}_i$. Clearly, any l-cut (a set of l edges removing which would separate s_i and t_i in H) in H must pick exactly one edge from each path in $\mathcal{P}_i$: we define $\mathsf{viol}_H(i)$ to be the set of all such l-cuts.

Labeling: Consider any cut $Q \in \mathsf{viol}_H(i)$. Since Q is a minimal l-cut for the demand pair s_i-t_i in H, it must be that any end vertex of a cut edge is reachable from one of s_i or t_i in $H \setminus Q$. We label each end vertex v reachable from s_i in $H \setminus Q$ by L (i.e., we set $\mathsf{label}(v) = L$), and each end vertex v reachable from t_i by R (we set $\mathsf{label}(v) = R$). Every other vertex in $V(G)$ has a label U; hence all but at most $2|Q|$ nodes are labeled U, which we denote by a "trivial label". Note that the labeling of the end vertices of a cut Q depends on the subgraph H and not just the set of edges in Q.

Theorem 2.4.3 (Cut Cover Theorem) *Consider a* kECND *instance $\mathcal{I}$, and let* OPT *denote any optimal solution. Let $H \subseteq G$ be any subgraph that l-edge-connects terminal pair $\{s_i, t_i\}$ for some $l < k$, such that the base tree path $P_T(s_i, t_i) \subseteq H$. Then for any l-cut $Q \in \mathsf{viol}_H(i)$, given the labeling of the endpoints of Q as described above, we can find an edge $e = (u, v) \in E(\mathsf{OPT})$ such that $O_e \setminus Q$ connects some L-vertex to some R-vertex. This ensures that s_i and t_i are connected in $(H \cup O_e) \setminus Q$.*

Note that the algorithms in Section 2.4.6 and 2.4.7 depend only on the statement of the above Cut Cover Theorem 2.4.3, so readers strapped for time can jump straight to the algorithms.

Proof Outline. The proof is by contradiction and assumes that there is no edge e s.t the base cycle O_e can cover this cut. We first give the outline of the proof, which would help in following the sequence of Lemmas 2.4.4–2.4.7. Suppose we delete all the edges of some minimal cut $Q \in \mathsf{viol}_H(i)$ from the current subgraph H. Such a cut (in particular, removing the edges $Q \cap T$) will separate the base tree into $|Q \cap T| + 1$ components (denoted by $\mathcal{C}$), with s_i and t_i belonging to different components (see Figure 2.4 for an illustration where the different circles are the tree components).

Firstly, observe that every component $C \in \mathcal{C}$ will have at least one vertex with a non-trivial label (a vertex with a label not U) since it has at least one edge from $Q \cap T$ in its boundary. To get the main intuition behind the proof, let us make the simplifying assumption that each component contains vertices of only one non-trivial label. If a component has its only non-trivial labels as L-vertices, we refer to it as an L-component, and likewise R-components only contain R-vertices.

Now, since OPT can $(l+1)$-edge-connect s_i and t_i, it means that there must exist a path P^* including which would connect these 2 vertices in H. We focus on this path and traverse it edge by edge, starting from s_i. Suppose we are considering the j^{th} edge on this path, and suppose it begins from an L-component. Then, in Lemma 2.4.4, we show that this edge must also end in an L-component—otherwise, its base cycle would cover this cut Q. This then lets us inductively proceed and show that each edge will always terminate in an L-component, since we begin from s_i and it belongs to an L-component. Hence we would never reach t_i (contained in an R-component), which gives us the desired contradiction.

In general, it may not be the case that a component has only L or R-vertices. To handle this, we extract out a subset of edges of the path P^* (called the Canonical Sequence), and argue that the tree edges in $Q \cap T$ which are induced by the base cycles w.r.t the canonical sequence, will satisfy this "consistency" property (Lemma 2.4.6). This is sufficient to push the induction through and we would arrive at the same contradiction. We now present the complete details.

Proof: Consider a cut $Q \in \mathsf{viol}_H(i)$. Note that $Q \cap T \neq \emptyset$, since by our assumption the base tree path $P_T(s_i, t_i) \subseteq H$ and hence the cut Q must contain some edge on it. Let the edges $Q \cap T$ separate the base tree into into $t \leq l+1$ components $\mathcal{C} = \{C_1, C_2, \ldots, C_t\}$. The terminals s_i and t_i must belong to different components: let $C(s_i)$ and $C(t_i)$ denote the components containing them. In general, let $C(v)$ denote the component containing vertex v. A component $C \in \mathcal{C}$ is called a *star component* if it contains some vertex from $P_T(s_i, t_i)$. We refer to the edges in $Q \cap T$ as *portal edges*. For every component $C \neq C(t_i) \in \mathcal{C}$, let the *parent edge* $\mathsf{head}(C)$ be the first portal edge on the base tree path from any vertex in C to t_i; note that the component $C(t_i)$ does not have a parent edge. Also note that for non-star components, $\mathsf{head}(C)$ also happens to be the first portal edge on the base tree path from any vertex in C to s_i.

For example, in Figure 2.4, the dashed edges are the portal edges, $\mathsf{head}(C_2)$ is the portal edge between C_2 and C_1, $\mathsf{head}(C_6)$ is the portal edge between C_6 and C_5, and C_1, C_4, C_5, C_{10} are the star components.

Since each edge in Q belongs to a distinct path in $\mathcal{P}_i$ (Q is a minimal l-cut separating s_i from t_i), the end vertices of any portal edge—and indeed of any edge in Q—have distinct labels from the set $\{L, R\}$. For a portal edge e, say its L-vertex is its unique endpoint labeled L, and its other endpoint is its R-vertex.

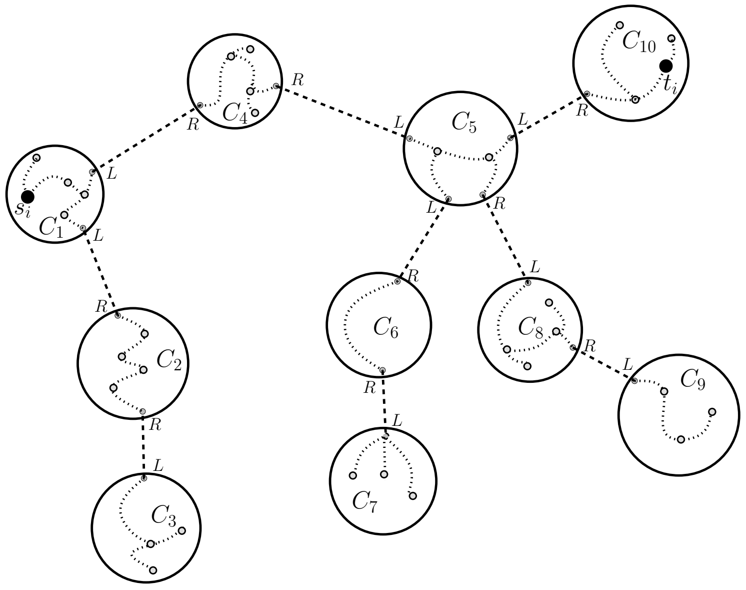


Figure 2.4: Example of the Portal Graph: circles are components, the dashed edges are portal edges, and dotted edges are other base-tree edges.

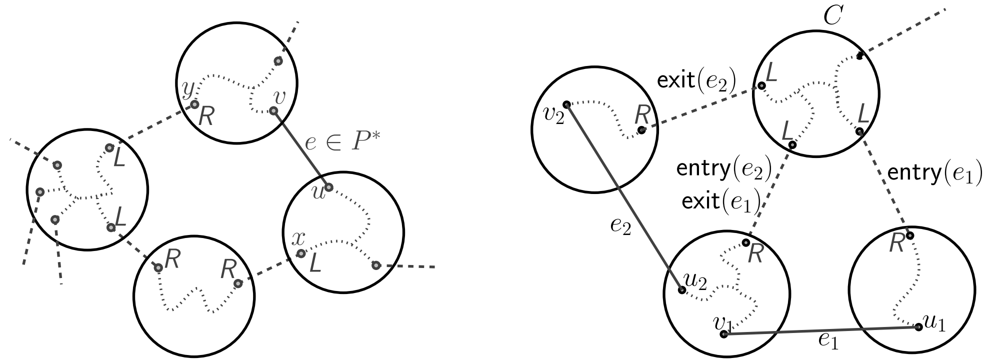
To prove Theorem 2.4.3, we will show that there exists an edge $e = (u, v) \notin Q$ which lies in an optimal solution such that $O_e \setminus Q$ contains a path between an L-vertex and an R-vertex in Q; in turn, this will ensure that s_i and t_i are connected in $(H \cup O_e) \setminus Q$, completing the proof. For the remainder of the proof, an edge (u, v) which satisfies this property is said to *cover* the cut Q.

The Canonical Sequence: Since s_i and t_i can be k-edge-connected in G, there must be a s_i-t_i path P^* *contained in the optimal* kEC *subgraph* with $P^* \cap Q = \emptyset$. We first eliminate some "redundant" edges from P^* and show that among the other edges, there is one that covers Q. First remove all edges from P^* that are internal to some component in $\mathcal{C}$. Now consider a new undirected graph—the *component graph*—whose vertex set is the collection $\mathcal{C}$ of components, and there is an edge (C_i, C_j) when there is an edge $(u, v) \in P^*$ such that $u \in C_i$ and $v \in C_j$. The edges in P^* now correspond to a path (not necessarily simple) between $C(s_i)$ and $C(t_i)$ in the component graph. We then remove edges from P^* that correspond to cycles in the component graph, and are left with a set of edges P^* corresponding to a simple path between $C(s_i)$ and $C(t_i)$ in the component graph. Say the edges of P^* in this order are $\langle e_1 = (u_1, v_1), e_2 = (u_2, v_2), \ldots, e_p = (u_p, v_p)\rangle$. Note that $C(u_i) = C(v_{i-1})$ for $2 \leq i \leq p$; however u_i need not be the same as v_{i-1} in general. We refer to this resulting sequence of edges also as P^* and call it the *canonical* sequence, and all the components $C(u_j)$ the *canonical* components.

For a contradiction, suppose there is no edge $(u, v) \in P^*$ that covers the cut Q. We now prove a set of lemmas about the canonical sequence and the labeling of the portal edges to show that this cannot happen. Recall that each portal edge has different labels from $\{L, R\}$ on its endpoints. When tracing some u-v path in the base tree T, we say some portal edge is crossed with *signature* $(L \to R)$ if the endpoint labeled L is closer to u than to v in the base tree T. Clearly, the signature of the portal edge depends on the starting vertex u and ending vertex v of the path.

Lemma 2.4.4 (Alternating Paths Lemma) *Suppose none of the edges $(u', v') \in P^*$ covers Q. For any edge $(u, v) \in P^*$, if the first portal edge on $P_T(u, v)$ is crossed with signature $(L \to R)$, then the final portal edge on $P_T(u, v)$ is crossed with signature $(R \to L)$. Also, the portal edges crossed along the way have alternating signatures $(L \to R)$, $(R \to L)$, . . ., $(L \to R)$, $(R \to L)$. An analogous statement is true in the case the first portal edge is crossed with signature $(R \to L)$.*

Proof: If the u-v base tree path $P_T(u, v)$ first crosses a portal edge with signature $(L \to R)$ and also ends by crossing a portal edge with signature $(L \to R)$, the base cycle $O_{(u,v)}$ would connect the first L-vertex in $C(u)$ to the final R-vertex in $C(v)$. Moreover, the portions of $O_{(u,v)}$ within $C(u)$ and $C(v)$ are disjoint from Q, and hence $O_{(u,v)} \setminus Q$ would connect these L and R vertices, contradicting the fact that (u, v) does not cover Q. For example, in Figure 2.5(a) we see that x and y would be connected in $O_{(u,v)} \setminus Q$.



(a) Alternating Paths Lemma (b) (u_1, v_1) and (u_2, v_2) transit C

Figure 2.5: Illustrative figures for the proof. Again, the dashed edges are portal edges, dotted edges are other base-tree edges, and solid edges belong to P^*.

Likewise, if the path $P_T(u, v)$ enters some component C through a portal edge signed $(L \to R)$ and also exits C through an $(L \to R)$ edge, the portion of $O_{(u,v)}$ within the component C would connect the entry vertex labeled R and exit vertex labeled L in $O_{(u,v)} \setminus Q$; as a result, including the edges of $O_{(u,v)}$ to H would connect s_i and t_i (even if the edges Q are deleted) by the definition of L and R-vertices. This is also a contradiction, completing the proof. ■

Lemma 2.4.5 (Star-Path Lemma) *Suppose no $(u', v') \in P^*$ covers Q. Consider the portal edges $e'_1, e'_2, \ldots, e'_s$ when traversing the s_i-t_i path $P_T(s_i, t_i)$ on the base tree T. Then the signatures of these edges alternate $(L \to R)$, $(R \to L)$, . . ., $(L \to R)$.*

Proof: If we have two consecutive portal edges e'_j and e'_{j+1} that are signed $(L \to R)$, then we would have an L-vertex and an R-vertex, both of which lie on the path $P_T(s_i, t_i)$, belonging to the same (star) component C. Since we assume that H contains $P_T(s_i, t_i)$, these two vertices would be connected in $H \setminus Q$, thus contradicting the fact that Q is itself a violated cut separating s_i from t_i. ■

Lemma 2.4.6 (Consistency Lemma) *Suppose no $(u', v') \in P^*$ covers Q. Consider a component $C \neq C(t_i)$ such that* $\mathsf{head}(C)$*'s L-vertex belongs to C. Then, for any $(u, v) \in P^*$, if $P_T(u, v)$ intersects the component C, the portal edge $P_T(u, v)$ takes when entering C (if any) has its L-vertex in C. The same is true for the portal edge $P_T(u, v)$ takes when exiting C (if any). An analogous statement holds if* $\mathsf{head}(C)$*'s R-vertex is contained in C.*

Proof: Let $\mathsf{entry}(u, v)$ and $\mathsf{exit}(u, v)$ denote the portal edges used by $P_T(u, v)$ to enter and exit C respectively if we traverse $P_T(u, v)$ *from u to v*. For an edge $(x, y) \in P^*$, we say (x, y) ***transits*** a component C if $P_T(x, y)$ intersects C, but neither x nor y belong to C. (see Figure 2.5(b) for an example.)

We first consider the case when C is not a star component *and* is a canonical component; the proof for the C not being canonical is only simpler and we later consider C being a star component.

Let $\langle e'_1, e'_2, \ldots, e'_a \rangle \subseteq P^*$ be the edges in P^* which transit C (in that order) before some edge $e_1 \in P^*$ has an endpoint in C. (Such an edge e_1 exists because we have assumed that C is a canonical component.) The subsequent edge $e_2 \in P^*$ exits C, and let $\langle e''_1, e''_2, \ldots, e''_b \rangle \subseteq P^*$ be the following edges that transit C. Since C is not a star component, $\mathsf{entry}(e'_1)$ and $\mathsf{exit}(e''_b)$ must be the edge $\mathsf{head}(C)$, which by the assumption of the lemma has its L-vertex in C. This is because, if we shrink all the components and trace the path taken by P^* along the tree formed by just the portal edges in $Q \cap T$, the first time we visit C has to be via its head edge, by the definition of head edges. Likewise, the final edge to visit C must leave along the same head edge, since eventually this path ends up in t_i. Furthermore, by Lemma 2.4.4, $\mathsf{exit}(e'_1)$ must have its L-vertex in C as well.

Moreover, since the base path traversals are all done along the tree T, it is not hard to see that $\mathsf{entry}(e'_{j+1}) = \mathsf{exit}(e'_j)$ for $1 \leq j < a$. Inductively applying the alternating paths lemma, all the portal edges $\mathsf{entry}(e'_j)$ and $\mathsf{exit}(e'_j)$ have their L-vertices in C. Since $\mathsf{entry}(e_1) = \mathsf{exit}(e'_a)$, we also get that $\mathsf{entry}(e_1)$ has its L-vertex in C. The same inductive argument applied starting with e''_b and working backwards shows that the entry and exit edges used by e''_j for all j, and e_2 all have their L-vertices in C. For the case when C is not a canonical component, the argument is only simpler, since we would not have the edges e_1 and e_2 and have only a set of transiting edges.

Finally, when C is a star component, it is no longer true that the edge $\mathsf{entry}(e'_1)$ is the same as $\mathsf{head}(C)$. However, either $C = C(s_i)$ (in which case the proof is the same as above without any edges e'_j or e_1), or else $\mathsf{entry}(e'_1)$ must be the head edge for the previous star component C_{prev} on the s_i-t_i path. Hence, since $\mathsf{head}(C)$ has its L-vertex in C, the Star-Path Lemma 2.4.5 implies that $\mathsf{entry}(e'_1) = \mathsf{head}(C_{prev})$ also has its L-vertex in C. Now the rest of the proof is identical to that above. ∎

Lemma 2.4.7 (Final Component Lemma) *Suppose there is no $(u', v') \in P^*$ that covers Q. For any $(u, v) \in P^*$ such that $P_T(u, v)$ intersects $C(t_i)$, the portal edge taken to enter $C(t_i)$ has its R-vertex in C_{t_i}. The same is the case for the portal edge taken to exit $C(t_i)$, if any.*

Proof: The proof of the lemma is very similar to that for Lemma 2.4.6. Since C_{t_i} is the final component on the path P^*, we would not have the edges of the form e_2 and e''_j (there is only one edge in P^* that has an end vertex in C_{t_i}, since we have eliminated all cycles). Also, because

e'_1 is the first edge in P^* to transit $C(t_i)$, $\mathsf{entry}(e'_1)$ must be the head edge for the previous star component C_{prev} on the s_i-t_i path. From the Star-Path Lemma 2.4.5, we get that $\mathsf{entry}(e'_1)$ has its R-vertex in $C(t_i)$. By Lemma 2.4.4, $\mathsf{exit}(e'_1)$ must have its R-vertex in $C(t_i)$ as well. Like in the previous proof, $\mathsf{entry}(e'_{j+1}) = \mathsf{exit}(e'_j)$ for $1 \leq j < a$. Inductively applying the alternating paths lemma, all the portal edges $\mathsf{entry}(e'_j)$ and $\mathsf{exit}(e'_j)$ have their R-vertices in C. Since $\mathsf{entry}(e_1) = \mathsf{exit}(e'_a)$, we also get that $\mathsf{entry}(e_1)$ has its R-vertex in C. ■

To complete the proof of Theorem 2.4.3, we argue the following.

Lemma 2.4.8 *Suppose no $(u', v') \in P^*$ covers Q. Then for all vertices v_j belonging to P^* (for $1 \leq j \leq p$), we have $C(v_j) \neq C(t_i)$.*

Proof: The proof is an induction on j, for $1 \leq j \leq p$. Since $C(u_1) = C(s_i)$, we know that $\mathsf{head}(C(u_1))$ has its L-vertex in C_{u_1}. By the Alternating Paths Lemma 2.4.6, we know that the final portal edge on $P_T(u_1, v_1)$ must have its L-vertex in $C(v_1)$. This implies that $C(v_1) \neq C(t_i)$, otherwise we would violate the Final Component Lemma 2.4.7. This establishes the base case. Now, since $C(v_1) \neq C(t_i)$, the Consistency Lemma 2.4.6 implies that $\mathsf{head}(C(v_1))$ must have its L-vertex in the component $C(v_1)$. But because $C(u_{j+1}) = C(v_j)$ for all j, we have that $\mathsf{head}(C(u_2))$ must have its L-vertex in $C(u_2)$, and therefore we can proceed inductively. ■

But note that Lemma 2.4.8 implies that we never reach $C(t_i)$ while following the canonical path, which contradicts the fact that P^* corresponds to a path between $C(s_i)$ and $C(t_i)$ in the component graph. This contradiction completes the proof, and hence implies that there must be some edge $(u, v) \in P^*$ that covers the cut Q. ■

2.4.6 Augmentation using Hitting Sets

We now show how we can use the covering property to get a low-cost augmentation. Given an instance $G, c(\cdot)$ of the kECND problem, suppose we have a subgraph H such that all terminal pairs $\{s_i, t_i\}$ are l-edge-connected in H: we now identify *sets* and *elements* such that the Hitting Set problem exactly captures the problem of augmenting H to $(l+1)$-edge-connect every s_i to t_i. Moreover, we want to do this in a way such that the number of sets and elements is small; if we were allowed exponentially many sets, we could imagine each l-cut $(U, V \setminus U)$ that separates s_i from t_i to be a set, and the edges of $G \setminus H$ to be the elements, such that element/edge e belongs to the set/cut $(U, V \setminus U)$ if $e \in \partial U$. But this gives us too many sets, as mentioned in Section 2.4.2.

To do this more efficiently, consider this: we can imagine H already contains the base tree, since it costs at most as much as the optimum kEC solution. Now look at the following hitting set instance $\mathcal{I}_A$: for each violated l-cut $Q \in \mathsf{viol}_H(i)$ for a terminal pair $\{s_i, t_i\}$, we have a *set* in our instance. (Recall that now $Q \subseteq E$ can be just a set of edges.) In case the same set of edges Q separate several terminal pairs, we have a set for each terminal pair. For each edge $e = (u, v)$ in G we have an element. An element/edge e belongs to a set/cut Q if the edge e covers the cut Q—in other words, if the base cycle O_e satisfies the property that $(H \cup O_e) \setminus Q$ connects the terminal pair $\{s_i, t_i\}$. The cost of an element e is simply the cost of the base cycle O_e, which is at most $2c(e)$, by the properties of the backboned graph. Note that in this instance, the number of sets is at most $|\mathcal{D}| \cdot m^l = O(n^2 m^l)$ and the number of elements is at most m. A straightforward consequence of the Cut Cover Theorem 2.4.3 establishes the following:

Theorem 2.4.9 (Augmentation Theorem) *Given an backboned instance $G, c(\cdot)$ of the* kECND *problem, suppose we have a subgraph H containing the base tree such that the terminal pairs $\{s_i, t_i\}$ are l-edge-connected in H, for some $l < k$. Then the instance of the hitting set problem $\mathcal{I}_A$ created above has a solution costing at most $2\,c(\mathsf{OPT})$. Furthermore, if the set of elements/edges bought in a solution to the hitting set instance is F, then the subgraph $(H \cup (\cup_{e \in F} O_e))$ is a network that $(l+1)$-edge-connects every terminal pair $\{s_i, t_i\}$.*

As a warm-up, this shows that we can solve the kECND problem, and more generally the generalized Steiner connectivity problem, on any backboned graph by starting off with the base tree as the 1-edge-connected network, and repeatedly applying Theorem 2.4.9 (and a good approximation algorithm for hitting set) to augment the connectivity from l to $l+1$ at cost $O(\log(n^2 m^l))c(\mathsf{OPT})$. In total, this approach gives us an approximation guarantee of $\sum_l O(l \log m + \log n) = O(r_{\max}^2 \log m + r_{\max} \log n)$. Finally, translating this to general networks loses another almost-logarithmic factor via Theorem 2.4.2. However, we can do better (and even do it online), as we now show.

2.4.7 Online Algorithm using Hitting Sets

To give an online algorithm for kECND, let us consider the above proofs again. When we defined the hitting set instance $\mathcal{I}$ corresponding to the $(l+1)$-augmentation problem, it appeared as if the notion of an element/edge e hitting a set/cut Q depended on the subgraph H. However, this is not the case: recall that the Cut Cover Theorem 2.4.3 showed that for any l-cut $Q \in \mathsf{viol}_H(i)$, there exists an edge $e = (u, v) \in \mathsf{OPT}$ such that $O_e \setminus Q$ connects an L-vertex to an R-vertex. In fact, if we were only given some set of edges $\widehat{Q}$ and some labels on its endpoints, and the theorem gives us an edge e, then this edge e is good for *all* subgraphs H such that (i) $P_T(s_i, t_i) \subseteq H$, (ii) $\widehat{Q}$ is an l-cut separating s_i and t_i in H, and (iii) the labels are indeed the labels we would get given H and $\widehat{Q}$. Moreover, for any cut Q, once we know the L and R labels of its end vertices, we can also identify whether an element (u, v) covers the cut Q. These are the properties we exploit in our online algorithm.

For the online algorithm for backboned graph, we first set up an instance $\mathcal{I}$ of the HITTING SET problem:

- **Universe**. For each edge $e \in E$, we have an element; there are $N = m$ elements. The cost of element e is $c(O_e) \in [c(e), 2c(e)]$.
- **Sets**. For each $l \in \{1, 2, \ldots, (k-1)\}$, we have a collection $\mathcal{F}_l$ of $M_l \leq \binom{m}{l} 2^l$ sets, where each set Q^l is a set of l edges *along* with $\{L, R\}$ labels on the endpoints of these edges. Hence $\mathcal{F} = \cup_l \mathcal{F}_l$ are all the $M := O((2m)^k)$ sets.
- **Incidence**. A element $e = (u, v)$ hits a set Q^l if and only if the subgraph $O_{(u,v)} \setminus Q^l$ connects an L-vertex and an R-vertex in Q^l.

Now when a terminal pair $\{s_i, t_i\}$ arrives, we first buy the edges on s_i-t_i base tree path $P_T(s_i, t_i)$ that have not yet been bought, and then perform a series of $(k-1)$ augmentations. In round l, we feed all the minimal violated cuts with l edges in the current subgraph H along with their $\{L, R\}$ labels to the online hitting set algorithm, which results in a new subgraph with increased connectivity. Note that the deterministic online algorithm for weighted set cover given by Alon et al. [4] would be $O(\log N \log M) = O(k \log^2 m)$-competitive on this hitting set instance as well. Formally, the algorithm is presented below.

Algorithm 1 OnlineAlg($\mathcal{D}$) for online kECND on backboned graphs

```
1: let $H \leftarrow \emptyset$.
2: set up the instance $\mathcal{I}$ of online hitting set.
3: for each terminal pair $\{s_i, t_i\}$ that arrives do
4:     let $H \leftarrow H \cup P_T(s_i, t_i)$
5:     for $l = 1$ to $k - 1$ do
6:         while $\{s_i, t_i\}$ not $l + 1$-edge-connected in $H$ do
7:             find some violated $l$-cut $Q$ between $s_i$ and $t_i$ in $H$ and its labeling w.r.t. $H$
8:             feed ($Q$, labeling) to online hitting set algorithm; let its output be $F \subseteq E$
9:             let $H \leftarrow H \cup (\cup_{e \in F} O_e)$
```

Theorem 2.4.10 *The algorithm* OnlineAlg *is has a competitive ratio of* $O(k \log^2 m)$ *for the* kECND *problem on backboned graphs.*

Proof: The proof essentially reiterates the aforementioned facts. Consider the case where τ terminals have arrived, and let OPT be an optimal offline network k-edge-connecting the demand pairs $\{s_i, t_i\}_{i \leq \tau}$. Clearly, the total cost spent in Step 4 in buying base tree paths is at most $c(\mathsf{OPT})$. Moreover, since for each request we feed the online algorithm, there is an element/edge $e \in \mathsf{OPT}$ that hits it (Theorem 2.4.3), the optimal offline cost to hit all our requests is at most $2c(\mathsf{OPT})$. (The factor 2 arises because we buy O_e with cost at most $2c(e)$, even though OPT may only buy e.) Hence, from the $O(\log M \log N)$-competitiveness of the online hitting set algorithm, we get $O(k \log^2 m)$-competitiveness for our online algorithm. ■

Combining Theorem 2.4.10 with Theorem 2.4.2 (from the discussion in Section 2.4.3), we immediately get:

Corollary 2.4.11 (Result for General Graphs) *There is an* $\widetilde{O}(k \log^2 m \log n)$*-competitive randomized online algorithm for the* kECND *problem on general graphs.*

2.5 Two-Stage Stochastic SNDP On General Graphs

We now briefly explain how online algorithms lead to algorithms in the stochastic setting almost immediately. The main concept towards understanding this is the notion of *strict cost-sharing schemes*. Indeed, the work of Gupta et al. [64] shows the following result, stated informally. Suppose an algorithm for the deterministic instance can *apportion* its total solution cost (i.e., assign the so-called *cost shares*) among the different vertices of the demand set $\mathcal{D}$, such that it is always possible, for any $v \in \mathcal{D}$, to augment the algorithm's solution on input $\mathcal{D} \setminus \{v\}$ using a cheap solution of cost at most the "cost share of v". Then, they provide a black-box procedure to use such an algorithm for handling uncertain input demands. Intuitively, such a scheme assigns, to each demand, the fraction of the total cost of the algorithm's solution, that is devoted to satisfying that particular demand. We now provide the more formal definition.

Strict Cost-Sharing Schemes

An α-approximation algorithm Alg is said to be β-strict for the kECND problem if, for each $\{s_i, t_i\} \in \mathcal{D}$, there exist cost-shares $\xi(\mathcal{D}, \{s_i, t_i\})$ such that the following properties hold:

1. $\sum_{\{s_i,t_i\}\in\mathcal{D}} \xi(\mathcal{D}, \{s_i, t_i\}) \leq c(\mathsf{OPT})$, where OPT is an optimal solution which k-edge-connects all terminal pairs in $\mathcal{D}$.
2. There is an efficient augmenting procedure $\mathsf{Augment}$ (which takes as input a terminal pair and outputs a set of edges) such that s_i and t_i are k-edge-connected in $\mathsf{Augment}(\{s_i, t_i\}) \cup \mathsf{Alg}(\mathcal{D} \setminus \{s_i, t_i\})$.
3. For each $\{s_i, t_i\} \in \mathcal{D}$, the total cost of edges output by $\mathsf{Augment}(\{s_i, t_i\})$ is at most $\beta\, \xi(\mathcal{D}, \{s_i, t_i\})$.

By the results of [64], it is known that if we have an α-approximation algorithm for a problem that admits β-strict cost-shares, we can then get randomized $(\alpha + \beta)$-approximation algorithms for the two-stage stochastic version of the problem. In the following, we explain how an online algorithm for a problem immediately implies strict cost-sharing schemes.

Cost-Shares from Online Algorithms

Given an α-competitive online algorithm Alg for kECND, order all possible vertex pairs in some universal canonical ordering, and feed the actual demands $\mathcal{D}$ in the induced ordering to Alg. Then for any $(s_i, t_i) \in \mathcal{D}$, define the cost-share $\xi(\mathcal{D}, \{s_i, t_i\})$ to be $\frac{1}{\alpha}$ times the increase in total cost incurred by the online algorithm. By the α-competitiveness of the online algorithm, we have $\sum_i \xi(\mathcal{D}, \{s_i, t_i\}) \leq \frac{1}{\alpha} \cdot \alpha\mathsf{OPT} = \mathsf{OPT}$. Moreover, the fixed ordering of the demands means that the augmentation cost for a demand pair s_i-t_i to $\mathsf{Alg}(\mathcal{D} \setminus \{s_i, t_i\})$ is at most the online algorithm's cost increase when we had presented s_i-t_i to it, i.e., $\alpha \cdot \xi(\mathcal{D}, \{s_i, t_i\})$.

Therefore, by combining the results of this section with Corollary 2.4.11, we can get our main theorem (Theorem 2.1.2) for the stochastic setting on general graphs.

2.6 Online SNDP on Metric Instances

In this section, we consider the *rooted* version of the online kECND problem on complete metrics, and give a deterministic online $O(\log n)$-competitive algorithm. Formally, we are given a complete graph G with the costs on edges $c(\cdot)$ satisfying the triangle inequality, and a root vertex r; a new demand vertex v_i arrives on the i^{th} day requiring k-edge-connectivity to the root r. The goal is to buy a set of edges $\mathcal{E}_i$ on the i^{th} day such that the collection $\cup_{j=1}^{i} \mathcal{E}_j$ contains k-edge-disjoint paths from r to $v_{j'}$ for all $j' \in [1, i]$, and the cost $c(\cup_{j=1}^{i} \mathcal{E}_j)$ is competitive with the cost of an optimal offline subgraph which k-edge-connects the demand vertices $v_1, v_2, \ldots, v_i$ with root r.

At a high level, our idea for the online algorithm is the following: when a new terminal arrives, run the online algorithm for Steiner tree; for each edge (u, v) that it buys, we buy a *minimum cost* set of edges to k-edge-connect u and v. While such an algorithm would indeed return a feasible solution to the kECND instance, it may not always be $O(\log n)$-competitive. However, what we can show is that it would be $O(\log n)$-competitive provided the online Steiner tree constructed has *bounded degree* (i.e., every vertex has degree bounded by a constant).

We first describe our kECND algorithm assuming such a *good* online algorithm for the Steiner forest problem, and then show how we can modify the algorithm of Imase and Waxman [74] to get the properties we desire. In the following, let Alg denote the $O(\log n)$-competitive online algorithm for Steiner tree which always maintains a feasible solution of bounded degree, and let N_v represent the set of the k nearest neighbors around any vertex v in G.

Algorithm 2 OnlineMetricAlg($\mathcal{D}$) for metric kECND

```
1: let network $S \leftarrow \emptyset$.
2: for each terminal pair $s_i$ that arrives do
3:    let $\mathcal{E}_i$ denote the set of edges bought by online Alg when we give it demand vertex $s_i$.
4:    for each edge $e = (u, v) \in \mathcal{E}_i$ do
5:       let $S \leftarrow S \cup \{(u, x) \mid x \in N_u\} \cup \{(v, x) \mid x \in N_v\}$
6:       let $S \leftarrow S \cup$ min-cost matching between $(N_u \setminus N_v)$ and $(N_v \setminus N_u)$
```

Theorem 2.6.1 *For any set of demands $\mathcal{D}$ that arrive, the network $S_\mathcal{D}$ output by algorithm* OnlineMetricAlg *is feasible to k-edge-connecting the root with the demands in $\mathcal{D}$, and has cost $c(S_\mathcal{D})$ at most $O(\log n)\mathsf{OPT}_\mathcal{D}$, where $\mathsf{OPT}_\mathcal{D}$ is the cost of an optimal offline subgraph which k-edge-connects vertices in $\mathcal{D} \cup \{r\}$.*

Proof: We first show that the network $S_\mathcal{D}$ is indeed a feasible solution. Let the terminals in $\mathcal{D}$ be $s_1, s_2, \ldots, s_i$, indexed by their arrival times. For any $j \in [1, i]$, let $F_j = \cup_{l=1}^{j} \mathcal{E}_l$ denote the Steiner subgraph bought by Alg in Step 3. Now consider some demand vertex $s_j \in \mathcal{D}$. Since Alg is an online algorithm for Steiner tree, the subgraph F_j contains a path from s_j to r. Consider vertices u and v such that $(u, v) \in F_j$. Since we connect u to N_u and v to N_v and add in a perfect matching between the vertices of $(N_u \setminus N_v)$ and $(N_v \setminus N_u)$ in S, it is easy to see that u and v are k-edge-connected. Therefore, from the transitivity of edge-connectivity, we see that any two vertices that are connected in F_j are in fact k-edge-connected, and hence $S_\mathcal{D}$ is a feasible solution.

To bound the cost, we use the following lower bounds on the cost of an optimal solution (similar bounds were also used by Cheriyan and Vetta [35] for node-connectivity SNDP):

- $c(\mathsf{OPT}_\mathcal{D}) \geq \frac{1}{2} \sum_{v \in D} c(v, N_v)$, and
- $c(F_i) \leq \frac{O(\log n)}{k} c(\mathsf{OPT}_\mathcal{D})$.

Let us explain why these are true: In any feasible solution, each terminal has to connect to at least k distinct neighbors. So if we add up the cost of some k outgoing edges from each vertex, the sum should be at most $2c(\mathsf{OPT}_\mathcal{D})$ since we can include each edge in $\mathsf{OPT}_\mathcal{D}$ at most twice. This gives us the first bound, since $c(v, N_v)$ is at most the sum of the costs on *some* k outgoing edges. For the second bound, consider the optimal fractional solution to the LP relaxation for the rooted kECND problem on demand set $\mathcal{D}$; clearly the cost of the fractional solution is at most $c(\mathsf{OPT}_\mathcal{D})$. Now, if we scale the fractional solution by a factor of k, we obtain a feasible fractional solution for the Steiner tree LP on demand set $\mathcal{D} \cup \{r\}$ of cost at most $\frac{1}{k} c(\mathsf{OPT}_\mathcal{D})$. But now because the LP for the minimum cost Steiner tree problem (which is a special case of SNDP) has a constant integrality gap, we get that the cost of an optimal offline Steiner tree feasible to the demand set $\mathcal{D}$ is at most $\frac{2}{k}c(\mathsf{OPT}_{\mathcal{D}})$. The second inequality then follows as a consequence of the $O(\log n)$-competitiveness of the online Steiner tree algorithm Alg.

The total cost of the subgraph $S_{\mathcal{D}}$ is then

$$\begin{aligned} c(S_{\mathcal{D}}) &\leq \textstyle\sum_{u\in F_i} c(u, N_u) + \sum_{(u,v)\in F_i} (c(u, N_u) + c(v, N_v) + k \cdot c(u, v)) \\ &\leq O(1) \textstyle\sum_{u\in F_i} c(u, N_u) + k \sum_{(u,v)\in F_i} c(u, v) \\ &\leq O(\log n)\, c(\mathsf{OPT}_{\mathcal{D}}) \end{aligned}$$

Here, the cost of the min-cost matching between N_u and N_v was bounded by $c(u, N_u) + c(v, N_v) + k \cdot c(u, v)$ by using the triangle inequality of the metric space. Also, the second inequality follows from the assumption that Alg always maintains a bounded-degree online Steiner tree. ■

It now remains to show how to design the bounded-degree $O(\log n)$-competitive online algorithm for Steiner tree; we now consider this sub-problem.

Online Degree-Bounded Steiner Tree

Imase and Waxman [74] show that the greedy algorithm (of each new demand connecting to its nearest vertex on the current solution) is $O(\log n)$-competitive for the online Steiner tree problem. The only problem is that if several terminals share a common vertex as their nearest neighbors, the common vertex would have a very high degree in the Steiner tree we maintain. To avoid this, the simple idea we use is to maintain a *chain* for each vertex v, and connect these terminals (which would have otherwise connected to the common vertex v) to the end of the chain instead. Because the graph is a complete metric, this would allow us to bound the cost of the chain by the cost of the star around each vertex v, and hence let us maintain a low-cost degree bounded solution. In the following, let GreedySteiner denote the online greedy algorithm for the Steiner tree problem.

Algorithm 3 LowDegAlg($\mathcal{D}$) for bounded-degree online Steiner tree

```
1: let network $S \leftarrow \emptyset$; define a chain $C_v \leftarrow \emptyset$ for each $v \in V$.
2: for each terminal vertex $s_i$ that arrives do
3:     let $(v, s_i)$ denote the edge bought by GreedySteiner when we give it demand vertex $s_i$.
4:     if $C_v = \emptyset$ then
5:         set $S \leftarrow S \cup (v, s_i)$; add $s_i$ to the chain $C_v$
6:     else
7:         let $v'$ be the tail of the chain $C_v$; set $S \leftarrow S \cup (v', s_i)$ and add $s_i$ to the end of the chain $C_v$.
```

Theorem 2.6.2 *The network $S_{\mathcal{D}}$ output by algorithm* LowDegAlg *is feasible to the online Steiner tree problem on demand set $\mathcal{D}$, and has cost $c(S_{\mathcal{D}})$ at most $O(1)c(\mathsf{GreedySteiner}(\mathcal{D}))$. Furthermore, the degree of each vertex in $S_{\mathcal{D}}$ is at most* 3.

Proof: We first show that the algorithm outputs a feasible solution. Consider a newly arrived vertex s_i, and let (v, s_i) denote the edge bought by the online algorithm GreedySteiner (recall that the greedy online Steiner tree algorithm buys only the edge to the nearest neighbor on the current solution). If the chain C_v is empty when s_i arrives, our online algorithm LowDegAlg also buys the edge (v, s_i) and therefore connects s_i to the root r (since v was already connected to the root). If on the other hand C_v was non-empty, then let $C_v \equiv \{v_1, v_2, \ldots, v_t\}$, with the vertices ordered by time of addition to the chain. Then, by the way LowDegAlg adds edges within a chain (in step 7), we are guaranteed that the edges $(v, v_1), (v_1, v_2), \ldots, (v_{t-1}, v_t)$ are all present in the network $S_{\mathcal{D}}$ maintained. And since the edge (v_t, s_i) is also added to $S_{\mathcal{D}}$ when s_i is added to the end of the chain C_v, we see that s_i is connected to v, and therefore to r in $S_{\mathcal{D}}$. This proves the feasibility.

To bound the cost, consider a vertex v, and let the current chain be $C_v \equiv \{v_1, v_2, \ldots, v_t\}$. Now, by the triangle inequality, the total cost of all the edges $(v, v_1), (v_1, v_2), \ldots, (v_{t-1}, v_t)$ can be bounded by $2\sum_{i=1}^{t} c(v, v_i)$, which is precisely twice the total cost incurred by GreedySteiner when connecting $v_1, v_2, \ldots, v_t$ to the vertex v. Therefore, the total cost of $S_{\mathcal{D}}$ can be bounded by $2c(\text{GreedySteiner}(\mathcal{D}))$.

It is also easy to see that the degree of each vertex in $S_{\mathcal{D}}$ is at most 3. To see why, let us consider a vertex v and look at when edges incident at v are added in $S_{\mathcal{D}}$. There are two reasons why such edges are added: **(i)** an edge (v, v') is added when v' arrives to the chain that v is present in, and **(ii)** v can have a chain of its own. In the former case, we know that v belongs to the chain of only one other vertex v' (the vertex to which it was connected in GreedySteiner when it arrived), and therefore v can have at most 2 incident edges by being present in $C_{v'}$. In the latter case, since it is the head, it can have at most one edge incident on it (with the vertex which first entered v's chain).

Therefore, the network $S_{\mathcal{D}}$ constructed is feasible to the online Steiner tree, has a cost competitive with GreedySteiner, and the degree of each vertex is bounded by 3. ■

2.7 Stochastic SNDP on Metric Instances

In this section, we assume that G is a complete graph, and that the edge costs $c(\cdot)$ satisfy the triangle inequality, i.e., $c(u, v) \leq c(u, w) + c(w, v)$ for all $u, v, w \in V$. Under this assumption, we can improve over even our results on the online algorithm from the previous section, and show a constant-factor approximation algorithm for the stochastic version of kECND.

Our idea proceeds as follows: we first give a new $O(1)$-approximation algorithm for metric-kECND, one which admits $O(1)$-strict *cost-shares* (see Section 2.5 for their definition and how these cost-shares help). Such an algorithm would immediately imply constant-approximations for the stochastic version of the problem. We remark that while better algorithms are known for even the k-node-connected generalization of kECND [35] on metric graphs, we need to come up with one which provably admits good cost-shares.

Constant-Factor Approximation for Metric kECND

Consider an instance $G = (V, E)$ of kECND with terminal pairs in $\mathcal{D}$; let D represent the set of all terminals, i.e., $D = \cup_{(s_i,t_i)\in\mathcal{D}}\{s_i, t_i\}$. Call a set $S \subseteq V$ *valid* if there exist a demand $(s_i, t_i) \in \mathcal{D}$ such that $|S \cap \{s_i, t_i\}| = 1$. Define ∂S to be the set of edges with one endpoint in S, and $x(E') = \sum_{e \in E'} x_e$. Finally, let N_v represent the set of the k nearest neighbors of vertex v in G. The LP relaxation of the kECND problem is the following:

$$\begin{array}{llll} (LP_k) & \text{minimize} & \sum_{e \in E} c_e x_e & \\ & \text{subject to} & (1) \quad x(\partial S) \geq k & \forall \text{ valid } S \subseteq V \\ & & (2) \quad 0 \leq x_e \leq 1, & \forall\, e \in E \end{array}$$

Let OPT and $\mathsf{OPT_{LP}}$ be optimal integral and fractional solutions to the given instance; clearly $c(\mathsf{OPT_{LP}}) \leq c(\mathsf{OPT})$. Our algorithm follows the ideas used in the online algorithm, with the following changes: instead of running the online algorithm for Steiner tree, we run the AKR algorithm ([3]) for Steiner forest to 1-edge-connect the demand pairs in the first step. The AKR algorithm is a primal-dual algorithm for the Steiner forest problem, which when given a set of terminal pairs $\mathcal{D}$, outputs a Steiner forest of cost at most *twice* the cost of an optimal fractional solution to the LP relaxation of the Steiner forest problem.

Getting back to our algorithm, in our second step, in order to get a low-degree Steiner forest, we simply take an euler tour of the AKR solution. Finally, we k-edge-connect u and v (using nearby neighbors) for any edge (u, v) that the AKR algorithm buys, just like in the online algorithm.

Algorithm 4 MetricAlg($\mathcal{D}$) for metric kECND

```
1: let network $S \leftarrow \emptyset$. Run the AKR algorithm on $\mathcal{D}$ to get forest $F$.
2: let $\widetilde{F} \leftarrow$ subgraph obtained by taking Euler tour of each component of $F$.
3: for each edge $e = \{u, v\}$ in $\widetilde{F}$ do
4:     let $S \leftarrow S \cup \{\{u, x\} \mid x \in N_u\} \cup \{\{v, x\} \mid x \in N_v\}$
5:     let $S \leftarrow S \cup$ min-cost matching between $N_u$ and $N_v$
```

Theorem 2.7.1 *The network S output by algorithm* MetricAlg *k-edge-connects the terminal pairs in $\mathcal{D}$, and has cost $c(S) \leq 10\, c(\mathsf{OPT})$. Furthermore, if $\{u, v\} \in F$, then u and v are k-edge-connected in S.*

Proof: The proof is very similar to that of Theorem 2.6.1. We begin by showing that network S is indeed a feasible solution. Consider a terminal pair $(s, t) \in \mathcal{D}$: since F is a feasible Steiner forest solution for $\mathcal{D}$, we know that s and t belong to the same tree in F and therefore, to the same cycle in $\widetilde{F}$. We now show that any pair of vertices that lie on a cycle in $\widetilde{F}$ are k-edge-connected in S. Consider vertices u and v such that $\{u, v\} \in \widetilde{F}$. Since we connect u to N_u and v to N_v and add in a perfect matching between the vertices of $N_u \setminus N_v$ and $N_v \setminus N_u$ in S, it is easy to see that u and v are k-edge-connected. By transitivity of edge-connectivity, any two vertices on a cycle in $\widetilde{F}$ are k-edge-connected. This proves that S is a feasible solution. To bound the cost, we again use the following lower bounds (almost identical to the ones used in the online algorithm):

- $c(\mathsf{OPT}) \geq \frac{1}{2} \sum_{v \in D} c(v, N_v)$, and
- $c(\widetilde{F}) \leq \frac{4}{k}\, c(\mathsf{OPT})$.

For completeness, let us explain why these are true:

The first bound has already been established in the proof for the online algorithm in the earlier section (proof of Theorem 2.6.1). For the second bound, consider the optimal fractional solution to the LP relaxation for the rooted kECND problem on demand set $\mathcal{D}$; clearly the cost of the fractional solution is at most $c(\mathsf{OPT}_{\mathcal{D}})$. Now, if we scale the fractional solution by a factor of k, we obtain a feasible fractional solution for the Steiner forest LP on demand set $\mathcal{D} \cup \{r\}$ of cost at most $\frac{1}{k}c(\mathsf{OPT}_{\mathcal{D}})$. But now because the AKR for the minimum cost Steiner forest problem has a constant approximation factor (w.r.t the optimal LP solution), we get that the cost of the solution $\mathcal{F}$ is at most $\frac{2}{k}c(\mathsf{OPT}_{\mathcal{D}})$. Making an euler tour of F to get $\widetilde{F}$ only increases the cost by a factor of at most 2.

The total cost of S is then

$$\begin{aligned} c(S) &\leq \textstyle\sum_{u\in\widetilde{F}} c(u, N_u) + \sum_{\{u,v\}\in\widetilde{F}} \left(c(u, N_u) + c(v, N_v) + k \cdot c(u, v)\right) \\ &\leq 3\textstyle\sum_{u\in\widetilde{F}} c(u, N_u) + k\sum_{\{u,v\}\in\widetilde{F}} c(u, v) \\ &\leq 10\, c(\mathsf{OPT}) \end{aligned}$$

In the second step, we used the fact that because each vertex has degree 2 in $\widetilde{F}$, the term $c(u, N_u)$ can appear at most twice in the latter summation $\sum_{\{u,v\}\in\widetilde{F}} \left(c(u, N_u) + c(v, N_v) + k \cdot c(u, v)\right)$.

■

Getting Strict Cost-Shares

We now show how we can get strict cost-shares for the above algorithm. As basis for our cost-sharing scheme, we use the cost-shares associated with the AKR algorithm, as given by Fleisher et al. [49] (We refer to this cost-sharing scheme as the FKLS analysis.).

Let $F^{\mathcal{D}}$ denote the AKR solution on demand set $\mathcal{D}$. The FKLS analysis defines the cost-sharing functions $\xi' : E \times V \to \mathbb{R}$ and $\xi : \mathcal{D} \to \mathbb{R}$ in the following manner:

(i) Each edge $e \in F^{\mathcal{D}}$ is assigned two *witness* terminals w_1 and w_2 such that $\xi'(e, w_1) = \xi'(e, w_2) = c_e/4$. The function $\xi'(e, v)$ is set to 0 for all $v \in V \setminus \{w_1, w_2\}$.

(ii) For any vertex u and an edge e not in $F^{\mathcal{D}}$, $\xi'(e, u)$ is set to 0.

(iii) The total cost-share of a terminal pair (s_i, t_i) is then defined as $\xi((s_i, t_i)) = \sum_{e\in F^{\mathcal{D}}} \left(\xi'(e, s_i) + \xi'(e, t_i)\right)$.

In further notation, for any edge $e \in F^{\mathcal{D}}$, let τ_e denote the time at which e was bought by the AKR algorithm; also denote the time at which (s_i, t_i) gets connected in $F^{\mathcal{D}}$ by τ_i. The FKLS analysis shows that the witnesses satisfy the following properties:

1. Consider a demand (s_i, t_i) and any edge $e \in F^{\mathcal{D}}$ bought at time $\tau_e \leq \tau_i$. If neither s_i nor t_i is a witness for e, then e is also bought in the run of AKR $(\mathcal{D} \setminus (s_i, t_i))$. In particular, for any edge e on the unique path connecting s_i and t_i in $F^{\mathcal{D}}$, if s_i and t_i don't witness e, then $e \in F^{\mathcal{D}\setminus(s_i,t_i)}$.
2. $\sum_{i=1}^{|\mathcal{D}|} \xi((s_i, t_i)) \leq \frac{2}{k}c(\mathsf{OPT}_{\mathsf{LP}}(\mathcal{D}))$. Further, the solution obtained by running AKR on $\mathcal{D} \setminus (s_i, t_i)$ can be augmented with edges of cost $O(1)\xi((s_i, t_i))$ to get a subgraph which connects s_i and t_i. In fact, these "augmenting" edges are those which s_i or t_i witness.

Given that the 1-edge-connectivity problem has nice witness properties, the most natural thing to try would be to define cost-shares for k-edge-connectivity in the following way: for any edge e that s_i or t_i witness, the cost-share for (s_i, t_i) includes the cost of k-edge-connecting u and v (at most $2\,c(u, N_u) + 2\,c(v, N_v) + k\,c(u, v)$). When defined in this form, although we would be able to augment a solution of $\mathsf{MetricAlg}(\mathcal{D} \setminus (s_i, t_i))$ to k-edge-connect s_i-t_i by paying $O(1) \times \xi^k((s_i, t_i))$, we cannot directly bound $\sum_{i=1}^{|\mathcal{D}|} \xi^k((s_i, t_i))$, since the quantity $c(u, N_u)$ could be counted several times. However, this can happen only if the degree of u in the approximate solution is high (just like in the online algorithm). We therefore look at transforming the AKR solution into a low-degree one while *preserving* the witness properties. (An Euler tour would get us the low-degree tree, but it would not satisfy *good* witness properties we desire.)

Let $F^{\mathcal{D}}$ be the forest obtained by running the AKR algorithm on demand set $\mathcal{D}$. We apply the following modification step for each tree in $F^{\mathcal{D}}$. Consider a tree $T^{\mathcal{D}}$, and arbitrarily root it at r. We now perform a reverse breadth-first (bottom up) traversal, and create a modified solution F^{mod} ($= \emptyset$ initially).

Modification: Suppose we are at vertex u in our traversal: Nothing is done if it is a leaf. If it is an internal node having degree 2, then the edge between u and it's child is included in F^{mod}. If it has degree more than 2, then we perform the following local modification (u is said to be the *main vertex* being *altered* in the step, and the edges being altered are the children edges incident at u):

Let $v_1, v_2, \ldots, v_p$ be an ordering of the child vertices of u ordered such that $\tau_{\{u,v_1\}} \leq \tau_{\{u,v_2\}} \leq \ldots \leq \tau_{\{u,v_p\}}$. We anchor the edge $\{u, v_1\}$ and add it to F^{mod}. For every other edge $\{u, v_i\}$, we add the edge $\{v_i, v_{i-1}\}$ to F^{mod}. The witnesses of $\{u, v_1\}$ remain the same as those assigned by the FKLS algorithm, and the witnesses of the edge $\{v_i, v_{i-1}\}$ in F^{mod} are the FKLS witnesses of $\{u, v_i\}$ in $F^{\mathcal{D}}$. Note that the degree of u in F^{mod} is reduced to 2, whereas the degree of u's child vertices (which were 2 before this step) are increased by at most 1. After this step, u would never be the main vertex being altered in any step meaning that it's degree will be at most 3 in F^{mod}.

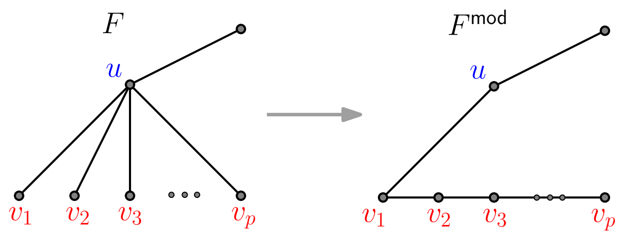


Figure 2.6: A step in the modification: u is the main vertex being altered

This local modification is performed in a reverse breadth first fashion. It is easy to see that we obtain a forest whose cost is at most twice the cost of the AKR solution. Further, each vertex has degree at most 3 and each edge has at most 2 witnesses.

Lemma 2.7.2 *The forest F^{mod} created by the above modification is such that the cost $c(F^{\mathsf{mod}})$ is at most $2c(F^{\mathcal{D}})$, and any vertex has degree at most 3 in F^{mod}. Furthermore, there exists witness definitions such that the following properties hold.*

(i) If W_i is the set of edges in F^{mod} for which either s_i or t_i is a witness, then for any edge $\{u,v\}$ in the unique path connecting s_i and t_i in $F^{\mathcal{D}}$, u and v remain connected in the subgraph $W_i \cup F^{\mathcal{D}\setminus(s_i,t_i)}$, where $F^{\mathcal{D}\setminus(s_i,t_i)}$ is the forest returned by $\mathsf{AKR}(\mathcal{D}\setminus(s_i,t_i))$.

(ii) At most 2 terminals witness any edge in F^{mod}.

Proof: The cost and degree bound follow directly as a consequence of the way our modification algorithm worked. We now prove the witness properties by showing that u and v are in fact connected by a path comprising of a sequence of edges in W_i followed by an edge which belongs to $F^{\mathcal{D}\setminus(s_i,t_i)}$. Consider the stage in the alteration procedure when $\{u,v\}$ is being altered. One of u or v has to be the main node being altered. Without loss of generality, we assume that u is the vertex being altered. Two cases are to be considered:
Case (1): $\{u,v\}$ is not witnessed by $\{s_i,t_i\}$: Since $\{u,v\}$ lies on the unique s_i-t_i path in $F^{\mathcal{D}}$, we know that $\tau_{\{u,v\}} \le \tau_i$. Therefore, by the first property of the FKLS analysis, this edge will be bought by the AKR algorithm when run on $\mathcal{D}\setminus(s_i,t_i)$, and therefore u and v are connected in $F^{\mathcal{D}\setminus(s_i,t_i)} \subseteq W_i \cup F^{\mathcal{D}\setminus(s_i,t_i)}$.
Case (2): $\{u,v\}$ is witnessed by one of $\{s_i,t_i\}$. Recall that we had assumed that u is main the vertex being altered. In the case that $\{u,v\}$ was the edge being anchored, we know that $\{u,v\}$ is present in the modified tree F^{mod} and has the same witnesses as before, meaning $\{u,v\} \in W_i \subseteq W_i \cup F^{\mathcal{D}\setminus(s_i,t_i)}$. If $\{u,v\}$ was not the edge being anchored, let $v_1, v_2, \ldots, v_p$ be the ordering of the child vertices of u chosen by the alteration procedure. Note that $v \in \{v_2, v_3, \ldots, v_p\}$. Without loss of generality, let v be v_q. Also, let r be the largest index such that $1 \le r < q$ and that $\{u, v_r\}$ is not witnessed by either s_i or t_i. There are two cases to be considered.

1. If such an r does not exist: it means that each of the edges $\{u,v_1\},\{u,v_2\},\ldots,\{u,v_q\}$ are witnessed by s_i or t_i, and therefore $\{u,v_1\},\{v_1,v_2\},\ldots,\{v_{q-1},v_q\}$ all belong to W_i, meaning that u and v are connected in $W_i \cup F^{\mathcal{D}\setminus(s_i,t_i)}$.
2. If such an r exists: we know that each of the edges $\{u,v_{r+1}\},\ldots,\{u,v_q\}$ are witnessed by s_i or t_i—this means that each of the edges $\{v_r,v_{r+1}\},\ldots,\{v_{q-1},v_q\}$ are in W_i. Further, by the way we ordered the children of u, it is clear that $\tau_{\{u,v_r\}} \le \tau_{\{u,v_q\}} \le \tau_i$. The latter inequality is because of the fact that $\{u,v_q\}$ is on the unique path connecting s_i and t_i, and therefore cannot be bought after s_i and t_i are connected. Hence, by property 1 of the FKLS algorithm, we know that $\{u,v_r\}$ is bought in the run of $\mathsf{AKR}(\mathcal{D}\setminus(s_i,t_i))$. Therefore, u and v are connected in $W_i \cup F^{\mathcal{D}\setminus(s_i,t_i)}$.

This proves the desired witness properties, and hence completes the proof. ■

We are now ready to define the $O(1)$-strict cost-shares for this problem.

Cost Shares: For each terminal pair (s_i,t_i), we set its cost-share to be

$$\xi^k((s_i,t_i)) = \sum_{\{u,v\}\in W_i} (2\,c(u,N_u) + 2\,c(v,N_v) + k\,c(u,v))$$

Recall that W_i is the set of edges which either s_i or t_i witness in F^{mod}. Since each vertex in F^{mod} has degree at most 3 and each edge e has at most 2 witnesses, we get $\sum_i \xi^k((s_i,t_i)) \le \sum_{u\in D} 12\,c(u,N_u) + 2k\sum_{e\in F^{\mathsf{mod}}} c(u,v) \le 32\,c(\mathsf{OPT}(\mathcal{D}))$. To show that these cost-shares are

$O(1)$-strict, we also need to give an algorithm which can augment edges of cost $\xi^k((s_i, t_i))$ to a subgraph returned by $\mathsf{MetricAlg}(\mathcal{D} \setminus (s_i, t_i))$ in order to k-edge-connect s_i and t_i.

Augmentation Algorithm: $\mathsf{Augment}\ ((s_i, t_i))$: For all $\{u, v\} \in W_i$, buy the set of edges of minimum cost to k-edge-connect u and v.

Analysis: From Lemma 2.7.2, we know that if an edge $\{u, v\}$ lies on the unique path connecting s_i and t_i in $F^{\mathcal{D}}$, then u and v are connected in $W_i \cup F^{\mathcal{D}\setminus(s_i,t_i)}$. Now consider any edge $\{u', v'\} \in W_i \cup F^{\mathcal{D}\setminus(s_i,t_i)}$. If $\{u', v'\}$ is in W_i, then the augmentation algorithm would k-edge-connect the vertices u' and v'. If it is in $F^{\mathcal{D}\setminus(s_i,t_i)}$, then from Theorem 2.7.1, $\mathsf{MetricAlg}(\mathcal{D} \setminus (s_i, t_i))$ would k-edge-connect u' and v'. Hence, each edge $\{u', v'\}$ on the $u - v$ path contained in $W_i \cup F^{\mathcal{D}\setminus(s_i,t_i)}$ is such that u' and v' are k-edge-connected in $\mathsf{Augment}((s_i, t_i)) \cup \mathsf{MetricAlg}(\mathcal{D} \setminus (s_i, t_i))$. Therefore, from transitivity of edge-connectivity, we get that s_i and t_i are k-edge-connected in $\mathsf{Augment}((s_i, t_i)) \cup \mathsf{MetricAlg}(\mathcal{D} \setminus (s_i, t_i))$. This, and the fact that the cost of the augmenting edges to k-edge-connect s_i-t_i is at most $\xi^k((s_i, t_i))$, establish the $O(1)$-strict cost-shares for Algorithm MetricAlg. The following theorem therefore follows.

Theorem 2.7.3 *The algorithm* MetricAlg *permits $O(1)$-strict cost-shares for* kECND*, implying $O(1)$ approximations for the two-stage stochastic* kECND *problem on metric instances.*

2.8 Conclusion

In this chapter, we considered the SNDP problem from the models of online and two-stage stochastic optimization. We presented the first non-trivial algorithms with polylogarithmic competitive and approximation ratios respectively. An interesting open problem that arises from this is that of designing online algorithms for the SNDP problem with vertex-connectivity requirements. Towards this end, following our work, Naor et al. [91] developed online (bi-criteria) algorithms for the single-source l-vertex-connectivity problem with polylogarithmic competitive ratio.

Chapter 3

Stochastic Knapsack

We now move on to the setting of adaptive stochastic optimization from the point of view of some basic scheduling problems. In this chapter, our focus will be on the *Stochastic Knapsack* problem, which is perhaps the simplest single-machine scheduling problem. Subsequent chapters generalize this problem in two very different but substantial ways.

Deterministic Knapsack. In the basic Knapsack problem, we are given a set of items/jobs, each with a size and reward, and a Knapsack of capacity B. The goal is to pick a collection of items whose size add up to at most B, and whose total reward is maximized. This fundamental problem is a classical problem in all of optimization, and has received much attention from various fields of study owing to both its simplicity as well as practical applicability. From a complexity perspective, the problem is NP-complete [51], but we know extremely good approximation algorithms known as FPTASs, i.e., those that return solutions with profit at least $(1 - \varepsilon)\mathsf{OPT}$ and whose running time is polynomial in n and $1/\varepsilon$.

The main focus of this chapter is, however, the following more general question: what if the sizes and rewards are random variables, and the algorithm only knows the distributions up front, and can only figure out the size (or reward) of an item/job only when it has been processed. In other words, can we approximate the *Stochastic Knapsack* problem just as well as we can, the deterministic problem?

Basic Stochastic Knapsack. The Stochastic Knapsack problem has garnered much attention in the recent past both from the practical and theoretical points of view, starting with the work of Dean et al. [41]. From the practical side, since they model the task of designing policies that can handle uncertainty in the job parameters, the common belief is that the algorithmic techniques developed could be useful in designing OS schedulers, real-time systems, etc. From the theoretical side, they introduce new challenges at all stages of algorithm design. Firstly, the optimal solution itself might require exponential size to describe (since it could be a complete decision tree, taking different actions for different random outcomes of the jobs). Secondly, our algorithm may also have to be adaptive (or else we need to bound the adaptivity gap to restrict our attention on designing non-adaptive solutions), and finally we would need to compare the expected profit of our (possibly adaptive) algorithm with that of an optimal (adaptive) algorithm. We note that the disparities over set of issues to tackle means that solution techniques for solving these adaptive stochastic optimization problems have a very different theme when compared to those for the two-stage stochastic problems.

While we now know good constant-factor approximations for the Stochastic Knapsack problem [18, 19, 41], all these algorithms work under two strong assumptions: (i) the (random) reward of a job is independent of its (random) size, and (ii) the algorithm cannot preempt or cancel jobs which are taking too long to complete. There are very simple (and often-arising) examples which do not have the independence property: for example, think of each stochastic job as being a randomized algorithm. Then, clearly its running time is a random variable, and so its resultant utility (or correctness probability). Moreover, these quantities are directly dependent on the internal coin-tosses of the algorithm, and are therefore correlated with each other. In a completely different setting, what if each random job requires delivering different articles at different places on a metric. Then depending on the traffic patterns, both the time, as well as the reward (which could be the number of articles successfully delivered) are random variables, and they are correlated with each other.

Likewise, the second assumption is also crucial in case the jobs have very large variance in their running times. In such cases, there could be tremendous advantage by having the ability to cancel jobs, as the following example illustrates. Consider the setting where there are N jobs. Each job has a size of either 1 w.p $1/2$ or N w.p $1/2$, and has a fixed reward of 1. The Knapsack budget is N. Then, in this case, notice that if the algorithm cannot cancel a job, then it can only get an expected reward of $O(1)$. This is because, for the algorithm to get a reward of i jobs, all of them must finish with size 1 (which happens with probability $1/2^i$). A simple calculation then shows that the expected reward is at most a small constant. However, in the setting where cancellations are allowed, the algorithm can simply cancel a job if it runs for 1 unit of time *but has not completed by itself.* In this manner, our algorithm can schedule *all the jobs*, and gets a profit from each of them with probability $1/2$, giving us a total expected reward of $N/2$.

General Stochastic Knapsack. Motivated by the above-mentioned reasons, we look at the general version of the Stochastic Knapsack problem. In this problem, we are given a collection of jobs, each equipped with a distribution over $(\mathsf{size}, \mathsf{reward})$ pairs. The goal is to adaptively schedule these jobs so as to maximize the expected reward of all the jobs that have been successfully scheduled within a time budget B. The algorithm gets to know the actual size/reward of a job only when it completes. There are two variants within this basic framework. In the first one, the algorithm can't arbitrarily preempt jobs or even prematurely cancel jobs during their execution, i.e., its decision to schedule a job is irrevocable. In the second variant, the algorithm has the power to prematurely cancel or preempt jobs during execution. However, in both variants, the algorithm can take adaptive decisions on which jobs to run when, based on how the randomness has instantiated.

3.1 Our Results

We prove the following results in this thesis. Our first theorem concerns with the approximability (and adaptivity gap) of the basic correlated Stochastic Knapsack problem in the model when job cancellations are not allowed.

Theorem 3.1.1 *There is a polynomial-time (non-adaptive) randomized algorithm for the Stochastic Knapsack problem with correlated rewards, which obtains an expected reward of at least* $\frac{1}{8}\mathsf{OPT}$*, where* OPT *is the expected reward of an optimal adaptive algorithm.*

We then extend this result to the model of the problem where the jobs can be prematurely canceled, e.g., if they run long durations without completing. There are some simple examples which show that the expected reward can be larger by an arbitrary factor in this model over the former (see Section 3.4.1).

Theorem 3.1.2 *For the Stochastic Knapsack problem with correlated reward and cancellations, there is a polynomial-time (non-adaptive) randomized algorithm which obtains an expected reward of at least* $\frac{1}{16}\mathsf{OPT}$*, where* OPT *is the expected reward of an optimal adaptive algorithm.*

The subsequent chapter on MAB's will cover a much more general model which even captures job preemptions (i.e., switching between jobs) and shows a constant-approximation for that version as well.

3.2 Related Work

Stochastic scheduling problems (in fact, even those with correlated rewards) have been long studied since the 1960s (e.g., [20, 38, 82, 92]); however, there are fewer papers on approximation algorithms for such problems. These problems were first studied from an approximations perspective in an important paper of Dean et al. [42] (see also [40, 41]). They considered the Stochastic Knapsack problem. Via an LP relaxation and a rounding algorithm, they gave *non-adaptive* solutions with expected rewards that are (surprisingly) within a constant-factor of the best *adaptive* ones, resulting in a constant adaptivity gap (also a notion they introduced). However, the results required that (a) the random rewards and sizes are independent of each other, and (b) once an item was placed, it can not be prematurely cancelled.

Kleinberg et al. [81], and Goel and Indyk [53] consider stochastic Knapsack problems with chance constraints: find the max-profit set which will overflow the Knapsack with probability at most p. However, their results hold for deterministic profits and specific size distributions. The problem of minimizing average completion times with arbitrary job-size distributions was studied by [90, 97]. The work most relevant to us is that of Dean, Goemans and Vondrák [40, 41, 42] on Stochastic Knapsack and packing; apart from algorithms (for independent rewards and sizes), they show the problem to be PSPACE-hard when correlations are allowed. [27] study stochastic flow problems. Finally, the recent work of Bhalgat et al. [17, 19] presents a PTAS but violate the capacity by a factor $(1+\varepsilon)$; they also get improved guarantees when there are no violations.

3.3 Chapter Roadmap

In the following section, we present an overview of why previous techniques don't work, and what our new technical contributions are. Then in Section 3.5, we present our algorithm for the case when cancellations are not allowed. Finally, we extend this to the setting where cancellations are allowed in Section 3.6. Finally, in Sections 3.7-3.8, we furnish complete details of the omitted proofs from the main sections.

3.4 Techniques

One reason why stochastic packing problems are more difficult than deterministic ones is that, unlike in the deterministic setting, we cannot simply take a solution with expected reward R^* that packs into a Knapsack of size $2B$ and get one with reward $\Omega(R^*)$ whilst fitting within the budget of B (by appropriately sub-selecting some items). In fact, in stochastic settings, there are examples where a budget of $2B$ can fetch much more reward than what a budget of size B can (see Section 3.4.2 below).

The work of Dean et al. [40, 42] assumes that the reward of an item is independent of its size. Moreover, their model does not consider the possibility of canceling items in the middle. These assumptions simplify the structure of the optimal (adaptive) decision tree and make it possible to formulate a Knapsack-style LP which captures even adaptive optimal solutions, and subsequently round it. We present their high-level idea, both for completeness and the fact that our techniques build on theirs.

$$\max \textstyle\sum_i \mathbb{E}[r_i] \cdot x_i \tag{$\mathsf{LP_{DGV}}$}$$
$$\textstyle\sum_i x_i \cdot \mathbb{E}[\min(S_i, B)] \le 2B \tag{3.1}$$
$$x_i \in [0,1] \qquad \forall i \tag{3.2}$$

Above, the variable $x_i \in [0,1]$ indicates that item i is included in the Knapsack; S_i denotes the random variable for the size of item i, and $\mathbb{E}[r_i]$ denotes the expected reward if item i completes successfully within the budget of B. Intuitively, their proof proceeds by showing that $x_i^* = \mathbb{P}\left[\mathsf{OPT} \text{ inserts item } i \text{ into the Knapsack}\right]$ is a feasible solution for the above LP, and that it has value at least $\mathbb{E}\left[\mathsf{OPT}\right]$. Notice that, unlike deterministic problems where showing that an LP is feasible is typically a trivial argument, it is not as immediate in the stochastic setting, because the LP we write uses expected values, whereas OPTdeals with the actual distributions. In other words, it would be a misnomer to call the aforementioned LP a *relaxation*, since it actually *strengthens* the problem by assuming that the random variables are replaced by their expectations.

Once this has been established, we can use the fact that the Knapsack LP is almost integral, and therefore argue that $\mathsf{OPT}$is, essentially, not adaptive — that is, it can decide on the jobs to include in the Knapsack regardless of how the previous' jobs sizes turned up. This also gives them a simple non-adaptive algorithm which performs as well as the best adaptive solution (up to constant-factors).

However, the above LP breaks down in the face of correlations or cancellations. The following two sections gives a couple of examples illustrating why.

3.4.1 Badness Due to Cancellations

We first observe that the LP relaxation for the StocK problem used in [42] has a large integrality gap in the model where cancellations are allowed, *even when the rewards are fixed for any item.* This was also noted in [40]. Consider the following example: there are n items, every item instantiates to a size of 1 with probability 0.5 or a size of $n/2$ with probability 0.5, and its reward is always 1. Let the total size of the Knapsack be $B = n$. For such an instance, a good solution would cancel any item that does not terminate at size 1; this way, it can collect a reward of at least $n/2$ in expectation, because an average of $n/2$ items will instantiate with a size 1 and these will all contribute to the reward. On the other hand, the LP from [42] has value $O(1)$, since the mean size of any item is at least $n/4$. In fact, any strategy that does not cancel items will also accrue only $O(1)$ reward.

3.4.2 Badness Due to Correlated Rewards

We next present an example of Stochastic Knapsack (where the reward is correlated with the actual size) for which the existing LP formulations for StocK (and even more general MAB problems) all have a large integrality gap.

Consider the following example: there are n items, every item instantiates to a size of 1 with probability $1 - 1/n$ or a size of n with probability $1/n$, and its reward is 1 only if its size is n, and 0 otherwise. Let the total size of the Knapsack be $B = n$. Clearly, any integral solution can fetch an expected reward of $1/n$ — if the first item it schedules instantiates to a large size, then it gives us a reward. Otherwise, no subsequent item can be fit within our budget even if it instantiates to its large size. The issue with the existing LPs is that the *arm-pull* constraints are ensured locally, and there is one global budget. That is, even if we play each arm to completion individually, the expected size (i.e., number of pulls) they occupy is $1 \cdot (1 - 1/n) + n \cdot (1/n) \leq 2$. Therefore, such LPs can accommodate $n/2$ items, fetching a total reward of $\Omega(1)$. This example brings to attention the fact that all these item are competing to be pulled in the first time slot (if we begin an item in any later time slot it fetches zero reward), thus naturally motivating our time-indexed LP formulation in Section 3.6.1.

In fact, the above example also shows that if we allow ourselves a budget of $2B$, i.e., $2n$ in this case, we can in fact achieve an expected reward of $O(1)$ (much higher than what is possible with a budget of B) — keep playing all items one by one, until one of them does not step after size 1 and then play that to completion; this event happens with probability $\Omega(1)$.

3.4.3 Intuition behind our LP

For the (correlated) Stochastic Knapsack problem, the main issue is that the earlier LPs do not capture the case when all the items have high *contention*, i.e., they may all want to be played early in order to collect a huge profit from their large sizes. We resolve these issues in the following manner. To handle the issues of contention, we formulate a *global time-indexed* formulation for Knapsack that forces the LP solution to commit each item to begin at a time, and places constraints on the maximum expected reward that can be obtained if the LP begins an item at a particular time. We also additionally add a family of consistency constraints for each sub-Knapsack of sizes in $\{1, 2, \ldots, B\}$. While these are completely redundant for the problem without correlations, they are crucial in resolving the contention issue. Incorporating item cancellations into Stochastic Knapsack can be done by adapting the flow-like LPs from earlier works on MABs.

3.5 Correlated Stochastic Knapsack without Cancellation

We begin by considering the Stochastic Knapsack problem (StocK), when the item rewards may be correlated with its size. In this section, we do not allow item cancellations prematurely, i.e., once the algorithm commits to running an item, it has to wait for the item to complete. As mentioned earlier, this generalizes the problem studied by Dean et al. [41] who assume that the rewards are independent of the size of the item.

3.5.1 Problem Definitions and Notation

We are given a knapsack of total budget B and a collection of n stochastic items. For any item $i \in [1, n]$, we are given a probability distribution over (size, reward) pairs specified as follows: for each integer value of $t \in [1, B]$, the tuple $(\pi_{i,t}, R_{i,t})$ denotes the probability $\pi_{i,t}$ that item i has a size t, and the corresponding reward is $R_{i,t}$; The interpretation for $R_{i,t}$ is the conditional expected reward of item i given that its size is t. Note that the $(\mathsf{size}, \mathsf{reward})$ pairs for two different items are still independent of each other.

An adaptive algorithm can take the following actions at the end of each timestep;

(i) an item may complete at a certain size (giving us the corresponding reward), and the algorithm may choose a new item to start, or

(ii) the knapsack becomes full, at which point the algorithm stops, and the item being processed does not fetch any reward.

The objective is to maximize the total expected reward obtained from all completed items.

3.5.2 LP Relaxation

The LP formulation in [41] was (essentially) a knapsack LP where the sizes of items are replaced by the expected sizes, and the rewards are replaced by the expected rewards. While this was sufficient when an item's reward is fixed (or chosen randomly but independent of its size), we showed an example in Section 3.4.2 where their LP (and in fact, the class of more general LPs used for approximating MAB problems) have a large integrality gap. Moreover, as mentioned in Section 3.4, the reason why local LPs don't work is that there could be high contention for being scheduled early (i.e., there could be a large number of items which all fetch reward if they instantiate to a large size, but these events occur with low probability). In order to capture this contention, we write a global time-indexed LP formulation.

The variable $x_{i,t} \in [0, 1]$ indicates that item i is scheduled at (global) time t; S_i denotes the random variable for the size of item i, and $\mathsf{ER}_{i,t} = \sum_{s \le B-t} \pi_{i,s} R'_{i,s}$ captures the expected reward that can be obtained from item i *if it begins* at time t; (no reward is obtained for sizes that cannot fit the (remaining) budget.)

$$\begin{array}{lll} \max \sum_{i,t} \mathsf{ER}_{i,t} \cdot x_{i,t} & & (\mathsf{LP}_{\mathsf{NoCancel}}) \\ \sum_t x_{i,t} \leq 1 & \forall i & (3.3) \\ \sum_{i,t' \leq t} x_{i,t'} \cdot \mathbb{E}[\min(S_i, t)] \leq 2t & \forall t \in [B] & (3.4) \\ x_{i,t} \in [0,1] & \forall t \in [B], \forall i & (3.5) \end{array}$$

While the size of the above LP (and the running time of the rounding algorithm below) polynomially depend on B, i.e., pseudo-polynomial, it is possible to write a compact (approximate) LP and then round it; details on the polynomial time implementation appear in Section 3.7.1.

Notice the constraints involving the *truncated random variables* in equation (3.4): these are crucial for showing the correctness of the rounding algorithm StocK-NoCancel. Furthermore, the ideas used here will appear subsequently in the MAB algorithm later; for MAB, even though we can't explicitly enforce such a constraint in the LP, we will end up inferring a similar family of inequalities from a near-optimal LP solution.

Lemma 3.5.1 *The formulation* $\mathsf{LP}_{\mathsf{NoCancel}}$ *is valid for the* StocK *problem when cancellations are not permitted, and has objective value* $\mathsf{OPT}_{\mathsf{LP}} \geq \mathsf{OPT}$*, where* OPT *is the expected profit of an optimal adaptive policy.*

Proof: Consider an optimal policy OPT and let $x^*_{i,t}$ denote the probability that item i is scheduled at time t. We first show that $\{x^*\}$ is a feasible solution for the LP formulation $\mathsf{LP}_{\mathsf{NoCancel}}$. It is easy to see that constraints (3.3) and (3.5) are satisfied. To prove that (3.4) are also satisfied, consider some $t \in [B]$ and some run (over random choices of item sizes) of the optimal policy. Let $\mathbf{1}^{\mathsf{sched}}_{i,t'}$ be indicator variable that item i is scheduled at time t' and let $\mathbf{1}^{\mathsf{size}}_{i,s}$ be the indicator variable for whether the size of item i is s. Also, let L_t be the random variable indicating the last item scheduled at or before time t. Notice that L_t is the only item scheduled before or at time t whose execution may go over time t. Therefore, we get that

$$\sum_{i \neq L_t} \sum_{t' \leq t} \sum_{s \leq B} \mathbf{1}^{\mathsf{sched}}_{i,t'} \cdot \mathbf{1}^{\mathsf{size}}_{i,s} \cdot s \leq t.$$

Including L_t in the summation and truncating the sizes by t, we immediately obtain

$$\sum_{i} \sum_{t' \leq t} \sum_{s} \mathbf{1}^{\mathsf{sched}}_{i,t'} \cdot \mathbf{1}^{\mathsf{size}}_{i,s} \cdot \min(s,t) \leq 2t.$$

Now, taking expectation (over all of OPT's sample paths) on both sides and using linearity of expectation we have

$$\sum_{i} \sum_{t' \leq t} \sum_{s} \mathbb{E}\left[\mathbf{1}^{\mathsf{sched}}_{i,t'} \cdot \mathbf{1}^{\mathsf{size}}_{i,s}\right] \cdot \min(s,t) \leq 2t.$$

However, because OPT decides whether to schedule an item before observing the size it instantiates to, we have that $\mathbf{1}^{\mathsf{sched}}_{i,t'}$ and $\mathbf{1}^{\mathsf{size}}_{i,s}$ are independent random variables; hence, the LHS above can be re-written as

$$\begin{aligned}
&\sum_i \sum_{t' \le t} \sum_s \mathbb{P}[\mathbf{1}^{\mathsf{sched}}_{i,t'} = 1 \wedge \mathbf{1}^{\mathsf{size}}_{i,s} = 1] \min(s,t) \\
&= \sum_i \sum_{t' \le t} \mathbb{P}[\mathbf{1}^{\mathsf{sched}}_{i,t'} = 1] \sum_s \mathbb{P}[\mathbf{1}^{\mathsf{size}}_{i,s} = 1] \min(s,t) \\
&= \sum_i \sum_{t' \le t} x^*_{i,t'} \cdot \mathbb{E}[\min(S_i, t)]
\end{aligned}$$

Hence constraints (3.4) are satisfied. Now we argue that the expected reward of OPT is equal to the value of the solution x^*. Let O_i be the random variable denoting the reward obtained by OPT from item i. Again, due to the independence between OPT scheduling an item and the size it instantiates to, we get that the expected reward that OPT gets from executing item i at time t is

$$\mathbb{E}[O_i | \mathbf{1}^{\mathsf{sched}}_{i,t} = 1] = \sum_{s \le B-t} \pi_{i,s} R_{i,s} = \mathsf{ER}_{i,t}.$$

Thus the expected reward from item i is obtained by considering all possible starting times for i:

$$\mathbb{E}[O_i] = \sum_t \mathbb{P}[\mathbf{1}^{\mathsf{sched}}_{i,t} = 1] \cdot \mathbb{E}[O_i | \mathbf{1}^{\mathsf{sched}}_{i,t} = 1] = \sum_t \mathsf{ER}_{i,t} \cdot x^*_{i,t}.$$

This shows that $\mathsf{LP}_{\mathsf{NoCancel}}$ is a valid formulation for our problem and completes the proof of the lemma. ■

3.5.3 Rounding Algorithm

Now, given an optimal fractional solution, our rounding algorithm StocK-NoCancel (Algorithm 5) is very simple: (i) pick a random start deadline for each item according to the corresponding distribution in the optimal LP solution, and (ii) play the items in order of the (random) deadlines. To ensure that the budget is not violated, we also drop each item independently with some constant probability.

Algorithm 5 Algorithm StocK-NoCancel

```
1: for each item $i$, **assign** a random start deadline $D_i = t$ with probability $\frac{x^*_{i,t}}{4}$; with probability
   $1 - \sum_t \frac{x^*_{i,t}}{4}$, completely ignore item $i$ ($D_i = \infty$ in this case).
2: **for** $j$ from 1 to $n$ **do**
3:     Consider the item $i$ which has the $j$th smallest deadline (and $D_i \neq \infty$)
4:     **if** the items added so far to the knapsack occupy at most $D_i$ space **then**
5:         add $i$ to the knapsack.
```

Notice that the rounding strategy obtains reward from all items which are not dropped and which do not fail (i.e. they can start being scheduled before the sampled deadline D_i in Step 1); we now bound the failure probability.

Lemma 3.5.2 *For every i, $\mathbb{P}(i \text{ fails} \mid D_i = t) \leq 1/2$.*

Proof: Consider an item i and time $t \neq \infty$ and condition on the event that $D_i = t$. Let us consider the execution of the algorithm when it tries to add item i to the knapsack in steps 3-5. Now, let Z be a random variable denoting *how much of the interval* $[0, t]$ of the knapsack is occupied by previously scheduling items, at the time when i is considered for addition; since i does not fail when $Z < t$, it suffices to prove that $\mathbb{P}(Z \geq t) \leq 1/2$.

For some item $j \neq i$, let $\mathbf{1}_{D_j \leq t}$ be the indicator variable that $D_j \leq t$; notice that by the order in which algorithm StocK-NoCancel adds items into the knapsack, it is also the indicator that j was considered before i. In addition, let $\mathbf{1}^{\mathsf{size}}_{j,s}$ be the indicator variable that $S_j = s$. Now, if Z_j denotes the total amount of the interval $[0, t]$ that that j occupies, we have

$$Z_j \leq \mathbf{1}_{D_j \leq t} \sum_s \mathbf{1}^{\mathsf{size}}_{j,s} \min(s, t).$$

Now, using the independence of $\mathbf{1}_{D_j \leq t}$ and $\mathbf{1}^{\mathsf{size}}_{j,s}$, we have

$$\mathbb{E}[Z_j] \leq \mathbb{E}[\mathbf{1}_{D_j \leq t}] \cdot \mathbb{E}[\min(S_j, t)] = \tfrac{1}{4} \textstyle\sum_{t' \leq t} x^*_{j,t'} \cdot \mathbb{E}[\min(S_j, t)] \tag{3.6}$$

Since $Z = \sum_j Z_j$, we can use linearity of expectation and the fact that $\{x^*\}$ satisfies LP constraint (3.4) to get

$$\mathbb{E}[Z] \leq \tfrac{1}{4} \textstyle\sum_j \sum_{t' \leq t} x^*_{j,t'} \cdot \mathbb{E}[\min(S_j, t)] \leq \tfrac{t}{2} \ .$$

To conclude the proof of the lemma, we apply Markov's inequality to obtain $\mathbb{P}(Z \geq t) \leq 1/2$. ■

To complete the analysis, we use the fact that any item chooses a random start time $D_i = t$ with probability $x^*_{i,t}/4$; conditioned on this, it is added to the knapsack with probability at least $1/2$ from Lemma 3.5.2; in this case, we get expected reward at least $\mathsf{ER}_{i,t}$. The theorem below follows by linearity of expectations.

Theorem 3.5.3 *The expected reward of algorithm* StocK-NoCancel *is at least $\frac{1}{8}$ of* $\mathsf{OPT}_{\mathsf{LP}}$.

Proof: Let add_i denote the event that item i was added to the knapsack in Step 5. Also, let V_i denote the random variable corresponding to the reward that our algorithm gets from item i.

Clearly if item i has $D_i = t$ and was added, then it is added to the knapsack before time t. In this case it is easy to see that $\mathbb{E}[V_i \mid \mathsf{add}_i \wedge (D_i = t)] \geq R_{i,t}$ (because its random size is independent of when the algorithm started it). Moreover, from the previous lemma we have that $\mathbb{P}(\mathsf{add}_i \mid (D_i = t)) \geq 1/2$ and from Step 1 we have $\mathbb{P}(D_i = t) = \frac{x^*_{i,t}}{4}$; hence $\mathbb{P}(\mathsf{add}_i \wedge (D_i = t)) \geq x^*_{i,t}/8$. Finally adding over all possibilities of t, we lower bound the expected value of V_i by

$$\mathbb{E}[V_i] \geq \sum_t \mathbb{E}[V_i \mid \mathsf{add}_i \wedge (D_i = t)] \cdot \mathbb{P}(\mathsf{add}_i \wedge (D_i = t)) \geq \frac{1}{8} \sum_t x^*_{i,t} R_{i,t}.$$

Finally, linearity of expectation over all items shows that the total expected reward of our algorithm is at least $\frac{1}{8} \cdot \sum_{i,t} x^*_{i,t} R_{i,t} = \mathsf{OPT}_{\mathsf{LP}}/8$, thus completing the proof. ■

3.6 Correlated Stochastic Knapsack with Cancellations

In this section, we present our algorithm for Stochastic Knapsack (StocK) where we allow correlations between rewards and sizes, and also allow cancellation of items. Recall that the example in Section 3.4.1 shows that there can be an arbitrarily large gap in the expected profit between strategies that can cancel items and those that can't. Hence we need to write new LPs to capture the benefit of cancellation, which we do in the following manner.

Consider any item i: we can create two items from it, the "early" version of the item, where we discard profits from any instantiation where the size of the item is more than $B/2$, and the "late" version of the item where we discard profits from instantiations of size at most $B/2$. Hence, we can get at least half the optimal value by flipping a fair coin and either collecting rewards from either the early or late versions of items, based on the outcome. For the first kind, we make use of the fact that contention is not really an issue (since rewards are only for small size instantiations) and write flow-like LPs akin to those for MAB problems [61]. For the second kind, we argue that cancellations don't help, and hence we can reduce it to StocK without cancellations (considered above).

Case I: Items with Early Rewards

We begin with the setting in which only small-size instantiations of items may fetch reward, i.e., the rewards $R_{i,t}$ of every item i are assumed to be 0 for $t > B/2$. In the following LP formulation LP_S, $v_{i,t} \in [0,1]$ tries to capture the probability with which OPT will process item i for *at least* t timesteps[1], $s_{i,t} \in [0,1]$ is the probability that OPT stops processing item i *exactly* at t timesteps. The time-indexed formulation causes the algorithm to have running times of $\mathrm{poly}(B)$—however, it is easy to write compact (approximate) LPs and then round them; we describe the necessary changes to obtain an algorithm with running time $\mathrm{poly}(n, \log B)$ in Section 3.8.1.

$$\max \textstyle\sum_{1\le t\le B/2} \sum_{1\le i\le n} v_{i,t} \cdot R_{i,t} \frac{\pi_{i,t}}{\sum_{t'\ge t}\pi_{i,t'}} \qquad (\mathsf{LP}_S)$$

$$v_{i,t} = s_{i,t} + v_{i,t+1} \qquad \forall\, t \in [0,B],\ i \in [n] \qquad (3.7)$$

$$s_{i,t} \ge \frac{\pi_{i,t}}{\sum_{t'\ge t}\pi_{i,t'}} \cdot v_{i,t} \qquad \forall\, t \in [0,B],\ i \in [n] \qquad (3.8)$$

$$\textstyle\sum_{i\in[n]} \sum_{t\in[0,B]} t \cdot s_{i,t} \le B \qquad (3.9)$$

$$v_{i,0} = 1 \qquad \forall\, i \qquad (3.10)$$

$$v_{i,t}, s_{i,t} \in [0,1] \qquad \forall\, t \in [0,B],\ i \in [n] \qquad (3.11)$$

Theorem 3.6.1 *The linear program* (LP_S) *is a valid formulation for the* StocK *problem, and hence the optimal value* $\mathsf{OPT}_{\mathsf{LP}}$ *of the LP is at least the total expected reward* OPT *of an optimal solution.*

Proof: Consider an optimal solution OPT and let $v^*_{i,t}$ and $s^*_{i,t}$ denote the probability that OPT processes item i for at least t timesteps, and the probability that OPT stops processing item i at exactly t timesteps. We will now show that all the constraints of LP_S are satisfied one by one.

[1]In the following two sections, we use the word timestep to refer to processing one unit of some item.

To this end, let R_i denote the random variable (over different executions of OPT) for the amount of processing done on item i. Notice that $\mathbb{P}[R_i \geq t] = \mathbb{P}[R_i \geq (t+1)] + \mathbb{P}[R_i = t]$. But now, by definition we have $\mathbb{P}[R_i \geq t] = v^*_{i,t}$ and $\mathbb{P}[R_i = t] = s^*_{i,t}$. This shows that $\{v^*, s^*\}$ satisfies these constraints.

For the next constraint, observe that conditioned on OPT running an item i for at least t time steps, the probability of item i stopping due to its size having instantiated to exactly equal to t is $\pi_{i,t}/\sum_{t' \geq t} \pi_{i,t'}$, i.e., $\mathbb{P}[R_i = t \mid R_i \geq t] \geq \pi_{i,t}/\sum_{t' \geq t} \pi_{i,t'}$. This shows that $\{v^*, s^*\}$ satisfies constraints (3.8).

Finally, to see why constraint (3.9) is satisfied, consider any particular run of the optimal algorithm and let $\mathbf{1}^{stop}_{i,t}$ denote the indicator random variable of the event $R_i = t$. Then we have

$$\sum_i \sum_t \mathbf{1}^{stop}_{i,t} \cdot t \leq B$$

Now, taking expectation over all runs of OPT and using linearity of expectation and the fact that $\mathbb{E}[\mathbf{1}^{stop}_{i,t}] = s^*_{i,t}$, we get constraint (3.9). As for the objective function, we again consider a particular run of the optimal algorithm and let $\mathbf{1}^{proc}_{i,t}$ now denote the indicator random variable for the event $(R_i \geq t)$, and $\mathbf{1}^{size}_{i,t}$ denote the indicator variable for whether the size of item i is instantiated to exactly t in this run. Then we have the total reward collected by OPT in this run to be exactly

$$\sum_i \sum_t \mathbf{1}^{proc}_{i,t} \cdot \mathbf{1}^{size}_{i,t} \cdot R_{i,t}$$

Now, we simply take the expectation of the above random variable over all runs of OPT, and then use the following fact about $\mathbb{E}[\mathbf{1}^{proc}_{i,t}\mathbf{1}^{size}_{i,t}]$:

$$\begin{aligned}
\mathbb{E}[\mathbf{1}^{proc}_{i,t}\mathbf{1}^{size}_{i,t}] &= \mathbb{P}[\mathbf{1}^{proc}_{i,t} = 1 \wedge \mathbf{1}^{size}_{i,t} = 1] \\
&= \mathbb{P}[\mathbf{1}^{proc}_{i,t} = 1]\, \mathbb{P}[\mathbf{1}^{size}_{i,t} = 1 \mid \mathbf{1}^{proc}_{i,t} = 1] \\
&= v^*_{i,t} \frac{\pi_{i,t}}{\sum_{t' \geq t} \pi_{i,t'}}
\end{aligned}$$

We thus get that the expected reward collected by OPT is exactly equal to the objective function value of the LP formulation for the solution (v^*, s^*). ■

3.6.1 Rounding Algorithm

Our rounding algorithm is very natural, and simply tries to mimic the probability distribution (over when to stop each item) as suggested by the optimal LP solution. In order to make sure the knapsack is not violated with constant probability, we introduce some damping in the selection probabilities up front. Let (v^*, s^*) denote an optimal fractional solution.

Notice that while we let the algorithm proceed even if its budget is violated, we will collect reward only from items that complete before time B. This is purely for ease of analysis. In Lemma 3.6.2 below, we show that for any item that is not dropped in step 2, its probability distribution over stopping times is identical to the distribution $\{s^*_{i,t}\}$. We then use this to argue that the expected reward of our algorithm is $\Omega(\mathsf{OPT}_{\mathsf{LP}})$.

Algorithm 6 Algorithm StocK-Small

```
1: for each item $i$ do
2:     ignore $i$ with probability $3/4$ (i.e., do not schedule it at all).
3:     for $0 \leq t \leq B/2$ do
4:         cancel item $i$ at this step with probability $\frac{s^*_{i,t}}{v^*_{i,t}} - \frac{\pi_{i,t}}{\sum_{t' \geq t} \pi_{i,t'}}$ and continue to next item.
5:         process item $i$ for its $(t+1)^{st}$ timestep.
6:         if item $i$ terminates after being processed for exactly $(t+1)$ timesteps then
7:             collect a reward of $R_{i,t+1}$ from this item; continue onto next item;
```

Lemma 3.6.2 *Consider item i that was not dropped in step 2, Then, for any timestep $t \geq 0$, the following hold:*

(i) The probability (including cancellation& completion) of stopping at timestep t for item i is $s^*_{i,t}$.

(ii) The probability that item i gets processed for its $(t+1)^{st}$ timestep is exactly $v^*_{i,t+1}$

(iii) If item i has been processed for $(t+1)$ timesteps, the probability of completing successfully at timestep $(t+1)$ is $\pi_{i,t+1}/\sum_{t' \geq t+1} \pi_{i,t'}$

Proof: The proof works by induction. For the base case, consider $t = 0$. Clearly, this item is forcefully canceled in step 4 of Algorithm 6 StocK-Small (in the iteration with $t = 0$) with probability $s^*_{i,0}/v^*_{i,0} - \pi_{i,0}/\sum_{t' \geq 0} \pi_{i,t'}$. But since $\pi_{i,0}$ was assumed to be 0 and $v^*_{i,0}$ is 1, this quantity is exactly $s^*_{i,0}$, and this proves property (i). For property (ii), item i is processed for its $\mathbf{1}^{st}$ timestep if it did not get forcefully canceled in step 4. This therefore happens with probability $1 - s^*_{i,0} = v^*_{i,0} - s^*_{i,0} = v^*_{i,1}$. For property (iii), conditioned on the fact that it has been processed for its $\mathbf{1}^{st}$ timestep, clearly the probability that its (unknown) size has instantiated to 1 is exactly $\pi_{i,1}/\sum_{t' \geq 1} \pi_{i,t'}$. When this happens, the item stops in step 7, thereby establishing the base case.

Assuming this property holds for every timestep until some fixed value $t - 1$, we show that it holds for t; the proofs are very similar to the base case. Assume item i was processed for the t^{th} timestep (this happens w.p $v^*_{i,t}$ from property (ii) of the induction hypothesis). Then from property (iii), the probability that this item completes at this timestep is exactly $\pi_{i,t}/\sum_{t' \geq t} \pi_{i,t'}$. Furthermore, it gets forcefully canceled in step 4 with probability $s^*_{i,t}/v^*_{i,t} - \pi_{i,t}/\sum_{t' \geq t} \pi_{i,t'}$. Thus the total probability of stopping at time t, assuming it has been processed for its t^{th} timestep is exactly $s^*_{i,t}/v^*_{i,t}$; unconditionally, the probability of stopping at time t is hence $s^*_{i,t}$.

Property (ii) follows as a consequence of Property (i), because the item is processed for its $(t+1)^{st}$ timestep only if it did not stop at timestep t. Therefore, conditioned on being processed for the t^{th} timestep, it continues to be processed with probability $1 - s^*_{i,t}/v^*_{i,t}$. Therefore, removing the conditioning, we get the probability of processing the item for its $(t+1)^{st}$ timestep is $v^*_{i,t} - s^*_{i,t} = v^*_{i,t+1}$. Finally, for property (iii), conditioned on the fact that it has been processed for its $(t+1)^{st}$ timestep, clearly the probability that its (unknown) size has instantiated to exactly $(t+1)$ is $\pi_{i,t+1}/\sum_{t' \geq t+1} \pi_{i,t'}$. When this happens, the item stops in step 7 of the algorithm. ■

Theorem 3.6.3 *The expected reward of algorithm* StocK-Small *is at least $\frac{1}{8}$ of* $\mathsf{OPT}_{\mathsf{LP}}$.

Proof: Consider any item i. In the worst case, we process it after all other items. Then the total expected size occupied thus far is at most $\sum_{i' \neq i} \mathbf{1}_{i'}^{keep} \sum_{t \geq 0} t \cdot s^*_{i',t}$, where $\mathbf{1}_{i'}^{keep}$ is the indicator random variable denoting whether item i' is not dropped in step 2. Here we used Lemma 3.6.2 to argue that if an item i' is selected, its stopping-time distribution follows $s^*_{i',t}$. Taking expectation over the randomness in step 2, the expected space occupied thus far is at most $\sum_{i' \neq i} \frac{1}{3} \sum_{t \geq 0} t \cdot s^*_{i',t} \leq \frac{B}{4}$. Markov's inequality implies that this is at most $B/2$ with probability at least $1/2$. In this case, if item i is started (which happens w.p. $1/4$), it runs without violating the knapsack, with expected reward $\sum_{t \geq 1} v^*_{i,t} \cdot \pi_{i,t} / (\sum_{t' \geq t} \pi_{i,t'})$; the total expected reward is then at least $\sum_i \frac{1}{8} \sum_t v^*_{i,t} \pi_{i,t} / (\sum_{t' \geq t} \pi_{i,t'}) \geq \frac{\mathsf{OPT}_{\mathsf{LP}}}{8}$. ■

Case II: Items with Late Rewards

Now we handle instances in which only large-size instantiations of items may fetch reward, i.e., the rewards $R_{i,t}$ of every item i are assumed to be 0 for $t \leq B/2$. For such instances, we first argue that *cancellation is not helpful.* To see why, notice that as an algorithm processes an item for its t^{th} timestep for $t < B/2$, it gets no more information about the reward than when starting (since all rewards are at large sizes). Furthermore, there is no benefit of canceling an item once it has run for at least $B/2$ timesteps – we can't get any reward by starting some other item. As mentioned earlier, we can now appeal to the results of Section 3.5 and obtain a constant-factor approximation for the large-size instances. Finally we can combine the algorithms that handle the two different scenarios (or choose one at random and run it), and get a constant fraction of the expected reward that an optimal policy fetches.

3.7 Proofs from Section 3.5

3.7.1 Making StocK-NoCancel Fully Polynomial

Recall that our LP relaxation $\mathsf{LP}_{\mathsf{NoCancel}}$ in Section 3.5 uses a global time-indexed LP. In order to make it compact, our approach will be to group the B timeslots in $\mathsf{LP}_{\mathsf{NoCancel}}$ and show that the grouped LP has optimal value within constant factor of $\mathsf{LP}_{\mathsf{NoCancel}}$; furthermore, we show also that it can be rounded and analyzed almost identically to the original LP. To this end, consider the following LP relaxation:

$$\max \sum_i \sum_{j=0}^{\log B} \mathsf{ER}_{i,2^{j+1}} \cdot x_{i,2^j} \qquad (\mathsf{PolyLP}_L)$$

$$\sum_{j=0}^{\log B} x_{i,2^j} \leq 1 \qquad \forall i \qquad (3.12)$$

$$\sum_{i,j' \leq j} x_{i,2^{j'}} \cdot \mathbb{E}[\min(S_i, 2^{j+1})] \leq 2 \cdot 2^j \qquad \forall j \in [0, \log B] \qquad (3.13)$$

$$x_{i,2^j} \in [0,1] \qquad \forall j \in [0, \log B], \forall i \qquad (3.14)$$

The next two lemmas relate the value of (PolyLP_L) to that of the original LP $(\mathsf{LP}_{\mathsf{NoCancel}})$.

Lemma 3.7.1 *The optimum of* (PolyLP_L) *is at least half of the optimum of* $(\mathsf{LP}_{\mathsf{NoCancel}})$.

Proof: Consider a solution x for $(\mathsf{LP}_{\mathsf{NoCancel}})$ and define $\overline{x}_{i1} = x_{i,1}/2 + \sum_{t \in [2,4)} x_{i,t}/2$ and $\overline{x}_{i,2^j} = \sum_{t \in [2^{j+1}, 2^{j+2})} x_{i,t}/2$ for $1 < j \leq \log B$. It suffices to show that $\overline{x}$ is a feasible solution to (PolyLP_L) with value greater than of equal to half of the value of x.

For constraints (3.12) we have $\sum_{j=0}^{\log B} \overline{x}_{i,2^j} = \sum_{t\geq 1} x_{i,t}/2 \leq 1/2$; these constraints are therefore easily satisfied. We now show that $\{\overline{x}\}$ also satisfies constraints (3.13):

$$\sum_{i,j'\leq j} x_{i,2^{j'}} \cdot \mathbb{E}[\min(S_i, 2^{j+1})] = \sum_i \sum_{t=1}^{2^{j+2}-1} \frac{x_{i,t}\mathbb{E}[\min(S_i, 2^{j+1})]}{2}$$
$$\leq \sum_i \sum_{t=1}^{2^{j+2}-1} \frac{x_{i,t}\mathbb{E}[\min(S_i, 2^{j+2}-1)]}{2} \leq 2^{j+2}-1,$$

where the last inequality follows from feasibility of $\{x\}$.

Finally, noticing that $\mathsf{ER}_{i,t}$ is non-increasing with respect to t, it is easy to see that $\sum_i \sum_{j=0}^{\log B} \mathsf{ER}_{i,2^{j+1}} \cdot \overline{x}_{i,2^j} \geq \sum_{i,t} \mathsf{ER} i, t \cdot x_{i,t}/2$ and hence $\overline{x}$ has value greater than of equal to half of the value of x ad desired. ■

Lemma 3.7.2 *Let $\{\overline{x}\}$ be a feasible solution for* (PolyLP_L)*. Define $\{\hat{x}\}$ satisfying $\hat{x}_{i,t} = \overline{x}_{i,2^j}/2^j$ for all $t \in [2^j, 2^{j+1})$ and $i \in [n]$. Then $\{\hat{x}\}$ is feasible for* ($\mathsf{LP}_{\mathsf{NoCancel}}$) *and has value at least as large as $\{\overline{x}\}$.*

Proof: The feasibility of $\{\overline{x}\}$ directly imply that $\{\hat{x}\}$ satisfies constraints (3.3). For constraints (3.4), consider $t \in [2^j, 2^{j+1})$; then we have the following:

$$\sum_{i,t'\leq t} \hat{x}_{i,t'} \cdot \mathbb{E}[\min(S_i, t)] \leq \sum_i \sum_{j'\leq j} \sum_{t\in[2^{j'},2^{j'+1})} \frac{\overline{x}_{i,2^j}}{2^j}\mathbb{E}[\min(S_i, 2^{j+1})]$$
$$= \sum_i \sum_{j'\leq j} \overline{x}_{i,2^j}\mathbb{E}[\min(S_i, 2^{j+1})] \leq 2 \cdot 2^j \leq 2t.$$

Finally, again using the fact that $\mathsf{ER}_{i,t}$ is non-increasing in t we get that the value of $\{\hat{x}\}$ is

$$\sum_{i,t} \mathsf{ER}_{i,t} \cdot \hat{x}_{i,t} = \sum_i \sum_{j=0}^{\log B} \sum_{t\in[2^j,2^{j+1})} \mathsf{ER}_{i,t}\frac{\overline{x}_{i,2^j}}{2^j} \geq \sum_i \sum_{j=0}^{\log B} \sum_{t\in[2^j,2^{j+1})} \mathsf{ER}_{i,2^{j+1}}\frac{\overline{x}_{i,2^j}}{2^j} = \sum_i \sum_{j=0}^{\log B} \mathsf{ER}_{i,2^{j+1}}\overline{x}_{i,2^j},$$

which is then at least as large as the value of $\{\overline{x}\}$. This concludes the proof of the lemma. ■

The above two lemmas show that the PolyLP_L has value close to that of $\mathsf{LP}_{\mathsf{NoCancel}}$: let's now show that we can simulate the execution of Algorithm StocK-Large just given an optimal solution $\{\overline{x}\}$ for (PolyLP_L). Let $\{\hat{x}\}$ be defined as in the above lemma, and consider the Algorithm StocK-Large applied to $\{\hat{x}\}$. By the definition of $\{\hat{x}\}$, here's how to execute Step 1 (and hence the whole algorithm) in polynomial time: we obtain $D_i = t$ by picking $j \in [0, \log B]$ with probability $\overline{x}_{i,2^j}$ and then selecting $t \in [2^j, 2^{j+1})$ uniformly; notice that indeed $D_i = t$ (with $t \in [2^j, 2^{j+1})$) with probability $\overline{x}_{i,2^j}/2^j = \hat{x}_{i,t}$.

Using this observation we can obtain a $1/16$ approximation for our instance $\mathcal{I}$ in polynomial time by finding the optimal solution $\{\overline{x}\}$ for (PolyLP_L) and then running Algorithm StocK-Large over $\{\hat{x}\}$ as described in the previous paragraph. Using a direct modification of Theorem 3.5.3 we have that the strategy obtained has expected reward at least at large as $1/8$ of the value of $\{\hat{x}\}$, which by Lemmas 3.7.1 and 3.7.2 (and Lemma 3.5.1) is within a factor of $1/16$ of the optimal solution for $\mathcal{I}$.

3.8 Proofs from Section 3.6

3.8.1 StocK with Small Sizes: A Fully Polytime Algorithm

The idea is to quantize the possible sizes of the items in order to ensure that LP LP_S has polynomial size, then obtain a good strategy (via Algorithm StocK-Small) for the transformed instance, and finally to show that this strategy is actually almost as good for the original instance.

Consider an instance $\mathcal{I} = (\pi, R)$ where $R_{i,t} = 0$ for all $t > B/2$. Suppose we start scheduling an item at some time; instead of making decisions of whether to continue or cancel an item at each subsequent time step, we are going to do it in time steps which are powers of 2. To make this formal, define instance $\overline{\mathcal{I}} = (\overline{\pi}, \overline{R})$ as follows: set $\overline{\pi}_{i,2^j} = \sum_{t\in[2^j,2^{j+1})} \pi_{i,t}$ and $\overline{R}_{i,2^j} = (\sum_{t\in[2^j,2^{j+1})} \pi_{i,t}R_{i,t})/\overline{\pi}_{i,2^j}$ for all $i \in [n]$ and $j \in \{0, 1, \ldots, \lfloor \log B \rfloor\}$. The instances are coupled in the natural way: the size of item i in the instance $\overline{\mathcal{I}}$ is 2^j iff the size of item i in the instance $\mathcal{I}$ lies in the interval $[2^j, 2^{j+1})$.

In Section 3.6, a *timestep* of an item has duration of 1 time unit. However, due to the construction of $\overline{\mathcal{I}}$, it is useful to consider that the t^{th} time step of an item has duration 2^t; thus, an item can only complete at its 0^{th}, 1^{st}, 2^{nd}, etc. timesteps. With this in mind, we can write an LP analogous to (LP_S):

$$\max \sum_{1\le j\le \log(B/2)} \sum_{1\le i\le n} v_{i,2^j} \cdot \overline{R}_{i,2^j} \frac{\overline{\pi}_{i,2^j}}{\sum_{j'\ge j} \pi_{i,2^{j'}}} \qquad (\mathsf{PolyLP}_S)$$

$$v_{i,2^j} = s_{i,2^j} + v_{i,2^j+1} \qquad \forall\, j \in [0, \log B],\ i \in [n] \qquad (3.15)$$

$$s_{i,2^j} \ge \frac{\overline{\pi}_{i,2^j}}{\sum_{j'\ge j} \overline{\pi}_{i,2^{j'}}} \cdot v_{i,2^j} \qquad \forall\, t \in [0, \log B],\ i \in [n] \qquad (3.16)$$

$$\sum_{i\in[n]} \sum_{j\in[0,\log B]} 2^j \cdot s_{i,2^j} \le B \qquad (3.17)$$

$$v_{i,0} = 1 \qquad \forall\, i \qquad (3.18)$$

$$v_{i,2^j}, s_{i,2^j} \in [0,1] \qquad \forall\, j \in [0, \log B],\ i \in [n] \qquad (3.19)$$

Notice that this LP has size polynomial in the size of the instance $\mathcal{I}$.

Consider the LP (LP_S) with respect to the instance $\mathcal{I}$ and let (v, s) be a feasible solution for it with objective value z. Then define $(\overline{v}, \overline{s})$ as follows: $\overline{v}_{i,2^j} = v_{i,2^j}$ and $\overline{s}_{i,2^j} = \sum_{t\in[2^j,2^{j+1})} s_{i,j}$. It is easy to check that $(\overline{v}, \overline{s})$ is a feasible solution for (PolyLP_S) with value at least z, where the latter uses the fact that $v_{i,t}$ is non-increasing in t. Using Theorem 3.6.1 it then follows that the optimum of (PolyLP_S) with respect to $(\overline{\pi}, \overline{R})$ is at least as large as the reward obtained by the optimal solution for the Stochastic Knapsack instance (π, R).

Let $(\overline{v}, \overline{s})$ denote an optimal solution of (PolyLP_S). Notice that with the redefined notion of timesteps we can naturally apply Algorithm StocK-Small to the LP solution $(\overline{v}, \overline{s})$. Moreover, Lemma 3.6.2 still holds in this setting. Finally, modify Algorithm StocK-Small by ignoring items with probability $1 - 1/8 = 7/8$ (instead of 3/4) in Step 2 (we abuse notation slightly and shall refer to the modified algorithm also as StocK-Small) and notice that Lemma 3.6.2 still holds.

Consider the strategy $\overline{\mathbb{S}}$ for $\overline{\mathcal{I}}$ obtained from Algorithm StocK-Small. We can obtain a strategy

$\mathbb{S}$ for $\mathcal{I}$ as follows: whenever $\mathbb{S}$ decides to process item i of $\overline{\mathcal{I}}$ for its jth timestep, we decide to continue item i of $\mathcal{I}$ while it has size from 2^j to $2^{j+1} - 1$.

Lemma 3.8.1 *Strategy $\mathbb{S}$ is a $1/16$ approximation for $\mathcal{I}$.*

Proof: Consider an item i. Let $\overline{O}$ be the random variable denoting the total size occupied before strategy $\overline{\mathbb{S}}$ starts processing item i and similarly let O denote the total size occupied before strategy $\mathbb{S}$ starts processing item i. Since Lemma 3.6.2 still holds for the modified algorithm StocK-Small, we can proceed as in Theorem 3.6.3 and obtain that $\mathbb{E}[\overline{O}] \leq B/8$. Due to the definition of $\mathbb{S}$ we can see that $O \leq 2\overline{O}$ and hence $\mathbb{E}[O] \leq B/4$. From Markov's inequality we obtain that $\mathbb{P}(O \geq B/2) \leq 1/2$. Noticing that i is started by $\mathbb{S}$ with probability $1/8$ we get that the probability that i is started and there is at least $B/2$ space left on the knapsack at this point is at least $1/16$. Finally, notice that in this case $\overline{\mathbb{S}}$ and $\mathbb{S}$ obtain the same expected value from item i, namely $\sum_j \overline{v}_{i,2^j} \cdot \overline{R}_{i,2^j} \frac{\overline{\pi}_{i,2^j}}{\sum_{j' \geq j} \overline{\pi}_{i,2^{j'}}}$. Thus $\mathbb{S}$ get expected value at least that of the optimum of (PolyLP$_S$), which is at least the value of the optimal solution for $\mathcal{I}$ as argued previously. ■

3.9 Conclusions

In this chapter, we presented constant-factor approximation algorithms (and adaptivitiy gaps) for the Stochastic Knapsack problem with correlated rewards, both in the models where preemptions are allowed, and in that where they aren't. Despite the progress we have made in our understanding of the stochastic version of the Knapsack problem, a fundamentcal question that remains open is that of the limits of approximation, i.e., does the (basic uncorrelated) StocK problem admit a PTAS, or is it APX-hard? The latter result would establish a nice separation with the deterministic Knapsack problem, which indeed admits an FPTAS. As of this writing, the best known approximation algorithm for the basic StocK problem is a factor of $(2 + \varepsilon)$ [17].

Chapter 4

Multi-Armed Bandits

The Multi-Armed Bandits problem (MAB) is a classical problem which draws the interest of researchers in several areas of science (Statistics, Machine Learning, and of course, TCS) with research dating back to the 1950s. In the basic version of the MAB problem, there are a collection of n coins, each having some (unknown) bias of turning up heads (which counts as giving us unit reward). We have some a priori distributional information about the different bias values. At each time, the algorithm can choose a coin to toss, and collects a reward if the coin turns up heads. The goal is to decide on which coin to toss at each time, in order to maximize the total expected number of heads seen over a horizon of, say, time B. Notice that the algorithm can update the distribution of each coin's bias to the one *conditioned on the observed history*, thereby learning the true bias better over time.

The MAB problem has several different forms and variants and is widely studied by the several aforementioned fields owing to the fact that it models the *exploration-exploitation* trade-off, i.e., an agent that simultaneously attempts to acquire new knowledge (exploration) and to optimize its decisions based on existing knowledge (exploitation). There are many practical applications, including but not restricted to (i) clinical trials for investigating the effects of different experimental treatments while minimizing patient losses, (ii) adaptive routing to minimizing delays in a network, etc.

Since MAB aptly captures the exploration-exploitation trade-off so well, and both the application areas listed above require balancing the notions of reward maximization based on the knowledge already acquired, and attempting new actions to further increase knowledge, the MAB models are often deployed to design good policies/strategies for these problems.

As mentioned above, this problem has received much attention in many different areas of research. However, most attempts at studying the problem can be classified as being from an *information-theoretic* point of view, i.e., how well does a particular algorithm do when compared to the best policy *in hindsight*, i.e., one which knows the true bias of each coin (and therefore simply tosses the coin with highest bias at all times). This work, however, focuses on the problem from a *computational complexity* perspective: how well does an algorithm do when compared to the best possible algorithm (which is also on an equal playing-field information wise). Formally, we study the following problem.

The MAB Problem Definition. We are given a collection of n arms; each arm is associated with a Markov Chain (given completely as input). Each state in an arm's Markov Chain is also associated with some reward. At each timestep, the algorithm is given the current states that each of the Markov Chains are in, and it must decide which arm to play. The chosen arm then transitions into a new state as dictated by the Markov Chain, and the algorithm collects the reward from the new state. Our task is to design an adaptive algorithm which maximizes the total expected reward it can obtain over a horizon of B plays.

Moreover, the benchmark for measuring the performance of an algorithm is the ratio $\mathbb{E}[\mathsf{OPT}]/\mathbb{E}[\mathsf{Alg}]$, where OPT is an optimal adaptive algorithm for the problem. Here, the individual expectations are over the randomness inherent in the problem (e.g., the random transitions of the Markov Chains) and possibly any randomization used by the algorithm.

While the model we study does not directly reduce to the coin-bias problem mentioned above, the intuition on why they are related is as follows: Each arm's Markov chain corresponds to how our prior distribution on the bias is updated, depending on that coin turning up heads or tails. The states therefore correspond to different outcome sequences of the coin, and the reward at the corresponding states is the *expected conditional bias of the coin given the sequence of outcomes*. In this manner, most earlier works on MAB compare the performance of an algorithm with one that knows the actual bias of each coin. In our work however, we compare the performance of our algorithm against one which also has to learn the bias of each coin by tossing them.

4.1 Related Work

The general area of learning with costs is a rich and diverse one with a variety of papers studying different forms of MAB problems. The interested reader is referred to [14, 52] for some pointers. Since the results of this thesis concern with the *computational aspects* of the problem, we now focus on Approximation algorithms for the MAB problems. The first result in this direction is the work of Guha and Munagala [59], who gave LP-rounding algorithms for some budgeted learning problems. Further papers by these authors [47, 54, 58, 60, 61] and by Goel et al. [55] give improvements, and also relate the LP-based techniques and index-based policies (which are often used in regret algorithms) and also give new index policies. There are also variants which considers a non-uniform cost of switching between playing different arms [60], and also those where there is delayed feedback of playing an arm [62].

However, an important distinction between all these papers and ours is that they all assume what is known as the *Martingale property* of the different Markov Chains: if an arm is some state u, one pull of this arm would bring an expected payoff equal to the payoff of state u itself.

The motivation for such an assumption comes from the application of MABs in Bayesian learning: each arm i is associated with an unknown distribution $\mathcal{D}_i$; the states of its Markov chain correspond to different priors for $\mathcal{D}_i$ based on previous observations. We start with some known prior for $\mathcal{D}_i$ (the initial state of the Markov chain) and every pull gives a sample from $\mathcal{D}_i$, which is used to update our prior (i.e., causes a state transition)—the payoff of a state is the expected reward according to the conditional distribution. Under certain assumptions on the distributions and update rules, these Markov chains would satisfy the martingale property (see for instance [55]).

However, the martingale assumption does not hold in many interesting situations—the correlated Stochastic Knapsack being one such example. Perhaps more importantly, it is too restrictive when we use the arms to model objects which can *react* to our actions. Such examples appear in project allocation or marketing problems [15]: For example, arms may model costumers that require repeated "pulls" (marketing actions) before they transition to a high payoff state, while the intermediate states yield no payoff. In another example, arms could model advertisers competing for B ad impressions; pulling an arm corresponds to assigning an impression—however, each advertiser has a budget on the number of impression it is willing to pay for and the rewards are typically non-increasing (a similar model is considered in [24]), implying that such instances would violate the martingale property. On a technical side, obtaining guarantees without the martingale assumption requires new tools that depart from the LP-based methods used before our work. Our main result is then the following:

Theorem 4.1.1 *There is a polynomial-time (adaptive) randomized algorithm for the* MAB *problem, which obtains an expected reward of at least* $\Omega(1)\mathsf{OPT}$*, where* OPT *is the expected reward of an optimal adaptive algorithm for the given* MAB *instance.*

Notice that our algorithm is *adaptive*: in Section 4.3, we provide some intuition for reasons as to why this is the case, and also an example showing arbitrarily large adaptivity gaps, this showing that adaptivity is in fact, necessary to be a constant-approximation to the adaptive OPT.

4.2 Chapter Roadmap

We first begin with an overview of our techniques for the MAB problem in Section 4.3. Then in Section 4.4 we present the key details of our MAB algorithm, assuming that all the transition graphs are *treelike*. Subsequently, we furnish the complete details of this special case in Sections 4.5-4.7. Finally, we extend our algorithm to work for arbitrary transition graphs (i.e., general Markov chains) in Section 4.8.

4.3 Techniques

Obtaining approximations for MAB problems is a more complicated task than the Stochastic Knapsack problem for two crucial reasons. Firstly, cancellations are inherent in the problem formulation (i.e., any strategy may stop playing a particular arm and switch to another), and moreover, the payoff of an arm is naturally correlated with its current state. Secondly, the notion of preemptions is also a feature we did not tackle in the previous chapter on StocK— namely, the power to pause an arm at some state, run other arms, and then return to this arm and continue from where we left off.

While the first issue (of cancellations) can tackled by using more elaborate LPs with a flow-like structure that compute a probability distribution over the different times at which the LP stops playing an arm (e.g., much like the LP formulation in [59]), the latter issue is much less understood. Indeed, several papers on this topic present *non-preempting* strategies that fetch an expected reward which is a constant-factor of an optimal solution's reward, but which may violate the budget by a constant factor. Indeed, along the way, they show that there are *near-optimal* strategies which do not, in fact, switch back-and-forth between arms provided there is a slack in the budget. Then, in order to obtain a good solution without violating the budget, they critically make use of the *martingale property*—with this assumption at hand, they can truncate the last arm played to fit the budget without incurring any loss in the expected reward. However, such an idea fails without the martingale property, and all these LPs have large integrality gaps.

A major drawback with the previous LP relaxations is that the constraints are *local* for each item/arm, i.e., they track the probability distribution over how long each item/arm is processed, and there is a global constraint on the total number of pulls/Knapsack budget. We show that *no such localized solution* can be good, since they do not capture the notion of preempting an arm. Indeed, we show cases when any near-optimal strategy must repeatedly switch back-and-forth between arms—this is the crucial difference from previous work with the martingale property where there exist near-optimal strategies that never return to any arm [60, Lemma 2.1]. Hence our algorithm needs to make *highly adaptive decisions*, contrasting with previously existing index-based policies.

We overcome this by writing a time-indexed LP formulation that tracks the probability of each arm being in each of its states, at every time instant $t \in \{1, 2, \dots, B\}$. Such an LP enables our rounding scheme to extract information about when to preempt an arm and when to re-visit it based on the LP solution; in fact, these decisions will depend on the (random) outcomes of previous pulls, but the LP encodes the information for each eventuality. We believe that our techniques are fairly general and would be applicable for other problems in Stochastic optimization.

4.3.1 The Benefit of Preemption in Non-Martingale Bandits

There are n identical arms, each of them with the following (recursively defined) transition tree starting at $\rho(0)$:

When the root $\rho(j)$ is pulled for $j < m$, the following two transitions can happen:

(i) with probability $1/(n \cdot n^{m-j})$, the arm transitions to the "right-side", where if it makes $B - n(\sum_{k=0}^{j} L^k)$ plays, it will deterministically reach a state with reward n^{m-j}. All intermediate states have 0 reward.

(ii) with probability $1 - 1/(n \cdot n^{m-j})$, the arm transitions to the "left-side", where if it makes $L^{j+1} - 1$ plays, it will deterministically reach the state $\rho(j+1)$. No state along this path fetches any reward.

Finally, node $\rho(m)$ makes the following transitions when played: (i) with probability $1/n$, to a leaf state that has a reward of 1 and the arm ends there; (ii) with probability $1 - 1/n$, to a leaf state with reward of 0.

For the following calculations, assume that $B \gg L > n$ and $m \gg 0$.

Preempting Solutions. We first exhibit a preempting solution with expected reward $\Omega(m)$. The strategy plays $\rho(0)$ of all the arms until one of them transitions to the "right-side", in which case it continues to play this until it fetches a reward of n^m. Notice that any root which transitioned to the right-side can be played to completion, because the number of pulls we have used thus far is at most n (only those at the $\rho(0)$ nodes for each arm), and the size of the right-side is exactly $B - n$. Now, if all the arms transitioned to the left-side, then it plays the $\rho(1)$ of each arm until one of them transitioned to the right-side, in which case it continues playing this arm and gets a reward of n^{m-1}. Again, any root $\rho(1)$ which transitioned to the right-side *can be played* to completion, because the number of pulls we have used thus far is at most $n(1 + L)$ (for each arm, we have pulled the root $\rho(0)$, transitioned the walk of length $L - 1$ to $\rho(1)$ and then pulled $\rho(1)$), and the size of the right-side is exactly $B - n(1 + L)$. This strategy is similarly defined, recursively.

We now calculate the expected reward: if any of the roots $\rho(0)$ made a transition to the right-side, we get a reward of n^m. This happens with probability roughly $1/n^m$, giving us an expected reward of 1 in this case. If all the roots made the transition to the left-side, then at least one of the $\rho(1)$ states will make a transition to their right-side with probability $\approx 1/n^{m-1}$ in which case will will get reward of n^{m-1}, and so on. Thus, summing over the first $m/2$ such rounds, our expected reward is at least

$$\frac{1}{n^m}n^m + \left(1 - \frac{1}{n^m}\right)\frac{1}{n^{m-1}}n^{m-1} + \left(1 - \frac{1}{n^m}\right)\left(1 - \frac{1}{n^{m-1}}\right)\frac{1}{n^{m-2}}n^{m-2} + \ldots$$

Each term above is $\Omega(1)$ giving us a total of $\Omega(m)$ expected reward.

Non-Preempting Solutions. Consider any non-preempting solution. Once it has played the first node of an arm and it has transitioned to the left-side, it has to irrevocably decide if it abandons this arm or continues playing. But if it has continued to play (and made the transition of $L - 1$ steps), then it cannot get any reward from the right-side of $\rho(0)$ of any of the other arms, because $L > n$ and the right-side requires $B - n$ pulls before reaching a reward-state. Likewise, if it has decided to move from $\rho(i)$ to $\rho(i+1)$ on any arm, it cannot get *any* reward from the right-sides of $\rho(0), \rho(1), \ldots, \rho(i)$ on *any* arm due to budget constraints. Indeed, for any $i \geq 1$, to have reached $\rho(i+1)$ on any particular arm, it must have utilized $(1+L-1)+(1+L^2-1)+\ldots+(1+L^{i+1}-1)$ pulls in total, which exceeds $n(1 + L + L^2 + \ldots + L^i)$ since $L > n$. Finally, notice that if the strategy has decided to move from $\rho(i)$ to $\rho(i + 1)$ on any arm, the maximum reward that it can obtain is n^{m-i-1}, namely, the reward from the right-side transition of $\rho(i + 1)$.

Using these properties, we observe that an optimal non-preempting strategy proceeds in rounds as described next.

Strategy at round i. Choose a set N_i of n_i available arms and play them as follows: pick one of these arms, play until reaching state $\rho(i)$ and then play once more. If there is a right-side transition before reaching state $\rho(i)$, discard this arm since there is not enough budget to play until reaching a state with positive reward. If there is a right-side transition at state $\rho(i)$, play this arm until it gives reward of n^{m-i}. If there is no right-side transition and there is another arm in N_i which is still to be played, discard the current arm and pick the next arm in N_i.

In round i, at least $\max(0, n_i - 1)$ arms are discarded, hence $\sum_i n_i \leq 2n$. Therefore, the expected reward can be at most

$$\frac{n_1}{n \cdot n^m}n^m + \frac{n_2}{n \cdot n^{m-1}}n^{m-1} + \ldots + \frac{n_m}{n} \leq 2$$

4.4 Approximating MAB on Tree-like Transition Graphs

In this section, we will present our MAB algorithm, assuming that each arm's transition graph has the structure of an arborescence, i.e., a directed tree. We will subsequently relax this assumption in Section 4.8.

We begin with the formal definition of the problem: There are n *arms*: arm i has a collection of states denoted by $\mathcal{S}_i$, a starting state $\rho_i \in \mathcal{S}_i$; Without loss of generality, we assume that $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for $i \neq j$. Each arm also has a *transition graph* T_i, which is given as a polynomial-size (weighted) directed tree rooted at ρ_i. If there is an edge $u \to v$ in T_i, then the edge weight $p_{u,v}$ denotes the probability of making a transition from u to v if we play arm i when its current state is node u; hence $\sum_{v:(u,v)\in T_i} p_{u,v} = 1$. Each time we play an arm, we get a reward whose value depends on the state from which the arm is played. Let us denote the reward at a state u by r_u.

Each of the arms starts at the start state $\rho_i \in \mathcal{S}_i$. We get a reward from every state we reach, and the goal is to maximize the total expected reward, while making at most B plays across all arms. Our general framework can handle other problems (like the explore/exploit kind) as well, please refer to our paper [70] for more details.

Notation. The transition graph T_i for arm i is an out-arborescence defined on the states $\mathcal{S}_i$ rooted at ρ_i. Let $\mathsf{depth}(u)$ of a node $u \in \mathcal{S}_i$ be the depth of node u in tree T_i, where the root ρ_i has depth 0. The unique parent of node u in T_i is denoted by $\mathsf{parent}(u)$. Let $\mathcal{S} = \cup_i \mathcal{S}_i$ denote the set of all states in the instance, and $\mathsf{arm}(u)$ denote the arm to which state u belongs, i.e., the index i such that $u \in \mathcal{S}_i$. Finally, for $u \in \mathcal{S}_i$, we refer to the act of playing arm i when it is in state u as "playing state $u \in \mathcal{S}_i$", or "playing state u" if the arm is clear in context.

4.4.1 Global Time-indexed LP

Variable $z_{u,t} \in [0,1]$ indicates that the algorithm plays state $u \in \mathcal{S}_i$ at time t. For $u \in \mathcal{S}_i$ and time t, $w_{u,t} \in [0,1]$ indicates that arm i *first enters* state u at time t (happens if and only if the algorithm *played* $\mathsf{parent}(u)$ at time $t-1$ and the arm jumped to state u). The following lemma bounds the LP cost.

$$\max \textstyle\sum_{u,t} r_u \cdot z_{u,t} \qquad (\mathsf{LP}_{\mathsf{mab}})$$

$$w_{u,t} = z_{\mathsf{parent}(u),t-1} \cdot p_{\mathsf{parent}(u),u} \qquad \forall t \in [2,B],\ u \in \mathcal{S} \setminus \cup_i \{\rho_i\} \qquad (4.1)$$

$$\textstyle\sum_{t' \le t} w_{u,t'} \ge \sum_{t' \le t} z_{u,t'} \qquad \forall t \in [1,B],\ u \in \mathcal{S} \qquad (4.2)$$

$$\textstyle\sum_{u \in \mathcal{S}} z_{u,t} \le 1 \qquad \forall t \in [1,B] \qquad (4.3)$$

$$w_{\rho_i,1} = 1 \qquad \forall i \in [1,n] \qquad (4.4)$$

Lemma 4.4.1 *The optimal LP reward* $\mathsf{OPT}_{\mathsf{LP}}$ *is at least* OPT*, the expected reward of an optimal adaptive strategy.*

Proof: We convention that OPT starts playing at time 1. Let $z^*_{u,t}$ denote the probability that OPT plays state u at time t, namely, the probability that arm $\mathsf{arm}(u)$ is in state u at time t and is played at time t. Also let $w^*_{u,t}$ denote the probability that OPT "enters" state u at time t, and further let $w^*_{\rho_i,1} = 1$ for all i.

We first show that $\{z^*, w^*\}$ is a feasible solution for $\mathsf{LP}_{\mathsf{mab}}$ and later argue that its LP objective is at least OPT. Consider constraint (4.1) for some $t \in [2, B]$ and $u \in \mathcal{S}$. The probability of entering state u at time t conditioned on OPT playing state $\mathsf{parent}(u)$ at time $t-1$ is $p_{\mathsf{parent}(u),u}$. In addition, the probability of entering state u at time t conditioning on OPT not playing state $\mathsf{parent}(u)$ at time $t-1$ is zero. Since $z^*_{\mathsf{parent}(u),t-1}$ is the probability that OPT plays state $\mathsf{parent}(u)$ at time $t-1$, we remove the conditioning to obtain $w^*_{u,t} = z^*_{\mathsf{parent}(u),t-1} \cdot p_{\mathsf{parent}(u),u}$.

Now consider constraint (4.2) for some $t \in [1, B]$ and $u \in \mathcal{S}$. For any outcome of the algorithm (denoted by a sample path σ), let $\mathbf{1}^{enter}_{u',t'}$ be the indicator variable that OPT enters state u' at time t' and let $\mathbf{1}^{play}_{u',t'}$ be the indicator variable that OPT plays state u' at time t'. Since T_i is acyclic, state u is played at most once in σ and is also entered at most once in σ. Moreover, whenever u is played before or at time t, it must be that u was also entered before or at time t, and hence $\sum_{t' \le t} \mathbf{1}^{play}_{u,t'} \le \sum_{t' \le t} \mathbf{1}^{enter}_{u,t'}$. Taking expectation on both sides and using the fact that $\mathbb{E}[\mathbf{1}^{play}_{u,t'}] = z^*_{u,t'}$ and $\mathbb{E}[\mathbf{1}^{enter}_{u,t'}] = w^*_{u,t'}$, linearity of expectation gives $\sum_{t' \le t} z^*_{u,t'} \le \sum_{t' \le t} w^*_{u,t'}$.

To see that constraint (4.3) is satisfied, notice that we can play at most one arm (or alternatively one state) in each time step, hence $\sum_{u \in \mathcal{S}} \mathbf{1}^{play}_{u,t} \le 1$ holds for all $t \in [1, B]$; the claim then follows by taking expectation on both sides as in the previous paragraph. Finally, constraint (4.4) is satisfied by definition of the start states.

To conclude the proof of the lemma, it suffices to show that $\mathsf{OPT} = \sum_{u,t} r_u \cdot z^*_{u,t}$. Since OPT obtains reward r_u whenever it plays state u, it follows that OPT's reward is given by $\sum_{u,t} r_u \cdot \mathbf{1}^{play}_{u,t}$; by taking expectation we get $\sum_{u,t} r_u z^*_{u,t} = \mathsf{OPT}$, and hence $\mathsf{OPT}_{\mathsf{LP}} \ge \mathsf{OPT}$.

■

4.4.2 The Rounding Algorithm

In order to best understand the motivation behind our rounding algorithm, it would be useful to go over the example which illustrates the necessity of preemption (repeatedly switching back and forth between the different arms) mentioned earlier in Section 4.3.1. At a high level, the rounding algorithm proceeds as follows. In Phase I, given an optimal LP solution, we decompose the fractional solution for each arm into a convex[1] combination of integral "strategy forests" (which are depicted in Figure 4.1): each of these tells us at what times to play the arm, and in which states to abandon the arm. Now, if we sample a random strategy forest for each arm from this distribution, we may end up scheduling multiple arms to play at some of the timesteps, and hence we need to resolve these conflicts. A natural approach might be to (i) sample a strategy forest for each arm, (ii) play these arms in some order, and (iii) for any arm follow the decisions (about whether to abort or continue playing) as suggested by the sampled strategy forest. But this is inherently non-preemptive and therefore, by the example in Section 4.3.1, it must fail.

Another approach would be to play the sampled forests at their prescribed times; if multiple

[1]Strictly speaking, we do not get convex combinations that sum to one; our combinations sum to $\sum_t z_{\rho_i,t}$, the value the LP assigned to pick to play the root of the arm over all possible start times, which is at most one.

forests want to play at the same time slot, we round-robin over them. But now if some arm needs B contiguous steps to get to a state with very high reward, even a single play of some other arm in the middle would end up fetching us no reward!

Guided by these bad examples, we try to use continuity information in the sampled strategy forests—once we start playing some contiguous component (where the strategy forest plays the arm in every consecutive time step), we make decisions to switch arms only at the end of the component (i.e. at the leaves of the different trees in Figure 4.1(b)). The naïve implementation does not work, so we first alter the solution to make all strategy forests "nice"—loosely, these are forests where all the connected components of any strategy forest are separated by large gaps (Phase II). The final strategy is presented in Phase III, and the analysis appears in Section 4.4.2.

Phase I: Convex Decomposition

In this step, we decompose the fractional solution into a convex combination of "forest-like strategies" $\{\mathbb{T}(i,j)\}_{i,j}$, corresponding to the j^{th} forest for arm i. We first formally define what these forests look like: The j^{th} *strategy forest* $\mathbb{T}(i,j)$ for arm i is an assignment of values $\mathsf{time}(i,j,u)$ and $\mathsf{prob}(i,j,u)$ to each state $u \in \mathcal{S}_i$ such that:

(i) For $u \in \mathcal{S}_i$ and $v = \mathsf{parent}(u)$, it holds that $\mathsf{time}(i,j,u) \geq 1 + \mathsf{time}(i,j,v)$, and

(ii) For $u \in \mathcal{S}_i$ and $v = \mathsf{parent}(u)$, if $\mathsf{time}(i,j,u) \neq \infty$ then $\mathsf{prob}(i,j,u) = p_{v,u}\,\mathsf{prob}(i,j,v)$; else if $\mathsf{time}(i,j,u) = \infty$ then $\mathsf{prob}(i,j,u) = 0$.

We call a triple (i,j,u) a *tree-node* of $\mathbb{T}(i,j)$.[2] For any state $u \in \mathcal{S}_i$, the values $\mathsf{time}(i,j,u)$ and $\mathsf{prob}(i,j,u)$ denote the time at which arm i is played from state u, and the probability with which the arm is played from state u, according to strategy forest $\mathbb{T}(i,j)$.

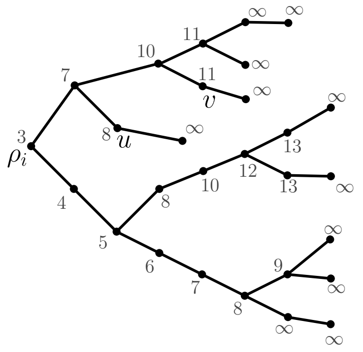
Observe that the probability values are particularly simple: if $\mathsf{time}(i,j,u) = \infty$ then this strategy does not play the arm at u, and hence the probability is zero, else $\mathsf{prob}(i,j,u)$ is equal to the probability of reaching u over the random transitions according to T_i if we play the root with probability $\mathsf{prob}(i,j,\rho_i)$. Hence, we can compute $\mathsf{prob}(i,j,u)$ just given $\mathsf{prob}(i,j,\rho_i)$ and whether or not $\mathsf{time}(i,j,u) = \infty$. Note that the time values are not necessarily consecutive, plotting these on the timeline and connecting a state to its parents only when they are in consecutive timesteps (as in Figure 4.1) gives us forests, hence the name.

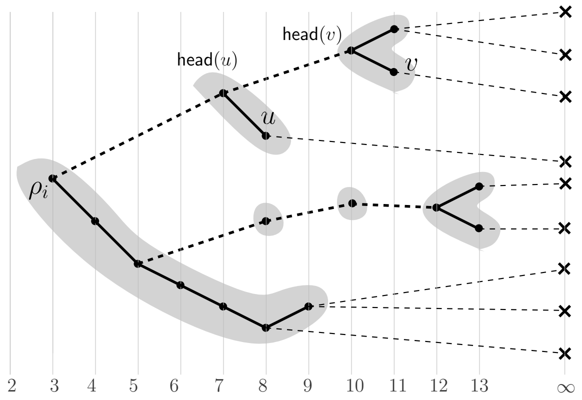
The algorithm to construct such a decomposition proceeds in rounds for each arm i; in a particular round, it "peels" off such a strategy as described above, and ensures that the residual fractional solution continues to satisfy the LP constraints, guaranteeing that we can repeat this process, which is similar to (but slightly more involved than) performing flow-decompositions. The decomposition lemma is proved in Section 4.5:

Lemma 4.4.2 *Given a solution to (*$\mathsf{LP}_{\mathsf{mab}}$*), there exists a collection of at most* $nB|\mathcal{S}|$ *strategy forests* $\{\mathbb{T}(i,j)\}$ *such that* $z_{u,t} = \sum_{j:\mathsf{time}(i,j,u)=t} \mathsf{prob}(i,j,u)$.[3] *Hence,* $\sum_{(i,j,u):\mathsf{time}(i,j,u)=t} \mathsf{prob}(i,j,u) \leq 1$ *for all* t.

[2]When i and j are understood from context, we identify the tree-node (i,j,u) with the state u.

[3]To reiterate, even though we call this a convex decomposition, the sum of the probability values of the root state of any arm is at most one by constraint 4.3, and hence the sum of the probabilities of the root over the decomposition could be less than one in general.


(a) Strategy forest: numbers are times



(b) Strategy forest shown on a timeline

Figure 4.1: Strategy forests and how to visualize them: grey blobs are connected components

For any $\mathbb{T}(i,j)$, $\mathsf{prob}(\cdot)$ satisfies "preflow" conditions: the in-flow at any node v is at least the out-flow, namely $\mathsf{prob}(i,j,v) \geq \sum_{u:\mathsf{parent}(u)=v} \mathsf{prob}(i,j,u)$, which leads to the following simple but crucial observation.

Observation 4.4.3 *For any arm i, for any set of states $X \subseteq \mathcal{S}_i$ such that no state in X is an ancestor of another in the transition tree T_i, and for any $z \in \mathcal{S}_i$ that is an ancestor of all states in X, $\mathsf{prob}(i,j,z) \geq \sum_{x\in X} \mathsf{prob}(i,j,x)$. More generally, if Z is a set of states such that for any $x \in X$, there exists $z \in Z$ such that z is an ancestor of x, we have $\sum_{z\in Z} \mathsf{prob}(i,j,z) \geq \sum_{x\in X} \mathsf{prob}(i,j,x)$*

Phase II: Eliminating Small Gaps

While Section 4.3.1 shows that switching between arms is necessary, we also should not get "tricked" into switching arms during very short breaks taken by the LP strategy forest, e.g., if an arm of length $(B-1)$ with high reward at the end was played in two continuous segments with a small gap in the middle, we should not lose profit from this arm by starting some other arms' plays during the gap. We now handle this, by eliminating such small gaps between contiguous segments of the strategy forest.

The motivation for the procedure comes from the following proof argument: we would like to claim that our algorithm begins playing any component C before the start-time in the LP, with probability at least $1/2$. But the issue is that conditioned on playing C, we also get to know that all its ancestors have been played; and since other arms may have also been scheduled before C, the desired claim would be false. But if we ensure that the number of ancestors is small (say at most $t/2$, where t is the time when the LP begins playing C), this problem disappears—other arms use upto t plays on average (which we can make $t/2$ by sampling), leaving enough room for the ancestors' plays. This is precisely the condition we use to advance some components to fill small gaps.

Before we make this formal, here is some useful notation: Given $u \in \mathcal{S}_i$, let $\mathsf{head}(i,j,u)$ be its ancestor node $v \in \mathcal{S}_i$ of least depth such that the plays from v through u occur in consecutive time values. More formally, the path $v = v_1, v_2, \ldots, v_l = u$ in T_i is such that $\mathsf{time}(i, j, v_{l'}) = \mathsf{time}(i, j, v_{l'-1}) + 1$ for all $l' \in [2, l]$. We also define the *connected component* of a node u, denoted by $\mathsf{comp}(i, j, u)$, as the set of all nodes u' such that $\mathsf{head}(i, j, u) = \mathsf{head}(i, j, u')$. Figure 4.1 shows the connected components and heads.

The *gap-filling* procedure works as follows: if a head state $v = \mathsf{head}(i, j, u)$ is played at time $t = \mathsf{time}(i, j, v)$ s.t. $t < 2 \cdot \mathsf{depth}(v)$, then we "advance" the $\mathsf{comp}(i, j, v)$ and get rid of the gap between v and its parent (and recursively apply this rule)[4]. The procedure is formally described in Section 4.6. By construction this guarantees that the components have large gaps between them. Additionally we show that the fractional number of plays made at any time t does not increase by too much due to these "advances". Intuitively this is because if for some time slot t we "advance" a set of components that were originally scheduled after t to now cross time slot t, these components moved because their ancestor paths (fractionally) used up at least $t/2$ of the time slots before t; since there are only a total of t time slots to be used up, there can be at most 2 units of components that were advanced across t. Hence, in the following, we assume that our $\mathbb{T}$'s satisfy the properties in the following lemma (whose proof is also in Section 4.6):

Lemma 4.4.4 *Algorithm* GapFill *produces a modified collection of* $\mathbb{T}$*'s such that*

(i) For each i, j, u *such that* $r_u > 0$*,* $\mathsf{time}(\mathsf{head}(i, j, u)) \geq 2 \cdot \mathsf{depth}(\mathsf{head}(i, j, u))$.

(ii) The total extent of plays at any time t*, i.e.,* $\sum_{(i,j,u):\mathsf{time}(i,j,u)=t} \mathsf{prob}(i, j, u)$ *is at most* 3.

Phase III: Scheduling the Arms

After the above processing, the final algorithm is as follows: it samples a strategy forest from the collection $\{\mathbb{T}(i, j)\}_j$ for each arm i. Then, it picks an arm with the earliest connected component (i.e., the one with smallest $\mathsf{time}(\mathsf{head}(i, j, u))$) that contains the current state (which is the root state to begin with), plays it to the end of the component, and repeats this step—note that we may switch out of an arm only if it jumps to a state played much later in time. Again we let the algorithm run as long as there is some active node, regardless of whether or not the budget is exceeded—however, we only count the profit from the first B plays in the analysis.

Observe that Steps 7-9 play a connected component of a strategy forest contiguously. In particular, this means that all $\mathsf{currstate}(i)$'s considered in Step 5 are head vertices of the corresponding strategy forests. These facts will be crucial in the analysis.

Lemma 4.4.5 *For arm* i *and strategy* $\mathbb{T}(i, j)$*, conditioned on* $\sigma(i) = j$ *after Step 1 of* AlgMAB*, the probability of playing state* $u \in \mathcal{S}_i$ *is* $\mathsf{prob}(i, j, u)/\mathsf{prob}(i, j, \rho_i)$*, where the probability is over the random transitions of arm* i.

The above lemma is relatively simple, and proved in Section 4.7. The rest of the section proves that in expectation, we collect a constant factor of the LP reward of each strategy $\mathbb{T}(i, j)$ before running out of budget; the analysis is inspired by our StocK rounding procedure. We mainly focus on the following lemma.

Lemma 4.4.6 *Consider any arm* i *and strategy* $\mathbb{T}(i, j)$*. Then, conditioned on* $\sigma(i) = j$ *and on the*

[4]The intuition is that such vertices have only a small gap in their play and should rather be played contiguously.

Algorithm 7 Scheduling the Connected Components: Algorithm AlgMAB

```
1: for arm $i$, sample strategy $\mathbb{T}(i,j)$ with probability $\frac{\mathsf{prob}(i,j,\rho_i)}{24}$; ignore arm $i$ w.p. $1 - \sum_j \frac{\mathsf{prob}(i,j,\rho_i)}{24}$.
2: let $A \leftarrow$ set of "active" arms which chose a strategy in the random process.
3: for each $i \in A$, let $\sigma(i) \leftarrow$ index $j$ of the chosen $\mathbb{T}(i,j)$ and let $\mathsf{currstate}(i) \leftarrow$ root $\rho_i$.
4: while active arms $A \neq \emptyset$ do
5:   let $i^* \leftarrow$ arm with state played earliest in the LP (i.e., $i^* \leftarrow \operatorname{argmin}_{i \in A}\{\mathsf{time}(i,\sigma(i),\mathsf{currstate}(i))\}$.
6:   let $\tau \leftarrow \mathsf{time}(i^*,\sigma(i^*),\mathsf{currstate}(i^*))$.
7:   while $\mathsf{time}(i^*,\sigma(i^*),\mathsf{currstate}(i^*)) \neq \infty$ and $\mathsf{time}(i^*,\sigma(i^*),\mathsf{currstate}(i^*)) = \tau$ do
8:     play arm $i^*$ at state $\mathsf{currstate}(i^*)$
9:     update $\mathsf{currstate}(i^*)$ be the new state of arm $i^*$; let $\tau \leftarrow \tau + 1$.
10:  if $\mathsf{time}(i^*,\sigma(i^*),\mathsf{currstate}(i^*)) = \infty$ then
11:    let $A \leftarrow A \setminus \{i^*\}$
```

algorithm playing state $u \in \mathcal{S}_i$, the probability that this play happens before time $\mathsf{time}(i,j,u)$ *is at least* $1/2$.

Proof: Fix an arm i and an index j for the rest of the proof. Given a state $u \in \mathcal{S}_i$, let $\mathcal{E}_{iju}$ denote the event $(\sigma(i) = j) \wedge (\text{state } u \text{ is played})$. Also, let $\mathbf{v} = \mathsf{head}(i,j,u)$ be the head of the connected component containing u in $\mathbb{T}(i,j)$. Let r.v. τ_u (respectively $\tau_{\mathbf{v}}$) be the actual time at which state u (respectively state $\mathbf{v}$) is played—these random variables take value ∞ if the arm is not played in these states. Then

$$\mathbb{P}[\tau_u \leq \mathsf{time}(i,j,u) \mid \mathcal{E}_{iju}] \geq \tfrac{1}{2} \iff \mathbb{P}[\tau_{\mathbf{v}} \leq \mathsf{time}(i,j,\mathbf{v}) \mid \mathcal{E}_{iju}] \geq \tfrac{1}{2}, \tag{4.5}$$

because the time between playing u and $\mathbf{v}$ is exactly $\mathsf{time}(i,j,u) - \mathsf{time}(i,j,\mathbf{v})$ since the algorithm plays connected components continuously (and we have conditioned on $\mathcal{E}_{iju}$). Hence, we can just focus on proving the right inequality in (4.5) for vertex $\mathbf{v}$.

For brevity of notation, let $t_{\mathbf{v}} = \mathsf{time}(i,j,\mathbf{v})$. In addition, we define the order $\preceq$ to indicate which states can be played before $\mathbf{v}$. That is, again making use of the fact that the algorithm plays connected components contiguously, we say that $(i',j',v') \preceq (i,j,\mathbf{v})$ iff $\mathsf{time}(\mathsf{head}(i',j',v')) \leq \mathsf{time}(\mathsf{head}(i,j,\mathbf{v}))$. Notice that this order is independent of the run of the algorithm; also it could be that $\mathsf{time}(i',j',v') > \mathsf{time}(i,j,\mathbf{v})$ yet $(i',j',v') \preceq (i,j,\mathbf{v})$.

For each arm $i' \neq i$ and index j', we define random variables $Z_{i'j'}$ used to count the number of plays that can possibly occur before the algorithm plays state $\mathbf{v}$. If $\mathbf{1}_{(i',j',v')}$ is the indicator variable of event $\mathcal{E}_{i'j'v'}$, define

$$Z_{i',j'} = \min\left(t_{\mathbf{v}}\ ,\ \textstyle\sum_{v':(i',j',v')\preceq(i,j,\mathbf{v})} \mathbf{1}_{(i',j',v')}\right). \tag{4.6}$$

We truncate $Z_{i',j'}$ at $t_{\mathbf{v}}$ because we just want to capture how much time *up to* $t_{\mathbf{v}}$ is being used. Now consider the sum $Z = \sum_{i' \neq i} \sum_{j'} Z_{i',j'}$. Note that for arm i', at most one of the $Z_{i',j'}$ values will be non-zero in any scenario, namely the index $\sigma(i')$ sampled in Step 1. The first claim below shows that it suffices to consider the upper tail of Z, and show that $\mathbb{P}[Z \geq t_{\mathbf{v}}/2] \leq 1/2$, and the second gives a bound on the conditional expectation of $Z_{i',j'}$. The proofs appear in Section 4.7.

Claim 4.4.7 $\mathbb{P}[\tau_{\mathbf{v}} \leq t_{\mathbf{v}} \mid \mathcal{E}_{iju}] \geq \mathbb{P}[Z \leq t_{\mathbf{v}}/2]$.

Claim 4.4.8

$$\mathbb{E}[Z_{i',j'} \mid \sigma(i') = j'] \leq \sum_{v' \text{ s.t } \mathsf{time}(i',j',v') \leq t_{\mathbf{v}}} \frac{\mathsf{prob}(i',j',v')}{\mathsf{prob}(i',j',\rho_{i'})} + t_{\mathbf{v}} \left(\sum_{v' \text{ s.t } \mathsf{time}(i',j',v') = t_{\mathbf{v}}} \frac{\mathsf{prob}(i',j',v')}{\mathsf{prob}(i',j',\rho_{i'})} \right)$$

Equipped with the above claims, we are ready to complete the proof of Lemma 4.4.6. Employing Claim 4.4.8 we get

$$\begin{aligned}
\mathbb{E}[Z] &= \sum_{i' \neq i} \sum_{j'} \mathbb{E}[Z_{i',j'}] = \sum_{i' \neq i} \sum_{j'} \mathbb{E}[Z_{i',j'} \mid \sigma(i') = j'] \cdot \mathbb{P}[\sigma(i') = j'] \\
&= \frac{1}{24} \sum_{i' \neq i} \sum_{j'} \left\{ \sum_{v' : \mathsf{time}(i',j',v') \leq t_{\mathbf{v}}} \mathsf{prob}(i',j',v') + t_{\mathbf{v}} \Bigg(\sum_{v' : \mathsf{time}(i',j',v') = t_{\mathbf{v}}} \mathsf{prob}(i',j',v') \Bigg) \right\} \qquad (4.7) \\
&= \frac{1}{24} (3 \cdot t_{\mathbf{v}} + 3 \cdot t_{\mathbf{v}}) \leq \frac{1}{4} t_{\mathbf{v}} \ . \qquad (4.8)
\end{aligned}$$

Equation (4.7) follows from the fact that each tree $\mathbb{T}(i,j)$ is sampled with probability $\frac{\mathsf{prob}(i,j,\rho_i)}{24}$ and (4.8) follows from Lemma 4.4.4. Applying Markov's inequality, we have that $\mathbb{P}[Z \geq t_{\mathbf{v}}/2] \leq 1/2$. Finally, Claim 4.4.7 says that $\mathbb{P}[\tau_{\mathbf{v}} \leq t_{\mathbf{v}} \mid \mathcal{E}_{iju}] \geq \mathbb{P}[Z \leq t_{\mathbf{v}}/2] \geq 1/2$, which completes the proof. ■

Theorem 4.4.9 *The reward obtained by the algorithm* AlgMAB *is at least* $\Omega(\mathsf{OPT}_{\mathsf{LP}})$.

Proof: The theorem follows by a simple linearity of expectation. Indeed, the expected reward obtained from any state $u \in \mathcal{S}_i$ is at least $\sum_j \mathbb{P}[\sigma(i) = j]\, \mathbb{P}[\text{state } u \text{ is played} \mid \sigma(i) = j]\, \mathbb{P}[\tau_u \leq t_u | \mathcal{E}_{iju}] \cdot R_u \geq \sum_j \frac{\mathsf{prob}(i,j,u)}{24} \frac{1}{2} \cdot R_u$. Here, we have used Lemmas 4.4.5 and 4.4.6 for the second and third probabilities. But now we can use Lemma 4.4.2 to infer that $\sum_j \mathsf{prob}(i,j,u) = \sum_t z_{u,t}$; Making this substitution and summing over all states $u \in \mathcal{S}_i$ and arms i completes the proof. ■

4.5 Details of Phase I (from Section 4.4.2)

We first begin with some notation that will be useful in the algorithm below. For any state $u \in \mathcal{S}_i$ such that the path from ρ_i to u follows the states $u_1 = \rho_i, u_2, \ldots, u_k = u$, let $\pi_u = \Pi_{l=1}^{k-1} p_{u_i,u_{i+1}}$.

Fix an arm i, for which we will perform the decomposition. Let $\{z, w\}$ be a feasible solution to $\mathsf{LP}_{\mathsf{mab}}$ and set $z^0_{u,t} = z_{u,t}$ and $w^0_{u,t} = w_{u,t}$ for all $u \in \mathcal{S}_i$, $t \in [B]$. We will gradually alter the fractional solution as we build the different forests. We note that in a particular iteration with index j, all z^{j-1}, w^{j-1} values that are not updated in Steps 12 and 13 are retained in z^j, w^j respectively. For brevity of notation, we shall use "iteration j of Step 2" to denote the execution of the entire block (Steps 3 – 14) which constructs strategy forest $\mathbb{T}(i,j)$.

Algorithm 8 Convex Decomposition of Arm i

```
1: set $\mathcal{C}_i \leftarrow \emptyset$ and set loop index $j \leftarrow 1$.
2: while $\exists$ a node $u \in \mathcal{S}_i$ s.t $\sum_t z^{j-1}_{u,t} > 0$ do
3:     initialize a new tree $\mathbb{T}(i,j) = \emptyset$.
4:     set $A \leftarrow \{u \in \mathcal{S}_i \text{ s.t } \sum_t z^{j-1}_{u,t} > 0\}$.
5:     for all $u \in \mathcal{S}_i$, set $\mathsf{time}(i,j,u) \leftarrow \infty$, $\mathsf{prob}(i,j,u) \leftarrow 0$, and set $\varepsilon_u \leftarrow \infty$.
6:     for every $u \in A$ do
7:         update $\mathsf{time}(i,j,u)$ to the smallest time $t$ s.t $z^{j-1}_{u,t} > 0$.
8:         update $\varepsilon_u = z^{j-1}_{u,\mathsf{time}(i,j,u)}/\pi_u$
9:     let $\varepsilon = \min_u \varepsilon_u$.
10:    for every $u \in A$ do
11:        set $\mathsf{prob}(i,j,u) = \varepsilon \cdot \pi_u$.
12:        update $z^j_{u,\mathsf{time}(i,j,u)} = z^{j-1}_{u,\mathsf{time}(i,j,u)} - \mathsf{prob}(i,j,u)$.
13:        update $w^j_{v,\mathsf{time}(i,j,u)+1} = w^{j-1}_{v,\mathsf{time}(i,j,u)+1} - \mathsf{prob}(i,j,u) \cdot p_{u,v}$ for all $v$ s.t $\mathsf{parent}(v) = u$.
14:    set $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \mathbb{T}(i,j)$.
15:    increment $j \leftarrow j+1$.
```

Lemma 4.5.1 *Consider an integer j and suppose that $\{z^{j-1}, w^{j-1}\}$ satisfies constraints* (4.1)-(4.3) *of* $\mathsf{LP_{mab}}$. *Then after iteration j of Step 2, the following properties hold:*

(a) $\mathbb{T}(i,j)$ *(along with the associated* $\mathsf{prob}(i,j,.)$ *and* $\mathsf{time}(i,j,.)$ *values) is a valid strategy forest, i.e., satisfies the conditions (i) and (ii) presented in Section 4.4.2.*

(b) The residual solution $\{z^j, w^j\}$ satisfies constraints (4.1)-(4.3).

(c) For any time t and state $u \in \mathcal{S}_i$, $z^{j-1}_{u,t} - z^j_{u,t} = \mathsf{prob}(i,j,u)\mathbf{1}_{\mathsf{time}(i,j,u)=t}$.

Proof: We show the properties stated above one by one.

Property (a): We first show that the time values satisfy $\mathsf{time}(i,j,u) \geq \mathsf{time}(i,j,\mathsf{parent}(u)) + 1$, i.e. condition (i) of strategy forests. For sake of contradiction, assume that there exists $u \in \mathcal{S}_i$ with $v = \mathsf{parent}(u)$ where $\mathsf{time}(i,j,u) \leq \mathsf{time}(i,j,v)$. Define $t_u = \mathsf{time}(i,j,u)$ and $t_v = \mathsf{time}(i,j,\mathsf{parent}(u))$; the way we updated $\mathsf{time}(i,j,u)$ in Step 7 gives that $z^{j-1}_{u,t_u} > 0$.

Then, constraint (4.2) of the LP implies that $\sum_{t' \leq t_u} w^{j-1}_{u,t'} > 0$. In particular, there exists a time $t' \leq t_u \leq t_v$ such that $w^{j-1}_{u,t'} > 0$. But now, constraint (4.1) enforces that $z^{j-1}_{v,t'-1} = w^{j-1}_{u,t'}/p_{v,u} > 0$ as well. But this contradicts the fact that t_v was the first time s.t $z^{j-1}_{v,t} > 0$. Hence we have $\mathsf{time}(i,j,u) \geq \mathsf{time}(i,j,\mathsf{parent}(u)) + 1$.

As for condition (ii) about $\mathsf{prob}(i,j,.)$, notice that if $\mathsf{time}(i,j,u) \neq \infty$, then $\mathsf{prob}(i,j,u)$ is set to $\varepsilon \cdot \pi_u$ in Step 11. It is now easy to see from the definition of π_u (and from the fact that $\mathsf{time}(i,j,u) \neq \infty \Rightarrow \mathsf{time}(i,j,\mathsf{parent}(u)) \neq \infty$) that $\mathsf{prob}(i,j,u) = \mathsf{prob}(i,j,\mathsf{parent}(u)) \cdot p_{\mathsf{parent}(u),u}$.

Property (b): Constraint (4.1) of $\mathsf{LP_{mab}}$ is clearly satisfied by the new LP solution $\{z^j, w^j\}$ because of the two updates performed in Steps 12 and 13: if we decrease the z value of any node at any time, the w of all children are appropriately reduced (for the subsequent timestep).

Before showing that the solution $\{z^j, w^j\}$ satisfies constraint (4.2), we first argue that they remain non-negative. By the choice of ε in step 9, we have $\mathsf{prob}(i,j,u) = \varepsilon\pi_u \leq \varepsilon_u\pi_u = z^{j-1}_{u,\mathsf{time}(i,j,u)}$ (where ε_u was computed in Step 8); consequently even after the update in step 12, $z^{j}_{u,\mathsf{time}(i,j,u)} \geq 0$ for all u. This and the fact that the constraints (4.1) are satisfied implies that $\{z^j, w^j\}$ satisfies the non-negativity requirement.

We now show that constraint (4.2) is satisfied. For any time t and state $u \notin A$ (where A is the set computed in step 4 for iteration j), clearly it must be that $\sum_{t' \leq t} z^{j-1}_{u,t} = 0$ by definition of the set A; hence just the non-negativity of w^j implies that these constraints are trivially satisfied.

Therefore consider some $t \in [B]$ and a state $u \in A$. We know from step 7 that $\mathsf{time}(i,j,u) \neq \infty$. If $t < \mathsf{time}(i,j,u)$, then the way $\mathsf{time}(i,j,u)$ is updated in step 7 implies that $\sum_{t' \leq t} z^{j}_{u,t'} = \sum_{t' \leq t} z^{j-1}_{u,t'} = 0$, so the constraint is trivially satisfied because w^j is non-negative. If $t \geq \mathsf{time}(i,j,u)$, we claim that the change in the left hand side and right hand side (between the solutions $\{z^{j-1}, w^{j-1}\}$ and $\{z^j, w^j\}$) of the constraint under consideration is the same, implying that it will be still satisfied by $\{z^j, w^j\}$.

To prove this claim, observe that the right hand side has decreased by exactly $z^{j-1}_{u,\mathsf{time}(i,j,u)} - z^{j}_{u,\mathsf{time}(i,j,u)} = \mathsf{prob}(i,j,u)$. But the only value which has been modified in the left hand side is $w^{j-1}_{u,\mathsf{time}(i,j,\mathsf{parent}(u))+1}$, which has gone down by $\mathsf{prob}(i,j,\mathsf{parent}(u)) \cdot p_{\mathsf{parent}(u),u}$. Because $\mathbb{T}(i,j)$ forms a valid strategy forest, we have $\mathsf{prob}(i,j,u) = \mathsf{prob}(i,j,\mathsf{parent}(u)) \cdot p_{\mathsf{parent}(u),u}$, and thus the claim follows.

Finally, constraint (4.3) are also satisfied as the z variables only decrease in value over iterations.

Property (c): This is an immediate consequence of the Step 12. ■

To prove Lemma 4.4.2, firstly notice that since $\{z^0, w^0\}$ satisfies constraints (4.1)-(4.3), we can proceed by induction and infer that the properties in the previous lemma hold for every strategy forest in the decomposition; in particular, each of them is a valid strategy forest.

In order to show that the marginals are preserved, observe that in the last iteration j^* of procedure we have $z^{j^*}_{u,t} = 0$ for all u, t. Therefore, adding the last property in the previous lemma over all j gives

$$z_{u,t} = \sum_{j \geq 1} (z^{j-1}_{u,t} - z^{j}_{u,t}) = \sum_{j \geq 1} \mathsf{prob}(i,j,u)\mathbf{1}_{\mathsf{time}(i,j,u)=t} = \sum_{j:\mathsf{time}(i,j,u)=t} \mathsf{prob}(i,j,u).$$

Finally, since some $z^{j}_{u,t}$ gets altered to 0 since in each iteration of the above algorithm, the number of strategies for each arm in the decomposition is upper bounded by $B|\mathcal{S}|$. This completes the proof of Lemma 4.4.2.

4.6 Details of Phase II (from Section 4.4.2)

Proof of Lemma 4.4.4: Let $\mathsf{time}^t(u)$ denote the time assigned to node u by the end of round $\tau = t$ of the algorithm; $\mathsf{time}^{B+1}(u)$ is the initial time of u. Since the algorithm works backwards

Algorithm 9 Gap Filling Algorithm GapFill

```
1: for $\tau = B$ to $1$ do
2:   while there exists a tree-node $u \in \mathbb{T}(i,j)$ such that $\tau = \mathsf{time}(\mathsf{head}(u)) < 2 \cdot \mathsf{depth}(\mathsf{head}(u))$ do
3:     let $v = \mathsf{head}(u)$.
4:     if $v$ is not the root of $\mathbb{T}(i,j)$ then
5:       let $v' = \mathsf{parent}(v)$.
6:       advance the component $\mathsf{comp}(v)$ rooted at $v$ such that $\mathsf{time}(v) \leftarrow \mathsf{time}(v') + 1$, to make $\mathsf{comp}(v)$ contiguous with the ancestor forming one larger component. Also alter the times of $w \in \mathsf{comp}(v)$ appropriately to maintain contiguity with $v$ (and now with $v'$).
```

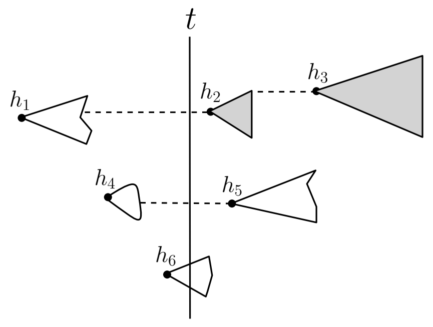
in time, our round index will start at B and end up at 1. To prove property (i) of the statement of the lemma, notice that the algorithm only converts head nodes to non-head nodes and not the other way around. Moreover, heads which survive the algorithm have the same time as originally. So it suffices to show that heads which originally did not satisfy property (i)—namely, those with $\mathsf{time}^{B+1}(v) < 2 \cdot \mathsf{depth}(v)$—do not survive the algorithm; but this is clear from the definition of Step 2.

To prove property (ii), fix a time t, and consider the execution of GapFill at the end of round $\tau = t$. We claim that the total extent of fractional play at time t does not increase as we continue the execution of the algorithm from round $\tau = t$ to round 1. To see why, let C be a connected component at the end of round $\tau = t$ and let h denote its head. If $\mathsf{time}^t(h) > t$ then no further **advance** affects C and hence it does not contribute to an increase in the number of plays at time t. On the other hand, if $\mathsf{time}^t(h) \leq t$, then even if C is advanced in a subsequent round, each node w of C which ends up being played at t, i.e., has $\mathsf{time}^1(w) = t$ must have an ancestor w' satisfying $\mathsf{time}^t(w') = t$, by the contiguity of C. Thus, Observation 4.4.3 gives that $\sum_{u \in C : \mathsf{time}^1(u) = t} \mathsf{prob}(u) \leq \sum_{u \in C : \mathsf{time}^t(u) = t} \mathsf{prob}(u)$. Applying this for each connected component C, proves the claim. Intuitively, any component which advances forward in time is only reducing its load/total fractional play at any fixed time t.

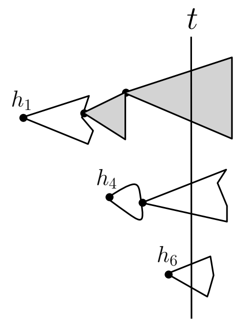
Then consider the end of iteration $\tau = t$ and we now prove that the fractional extent of play at time t is at most 3. Due to Lemma 4.4.2, it suffices to prove that $\sum_{u \in U} \mathsf{prob}(u) \leq 2$, where U is the set of nodes which caused an increase in the number of plays at time t, namely, $U = \{u : \mathsf{time}^{B+1}(u) > t \text{ and } \mathsf{time}^t(u) = t\}$.

Notice that a connected component of the original forest can only contribute to this increase if its head h crossed time t, that is $\mathsf{time}^{B+1}(h) > t$ and $\mathsf{time}^t(h) \leq t$. However, it may be that this crossing was not directly caused by an **advance** on h (i.e. h advanced till $\mathsf{time}^{B+1}(\mathsf{parent}(h)) \geq t$), but an **advance** to a head h' in a subsequent round was responsible for h crossing over t. But in this case h must be part of the connected component of h' when the latter **advance** happens, and we can use h''s advance to bound the congestion.

To make this more formal, let H be the set of heads of the original forest whose **advances** made them cross time t, namely, $h \in H$ iff $\mathsf{time}^{B+1}(h) > t$, $\mathsf{time}^t(h) \leq t$ and $\mathsf{time}^{B+1}(\mathsf{parent}(h)) <$



(a) Connected components in the beginning of the algorithm



(b) Configuration at the end of iteration $\tau = t$

Figure 4.2: Depiction of a strategy forest $\mathbb{T}(i,j)$ on a timeline where each triangle is a connected component. In this example, $H = \{h_2, h_5\}$ and C_{h_2} consists of the grey nodes. From Observation 4.4.3 the number of plays at t do not increase as components are moved to the left

t. Moreover, for $h \in H$ let C_h denote the connected component of h in the beginning of the iteration where an **advance** was executed on h, that is, when v was set to h in Step 3. The above argument shows that these components C_h's contain all the nodes in U, hence it suffices to see how they increase the congestion at time t.

In fact, it is sufficient to focus just on the heads in H. To see this, consider $h \in H$ and notice that no node in $U \cap C_h$ is an ancestor of another. Then Observation 4.4.3 gives $\sum_{u \in U \cap C_h} \mathsf{prob}(u) \leq \mathsf{prob}(h)$, and adding over all h in H gives $\sum_{u \in U} \mathsf{prob}(u) \leq \sum_{h \in H} \mathsf{prob}(h)$.

To conclude the proof, we upper bound the right hand side of the previous inequality. The idea now is that the play probabilities on the nodes in H cannot be too large since their parents have $\mathsf{time}^{B+1} < t$ (and each head has a large number of ancestors in $[1,t]$ because it was considered for an advance). More formally, fix i,j and consider a head h in $H \cap \mathbb{T}(i,j)$. From Step 2 of the algorithm, we obtain that $\mathsf{depth}(h) > (1/2)\mathsf{time}^{B+1}(h) \geq t/2$. Since $\mathsf{time}^{B+1}(\mathsf{parent}(h)) < t$, it follows that for every $d \leq \lfloor t/2 \rfloor$, h has an ancestor $u \in \mathbb{T}(i,j)$ with $\mathsf{depth}(u) = d$ and $\mathsf{time}^{B+1}(u) \leq t$. Moreover, the definition of H implies that no head in $H \cap \mathbb{T}(i,j)$ can be an ancestor of another. Then again employing Observation 4.4.3 we obtain

$$\sum_{h \in H \cap \mathbb{T}(i,j)} \mathsf{prob}(h) \leq \sum_{u \in \mathbb{T}(i,j) : \mathsf{depth}(u) = d, \mathsf{time}^{B+1}(u) \leq t} \mathsf{prob}(u) \qquad (\forall d \leq \lfloor t/2 \rfloor).$$

Adding over all i,j and $d \leq \lfloor t/2 \rfloor$ leads to the bound $(t/2) \cdot \sum_{h \in H} \mathsf{prob}(h) \leq \sum_{u : \mathsf{time}^{B+1}(u) \leq t} \mathsf{prob}(u)$. Finally, using Lemma 4.4.2 we can upper bound the right hand side by t, which gives $\sum_{u \in U} \mathsf{prob}(u) \leq \sum_{h \in H} \mathsf{prob}(u) \leq 2$ as desired. ■

4.7 Details of Phase III (from Section 4.4.2)

Proof of Lemma 4.4.5: The proof is quite straightforward. Intuitively, it is because AlgMAB (Algorithm 7) simply follows the probabilities according to the transition tree T_i (unless $\mathsf{time}(i,j,u) = \infty$ in which case it abandons the arm). Consider an arm i such that $\sigma(i) = j$, and any state

$u \in \mathcal{S}_i$. Let $\langle v_1 = \rho_i, v_2, \ldots, v_t = u\rangle$ denote the unique path in the transition tree for arm i from ρ_i to u. Then, if $\mathsf{time}(i,j,u) \neq \infty$ the probability that state u is played is exactly the probability of the transitions reaching u (because in Steps 8 and 9, the algorithm just keeps playing the states[5] and making the transitions, unless $\mathsf{time}(i,j,u) = \infty$). But this is precisely $\Pi_{k=1}^{t-1} p_{v_k,v_{k+1}} = \mathsf{prob}(i,j,u)/\mathsf{prob}(i,j,\rho_i)$ (from the properties of each strategy in the convex decomposition). If $\mathsf{time}(i,j,u) = \infty$ however, then the algorithm terminates the arm in Step 10 without playing u, and so the probability of playing u is $0 = \mathsf{prob}(i,j,u)/\mathsf{prob}(i,j,\rho_i)$. This completes the proof. ∎

Proof of Claim 4.4.7: We first claim that $\mathbb{P}[\tau_{\mathbf{v}} \le t_{\mathbf{v}} \mid \mathcal{E}_{iju}] \ge \mathbb{P}[Z \le t_{\mathbf{v}}/2 \mid \mathcal{E}_{iju}]$. So, let us condition on $\mathcal{E}_{iju}$. Then if $Z \le t_{\mathbf{v}}/2$, none of the $Z_{i',j'}$ variables were truncated at $t_{\mathbf{v}}$, and hence Z exactly counts the total number of plays (by all other arms $i' \neq i$, from any state) that could possibly be played before the algorithm plays v in strategy $\mathbb{T}(i,j)$. Therefore, if Z is smaller than $t_{\mathbf{v}}/2$, then combining this with the fact that $\mathsf{depth}(v) \le t_{\mathbf{v}}/2$ (from Lemma 4.4.4(i)), we can infer that all the plays (including those of v's ancestors) that can be made before playing v can indeed be completed within $t_{\mathbf{v}}$. In this case the algorithm will definitely play v before $t_{\mathbf{v}}$; hence we get that conditioning on $\mathcal{E}_{iju}$, the event $\tau_{\mathbf{v}} \le t_{\mathbf{v}}$ holds when $Z \le t_{\mathbf{v}}/2$.

Finally, to remove the conditioning: note that $Z_{i'j'}$ is just a function of (i) the random variables $\mathbf{1}_{(i',j',v')}$, i.e., the random choices made by playing $\mathbb{T}(i',j')$, and (ii) the constant $t_{\mathbf{v}} = \mathsf{time}(i,j,v)$. However, the r.vs $\mathbf{1}_{(i',j',v')}$ are clearly independent of the event $\mathcal{E}_{iju}$ for $i' \neq i$ since the plays of AlgMAB in one arm are independent of the others, and $\mathsf{time}(i,j,v)$ is a constant determined once the strategy forests are created in Phase II. Hence the event $Z \le t_{\mathbf{v}}/2$ is independent of $\mathcal{E}_{iju}$; hence $\mathbb{P}[Z \le t_{\mathbf{v}}/2 \mid \mathcal{E}_{iju}] = \mathbb{P}[Z \le t_{\mathbf{v}}/2]$, which completes the proof. ∎

Proof of Claim 4.4.8: Recall the definition of $Z_{i'j'}$ in Eq (4.6): any state v' with $\mathsf{time}(i',j',v') > t_{\mathbf{v}}$ may contribute to the sum only if it is part of a connected component with head $\mathsf{head}(i',j',v')$ such that $\mathsf{time}(\mathsf{head}(i',j',v')) \le t_{\mathbf{v}}$, by the definition of the ordering $\preceq$. Even among such states, if $\mathsf{time}(i',j',v') > 2t_{\mathbf{v}}$, then the truncation implies that $Z_{i',j'}$ is unchanged whether or not we include $\mathbf{1}_{(i',j',v')}$ in the sum. Indeed, if $\mathbf{1}_{(i',j',v')} = 1$ then all of v''s ancestors will have their indicator variables at value 1; moreover $\mathsf{depth}(v') > t_{\mathbf{v}}$ since there is a contiguous collection of nodes that are played from this tree $\mathbb{T}(i',j')$ from time $t_{\mathbf{v}}$ onwards till $\mathsf{time}(i',j',v') > 2t_{\mathbf{v}}$; so the sum would be truncated at value $t_{\mathbf{v}}$ whenever $\mathbf{1}_{(i',j',v')} = 1$. Therefore, we can write

$$Z_{i',j'} \le \sum_{v' : \mathsf{time}(i',j',v') \le t_{\mathbf{v}}} \mathbf{1}_{(i',j',v')} + \sum_{\substack{v' : t_{\mathbf{v}} < \mathsf{time}(i',j',v') \le 2t_{\mathbf{v}} \\ (i',j',v') \preceq (i,j,v)}} \mathbf{1}_{(i',j',v')} \tag{4.9}$$

Recall we are interested in the conditional expectation given $\sigma(i') = j'$. Note that $\mathbb{P}[\mathbf{1}_{(i',j',v')} \mid \sigma(i') = j'] = \mathsf{prob}(i',j',v')/\mathsf{prob}(i',j',\rho_{i'})$ by Lemma 4.4.5, hence the first sum in (4.9) gives the first part of the claimed bound. Now the second part: observe that for any arm i', any fixed value of $\sigma(i') = j'$, and any value of $t' \ge t_{\mathbf{v}}$,

$$\sum_{\substack{v' \text{ s.t } \mathsf{time}(i',j',v') = t' \\ (i',j',v') \preceq (i,j,v)}} \mathsf{prob}(i',j',v') \le \sum_{v' \text{ s.t } \mathsf{time}(i',j',v') = t_{\mathbf{v}}} \mathsf{prob}(i',j',v')$$

[5]We remark that while the plays just follow the transition probabilities, they may not be made contiguously.

This is because of the following argument: Any state that appears on the LHS of the sum above is part of a connected component which crosses $t_{\mathbf{v}}$, they must have an ancestor which is played at $t_{\mathbf{v}}$. Also, since all states which appear in the LHS are played at t', no state can be an ancestor of another. Hence, we can apply the second part of Observation 4.4.3 and get the above inequality. Combining this with the fact that $\mathbb{P}[\mathbf{1}_{(i',j',v')} \mid \sigma(i') = j'] = \mathsf{prob}(i',j',v')/\mathsf{prob}(i',j',\rho_{i'})$, and applying it for each value of $t' \in (t_{\mathbf{v}}, 2t_{\mathbf{v}}]$, gives us the second term. ■

4.8 MABs with Arbitrary Transition Graphs

We now show how we can use techniques akin to those we described for the case when the transition graph is a tree, to handle the case when it can be an arbitrary directed graph. A naïve way to do this is to expand out the transition graph as a tree, but this incurs an exponential blowup of the state space which we want to avoid. We can assume we have a layered DAGs, though, since the conversion from a digraph to a layered DAG only increases the state space by a factor of the horizon B.

4.8.1 Layered DAGs capture all Graphs

We first show that *layered DAGs* can capture all transition graphs, with a blow-up of a factor of B in the state space. For each arm i, for each state u in the transition graph $\mathcal{S}_i$, create B copies of it indexed by (v,t) for all $1 \leq t \leq B$. Then for each u and v such that $p_{u,v} > 0$ and for each $1 \leq t < B$, place an arc $(u,t) \to (v,t+1)$. Finally, delete all vertices that are not reachable from the state $(\rho_i, 1)$ where ρ_i is the starting state of arm i. There is a clear correspondence between the transitions in $\mathcal{S}_i$ and the ones in this layered graph: whenever state u is played at time t and $\mathcal{S}_i$ transitions to state v, we have the transition from (u,t) to $(v,t+1)$ in the layered DAG. Henceforth, we shall assume that the layered graph created in this manner is the transition graph for each arm.

4.8.2 Our Techniques

While we can again write an LP relaxation of the problem for layered DAGs, the challenge arises in the rounding algorithm: specifically, in (i) obtaining the convex decomposition of the LP solution as in Phase I, and (ii) eliminating small gaps as in Phase II by advancing forests in the strategy.

- We handle the first difficulty by considering convex decompositions not just over strategy forests, but over slightly more sophisticated strategy DAGs. Recall (from Figure 4.1) that in the tree case, each state in a strategy forest was labeled by a unique time and a unique probability associated with that time step. As the name suggests, we now have labeled DAGs—but the change is more than just that. Now each state has a copy associated with *each* time step in $\{1,\ldots,B\}$. This change tries to capture the fact that our strategy may play from a particular state u at different times depending on the path taken by the random transitions used to reach this state. (This path was unique in the tree case.)
- Now having sampled a strategy DAG for each arm, one can expand them out into strategy forests (albeit with an exponential blow-up in the size), and use Phases II and III from

our previous algorithm—it is not difficult to prove that this algorithm is a constant-factor approximation. However, the above algorithm would not be efficient, since the size of the strategy forests may be exponentially large. If we don't expand the DAG, then we do not see how to define gap elimination for Phase II.

The following observation though, comes to our rescue to overcome this issue: instead of explicitly performing the advance steps in Phase II, it in fact suffices to perform them purely as *thought experiments*—i.e., to not alter the strategy forest at all, but merely to infer when these advances would have happened, and play accordingly in the Phase III [6]. Using this, we can give an algorithm that plays just on the DAG, and argue that the sequence of plays made by our DAG algorithm faithfully mimics the execution if we had constructed the exponential-size tree from the DAG, and executed Phases II and III on that tree.

The details of the LP rounding algorithm for layered DAGs follows in Sections 4.8.3-4.8.5.

4.8.3 LP Relaxation

There is only one change in the LP—constraint (4.10) now says that if a state u is visited at time t, then one of its ancestors must have been pulled at time $t-1$; this ancestor was unique in the case of trees.

$$\max \textstyle\sum_{u,t} r_u \cdot z_{u,t} \qquad (\mathsf{LP}_{\mathsf{mabdag}})$$

$$w_{u,t} = \sum_v z_{v,t-1} \cdot p_{v,u} \qquad \forall t \in [2,B],\ u \in \mathcal{S} \setminus \cup_i \{\rho_i\},\ v \in \mathcal{S} \qquad (4.10)$$

$$\textstyle\sum_{t' \le t} w_{u,t'} \ge \sum_{t' \le t} z_{u,t'} \qquad \forall t \in [1,B],\ u \in \mathcal{S} \qquad (4.11)$$

$$\textstyle\sum_{u \in \mathcal{S}} z_{u,t} \le 1 \qquad \forall t \in [1,B] \qquad (4.12)$$

$$w_{\rho_i,1} = 1 \qquad \forall i \in [1,n] \qquad (4.13)$$

Again, a similar analysis to the tree case shows that this is a valid relaxation, and hence the LP value is at least the optimal expected reward.

4.8.4 Convex Decomposition: The Altered Phase I

This is the step which changes the most—we need to incorporate the notion of peeling out a "strategy DAG" instead of just a tree. The main complication arises from the fact that a play of a state u may occur at different times in the LP solution, depending on the path to reach state u in the transition DAG. However, we don't need to keep track of the entire history used to reach u, just how much time has elapsed so far. With this in mind, we create B copies of each state u (which will be our nodes in the strategy DAG), indexed by (u,t) for $1 \le t \le B$.

The j^{th} *strategy dag* $\mathbb{D}(i,j)$ for arm i is an assignment of values $\mathsf{prob}(i,j,u,t)$ and a relation '$\to$' from 4-tuples to 4-tuples of the form $(i,j,u,t) \to (i,j,v,t')$ such that the following properties hold:

[6]This is similar to the idea of lazy evaluation of strategies. The DAG contains an implicit randomized strategy which we make explicit as we toss coins of the various outcomes using an algorithm.

(i) For $u, v \in \mathcal{S}_i$ such that $p_{u,v} > 0$ and any time t, there is exactly one time $t' \geq t + 1$ such that $(i, j, u, t) \to (i, j, v, t')$. Intuitively, this says if the arm is played from state u at time t and it transitions to state v, then it is played from v at a unique time t', if it played at all. If $t' = \infty$, the play from v never happens.

(ii) For any $u \in \mathcal{S}_i$ and time $t \neq \infty$, $\mathsf{prob}(i, j, u, t) = \sum_{(v,t') \text{ s.t } (i,j,v,t') \to (i,j,u,t)} \mathsf{prob}(i, j, v, t') \cdot p_{v,u}$.

For clarity, we use the following notation throughout the remainder of the section: *states* refer to the states in the original transition DAG, and *nodes* correspond to the tuples (i, j, u, t) in the strategy DAGs. When i and j are clear in context, we may simply refer to a node of the strategy DAG by (u, t).

Equipped with the above definition, our convex decomposition procedure appears in Algorithm 11. The main subroutine involved is presented first (Algorithm 10). This subroutine, given a fractional solution, identifies the structure of the DAG that will be peeled out, depending on when the different states are first played fractionally in the LP solution. Since we have a layered DAG, the notion of the *depth* of a state is well-defined as the number of hops from the root to this state in the DAG, with the depth of the root being 0.

Algorithm 10 Sub-Routine PeelStrat (i,j)

```
1: mark $(\rho_i, t)$ where $t$ is the earliest time s.t. $z_{\rho_i,t} > 0$ and set $\mathsf{peelProb}(\rho_i, t) = 1$. All other
   nodes are un-marked and have $\mathsf{peelProb}(v, t') = 0$.
2: while $\exists$ a marked unvisited node do
3:    let $(u, t)$ denote the marked node of smallest depth and earliest time; update its status to
      visited.
4:    for every $v$ s.t. $p_{u,v} > 0$ do
5:       if there is $t'$ such that $z_{v,t'} > 0$, consider the earliest such $t'$ and then
6:          mark $(v, t')$ and set $(i, j, u, t) \to (i, j, v, t')$; update $\mathsf{peelProb}(v, t') :=$
            $\mathsf{peelProb}(v, t') + \mathsf{peelProb}(u, t) \cdot p_{u,v}$.
7:       else
8:          set $(i, j, u, t) \to (i, j, v, \infty)$ and leave $\mathsf{peelProb}(v, \infty) = 0$.
```

The convex decomposition algorithm is now very easy to describe with the sub-routine in Algorithm 10 in hand.

An illustration of a particular DAG and a strategy dag $\mathbb{D}(i, j)$ peeled off is given in Figure 4.3 (notice that the states w, y and z appear more than once depending on the path taken to reach them).

Now we analyze the solutions $\{z^j, w^j\}$ created by Algorithm 11.

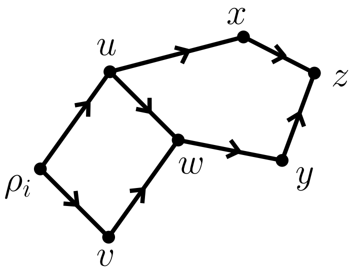
Lemma 4.8.1 *Consider an integer j and suppose that $\{z^{j-1}, w^{j-1}\}$ satisfies constraints* (4.1)-(4.3) *of* $\mathsf{LP}_{\mathsf{mabdag}}$. *Then after iteration j of Step 2, the following properties hold:*

(a) $\mathbb{D}(i, j)$ (along with the associated $\mathsf{prob}(i, j, ., .)$ *values) is a valid strategy dag, i.e., satisfies the conditions (i) and (ii) presented above.*

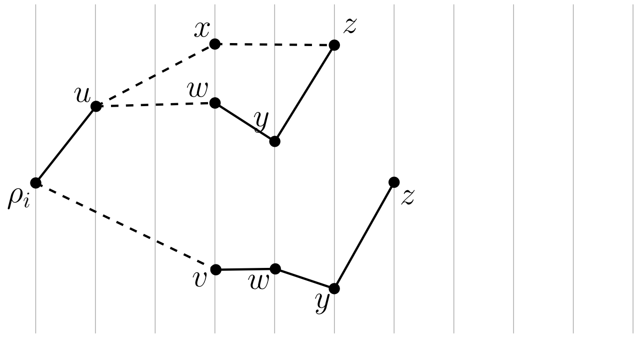
(b) The residual solution $\{z^j, w^j\}$ satisfies constraints (4.10)-(4.12).

Algorithm 11 Convex Decomposition of Arm i

```
1: set $\mathcal{C}_i \leftarrow \emptyset$ and set loop index $j \leftarrow 1$.
2: while $\exists$ a state $u \in \mathcal{S}_i$ s.t. $\sum_t z^{j-1}_{u,t} > 0$ do
3:     run sub-routine PeelStrat to extract a DAG $\mathbb{D}(i,j)$ with the appropriate $\mathsf{peelProb}(u,t)$ values.
4:     let $A \leftarrow \{(u,t)$ s.t $\mathsf{peelProb}(u,t) \neq 0\}$.
5:     let $\varepsilon = \min_{(u,t)\in A} z^{j-1}_{u,t}/\mathsf{peelProb}(u,t)$.
6:     for every $(u,t)$ do
7:         set $\mathsf{prob}(i,j,u,t) = \varepsilon \cdot \mathsf{peelProb}(u,t)$.
8:         update $z^j_{u,t} = z^{j-1}_{u,t} - \mathsf{prob}(i,j,u,t)$.
9:         update $w^j_{v,t+1} = w^{j-1}_{v,t+1} - \mathsf{prob}(i,j,u,t) \cdot p_{u,v}$ for all $v$.
10:    set $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \mathbb{D}(i,j)$.
11:    increment $j \leftarrow j+1$.
```



(a) DAG for some arm i



(b) Strategy dag $\mathbb{D}(i,j)$

Figure 4.3: Strategy DAGs and how to visualize them: notice the same state played at different times

(c) For any time t and state $u \in \mathcal{S}_i$, $z^{j-1}_{u,t} - z^j_{u,t} = \mathsf{prob}(i,j,u,t)$.

Proof: We show the properties stated above one by one.

Property (a): This follows from the construction of Algorithm 10. More precisely, condition (i) is satisfied because in Algorithm 10 each (u,t) is visited at most once and that is the only time when a pair $(u,t) \to (v,t')$ (with $t' \geq t+1$) is added to the relation. For condition (ii), notice that every time a pair $(u,t) \to (v,t')$ is added to the relation we keep the invariant $\mathsf{peelProb}(v,t') = \sum_{(w,\tau) \text{ s.t } (i,j,w,\tau)\to(i,j,v,t')} \mathsf{peelProb}(w,\tau) \cdot p_{w,v}$; condition (ii) then follows since $\mathsf{prob}(.)$ is a scaling of $\mathsf{peelProb}(.)$.

Property (b): Constraint (4.10) of $\mathsf{LP}_{\mathsf{mabdag}}$ is clearly satisfied by the new LP solution $\{z^j, w^j\}$ because of the two updates performed in Steps 8 and 9: if we decrease the z value of any state at any time, the w of all children are appropriately reduced for the subsequent timestep.

Before showing that the solution $\{z^j, w^j\}$ satisfies constraint (4.11), we first argue that after every round of the procedure they remain non-negative. By the choice of ε in Step 5, we have

$\mathsf{prob}(i,j,u,t) = \varepsilon \cdot \mathsf{peelProb}(u,t) \le \frac{z^{j-1}_{u,t}}{\mathsf{peelProb}(u,t)} \mathsf{peelProb}(u,t) = z^{j-1}_{u,t}$ (notice that this inequality holds even if $\mathsf{peelProb}(u,t) = 0$); consequently even after the update in Step 8, $z^j_{u,t} \ge 0$ for all u, t. This and the fact that the constraints (4.10) are satisfied implies that $\{z^j, w^j\}$ satisfies the non-negativity requirement.

We now show that constraint (4.11) is satisfied. Suppose for the sake of contradiction there exist some $u \in \mathcal{S}$ and $t \in [1, B]$ such that $\{z^j, w^j\}$ violates this constraint. Then, let us consider any such u and the earliest time t_u such that the constraint is violated. For such a u, let $t'_u \le t_u$ be the latest time before t_u where $z^{j-1}_{u,t'} > 0$. We now consider two cases.

Case (i): $t'_u < t_u$. This is the simpler case of the two. Because t_u was the earliest time where constraint (4.11) was violated, we know that $\sum_{t' \le t'_u} w^j_{u,t'} \ge \sum_{t' \le t'_u} z^j_{u,t'}$. Furthermore, since $z_{u,t}$ is never increased during the course of the algorithm we know that $\sum_{t'=t'_u+1}^{t_u} z^j_{u,t'} = 0$. This fact coupled with the non-negativity of $w^j_{u,t}$ implies that the constraint in fact is not violated, which contradicts our assumption about the tuple u, t_u.

Case (ii): $t'_u = t_u$. In this case, observe that there cannot be any pair of tuples $(v, t_1) \to (u, t_2)$ s.t. $t_1 < t_u$ and $t_2 > t_u$, because any copy of v (some ancestor of u) that is played before t_u, will mark a copy of u that occurs before t_u or the one being played at t_u in Step 6 of PeelStrat. We will now show that summed over all $t' \le t_u$, the decrease in the LHS is counter-balanced by a corresponding drop in the RHS, between the solutions $\{z^{j-1}, w^{j-1}\}$ and $\{z^j, w^j\}$ for this constraint (4.11) corresponding to u and t_u. To this end, notice that the only times when $w_{u,t'}$ is updated (in Step 9) for $t' \le t_u$, are when considering some (v, t_1) in Step 6 such that $(v, t_1) \to (u, t_2)$ and $t_1 < t_2 \le t_u$. The value of w_{u,t_1+1} is dropped by exactly $\mathsf{prob}(i,j,v,t_1) \cdot p_{v,u}$. But notice that the corresponding term z_{u,t_2} drops by $\mathsf{prob}(i,j,u,t_2) = \sum_{(v'',t'') \text{ s.t } (v'',t'') \to (u,t_2)} \mathsf{prob}(i,j,v'',t'') \cdot p_{v'',u}$. Therefore, the total drop in w is balanced by a commensurate drop in z on the RHS.

Finally, constraint (4.12) is also satisfied as the z variables only decrease in value.

Property (c): This is an immediate consequence of the Step 8 of the convex decomposition algorithm. ■

As a consequence of the above lemma, we get the following.

Lemma 4.8.2 *Given a solution to* ($\mathsf{LP}_{\mathsf{mabdag}}$)*, there exists a collection of at most $nB^2|\mathcal{S}|$ strategy dags $\{\mathbb{D}(i,j)\}$ such that $z_{u,t} = \sum_j \mathsf{prob}(i,j,u,t)$. Hence, $\sum_{(i,j,u)} \mathsf{prob}(i,j,u,t) \le 1$ for all t.*

4.8.5 Phases II and III

We now show how to execute the strategy dags $\mathbb{D}(i,j)$. At a high level, the development of the plays mirrors that of Sections 4.4.2 and 4.4.2. First we transform $\mathbb{D}(i,j)$ into a (possibly exponentially large) blown-up tree and show how this playing these exactly captures playing the strategy dags. Hence (if running time is not a concern), we can simply perform the gap-filling algorithm and make plays on these blown-up trees following Phases II and III in Sections 4.4.2 and 4.4.2. To achieve polynomial running time, we then show that we can *implicitly execute* the gap-filling phase while playing this tree, thus getting rid of actually performing Phase 4.4.2.

Finally, to complete our argument, we show how we do not need to explicitly construct the blown-up tree, and can generate the required portions depending on the transitions made thus far *on demand.*

Transforming the DAG into a Tree

Consider any strategy dag $\mathbb{D}(i,j)$. We first transform this dag into a (possibly exponential) tree by making as many copies of a node (i,j,u,t) as there are paths from the root to (i,j,u,t) in $\mathbb{D}(i,j)$. More formally, define $\mathbb{DT}(i,j)$ as the tree whose vertices are the simple paths in $\mathbb{D}(i,j)$ which start at the root. To avoid confusion, we will explicitly refer to vertices of the tree $\mathbb{DT}$ as tree-nodes, as distinguished from the *nodes* in $\mathbb{D}$; to simplify the notation we identify each tree-node in $\mathbb{DT}$ with its corresponding path in $\mathbb{D}$. Given two tree-nodes P, P' in $\mathbb{DT}(i,j)$, add an arc from P to P' if P' is an immediate extension of P, i.e., if P corresponds to some path $(i,j,u_1,t_1) \to \ldots \to (i,j,u_k,t_k)$ in $\mathbb{D}(i,j)$, then P' is a path $(i,j,u_1,t_1) \to \ldots \to (i,j,u_k,t,k) \to (i,j,u_{k+1},t_{k+1})$ for some node (i,j,u_{k+1},t_{k+1}).

For a tree-node $P \in \mathbb{DT}(i,j)$ which corresponds to the path $(i,j,u_1,t_1) \to \ldots \to (i,j,u_k,t_k)$ in $\mathbb{D}(i,j)$, we define $\mathsf{state}(P) = u_k$, i.e., $\mathsf{state}(\cdot)$ denotes the final state (in $\mathcal{S}_i$) in the path P. Now, for tree-node $P \in \mathbb{DT}(i,j)$, if $u_1, \ldots, u_k$ are the children of $\mathsf{state}(P)$ in $\mathcal{S}_i$ with positive transition probability from $\mathsf{state}(P)$, then P has exactly k children $P_1, \ldots, P_k$ with $\mathsf{state}(P_l)$ equal to u_l for all $l \in [k]$. The *depth* of a tree-node P is defined as the depth of $\mathsf{state}(P)$.

We now define the quantities time and prob for tree-nodes in $\mathbb{DT}(i,j)$. Let P be a path in $\mathbb{D}(i,j)$ from ρ_i to node (i,j,u,t). We define $\mathsf{time}(P) := t$ and $\mathsf{prob}(P) := \mathsf{prob}(P')p_{(\mathsf{state}(P'),u)}$, where P' is obtained by dropping the last node from P. The blown-up tree $\mathbb{DT}(i,j)$ of our running example $\mathbb{D}(i,j)$ (Figure 4.3) is given in Figure 4.4.

Lemma 4.8.3 *For any state u and time t, $\sum_{P \text{ s.t } \mathsf{time}(P)=t \text{ and } \mathsf{state}(P)=u} \mathsf{prob}(P) = \mathsf{prob}(i,j,u,t)$.*

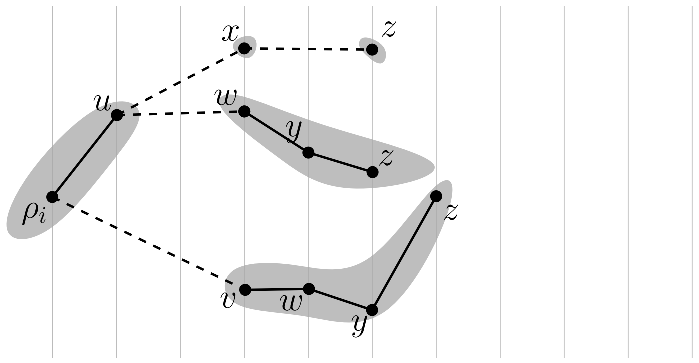


Figure 4.4: Blown-up Strategy Forest $\mathbb{DT}(i,j)$

Now that we have a tree labeled with prob and time values, the notions of connected components and heads from Section 4.4.2 carry over. Specifically, we define $\mathsf{head}(P)$ to be the ancestor P' of P in $\mathbb{DT}(i,j)$ with least depth such that there is a path $(P' = P_1 \to \ldots \to P_l = P)$ satisfying $\mathsf{time}(P_i) = \mathsf{time}(P_{i-1}) + 1$ for all $i \in [2,l]$, i.e., the plays are made contiguously from $\mathsf{head}(P)$ to P in the blown-up tree. We also define $\mathsf{comp}(P)$ as the set of all tree-nodes P' such that $\mathsf{head}(P) = \mathsf{head}(P')$.

In order to play the strategies $\mathbb{DT}(i,j)$ we first eliminate small gaps. The algorithm GapFill presented in Section 4.4.2 can be employed for this purpose and returns trees $\mathbb{DT}'(i,j)$ which satisfy the analog of Lemma 4.4.4.

Lemma 4.8.4 *The trees returned by* GapFill *satisfy the followings properties.*

(i) For each tree-node P such that $r_{\mathsf{state}(P)} > 0$, $\mathsf{time}(\mathsf{head}(P)) \geq 2 \cdot \mathsf{depth}(\mathsf{head}(P))$.

(ii) The total extent of plays at any time t, i.e., $\sum_{P:\mathsf{time}(P)=t} \mathsf{prob}(P)$ is at most 3.

Now we use Algorithm 7 to play the trees $\mathbb{DT}(i,j)$. We restate the algorithm to conform with the notation used in the trees $\mathbb{DT}(i,j)$.

Algorithm 12 Scheduling the Connected Components: Algorithm AlgDAG

```
1:  for arm $i$, sample strategy $\mathbb{DT}(i,j)$ with probability $\frac{\mathsf{prob}(\mathsf{root}(\mathbb{DT}(i,j)))}{24}$; ignore arm $i$ w.p. $1 - \sum_j \frac{\mathsf{prob}(\mathsf{root}(\mathbb{DT}(i,j)))}{24}$.
2:  let $A \leftarrow$ set of "active" arms which chose a strategy in the random process.
3:  for each $i \in A$, let $\sigma(i) \leftarrow$ index $j$ of the chosen $\mathbb{DT}(i,j)$ and let $\mathsf{currnode}(i) \leftarrow$ root of $\mathbb{DT}(i,\sigma(i))$.
4:  while active arms $A \neq \emptyset$ do
5:     let $i^* \leftarrow$ arm with tree-node played earliest (i.e., $i^* \leftarrow \mathrm{argmin}_{i \in A}\{\mathsf{time}(\mathsf{currnode}(i))\}$).
6:     let $\tau \leftarrow \mathsf{time}(\mathsf{currnode}(i^*))$.
7:     while $\mathsf{time}(\mathsf{currnode}(i^*)) \neq \infty$ and $\mathsf{time}(\mathsf{currnode}(i^*)) = \tau$ do
8:        play arm $i^*$ at state $\mathsf{state}(\mathsf{currnode}(i^*))$
9:        let $u$ be the new state of arm $i^*$ and let $P$ be the child of $\mathsf{currnode}(i^*)$ satisfying $\mathsf{state}(P) = u$.
10:       update $\mathsf{currnode}(i^*)$ to be $P$; let $\tau \leftarrow \tau + 1$.
11:    if $\mathsf{time}(\mathsf{currnode}(i^*)) = \infty$ then
12:       let $A \leftarrow A \setminus \{i^*\}$
```

Now an argument identical to that for Theorem 4.4.9 gives us the following:

Theorem 4.8.5 *The reward obtained by the algorithm* AlgDAG *is at least a constant fraction of the optimum for* ($\mathsf{LP}_{\mathsf{mabdag}}$).

Implicit gap filling

Our next goal is to execute GapFill implicitly, that is, to incorporate the gap-filling within Algorithm AlgDAG without having to explicitly perform the advances.

To do this, let us review some properties of the trees returned by GapFill. For a tree-node P in $\mathbb{DT}(i,j)$, let $\mathsf{time}(P)$ denote the associated time in the original tree (i.e., before the application of GapFill) and let $\mathsf{time}'(P)$ denote the time in the modified tree (i.e., after $\mathbb{DT}(i,j)$ is modified by GapFill).

Claim 4.8.6 *For a non-root tree-node P and its parent P',* $\mathsf{time}'(P) = \mathsf{time}'(P') + 1$ *if and only if, either* $\mathsf{time}(P) = \mathsf{time}(P') + 1$ *or* $2 \cdot \mathsf{depth}(P) > \mathsf{time}(P)$.

Proof: Let us consider the forward direction. Suppose $\mathsf{time}'(P) = \mathsf{time}'(P') + 1$ but $\mathsf{time}(P) > \mathsf{time}(P') + 1$. Then P must have been the head of its component in the original tree and an **advance** was performed on it, so we must have $2 \cdot \mathsf{depth}(P) > \mathsf{time}(P)$.

For the reverse direction, if $\mathsf{time}(P) = \mathsf{time}(P') + 1$ then P could not have been a head since it belongs to the same component as P' and hence it will always remain in the same component as P' (as GapFill only merges components and never breaks them apart). Therefore, $\mathsf{time}'(P) = \mathsf{time}'(P') + 1$. On the other hand, if $\mathsf{time}(P) > \mathsf{time}(P') + 1$ and $2 \cdot \mathsf{depth}(P) > \mathsf{time}(P)$, then P was a head in the original tree, and because of the above criterion, GapFill must have made an advance on P' thereby including it in the same component as P; so again it is easy to see that $\mathsf{time}'(P) = \mathsf{time}'(P') + 1$. ■

The crucial point here is that whether or not P is in the same component as its predecessor after the gap-filling (and, consequently, whether it was played contiguously along with its predecessor should that transition happen in AlgDAG) can be inferred from the time values of P, P' before gap-filling and from the depth of P—it does not depend on any other **advance**s that happen during the gap-filling.

Algorithm 13 is a procedure which plays the original trees $\mathbb{DT}(i, j)$ while implicitly performing the **advance** steps of GapFill (by checking if the properties of Claim 4.8.6 hold). This change is reflected in Step 7 where we may play a node even if it is not contiguous, so long it satisfies the above stated properties. Therefore, as a consequence of Claim 4.8.6, we get the following Lemma that the plays made by ImplicitFill are identical to those made by AlgDAG after running GapFill.

Algorithm 13 Filling gaps implicitly: Algorithm ImplicitFill

```
1:  for arm $i$, **sample** strategy $\mathbb{DT}(i,j)$ with probability $\frac{\mathsf{prob}(\mathsf{root}(\mathbb{DT}(i,j)))}{24}$; ignore arm $i$ w.p. $1 - \sum_j \frac{\mathsf{prob}(\mathsf{root}(\mathbb{DT}(i,j)))}{24}$.
2:  let $A \leftarrow$ set of "active" arms which chose a strategy in the random process.
3:  for each $i \in A$, **let** $\sigma(i) \leftarrow$ index $j$ of the chosen $\mathbb{DT}(i,j)$ and **let** $\mathsf{currnode}(i) \leftarrow$ root of $\mathbb{DT}(i,\sigma(i))$.
4:  **while** active arms $A \neq \emptyset$ **do**
5:      **let** $i^* \leftarrow$ arm with state played earliest (i.e., $i^* \leftarrow \mathrm{argmin}_{i \in A}\{\mathsf{time}(\mathsf{currnode}(i))\}$).
6:      **let** $\tau \leftarrow \mathsf{time}(\mathsf{currnode}(i^*))$.
7:      **while** $\mathsf{time}(\mathsf{currnode}(i^*)) \neq \infty$ **and** ($\mathsf{time}(\mathsf{currnode}(i^*)) = \tau$ **or** $2 \cdot \mathsf{depth}(\mathsf{currnode}(i^*)) > \mathsf{time}(\mathsf{currnode}(i^*))$) **do**
8:          **play** arm $i^*$ at state $\mathsf{state}(\mathsf{currnode}(i^*))$
9:          **let** $u$ be the new state of arm $i^*$ and **let** $P$ be the child of $\mathsf{currnode}(i^*)$ satisfying $\mathsf{state}(P) = u$.
10:         **update** $\mathsf{currnode}(i^*)$ to be $P$; **let** $\tau \leftarrow \tau + 1$.
11:     **if** $\mathsf{time}(\mathsf{currnode}(i^*)) = \infty$ **then**
12:         **let** $A \leftarrow A \setminus \{i^*\}$
```

Lemma 4.8.7 *Algorithm* ImplicitFill *obtains the same reward as algorithm* AlgDAG $\circ$ GapFill.

Running ImplicitFill in Polynomial Time

With the description of ImplicitFill, we are almost complete with our proof with the exception of handling the exponential blow-up incurred in moving from $\mathbb{D}$ to $\mathbb{DT}$. To resolve this, we now argue that while the blown-up $\mathbb{DT}$ made it easy to visualize the transitions and plays made, all of it can be done implicitly from the strategy DAG $\mathbb{D}$. Recall that the tree-nodes in $\mathbb{DT}(i,j)$ correspond to simple paths in $\mathbb{D}(i,j)$. In the following, the final algorithm we employ (called ImplicitPlay) is simply the algorithm ImplicitFill, but with the exponentially blown-up trees $\mathbb{DT}(i,\sigma(i))$ being generated *on-demand*, as the different transitions are made. We now describe how this can be done.

In Step 3 of ImplicitFill, we start off at the roots of the trees $\mathbb{DT}(i,\sigma(i))$, which corresponds to the single-node path corresponding to the root of $\mathbb{D}(i,\sigma(i))$. Now, at some point in time in the execution of ImplicitFill, suppose we are at the tree-node currnode(i^*), which corresponds to a path Q in $\mathbb{D}(i,\sigma(i))$ that ends at $(i,\sigma(i),v,t)$ for some v and t. The invariant we maintain is that, in our algorithm ImplicitPlay, we are at node $(i,\sigma(i),v,t)$ in $\mathbb{D}(i,\sigma(i))$. Establishing this invariant would show that the two runs ImplicitPlay and ImplicitFill would be identical, which when coupled with Theorem 4.8.5 would complete the proof—the information that ImplicitFill uses of Q, namely time(Q) and depth(Q), can be obtained from $(i,\sigma(i),v,t)$.

The invariant is clearly satisfied at the beginning, for the different root nodes. Suppose it is true for some tree-node currnode(i), which corresponds to a path Q in $\mathbb{D}(i,\sigma(i))$ that ends at $(i,\sigma(i),v,t)$ for some v and t. Now, suppose upon playing the arm i at state v (in Step 8), we make a transition to state u (say), then ImplicitFill would find the unique child tree-node P of Q in $\mathbb{DT}(i,\sigma(i))$ with state(P) $= u$. Then let $(i,\sigma(i),u,t')$ be the last node of the path P, so that P equals Q followed by $(i,\sigma(i),u,t')$.

But, since the tree $\mathbb{DT}(i,\sigma(i))$ is just an expansion of $\mathbb{D}(i,\sigma(i))$, the unique child P in $\mathbb{DT}(i,\sigma(i))$ of tree-node Q which has state(P) $= u$, is (by definition of $\mathbb{DT}$) the unique node $(i,\sigma(i),u,t')$ of $\mathbb{D}(i,\sigma(i))$ such that $(i,\sigma(i),v,t) \to (i,\sigma(i),u,t')$. Hence, just as ImplicitFill transitions to P in $\mathbb{DT}(i,\sigma(i))$ (in Step 9), we can transition to the state $(i,\sigma(i),u,t')$ with just $\mathbb{D}$ at our disposal, thus establishing the invariant.

For completeness, we present the implicit algorithm below.

4.9 Conclusions

In this chapter, we presented constant-factor approximation algorithms for the MAB problem, generalizing earlier results (e.g., [55, 60]) which relied on a Martingale-like assumption for the behavior of each arm. There are still several interesting problems in this area, however. Our algorithms crucially depend on the independence arcoss arms, and also the fact that the problem is *restless*— playing one arm does not affect the state of other arms. What if other arms passively make transitions as well? What if there is delayed feedback in the mechanism— we only get to see the resulting state of a random transition after a pre-specified delay? While there has been some work along these directions [62], we still lack a complete understanding.

Algorithm 14 Algorithm ImplicitPlay

```
1: for arm $i$, **sample** strategy $\mathbb{D}(i,j)$ with probability $\frac{\mathsf{prob}(\mathsf{root}(\mathbb{D}(i,j)))}{24}$; ignore arm $i$ w.p. $1 - \sum_j \frac{\mathsf{prob}(\mathsf{root}(\mathbb{D}(i,j)))}{24}$.
2: let $A \leftarrow$ set of "active" arms which chose a strategy in the random process.
3: for each $i \in A$, **let** $\sigma(i) \leftarrow$ index $j$ of the chosen $\mathbb{D}(i,j)$ and **let** $\mathsf{currnode}(i) \leftarrow$ root of $\mathbb{D}(i,\sigma(i))$.
4: **while** active arms $A \neq \emptyset$ **do**
5:     **let** $i^* \leftarrow$ arm with state played earliest (i.e., $i^* \leftarrow \operatorname{argmin}_{i \in A}\{\mathsf{time}(\mathsf{currnode}(i))\}$).
6:     **let** $\tau \leftarrow \mathsf{time}(\mathsf{currnode}(i^*))$.
7:     **while** $\mathsf{time}(\mathsf{currnode}(i^*)) \neq \infty$ **and** ($\mathsf{time}(\mathsf{currnode}(i^*)) = \tau$ **or** $2 \cdot \mathsf{depth}(\mathsf{currnode}(i^*)) > \mathsf{time}(\mathsf{currnode}(i^*))$) **do**
8:         **play** arm $i^*$ at state $\mathsf{state}(\mathsf{currnode}(i^*))$
9:         **let** $u$ be the new state of arm $i^*$.
10:        **update** $\mathsf{currnode}(i^*)$ to be $u$; **let** $\tau \leftarrow \tau + 1$.
11:    **if** $\mathsf{time}(\mathsf{currnode}(i^*)) = \infty$ **then**
12:        **let** $A \leftarrow A \setminus \{i^*\}$
```

Chapter 5

Stochastic Orienteering

Our final focus in this thesis is on the orienteering problem with stochastic items located at nodes. This problem is essentially a generalization of the Stochastic Knapsack problem, but where the jobs are now located at vertices on a metric, and the algorithm must actually be at the corresponding location to process the job.

For a practical motivation of this orienteering problem, consider the following setting: you start your day at home with a set of chores to run at various locations (e.g., at the bank, the post office, the grocery store), but you only have limited time to run those chores in (say, you have from 9am until 5pm, when all these shops close). Each successfully completed chore/job j gives you some fixed reward r_j. You know the time it takes you to travel between the various job locations: these distances are deterministic form a metric (V, d). However, you do not know the amount of time you will spend doing each job (e.g., standing in the queue, filling out forms). Instead, for each job j, you are only given the probability distribution π_j governing the random amount of time you need to spend performing j. That is, once you start performing the job j, the job finishes after size_j time units and you get the reward, where size_j is a random variable denoting the size, and distributed according to π_j.[1] (You may cancel processing the job j prematurely, but in that case you don't get any reward, and you are barred from trying j again.) The goal is now a natural one: given the metric (V, d), the starting point ρ, the time budget B, and the probability distributions for all the jobs, give a strategy for traveling around and doing the jobs that maximizes the expected reward accrued.

In a different example, let us again consider the processor scheduling setting. In the StocK problem, we had assumed that the jobs each have random processing times and rewards, and the goal is to maximize the total reward of jobs completed within a total time of B. Now, what if there is a cost associated with switching between tasks? That is, loading job j after completing job j' incurs a time of $d(j, j')$, and we have the same goal of maximizing the total reward of jobs completed in a time of B. Notice that this requirement corresponds to having the total processing times plus switching times being at most B. Furthermore, these $d(\cdot,\cdot)$ clearly form a metric since

[1]To clarify: before you reach the job, all you know about its size is what can be gleaned from the distribution π_j of size_j; and even having worked on j for t units of time, all you know about the actual size of j is what you can infer from the conditional $(\mathsf{size}_j \mid \mathsf{size}_j > t)$.

switching from j to j'' can not take more time than switching from j to j' and then from j' to j''. Therefore, this setting exactly corresponds to our Stochastic Orienteering problem.

Two Special Cases. The case when all the sizes are deterministic (i.e., $\mathsf{size}_j = s_j$ with probability 1) is the orienteering problem, for which we now know a $(2+\varepsilon)$-approximation algorithm [21, 31]. Another special case, where all the chores are located at the start node, but the sizes are random, is the Stochastic Knapsack problem, which also admits a $(2+\varepsilon)$-approximation algorithm [18, 42]. However, the stochastic orienteering problem above, which combines aspects of both these problems, seems to have been hitherto unexplored in the approximation algorithms literature.

Finally, just as in the previous sections, our theoretical motivation stems from analyzing the "adaptivity gap" of this problem, and devising good *non-adaptive* solutions[2].

5.1 Our Results

In this thesis we show we can achieve the following approximations for the stochastic orienteering problem.

Theorem 5.1.1 *There is an $O(\log\log B)$-approximation algorithm for the stochastic orienteering problem.*

Indeed, our proof proceeds by first showing the following structure theorem which bounds the adaptivity gap:

Theorem 5.1.2 *Given an instance of the stochastic orienteering problem, then*

1. *either there exists a single job which gives an $\Omega(\log\log B)$ fraction of the optimal reward, or*
2. *there exists a value W^* such that the optimal non-adaptive tour which spends at most W^* time waiting and $B - W^*$ time traveling, gets an $\Omega(\log\log B)$ fraction of the optimal reward.*

Note that naïvely we would expect only a logarithmic fraction of the reward, but the structure theorem shows we can do better. Indeed, this theorem is the technical heart of the analysis, and is proved via a martingale argument, that we believe could be of independent interest. Since the above theorem shows the existence of a non-adaptive solution close to the best adaptive solution, we can combine it with the following result to prove Theorem 5.1.1.

Theorem 5.1.3 *There exists a constant-factor approximation algorithm to the optimal non-adaptive policy for stochastic orienteering.*

Note that if we could show an existential proof of a constant adaptivity gap (which we conjecture to be true), the above approximation for non-adaptive problems that we show immediately implies an $O(1)$-approximation algorithm for the adaptive problem too.

We then generalize our model and results to the setting where *both the rewards and job sizes* are random, and could be correlated with each other. For this correlated problem, we show:

[2]A non-adaptive solution for stochastic orienteering is simply a tour P of points in the metric space starting at the root ρ: we visit the points in this fixed order, performing the jobs at the points we reach, until time runs out

Theorem 5.1.4 *There is a polynomial-time algorithm that outputs a non-adaptive policy for correlated stochastic orienteering, achieving an $O(\log n \log B)$-approximation to the best adaptive policy. Moreover, this problem is at least as hard as the orienteering-with-deadlines problem.*

The orienteering-with-deadlines problem [7] is the following: given a metric with deadlines at vertices, and a starting vertex ρ, compute a path originating from ρ (at time zero) that maximizes the number of vertices visited before their respective deadlines. The best known approximation algorithm for this problem achieves an $O(\log n)$ ratio [7]. Notice that this negative result is in contrast with our knowledge for the Stochastic Knapsack problem, where we can show $O(1)$-approximation algorithms for both the uncorrelated and correlated versions.

5.1.1 Related Work

The (deterministic) orienteering problem is known to be APX-hard, and the first constant-factor approximation algorithm was due to Blum et al. [21]. Their factor of 4 was improved by [7] and ultimately by [31] to $(2 + \varepsilon)$ for every $\varepsilon > 0$. There is a PTAS known for the orienteering problem on low-dimensional Euclidean space [33]. The orienteering problem has also been useful as a subroutine for obtaining approximation algorithms for other vehicle routing problems such as TSP with deadlines and time-windows [7, 29, 30].

To the best of our knowledge, the stochastic version of the orienteering problem has not been studied before from the perspective of approximation algorithms. Heuristics and empirical guarantees for a similar problem were given by Campbell et al. [23].

As mentioned earlier, the Stochastic Knapsack problem [42] is a special case of stochastic orienteering, where all the jobs are located at the root ρ itself. Dean et al. [42] gave the first constant factor approximation algorithm for this basic problem. In Chapter 3 of this thesis, we considered the extension with *correlated* rewards and sizes, and showed a constant-factor approximation algorithm for it as well.

Another very related body of work is on *budgeted learning with metric switching costs*. Specifically, in the work of Guha and Munagala [60], there is a collection of Markov chains located in a metric, each state of each chain having an associated reward. When at a Markov chain at location j, the policy can advance that chain one step every unit of time. Given a bound of L time units for traveling, and a bound of C time units for advancing Markov chains, the goal is maximize some function (say the sum or the max) of rewards of the final states in expectation. [60] gave an elegant constant factor approximation algorithm for this problem (under some mild conditions on the rewards) via a reduction to classical orienteering using Lagrangean multipliers. Our algorithm/analysis for the "knapsack orienteering" problem (defined in Section 5.4) is inspired by theirs; the analysis of our algorithm though is simpler, due to the problem itself being deterministic. This can be used to obtain a constant-factor approximation algorithm for the variant of stochastic orienteering with two *separate* budgets for travel time and processing time. However, it is unclear how to use the approach from [60] to obtain an approximation ratio better than $O(\log B)$ for the (single budget) stochastic orienteering problem that we consider.

Approximation algorithms have been studied for adaptive versions of a number of combinatorial optimization problems. Many of these results (machine scheduling [89], knapsack [42], budgeted learning [59], matchings [10], and also those in Chapters 3 and 4 in this thesis) are based on LP relaxations that capture certain expected values of the optimal adaptive policy. Such an LP-based approach was also used in earlier optimality proofs for some stochastic queuing problems [78] and the multi-armed bandit problem [16]. However, an LP-based approach is not directly useful for stochastic orienteering since we do not know good LP relaxations for even the deterministic orienteering problem.

On the other hand, there are also other papers (eg. stochastic matchings [34], stochastic knapsack [18, 19], optimal decision trees [2, 67, 85]) that have had to reason about the optimal adaptive policies directly. We hope that our martingale-based analysis for stochastic orienteering will add to the set of tools used for adaptive optimization problems.

5.2 Chapter Roadmap

We begin by highlighting the main technical challenges we encounter, and also our ideas to handle them in Section 5.3. We then define some useful notation in Section 5.4. En route to our algorithm for StocOrient, we first solve a crucial sub-routine which we call *Knapsack Orienteering* in Section 5.5. Then we show a constant-factor approximation to the optimal non-adaptive solution for StocOrient in Section 5.6, and conclude the results of the uncorrelated version by giving an existential proof bounding the adaptivity gap by $O(\log\log B)$ in Section 5.7. Finally, we show our algorithm for the StocOrient problem with correlated rewards in Section 5.9.

5.3 Techniques

A natural approach for StocOrient is to replace stochastic jobs by deterministic ones with size equal to the expected size $E[S_v]$, and find a near-optimal orienteering solution P to the deterministic instance which gets reward R. One can then use this path P to get a non-adaptive policy for the original StocOrient instance with expected reward $\Omega(R)$. Indeed, suppose the path P spends time L traveling and W processing the deterministic jobs such that $L + W \leq B$. Then, picking a random half of the jobs and visiting them results in a non-adaptive solution for StocOrient which travels at most L and processes jobs for time at most $W/2$ in expectation. Hence, Markov's inequality says that with probability at least $1/2$, all jobs finish processing within W time units and we get the entire reward of this sub-path, which is $\Omega(R)$.

However, the problem is in showing that $R = \Omega(\mathsf{OPT})$—i.e., that the deterministic instance has a solution with reward that is comparable to the StocOrient optimum.

The above simplistic reduction of replacing random jobs by deterministic ones with mean size fails even for stochastic knapsack: suppose the knapsack budget is B, and each of the n jobs has size Bn with probability $1/n$, and size 0 otherwise. Note that the expected size of every job is now B. Therefore, a deterministic solution can pick only one job, whereas the optimal solution would finish $\Omega(n)$ jobs with high probability. However, observe that this problem disappears if we truncate all sizes at the budget, i.e., set the deterministic size to be the expected "truncated" size $\mathbb{E}[\min(S_j, B)]$ where S_j is the random size of job j. (Of course, we also have to set the reward to be $r_j\,\mathbb{P}[\mathsf{size}_j \leq B]$ to discount the reward from impossible size realizations.) Now $\mathbb{E}[\min(W_j, B)]$

reduces to B/n and so the deterministic instance can now get $\Omega(n)$ reward. Indeed, this is the approach used by [42] to get an $O(1)$-approximation algorithm and adaptivity gap.

But for StocOrient, is there a good truncation threshold?

Considering $\mathbb{E}[\min(\mathsf{size}_j, B)]$ fails on the example where all jobs are co-located at a point at distance $B-1$ from the root. Each job v has size B with probability $1/B$, and 0 otherwise. Truncation by B gives an expected size $\mathbb{E}_{\mathsf{size}_v \sim \pi_v}[\min(\mathsf{size}_v, B)] = 1$ for every job, and so the deterministic instance gets reward from only one job, while the StocOrient optimum can collect $\Omega(B)$ jobs. Now noticing that any algorithm *has* to spend $B-1$ time traveling to reach any vertex that has some job, we can instead truncate each job j's size at $B - d(\rho, j)$, which is the maximum amount of time we can possibly spend at j (since we must reach vertex j from ρ). However, while this fix works for the aforementioned example, the following example shows that such a deterministic instance might only get an $O(\frac{\log\log B}{\log B})$ fraction of the optimal stochastic reward.

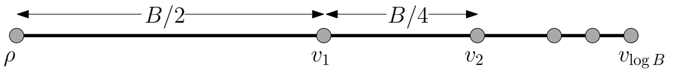


Figure 5.1: Bad example for replacing by expectations.

Consider $n = \log B$ jobs on a line as in Figure 5.3. For $i = 1, 2, \ldots, \log B$, the i^{th} job is at distance $B(1 - 1/2^i)$ from the root ρ; job i takes on size $B/2^i$ with probability $p := 1/\log B$ and size 0 otherwise. Each job has unit reward. The optimal (adaptive and non-adaptive) solution to this instance is to try all the jobs in order $1, 2, \ldots, \log B$: with probability $(1-p)^{\log B} \approx 1/e$, all the jobs instantiate to size 0 and we will accrue reward $\Omega(\log B)$.

In the deterministic orienteering instance, each job i has its expected truncated size $\mu_i = \mathbb{E}[\min\{S_i, B - d(\rho, i)\}] = B/(2^i \log B)$. A feasible solution consists of a subset of jobs where the total travel plus expected sizes is at most B. Suppose j is the first job we pick along the line, then because of its size being μ_j we cannot reach any jobs in the last μ_j length of the path. The number of these lost jobs is $\log \mu_j = \log B - j - \log\log B$ (because of the geometrically decreasing gaps between jobs), and hence we can reach only jobs $j, j+1, j+\log\log B - 1$—giving us a maximum profit of $\log\log B$ even if we ignore the space these jobs would take. (In fact, since their sizes decrease geometrically, we can indeed get all but a constant number of these jobs.)

This shows that replacing jobs in a StocOrient instance by their expected truncated sizes gives a deterministic instance whose optimal reward is smaller by an $\Omega(\frac{\log B}{\log\log B})$ factor.

5.3.1 Our Approach: Reduction to Knapsack Orienteering

The reason why the deterministic techniques described above worked for Stochastic Knapsack, but failed for Stochastic Orienteering is the following: the total sizes of jobs is always roughly B in knapsack (so truncating at B was the right thing to do). But in orienteering it depends on the total time spent traveling, *which in itself is a random quantity, even for a non-adaptive solution.* One way around this is to guess the amount of time W spent processing jobs (up to a factor of 2) which gets the largest profit, and use that as the truncation threshold, to define a knapsack orienteering instance. It seems that such an approach should lose an $\Omega(\log B)$ fraction of the optimal reward, since there are $\log_2 B$ choices for the truncation parameter W. Somewhat surprisingly, we show that this algorithm actually gives a much better reward: it achieves a constant factor approximation relative to a non-adaptive optimum, and an $O(\log \log B)$-approximation when compared to the adaptive optimum!

Step 1: Enumerate over all choices for the truncation threshold W. Construct a suitable instance $\mathcal{I}_{ko}(W)$ of *Knapsack Orienteering* ($\mathsf{KnapOrient}$), with the guarantee that the optimal reward from this $\mathsf{KnapOrient}$ instance $\mathcal{I}_{ko}(W)$ is at least OPT/α.

Step 2: Use Theorem 5.5.1 on $\mathcal{I}_{ko}$ to find a path P with reward $\Omega(\mathsf{OPT}/\alpha)$.

Step 3: Convert this $\mathsf{KnapOrient}$ solution P into a non-adaptive policy for $\mathsf{StocOrient}$ (Theorem 5.6.4).

Given an instance $\mathcal{I}_{so}$ of $\mathsf{StocOrient}$ with optimal (non-adaptive or adaptive) solution having expected reward OPT, our algorithm is outlined above. However, there are many details to be addressed, and we flesh out the details of this algorithm over the next two sections. We will prove that $\alpha = O(1)$ for non-adaptive $\mathsf{StocOrient}$, and $\alpha = O(\log \log B)$ in the adaptive case.

5.4 Definitions and Notation

Stochastic Orienteering. An instance of Stochastic Orienteering ($\mathsf{StocOrient}$) is defined on an underlying metric space (V, d) with ground set $|V| = n$ and symmetric integer distances $d : V \times V \to \mathbb{Z}^+$ (satisfying the triangle inequality) that represent travel times. Each vertex $v \in V$ is associated with a unique stochastic job, which we also call v. For the remainder of this chapter, each job v has a fixed reward $r_v \in \mathbb{Z}_{\geq 0}$; and a random processing time (a.k.a. size) size_v, which is distributed according to a known but arbitrary probability distribution $\pi_v : \mathbb{R}^+ \to [0, 1]$. We are also given a starting "root" vertex ρ, and a budget B on the total time available.

The only actions allowed to an algorithm are to travel to a vertex v and begin processing the job there: when the job finishes after its random length size_v of time, we get the reward r_v (so long as the total time elapsed, i.e., travel time plus processing time, is at most B), and we can then move to the next job. Recall that this is a non-preemptive model. We show in Section 5.8 that all our results extend to a related model that allows cancellations: here we can cancel any job at any time without receiving its reward, but we are not allowed to attempt this job again in the future.

Note that any solution (policy) corresponds to a decision tree where each "state" depends on which previous jobs were processed, and what information we obtained about their sizes. Now the goal is to devise a policy which, starting at the root ρ, decides for each possible state the next job to visit and process. Such a policy is called "non-anticipatory" due to the fact that its action at any point in time can only depend on already observed information. The objective is to obtain a policy that maximizes the expected sum of rewards of jobs successfully completed before the total time (travel and processing) reaches the threshold of B. The approximation ratio of an algorithm is defined to be the ratio of the expected reward of an optimal policy to that of the algorithm's policy.

Adaptive and Non-Adaptive Policies. We are interested in both adaptive and non-adaptive policies, and in particular, want to bound the ratio between the performance of the best adaptive and best non-adaptive policies. An *adaptive policy* is a decision tree where each node is labeled by a job/vertex of V, with the outgoing arcs from a node labeled by j corresponding to the possible sizes in the support of π_j. A *non-adaptive policy*, on the other hand, is simply given by a path P starting at ρ; we just traverse this path, processing the jobs that we encounter, until the total (random) size of the jobs plus the distance traveled exceeds B. A *randomized non-adaptive policy* may pick a path P at random from some distribution before it knows any of the size instantiations, and then follows this path as above. Note that in a non-adaptive policy, the order in which jobs are processed is independent of their processing time instantiations.

Finally, for any integer $m \geq 0$ we use $[m]$ to denote the set $\{0, 1, \ldots, m\}$.

5.5 The (Deterministic) Knapsack Orienteering Problem

We now define a variant of the orienteering problem which will be crucially used in the rest of the paper. Recall that in the basic orienteering problem, the input consists of a metric (V, d), the root vertex ρ, rewards r_v for each job v, and total budget B. The goal is to find a path P of length at most B starting at ρ that maximizes the total reward $\sum_{v \in P} r_v$ of vertices in P.

In the **knapsack orienteering** problem ($\mathsf{KnapOrient}$), we are given a metric (V, d), root vertex ρ, and two budgets: L which is the "travel" budget, and W which is the "knapsack" budget. Each job v has a reward $\widehat{r}_v$, and now also a "size" $\widehat{s}_v$. A feasible solution is a path P originating at ρ having length at most L, such that the total size $\widehat{s}(P) := \sum_{v \in P} \widehat{s}_v$ is at most W. The goal is to find a feasible solution of maximum reward $\sum_{v \in P} \widehat{r}_v$.

Theorem 5.5.1 *There is a polynomial time $O(1)$-approximation algorithm* AlgKO *for the* $\mathsf{KnapOrient}$ *problem.*

The idea of the proof is consider the *Lagrangian relaxation* of the knapsack constraint; we remark that such an approach was also taken in [60] for a related problem. This way we alter the rewards of items while still optimizing over the set of feasible orienteering solutions. For a suitable choice of the Lagrange parameter, we will show that we can recover a solution with large (unaltered) reward while meeting both the knapsack (W) and length (L) constraints.

Proof: For a value $\lambda \geq 0$, define an orienteering instance $\mathcal{I}(\lambda)$ on metric (V, d) with root ρ, travel budget L, and profits $r_v^\lambda := \widehat{r}_v - \lambda \cdot \widehat{s}_v$ at each $v \in V$. Note that the optimal solution to this orienteering instance has value at least $\mathsf{OPT} - \lambda \cdot W$, where OPT is the optimal value of the original $\mathsf{KnapOrient}$ instance.

Let $\mathsf{Alg}_o(\lambda)$ denote an α-approximate solution to $\mathcal{I}(\lambda)$ as well as its profit; we have $\alpha = 2 + \delta$ via the algorithm from [31]. By exhaustive search, let us find:

$$\lambda^* := \max \left\{ \lambda \geq 0 : \mathsf{Alg}_o(\lambda) \geq \frac{\lambda \cdot W}{\alpha} \right\} \tag{5.1}$$

Observe that by setting $\lambda = \frac{\mathsf{OPT}}{2W}$, we have $\mathsf{Alg}_o(\lambda) \geq \frac{1}{\alpha}(\mathsf{OPT} - \lambda W) = \frac{\mathsf{OPT}}{2\alpha}$. Thus $\lambda^* \geq \frac{\mathsf{OPT}}{2W}$.

Let σ denote the path in solution $\mathsf{Alg}_o(\lambda^*)$, and let $\sum_{v \in \sigma} \widehat{s}_v = y \cdot W$ for some $y \geq 0$. Partition the vertices of σ into $c = \max\{1, \lfloor 2y \rfloor\}$ parts $\sigma_1, \ldots, \sigma_c$ with $\sum_{v \in \sigma_j} \widehat{s}_v \leq W$ for all

$j \in \{1, \ldots, c\}$. This partition can be obtained by greedy aggregation since $\max_{v\in V} \widehat{s}_v \le W$ (all vertices with larger size can be safely excluded by the algorithm). Set $\sigma' \leftarrow \sigma_k$ for $k = \arg\max_{j=1}^{c} \widehat{r}(\sigma_j)$. We then output σ' (which follows path σ but only visits vertices in σ_k) as our approximate solution to the KnapOrient instance. Clearly σ' satisfies both the length and knapsack constraints. It remains to bound the reward we obtain.

$$\begin{aligned} \widehat{r}(\sigma') &\ge \frac{\widehat{r}(\sigma)}{c} \ge \frac{\lambda^* yW + \lambda^* W/\alpha}{c} = \lambda^* W \cdot \left(\frac{y + 1/\alpha}{c}\right) \\ &\ge \lambda^* W \cdot \min\left\{y + \frac{1}{\alpha}, \frac{1}{2} + \frac{1}{2\alpha y}\right\} \ge \frac{\lambda^* W}{\alpha} \end{aligned}$$

The second inequality is by $\widehat{r}(\sigma) - \lambda^* \cdot \widehat{s}(\sigma) = \mathsf{Alg}_o(\lambda^*) \ge \frac{\lambda^* W}{\alpha}$ due to the choice (5.1), which implies that $\widehat{r}(\sigma) \ge \lambda^* \cdot \widehat{s}(\sigma) + \frac{\lambda^* \cdot W}{\alpha} = \lambda^* yW + \frac{\lambda^* W}{\alpha}$ by the definition of y. The third inequality is by $c \le \max\{1, 2y\}$. The last inequality uses $\alpha \ge 2$. It follows that $\widehat{r}(\sigma') \ge \frac{\mathsf{OPT}}{2\alpha}$, thus giving us the desired approximation guarantee. ■

As an aside, this Lagrangian approach can be used to obtain a constant-factor approximation algorithm for a two-budget version of Stochastic Orienteering (with separate bounds on travel and processing times). But it is unclear if this can be extended to the single-budget version. In particular, we are not able to show that the Lagrangian relaxation (of processing times) has objective value $\Omega(\mathsf{OPT})$. This is because different decision paths in the OPTtree might vary a lot in their processing times, implying that there is no reasonable candidate for a Lagrange multiplier.

In the next subsection we discuss some simple reductions from StocOrient to deterministic orienteering that fail to achieve a good approximation ratio. This serves as a warm up for our algorithm which reduces StocOrient to KnapOrient; we outline this in Subsection 5.3.1.

5.6 Non-Adaptive Stochastic Orienteering

We first consider the *non-adaptive* StocOrient problem, and present an $O(1)$-approximation algorithm, which proves Theorem 5.1.3. This also contains many ideas used in the more involved analysis of the adaptive setting.

Recall that the input consists of metric (V, d) with each vertex $v \in V$ representing a stochastic job having a deterministic reward $r_v \in \mathbb{Z}_{\ge 0}$ and a random processing time/size size_v distributed according to $\pi_v : \mathbb{R}^+ \to [0, 1]$; we are also given a root ρ and budget B. A non-adaptive policy is an ordering σ of the vertices (starting with ρ), which corresponds to visiting vertices (and processing the respective jobs) in the order σ. The goal in the non-adaptive StocOrient problem is to compute an ordering that maximizes the expected reward, i.e., the total reward of all items which are completed within the budget of B (travel + processing times). We first perform some preprocessing on the input instance. Throughout, OPTwill denote the optimal non-adaptive solution to the given StocOrient instance, as well as its expected reward.

Assumption 5.6.1 *We may assume without loss of generality that:*

1. *No single-vertex solution has expected reward more than* $\mathsf{OPT}/8$.

2. *For each vertex $u \in V$, $\mathbb{P}_{S_u \sim \pi_u}[\, S_u > B - d(\rho, u)\,] \leq 1/2$.*

Proof: (1) Note that we can enumerate over all single vertex solutions (there are only n of them) and output the best one– if any such solution has value greater than $\mathsf{OPT}/8$ then we already have an 8-approximate solution. So the first assumption follows.

(2) For the second assumption, call a vertex u *bad* if $\mathbb{P}_{S_u \sim \pi_u}[\, S_u > B - d(\rho, u)\,] > 1/2$. We show below that all bad vertices can be ignored. Notice that if OPTvisits a bad vertex then the probability that it continues further decreases geometrically by a factor $1/2$, because the total budget is exceeded with probability at least $1/2$. Therefore, the total expected reward that OPTcollects from all bad jobs is at most twice the maximum expected reward from any single bad vertex. By the first assumption, the maximum expected reward from any single vertex is at most $\mathsf{OPT}/8$. So the expected reward obtained by ignoring bad vertices is at least $\frac{3}{4} \cdot \mathsf{OPT}$. ■

Definition 5.6.2 (Truncated Means) *For any vertex $u \in V$ and any positive value $Z \geq 0$, let $\mu_u(Z) := \mathbb{E}_{S_u \sim \pi_u}[\min(S_u, Z)]$ denote the expected size truncated at Z. Note that $\mu_u(Z_1) \leq \mu_u(Z_2)$ and $\mu_u(Z_1 + Z_2) \leq \mu_u(Z_1) + \mu_u(Z_2)$ for all $Z_2 \geq Z_1 \geq 0$.*

Definition 5.6.3 (Valid KnapOrient Instances) *Given an instance $\mathcal{I}_{so}$ of* StocOrient *and value $W \leq B$, define the valid* KnapOrient *instance $\mathcal{I}_{ko}(W) := \mathsf{KnapOrient}(V, d, \{(\widehat{s}_u, r_u) : \forall u \in V\}, L, W, \rho)$ where:*

(i) The travel budget $L = B - W$ and size budget is W.

(ii) For all $u \in V$, its deterministic size $\widehat{s}_u = \mu_u(W)$.

Recall that AlgKO is an $O(1)$-approximation algorithm for KnapOrient. Algorithm 15 for non-adaptive StocOrient proceeds in the following manner: (i) it enumerates over all possible powers-of-two for the choice of size budget W (see the definition of valid KnapOrient instances), (ii) uses AlgKO to find near-optimal solution for each of the valid KnapOrient instances, and finally (iii) converts the best of them into a non-adaptive StocOrient solution. The final part of this procedure is characterized by the following Theorem 5.6.4. The proof is similar to that used in earlier works on Stochastic Knapsack [42].

Theorem 5.6.4 *Given any solution P to* KnapOrient *instance $\mathcal{I}_{ko}(W)$ (for any $W \leq B$) having reward R, we can obtain in polynomial time a non-adaptive policy for* StocOrient *of expected reward $R/12$.*

Proof: To reduce notation, let P also denote the set of vertices visited in the solution to $\mathcal{I}_{ko}(W)$.

$$L := \left\{ u \in P : \mu_u(W) > \frac{W}{4} \right\} \quad \text{and} \quad S := \left\{ u \in P : \mu_u(W) \leq \frac{W}{4} \right\}$$

Notice that $|L| < 4$ since $\sum_{u \in P} \mu_u(W) \leq W$ by the size budget in $\mathcal{I}_{ko}(W)$. By averaging $\max\{r_u : u \in L\} \geq r(L)/3$. Moreover, by Observation 5.6.1 the best single vertex solution (to StocOrient) among L has expected reward at least $\frac{1}{2} \cdot \max\{r_u : u \in L\} \geq r(L)/6$.

Since each $v \in S$ has $\mu_v(W) \leq W/4$ and $\sum_{u \in S} \mu_u(W) \leq W$, we can partition S into 3 parts such that each part has total size at most $W/2$. Again by averaging, one of these parts

$S' \subseteq S$ satisfies $\sum_{u \in S'} \mu_u(W) \leq W/2$ and $r(S') \geq r(S)/3$. Consider the following non-adaptive policy for StocOrient: visit (and process) vertices in S' in the order of P. By triangle inequality, the travel time is at most that of P, namely $B - W$. By Markov's inequality, with probability at least $1/2$, the total processing time of S' is at most W. Hence the expected reward of this policy to StocOrient is at least $\frac{1}{2} \cdot r(S') \geq r(S)/6$.

The better of the two policies above (from L and S) has reward at least $R/12$. ■

Algorithm 15 Algorithm AlgSO for StocOrient on input $\mathcal{I}_{so} = (V, d, \{(\pi_u, r_u) : \forall u \in V\}, B, \rho)$

```
 1: for all $v \in V$ do
 2:     let $R_v := r_v \cdot \mathbb{P}_{S_v \sim \pi_v}[S_v \leq (B - d(\rho, v))]$ be the expected reward of the single-vertex
        solution to $v$.
 3: w.p. $1/2$, just visit the vertex $v$ with the highest $R_v$ and exit.
 4: delete all vertices $u \in V$ with $\mathbb{P}_{S_u \sim \pi_u}[\, S_u > B - d(\rho, u)\,] > 1/2$.
 5: for $i = 0, 1, \ldots, \lceil \log B \rceil$ do
 6:     set $W = B/2^i$
 7:     let $P_i$ be the path returned by AlgKO on the valid KnapOrient instance $\mathcal{I}_{ko}(W)$.
 8:     let $R_i$ be the reward of this KnapOrient solution $P_i$.
 9: let $P_{i^*}$ be the solution among $\{P_i\}_{i \in [\log B]}$ with maximum reward $R_i$.
10: output the non-adaptive StocOrient policy corresponding to $P_{i^*}$, using Theorem 5.6.4.
```

Therefore, in order to prove a constant approximation ratio, it suffices to show the existence of some $W = B/2^i$ for which the optimal value of $\mathcal{I}_{ko}(W)$ is $\Omega(\mathsf{OPT})$. Formally,

Theorem 5.6.5 (Structure Theorem 1) *Given any instance $\mathcal{I}_{so}$ of non-adaptive* StocOrient *satisfying Assumption 5.6.1, there exists $W = B/2^i$ for some $i \in \{0, 1, \ldots, \lceil \log B \rceil\}$ such that $\mathcal{I}_{ko}(W)$ has optimal value $\Omega(\mathsf{OPT})$.*

The rest of this section proves this result.

Without loss of generality, let the optimal non-adaptive ordering be $\{\rho = v_0, v_1, v_2, \ldots, v_n\}$. For any $v_j \in V$ let $D_j = \sum_{i=1}^{j} d(v_{i-1}, v_i)$ denote the *total distance* spent before visiting vertex v_j. Note that while the total time (travel plus processing) spent before visiting any vertex is a random quantity, the distance (i.e. travel time) is deterministic, since we deal with non-adaptive policies. Let v_{j^*} denote the first index j such that

$$\textstyle\sum_{i<j} \mu_{v_i}(B - D_j) \quad \geq \quad K \cdot (B - D_j) \tag{5.2}$$

Here K is some constant which we will fix later. Observe that this condition is trivially satisfied when $D_j = B$; so we may assume, without loss of generality, that $D_{j^*-1} \leq B - 1$.

Lemma 5.6.6 *For index j^* as in* (5.2)*, we have $\sum_{i \leq j^*-1} r_{v_i} \;\geq\; \mathsf{OPT}/2$.*

Proof: We first deal with the corner case that $D_{j^*} = B$. In this case v_{j^*} is the last possible vertex visited by OPT. By Assumption 5.6.1, the expected reward from vertex v_{j^*} even if it is visited directly from the root, is at most $\mathsf{OPT}/8$. So the expected reward from the first $j^* - 1$ vertices is at least $\frac{7}{8} \cdot \mathsf{OPT}$, which implies the lemma.

In the following, we assume that $D_{j^*} \leq B - 1$.

Claim 5.6.7 *The probability that* OPT*visits some vertex indexed* j^* *or higher is at most* $e^{1-K/2}$.

Proof: If OPTvisits vertex v_{j^*} then we have $\sum_{i<j^*} S_{v_i} \leq B - D_{j^*}$. This also implies that $\sum_{i<j^*} \min(S_{v_i}, B - D_{j^*}) \leq B - D_{j^*}$. Now, for each $i < j^*$ let us define a random variable $X_i := \frac{\min(S_{v_i}, B-D_{j^*})}{B-D_{j^*}}$. Note that the X_i's are independent $[0,1]$ random variables, and that $\mathbb{E}[X_i] = \mu_{v_i}(B - D_{j^*})/(B - D_{j^*})$. From this definition, it is also clear that the probability that OPTvisits v_{j^*} is upper bounded by the probability that $\sum_{i<j^*} X_i \leq 1$. To this end, we have from Inequality (5.2) that $\sum_{i<j^*} \mathbb{E}[X_i] \geq K$. Therefore we can apply a standard Chernoff bound to conclude that

$$\mathbb{P}\left[\mathsf{OPT} \text{ visits vertex } v_{j^*}\right] \quad \leq \quad \mathbb{P}\left[\sum_{i<j^*} X_i \leq 1\right] \quad \leq \quad e^{1-K/2}$$

This completes the proof. ■

Claim 5.6.8 *Conditional on reaching* v_{j^*}*, the expected reward obtained by the optimal policy from vertices* $\{v_{j^*}, v_{j^*+1}, \ldots\}$ *is at most* OPT.

Proof: Consider the alternate policy $\{\rho = v_0, v_{j^*}, v_{j^*+1}, \ldots, v_n\}$ that skips all vertices before v_{j^*}. By triangle inequality the distance $d(\rho, v_{j^*}) \leq D_{j^*}$. So the expected reward from this policy is at least the conditional reward of OPTobtained beyond vertex v_{j^*}. The claim now follows by optimality. ■

Combining these two claims, the expected reward from the first $j^* - 1$ vertices is at least $\left(1 - e^{1-K/2}\right) \cdot \mathsf{OPT}$. Setting $K \geq 4$, this implies the lemma. ■

Recall that $D_{j^*-1} \leq B - 1$; let $\ell \in \mathbb{Z}_+$ be such that $B/2^\ell < B - D_{j^*-1} \leq B/2^{\ell-1}$. Set $W^* = B/2^\ell$. We will show that the $\mathsf{KnapOrient}$ instance $\mathcal{I}_{ko}(W^*)$ has optimal value at least $\mathsf{OPT}/(8K+8)$, which would prove Theorem 5.6.5. Consider path $P^* = \langle \rho = v_0, v_1, \ldots, v_{j^*-1} \rangle$. The reward on this path is at least $\mathsf{OPT}/2$ and it satisfies the travel budget $B - W^*$ in $\mathcal{I}_{ko}(W^*)$. The total size on this path is:

$$\begin{aligned}\sum_{i \leq j^*-1} \mu_{v_i}(W^*) \quad &\leq \quad \sum_{i \leq j^*-1} \mu_{v_i}(B - D_{j^*-1}) \quad = \quad \sum_{i<j^*-1} \mu_{v_i}(B - D_{j^*-1}) + \mu_{v_{j^*-1}}(B - D_{j^*-1}) \\ &\leq \quad (K+1)(B - D_{j^*-1}) \quad = \quad 2(K+1)W^*\end{aligned}$$

The second inequality is by choice of j^* in equation (5.2). Although P^* may not satisfy the size budget of W^*, we obtain a subset $P' \subseteq P^*$ that does. Since each vertex has size at most W^* and the total size of P^* is at most $2(K+1)W^*$, there is a partition of P^* into at most $4(K+1)$ parts such that each part has size at most W^*. Choosing the maximum reward part amongst these yields a *feasible* solution $\mathcal{I}_{ko}(W^*)$ of value at least $\frac{\mathsf{OPT}}{8(K+1)}$.

Combining Theorem 5.6.4 and 5.6.5, we obtain Theorem 5.1.3.

5.7 Bounding the Adaptivity Gap

In this section we consider the adaptive StocOrient problem. We will show the same algorithm (Algorithm AlgSO) is an $O(3)$-approximation algorithm to the best adaptive solution, thus proving Theorem 5.1.1. Note that this also establishes an adaptivity gap of $O(\log\log B)$.

Here OPTdenotes the optimal adaptive policy, as well as its expected reward. Assumption 5.6.1 holds in this adaptive setting as well; the proof is almost identical and not repeated here. Recall the definition of valid KnapOrient instances and Theorem 5.6.4. The main result that we need is an analog of Theorem 5.6.5, namely:

Theorem 5.7.1 (Structure Theorem 2) *Given any instance $\mathcal{I}_{so}$ of adaptive* StocOrient *satisfying Assumption 5.6.1, there exists $W = B/2^i$ for some $i \in \{0, 1, \ldots, \lceil \log B \rceil\}$ such that $\mathcal{I}_{ko}(W)$ has optimal value $\Omega(\mathsf{OPT}/\log\log B)$.*

Before we begin, recall the typical instance $\mathcal{I}_{so} := \mathsf{StocOrient}(V, d, \{(\pi_u, r_u) : \forall u \in V\}, B, \rho)$ of the Stochastic Orienteering problem.

Roadmap. We begin by giving a roadmap of the proof. Let us view the optimal adaptive policy OPT as a decision tree where each node is labeled with a vertex/job, and the children correspond to different size instantiations of the job. For any sample path P in this decision tree, consider the first node x_P where the sum of expected sizes of the jobs processed until x_P exceeds the "budget remaining"—here, if $L_{x,P}$ is the total distance traveled from the root ρ to this node x_P by visiting vertices along P, then the remaining budget is $B - L_{x,P}$. Call such a node a *frontier node*, and the *frontier* is the union of all such frontier nodes. To make sense of this definition, note that if the orienteering instance was non-stochastic (and all the sizes would equal the expected sizes), then we would not be able to get any reward from portions of the decision tree on or below the frontier nodes. Unfortunately, since job sizes are random for us, this may not be the case.

The main idea in the proof is to show that we do not lose too much reward by truncation: i.e., even if we truncate OPT along this frontier, we still obtain an expected reward of $\Omega(\mathsf{OPT}/3)$ from the truncated tree. Now an averaging argument says that there exists *some* path P^* of length L where (i) the total rewards of jobs is $\Omega(\mathsf{OPT}/3)$, and (ii) the sum of expected sizes of the jobs is $O(B - L)$ and this gives us the candidate KnapOrient solution.

Viewing OPTas a Discrete Time Stochastic Process. Note that the transitions of the decision tree OPT represent travel between vertices: if the parent node is labeled with vertex u, and its child is labeled with v, the transition takes $d(u, v)$ time. To simplify matters, we take every such transition, and subdivide it into $d(u, v)$ unit length transitions. The intermediate nodes added in are labeled with new dummy vertices, with dummy jobs of deterministic size 0 and reward 0. We denote this tree as OPT', and note the amount of time spent traveling is exactly the number of edges traveled down this tree. (All this is only for analysis.) Now if we start a particle at the root, and let it evolve down the tree based on the random outcomes of job sizes, then the node reached at timestep t corresponds to some job with a random size and reward. This naturally gives us a discrete-time stochastic process $\mathcal{T}$, which at *every* timestep picks a job of size $\mathbf{S}_t \sim \mathcal{D}_t$ and reward $\mathbf{R}_t$. Note that $\mathbf{S}_t, \mathbf{R}_t$ and the probability distribution $\mathcal{D}_t$ all are random variables that depend on the outcomes of the previous timesteps $0, 1, \ldots, t-1$ (since the actual job that the particle sees after going t hops depends on past outcomes). We stop this process at the first (random) timestep t_{end} such that $\sum_{t=0}^{t_{end}} \mathbf{S}_t \geq (B - t_{end})$—this is the natural point to stop, since it is precisely the time step when the total processing plus the total distance traveled exceeds the budget B.

Some notation: *Nodes* will correspond to states of the decision tree OPT', whereas vertices are points in the metric (V, d). The *level* of a node x in OPT' is the number of hops in the decision tree from the root to reach x—this is the timestep when the stochastic process would reach x, or equivalently the travel time to reach the corresponding vertex in the metric. We denote this by $\mathsf{level}(x)$. Let $\mathsf{label}(x)$ be the vertex labeling x. We abuse notation and use S_x, R_x and π_x to denote the size, reward, and size distribution for node x—hence $S_x = S_{\mathsf{label}(x)}$, $R_x = R_{\mathsf{label}(x)}$ and $\pi_x = \pi_{\mathsf{label}(x)}$. We use $x' \preceq x$ to denote that x' is an ancestor of x.

Now to begin the proof of Theorem 5.7.1. Firstly, we assume that there are no co-located stochastic jobs (i.e., there is only one job at every vertex); note that this also implies that we have to travel for a non-zero integral distance between jobs. This is only to simplify the exposition of the proof; we explain how to discharge this assumption at the end of this section.

Defining the Frontiers. Henceforth, we will focus on the decision tree OPT' and the induced stochastic process $\mathcal{T}$. Consider any intermediate node x and the sample path from the root to x in OPT'. We call x a *star node* if x is the first node along this sample path for which the following condition is satisfied:

$$\textstyle\sum_{x' \prec x} \mu_{x'}\left(B - \mathsf{level}(x)\right) \quad \geq \quad 8K\left(B - \mathsf{level}(x)\right) \tag{5.3}$$

Observe that this condition obviously holds when $\mathsf{level}(x) = B$, and that no star node is an ancestor of another star node. To get a sense of this definition of star nodes, ignore the truncation for a moment: then x is a star node if the expected sizes of all the $\mathsf{level}(x)$ jobs on the sample path until x sum to at least $8K(B - \mathsf{level}(x))$. But since we have spent $\mathsf{level}(x)$ time traveling to reach x, the process only continues beyond vertex x if the actual sizes of the jobs is at most $B - \mathsf{level}(x)$; i.e., if the sizes of the jobs are a factor $8K$ smaller than their expectations. If this were an unlikely event, then pruning OPT' at the star nodes would result in little loss of reward. And that is precisely what we show.

Let OPT'' denote the subtree of OPT' obtained by pruning it at star nodes (OPT'' does not include rewards at star nodes). Note that leaf-nodes in OPT'' are either leaves of OPT' or *parents* of star nodes. In particular, $\mathsf{level}(s) \leq B - 1$ for each leaf-node $s \in \mathsf{OPT}''$.

Lemma 5.7.2 *The expected reward in* OPT'' *is at least* $\mathsf{OPT}/2$.

Before proving this lemma, we show how this implies Theorem 5.7.1.

Proof of Theorem 5.7.1: We start with the following claim that uses the definition of star nodes.

Claim 5.7.3 *Every leaf node* $s \in \mathsf{OPT}''$ *satisfies* $\sum_{x \preceq s} \mu_x\left(B - \mathsf{level}(s)\right) \leq 9K\left(B - \mathsf{level}(s)\right)$.

Proof: By definition of OPT'', leaf-node s is not a star node (nor a descendant of one). So:

$$\sum_{x \preceq s} \mu_x\left(B - \mathsf{level}(s)\right) \;=\; \sum_{x \prec s} \mu_x\left(B - \mathsf{level}(s)\right) + \mu_s\left(B - \mathsf{level}(s)\right) \quad < \quad (8K+1)\cdot\left(B - \mathsf{level}(s)\right).$$

The inequality is by (3.8). This proves the claim. ■

Now by an averaging argument, there exists a sample path P^* in OPT'' to leaf node s^* with travel time $\mathsf{level}(s^*)$ such that:

1. The sum of rewards of nodes visited along P^* is at least $\mathsf{OPT}/2$, using Lemma 5.7.2.
2. The sum of means (truncated at $B - \mathsf{level}(s^*)$) of jobs in P^* is at most $9K\,(B - \mathsf{level}(s^*))$, using Claim 5.7.3.

Recall that every leaf in OPT'' has level at most $B-1$, so $\mathsf{level}(s^*) \leq B-1$. Choose $\ell \in \{0, 1, \ldots, \lceil \log B \rceil\}$ so that $B/2^\ell \leq B - \mathsf{level}(s^*) \leq B/2^{\ell+1}$, and set $W^* = B/2^\ell$. Then we have:

$$\sum_{x \preceq s^*} \mu_x(W^*) \quad \leq \quad \sum_{x \preceq s^*} \mu_x(B - \mathsf{level}(s^*)) \quad \leq \quad 9K \cdot (B - \mathsf{level}(s^*)) \quad \leq \quad 18K \cdot W^*$$

Consider the $\mathsf{KnapOrient}$ instance $\mathcal{I}_{ko}(W^*)$; we will show that it has optimal value at least $\Omega(\mathsf{OPT}/K)$, which proves Theorem 5.7.1. Note that path P^* has length $\mathsf{level}(s^*)$ and satisfies the travel budget of $B - W^*$. The above calculation shows that the total size of P^* is at most $18K \cdot W^*$. Using the bin-packing-type argument as in the previous section, we obtain a subset $P' \subseteq P^*$ that has total size at most W^* and reward at least $r(P^*)/(36K) \geq \frac{\mathsf{OPT}}{72K}$. Thus we obtain Theorem 5.7.1. ■

We now prove Lemma 5.7.2. Group the star nodes into $\lceil \log B \rceil + 1$ *bands* based on the value of $B - \mathsf{level}(x)$. Star node x is in band i if $B - \mathsf{level}(x) \in (B/2^{i+1}, B/2^i]$ for $0 \leq i \leq \lceil \log B \rceil$, and in band $\lceil \log B \rceil + 1$ if $\mathsf{level}(x) = B$.

First consider star nodes of band $\lceil \log B \rceil + 1$. Note that the policy terminates after these nodes (since B time units have already been spent traveling). By Assumption 5.6.1, the loss in reward by ignoring star nodes of band $\lceil \log B \rceil + 1$ is at most $\mathsf{OPT}/8$.

Next we consider bands $\{0, \ldots, \lceil \log B \rceil\}$. We use the following key lemma that upper bounds the probability of reaching star nodes in any particular band i.

Lemma 5.7.4 *For any $i \in \{0, \ldots, \lceil \log B \rceil\}$, the probability of reaching band i is at most $\frac{1}{10\lceil \log B \rceil}$.*

Taking a union bound, the probability of reaching some band $\{0, \ldots, \lceil \log B \rceil\}$ is at most $\frac{1}{10}$. Then we have the following claim (similar to Claim 5.6.8 in the non-adaptive case).

Claim 5.7.5 *Conditional on reaching any node $x \in \mathsf{OPT}'$, the expected reward obtained by the optimal policy from nodes below x is at most OPT.*

Proof: Consider the alternate adaptive policy that visits node x directly from the root. Using triangle inequality, the expected reward from this policy is at least the conditional reward of OPT' obtained below vertex x. The claim now follows by optimality. ■

Thus we obtain that the loss in reward by truncating at star nodes in bands $\{0, \ldots, \lceil \log B \rceil\}$ is at most $\mathsf{OPT}/10$. Combined with the loss due to band $\lceil \log B \rceil + 1$, it follows that OPT'' has reward at least $\mathsf{OPT}/2$.

It only remains to prove Lemma 5.7.4, which we do in the rest of this section.

Proof of Lemma 5.7.4: In order to bound the probability, consider the following altered stochastic process $\mathcal{T}_i$: follow $\mathcal{T}$ as long as it could lead to a star node in band i. If we reach a node y such that there is no band-i star node as a descendant of y, then we stop the process $\mathcal{T}_i$ at y. Else we stop when we reach a star node in band i. An illustration of the optimal decision tree, the different bands and altered processes is given in Figure 5.2.

By a straightforward coupling argument, the probabilities of reaching a band-i star node in $\mathcal{T}$ and in $\mathcal{T}_i$ are identical, and hence it suffices to bound the corresponding probability of continuing beyond a band-i star node in $\mathcal{T}_i$. Therefore, let us fix a particular band i.

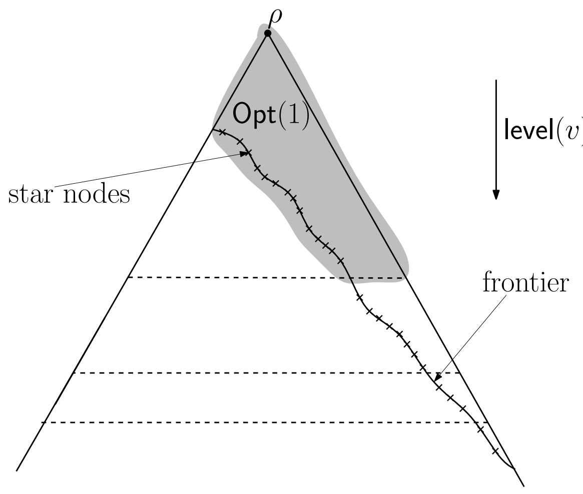


Figure 5.2: Optimal Decision Tree example: dashed lines indicate the bands, $\times$ indicates star nodes

Claim 5.7.6 *For each $i \in \{0, 1, \ldots, \lceil \log B \rceil\}$, and any star node x in band i,*

$$2K\frac{B}{2^i} \quad \leq \quad \sum_{x' \prec x} \mu_{x'}\left(B/2^{i+1}\right) \quad \leq \quad 17K\frac{B}{2^i}$$

Proof: By definition of a star node (5.3), and since node x is in band-i, $B/2^{i+1} \leq B - \mathsf{level}(x) \leq B/2^i$,

$$\begin{aligned}\sum_{x' \prec x} \mu_{x'}\left(B/2^{i+1}\right) \quad &\geq \quad \sum_{x' \prec x} \mu_{x'}\left(\frac{1}{2}(B - \mathsf{level}(x))\right) \quad \geq \quad \frac{1}{2}\sum_{x' \prec x} \mu_{x'}\left(B - \mathsf{level}(x)\right) \\ &\geq \quad 4K(B - \mathsf{level}(x)) \quad \geq \quad 2K\frac{B}{2^i}.\end{aligned}$$

The first two inequalities used the monotonicity and subadditivity of $\mu_{x'}(\cdot)$.

Moreover, since y, the parent node of x, is not a star node, it satisfies $\sum_{x' \prec y} \mu_{x'}\left(B - \mathsf{level}(y)\right) < 8K(B - \mathsf{level}(y)) = 8K(B - \mathsf{level}(x) + 1)$. But since we are not considering band number $\lceil \log B \rceil + 1$ and all distances are at least 1, $\mathsf{level}(x) \leq B - 1$, and hence $B - \mathsf{level}(x) + 1 \leq$

$2(B - \mathsf{level}(x)) \leq 2B/2^i$. Thus we have $\sum_{x' \prec y} \mu_{x'}\left(B - \mathsf{level}(y)\right) < 16K \cdot B/2^i$. Now,

$$\begin{aligned}\sum_{x' \prec x} \mu_{x'}\left(B/2^{i+1}\right) &= \sum_{x' \prec y} \mu_{x'}\left(B/2^{i+1}\right) + \mu_y\left(B/2^{i+1}\right) \quad \leq \quad \sum_{x' \prec y} \mu_{x'}\left(B - \mathsf{level}(y)\right) + \frac{B}{2^{i+1}} \\ &\leq 16K \cdot \frac{B}{2^i} + \frac{B}{2^{i+1}} \quad \leq \quad 17K \cdot \frac{B}{2^i}\end{aligned}$$

The first inequality uses $B - \mathsf{level}(y) \geq B - \mathsf{level}(x) \geq B/2^{i+1}$. This completes the proof. ■

Claim 5.7.7 *For any $i \in \{0, 1, \ldots, \lceil \log B \rceil\}$ and any star node x in band i, if process $\mathcal{T}_i$ reaches x:*

$$\sum_{x' \prec x} \min\left(S_{x'}, \frac{B}{2^{i+1}}\right) \quad \leq \quad \frac{B}{2^i}$$

Proof: Clearly, if process $\mathcal{T}_i$ reaches node x, it must mean that $\sum_{x' \prec x} S_{x'} \leq (B - \mathsf{level}(x)) \leq B/2^i$, else we would have run out of budget earlier. The claim follows because truncation can only decrease the left hand side. ■

We now finish upper bounding the probability of reaching a star node in band i using a Martingale analysis. Define a sequence of random variables $\{Z_t,\ t = 0, 1, \ldots\}$ where

$$Z_t = \sum_{t'=0}^{t} \left(\min\left\{ \mathbf{S}_{t'}, \frac{B}{2^{i+1}} \right\} - \mu_{\mathbf{S}_{t'}}\left(B/2^{i+1}\right) \right). \tag{5.4}$$

Since the subtracted term is precisely the expectation of the first term, the one-term expected change is zero and the sequence $\{Z_t\}$ forms a martingale. In turn, $\mathbb{E}[Z_\tau] = 0$ for any stopping time τ. We will define τ to be the time when the process $\mathcal{T}_i$ ends—recall that this is the first time when (a) either the process reaches a band-i star node, or (b) there is no way to get to a band-i star node in the future.

Claim 5.7.7 says that when $\mathcal{T}_i$ reaches any star node x, the sum over the first terms in (5.4) is at most $B/2^i$, whereas Claim 5.7.6 says the sums of the means is at least $2K\frac{B}{2^i}$. Because $K \geq 1$, we can infer that the $Z_t \leq -K(B/2^i)$ for any star node (at level t). To bound the probability of reaching a star node in $\mathcal{T}_i$, we appeal to Freedman's concentration inequality for martingales, which substantially generalizes the Azuma-Hoeffding inequality.

Theorem 5.7.8 (Freedman [50] (Theorem 1.6)) *Consider a real-valued martingale sequence $\{X_k\}_{k \geq 0}$ such that $X_0 = 0$, and $\mathbb{E}\left[X_{k+1} \mid X_k, X_{k-1}, \ldots, X_0\right] = 0$ for all k. Assume that the sequence is uniformly bounded, i.e., $|X_k| \leq M$ almost surely for all k. Now define the predictable quadratic variation process of the martingale to be*

$$W_k = \textstyle\sum_{j=0}^{k} \mathbb{E}\left[X_k^2 \mid X_{k-1}, X_{k-2}, \ldots X_0\right]$$

for all $k \geq 1$. Then for all $l \geq 0$ and $\sigma^2 > 0$, and any stopping time τ we have

$$\mathbb{P}\left[|\sum_{j=0}^{\tau} X_j| \geq l \text{ and } W_\tau \leq \sigma^2 \right] \quad \leq \quad 2\exp\left(-\frac{l^2/2}{\sigma^2 + Ml/3} \right)$$

We apply the above theorem to the Martingale difference sequence $\{X_t = Z_t - Z_{t-1}\}$. Now, since each term X_t is just $\min(S_t, \frac{B}{2^{i+1}}) - \mu_{S_t}(B/2^{i+1})$, we get that $\mathbb{E}[X_t \mid X_{t-1}, \ldots] = 0$ by definition of $\mu_{S_t}(B/2^{i+1})$. Moreover, since the sizes and means are both truncated at $B/2^{i+1}$, we have $|X_t| \leq B/2^{i+1}$ w.p 1; hence we can set $M = B/2^{i+1}$. Finally in order to bound the variance term W_t we appeal to Claim 5.7.6. Indeed, consider a single r.v $X_t = \min(S_t, \frac{B}{2^{i+1}}) - \mu_{S_t}(B/2^{i+1})$, and suppose we abbreviate $\min(S_t, \frac{B}{2^{i+1}})$ by Y for convenience. Then:

$$\begin{aligned}\mathbb{E}\left[X_t^2 \mid X_{t-1}, \ldots\right] &= \mathbb{E}\left[(Y - \mathbb{E}[Y])^2\right] &= \mathbb{E}\left[Y^2\right] - \mathbb{E}[Y]^2 \\ &\leq Y_{max} \cdot \mathbb{E}[Y] &\leq \frac{B}{2^{i+1}} \cdot \mu_{S_t}\left(B/2^{i+1}\right)\end{aligned}$$

Here, the first inequality uses $Y \geq 0$ and Y_{max} as the maximum value of Y. The last inequality uses the definition of Y. Hence the term W_t is at most $(B/2^{i+1})\sum_{t' \leq t} \mu_{S_{t'}}(B/2^{i+1})$ for the process at time t. Now from Claim 5.7.6 we have that for any star node (say at level t) in band i, we have $\sum_{t' \leq t} \mu_{t'}(B/2^{i+1}) \leq 17K(B/2^i)$. Therefore we have $W_t \leq 9K \cdot (B/2^i)^2$ for star nodes, and we set σ^2 to be this quantity.

So by setting $\ell = K(B/2^i)$, $\sigma^2 = 9K(B/2^i)^2$, and $M = B/2^{i+1}$, we get that

$$\mathbb{P}[\text{ reaching star node in } \mathcal{T}_i] \quad \leq \quad \mathbb{P}\left[|Z_\tau| \geq K(B/2^i) \textit{ and } W_\tau \leq 9K(B/2^i)^2\right] \quad \leq \quad 2\,e^{-K/20}.$$

Setting $K = \Omega(3)$ and performing a simple union bound calculation over the $\lceil \log B \rceil$ bands completes the proof of Lemma 5.7.4. ■

Handling Co-Located Jobs To help with the presentation in the above analysis, we assumed that a node x which is at depth l in the decision tree for OPTis actually processed after the adaptive policy has traveled a distance of l. In particular, this meant that there is at most one stochastic job per node (because if OPTis done with processing some job, it has to travel at least 1 timestep in the decision tree because its depth increases). However, if we are more careful in defining the truncations of any node (in equation (3.8)) by its *actual length along the path*, instead of simply its depth/level in the tree, then we will be able to handle co-located jobs in exactly the same manner as above. In this situation, there could be several nodes in a sample path which have the same truncation threshold but it is not difficult to see that the rest of the analysis would proceed in an identical fashion. We do not present additional details here– the analysis in the next section handles this issue in a much more general problem setting.

5.8 Stochastic Orienteering with Cancellations

Throughout the results in this chapter, we considered the non-preemptive model for processing jobs. In this section, we extend those results to the model where a policy can cancel/abort jobs during processing; however once a job is aborted, it can not be attempted again. As we have seen in an earlier chapter (Section 3.4.1), there are instances that demonstrate an arbitrarily large gap in the expected reward for policies which can cancel and those that can not, even in the special case of Stochastic Knapsack.

We now show that our algorithms for StocOrient can be extended to StocOrient with cancellation, while preserving our approximation guarantee of $O(\log\log B)$. The main idea is to slightly modify the instances we create of our deterministic subroutine, i.e. the KnapOrient problem. Specifically, we create up to B co-located copies of each job v, each of which corresponds to canceling the job v after a certain time t of processing it (the size and reward of copy $\langle v, t\rangle$ are appropriately defined to reflect this, i.e., the size distribution will be truncated at t, and any reward is assumed to be 0 if the size exceeds t, i.e., the truncation threshold is reached). It is easy to see that any adaptive optimal solution, when it visits a vertex in fact just plays *some copy* of it (it might be adaptive as to which copy it plays, i.e., the choice of copy might depend on the history of the sample path taken to reach this vertex). But because it does not play multiple copies of the same vertex, we can find a good deterministic solution with suitably large reward (this is the KnapOrient problem for the uncorrelated case. Now the only issue is when we translate back from the deterministic instance to a non-adaptive solution for the StocOrient instance: the deterministic solution which AlgKO computes might collect reward from multiple copies of the same job.

However, we can bound this gap by using the following geometric scaling idea: when we create the instance of KnapOrient, we also do a pre-processing that, if two copies of a vertex $\langle v, t\rangle$ and $\langle v, t'\rangle$ for $t' \geq t$ have their (modified) rewards within a factor of 2 of each other, then we do not place the copy correspond to $\langle v, t'\rangle$ in our KnapOrient instance. Furthermore, we do this pruning in an ordered left-to-right manner starting with $t = 1$ and ending at $t = B$. Now, our KnapOrient instance is such that for every vertex v, all its copies have rewards which are geometrically separated by at least a factor of 2. Moreover, for every pruned copy $\langle v, t'\rangle$, there is a copy $\langle v, t\rangle$ which (i) has not been pruned out, (ii) has $t \leq t'$, and (iii) has reward at least half the reward of $\langle v, t\rangle$.

We now see how this changes the reduction in both directions: in one way, it is easy to show the optimal solution for KnapOrient does not drop by more than a factor of 2. This is because, if the KnapOrient OPTuses a copy $\langle v, t'\rangle$ in the unpruned instance (and it has been pruned), we can use the copy $\langle v, t\rangle$ which is guaranteed to exist, has $t \leq t'$ and has reward at least half the reward of $\langle v, t'\rangle$. Therefore, we can get a feasible solution for the pruned KnapOrient instance with at least half the reward.

The beauty of this geometric scaling, though, is that it makes recovering a non-adaptive solution from a KnapOrient solution very simple. Indeed, suppose the KnapOrient instance "cheats" by collecting reward from multiple copies of the same vertex. Then, we simply throw away all the copies, with the exception of the one fetching most reward. By definition of our pruning strategy, all copies of a vertex which remain have rewards which are geometrically well-separated, and hence the total reward of all copies that a solution chooses is at most twice the reward of the best single copy it chooses! Therefore, we can apply our reduction in Theorem 5.6.4 and recover a non-adaptive solution for the StocOrient problem.

5.9 Stochastic Orienteering with Correlated Rewards

In this section we consider the StocOrient problem where the reward of each job is a random variable that may be correlated with its processing time (i.e. size). Crucially, the distributions at different vertices are still independent of each other. The input to CorrOrient consists of a metric

(V, d) with root vertex ρ and a bound B. At each vertex $v \in V$, there is a stochastic job with a given probability distribution over (size, reward) pairs: for each $t \in \{0, 1, \dots, B\}$, the job at v has size t and reward $r_{v,t}$ with probability $\pi_{v,t}$. Again we consider the non-preemptive setting, so once a job is started it must be run to completion unless the budget B is exhausted. The goal is to devise a (possibly adaptive) policy that maximizes the expected reward of completed jobs, subject to the total budget (travel time + processing time) being at most B.

When there is no metric in the problem instance, this is precisely the correlated stochastic knapsack problem, and [68] gave a non-adaptive algorithm which is a constant-factor approximation to the optimal adaptive policy; this used an LP-relaxation that is quite different from that in the uncorrelated setting. The trouble with extending that approach to StocOrient is again that we do not know of LP relaxations with good approximation guarantees even for deterministic orienteering. We circumvented this issue for the uncorrelated case by using a Martingale analysis to bypass the need for an LP relaxation, which gave a direct lower bound. We adopt a similar approach for CorrOrient, but as Theorem 5.1.4 says, our approximation ratio is only $O(\log n \log B)$: this is because our algorithm here relies on the "deadline orienteering" problem. Moreover, we show that CorrOrient is at least as hard to approximate as the deadline orienteering problem, for which the best known guarantee is an $O(\log n)$ approximation algorithm [7].

5.9.1 The Non-Adaptive Algorithm for CorrOrient

We now get into describing our approximation algorithm, which proceeds via a reduction to suitably constructed instances of the *deadline orienteering problem*. Recall that an instance of deadline orienteering consists of a metric (denoting travel times) with a reward and deadline at each vertex, and a root vertex. The objective is to compute a path starting from the root that maximizes the reward obtained from vertices that are visited before their deadlines. Our reduction proceeds via taking a *Lagrangian relaxation* of the processing times, and then performing an amortized analysis to argue that the reward is high and the budget is met with constant probability. (In this it is similar to the ideas of [60].) However, new ideas are required to handle correlations between sizes and rewards: indeed, this is where the deadline orienteering problem is needed. Also, we use an interesting counting property (Claim 5.9.1) to avoid double-counting reward.

Notation. Let OPTdenote an optimal decision tree. We classify every execution of a job in this decision tree as belonging to one of $(\log_2 B + 1)$ *types* thus: for $i \in [\log_2 B]$, a *type-i* job execution occurs when the processing time spent *before* running the job lies in the interval $[2^i - 1, 2^{i+1} - 1)$. So if t' is the distance spent before reaching a type-i job, its start time lies in $[t' + 2^i - 1,\ t' + 2^{i+1} - 1)$. Note that the same job might have different types on different sample paths of OPT, but for a fixed sample path down OPT, it can have at most one type. If $\mathsf{OPT}(i)$ is the expected reward obtained from job runs of type i, then we have $\mathsf{OPT} = \sum_i \mathsf{OPT}(i)$, and hence $\max_{i \in [\log_2 B]} \mathsf{OPT}(i) \geq \Omega(\frac{1}{\log B}) \cdot \mathsf{OPT}$. For all $v \in V$ and $t \in [B]$, let $R_{v,t} := \sum_{z=0}^{B-t} r_{v,z} \cdot \pi_{v,z}$ denote the expected reward when job v's size is restricted to being at most $B - t$. Note that this is the expected reward we can get from job v *if it starts at time* t. Recall that for any $v \in V$, S_v denotes its random size which has distribution $\{\pi_{v,t}\}_{t=0}^{B}$

Reducing CorrOrient **to (deterministic)** DeadlineOrient

The high-level idea is the following: for any fixed i, we create an instance of DeadlineOrient to get an $O(\log n)$ fraction of $\mathsf{OPT}(i)$ as reward; then choosing the best such setting of i gives us the $O(\log n \log B)$-approximation algorithm. To obtain the instance of DeadlineOrient, for each vertex v we create several copies of it: for each time t there is a copy corresponding to starting job v at time t (and hence has reward $R_{v,t}$). However, to prevent the DeadlineOrient solution from collecting reward from many different copies of the same vertex, we make copies of vertices *only when the reward changes by a geometric factor*. Indeed, for $t_1 < t_2$, if it holds that $R_{v,t_1} \leq 2R_{v,t_2}$, then we do not include the copy $\langle v, t_1 \rangle$ in our instance. The following claim is useful for defining such a minimal set of starting times for each job.

Claim 5.9.1 *Given any non-increasing function $f : [B] \to \mathbf{R}_+$, we can efficiently find a subset $I \subseteq [B]$:*

$$\frac{f(t)}{2} \leq \max_{\ell \in I : \ell \geq t} f(\ell) \qquad \text{and} \qquad \sum_{\ell \in I : \ell \geq t} f(\ell) \leq 2 \cdot f(t), \qquad \forall t \in [B].$$

Proof: The set I is constructed as follows.

Algorithm 16 Computing the support I in Claim 5.9.1.

```
1: let $i \leftarrow 0$, $k_0 \leftarrow 0$, $I \leftarrow \emptyset$.
2: while $k_i \in [B]$ and $f(k_i) > 0$ do
3:     $\ell_i \leftarrow \max\left\{\ell \in [B] : f(\ell) \geq \frac{f(k_i)}{2}\right\}$.
4:     $k_{i+1} \leftarrow \ell_i + 1$, $I \leftarrow I \bigcup \{\ell_i\}$.
5:     $i \leftarrow i + 1$.
6: output set $I$.
```

Observe that B is always contained in the set I, and hence for any $t \in [B]$, $\min\{\ell \geq t : \ell \in I\}$ is well-defined. To prove the claimed properties let $I = \{\ell_i\}_{i=0}^{p}$. For the first property, given any $t \in [B]$ let $\ell(t) = \min\{\ell \geq t : \ell \in I\}$. Observe that if this set is empty then it must be that $f(t) = 0$ and the claim trivially holds. Now assume that value $\ell(t)$ exists and that $\ell(t) = \ell_i$. By the definition of $\ell(t)$ it must be that $\ell_{i-1} < t$, and so $k_i \leq t$. Hence $f(\ell_i) \geq f(k_i)/2 \geq f(t)/2$; the first inequality is by choice ℓ_i, and the second as f is non-increasing.

We now show the second property. For any index i, we have $\ell_{i-1} < k_i \leq \ell_i < k_{i+1} \leq \ell_{i+1}$. Note that $f(k_{i+1}) = f(\ell_i + 1) < f(k_i)/2$ by choice of ℓ_i. So $f(\ell_{i+1}) \leq f(k_{i+1}) < f(k_i)/2 \leq f(\ell_{i-1})/2$. Given any $t \in [B]$ let q be the index such that $\ell_q = \min\{\ell \geq t : \ell \in I\}$. Consider the sum:

$$\sum_{i \geq q} f(\ell_i) \quad = \quad \sum_{j \geq 0} f(\ell_{q+2j}) + \sum_{j \geq 0} f(\ell_{q+1+2j}) \quad \leq \quad f(\ell_q) + f(\ell_{q+1}) \quad \leq \quad 2 \cdot f(t)$$

The first inequality uses $f(\ell_{i+1}) < f(\ell_{i-1})/2$ and a geometric summation; the last is by $t \leq \ell_q < \ell_{q+1}$. This completes the proof. ■

Consider any $i \in [\log_2 B]$. Now for each $v \in V$, apply Claim 5.9.1 to the function $f(t) := R_{v,t+2^i-1}$ to obtain a subset $I_v^i \subseteq [B]$. These subsets define the copies of each job that we will use.

For each i and parameter $\lambda \geq 0$ we define a deadline orienteering instance:

Definition 5.9.2 (DeadlineOrient **Instance** $\mathcal{I}_i(\lambda)$**)** *The metric is (V, d) with root vertex ρ. For each $v \in V$ and $\ell \in I_v^i$ there is a job $\langle v, \ell \rangle$ located at vertex v with deadline ℓ and reward $\hat{r}_i(v, \ell, \lambda) := R_{v,\ell+2^i-1} - \lambda \cdot \mathbb{E}[\min(S_v, 2^i)]$. The objective in $\mathcal{I}_i(\lambda)$ is to find a path originating at ρ that maximizes the reward of the jobs visited within their deadlines.*

Also define $N_i = \{\langle v, \ell \rangle : \ell \in I_v^i, v \in V\}$; for each job $\langle v, \ell \rangle \in N_i$, its size $s_i(\langle v, \ell \rangle) = s_i(v) := \mathbb{E}[\min(S_v, 2^i)]$, and $r_i(\langle v, \ell \rangle) = R_{v,\ell+2^i-1}$.

The co-located jobs $\{\langle v, \ell \rangle : \ell \in I_v^i\}$ in $\mathcal{I}_i(\lambda)$ are copies of job v in the original CorrOrient instance, where copy $\langle v, \ell \rangle$ corresponds to running v as a type-i job after distance ℓ. Also, λ should be thought of as a Lagrangean multiplier, and $\mathcal{I}_i(\lambda)$ is really a Lagrangean relaxation of a DeadlineOrient instance with an additional side constraint that the size is at most 2^i. It is immediate by the definition of rewards that $\mathsf{OPT}(\mathcal{I}_i(\lambda))$ is a non-increasing function of λ.

The idea of our algorithm is to argue that for the "right" setting of λ, the optimal DeadlineOrient solution for $I_i(\lambda)$ has value $\Omega(\mathsf{OPT}(i))$, which is shown in Lemma 5.9.4. Moreover, as shown in Theorem 5.9.8, we can recover a valid solution to CorrOrient from an approximate solution to $I_i(\lambda)$.

Lemma 5.9.3 *For any $i \in [\log B]$ and $\lambda > 0$, the optimal value of the* DeadlineOrient *instance $\mathcal{I}_i(\lambda)$ is at least $\mathsf{OPT}(i)/2 - \lambda\, 2^{i+1}$.*

Proof: Consider the optimal decision tree OPTof the CorrOrient instance, and label every node in OPTby a (dist, size) pair, where dist is the total time spent traveling and size the total time spent processing jobs *before* visiting that node. Note that both dist and size are non-decreasing as we move down OPT. Also, type-i nodes are those where $2^i - 1 \leq \mathsf{size} < 2^{i+1} - 1$, so zeroing out rewards from all but type-i nodes yields expected reward $\mathsf{OPT}(i)$.

For any vertex $v \in V$, let X_v^i denote the indicator r.v. that job v is run as type-i in OPT, and S_v be the r.v. denoting its instantiated size—note that these are independent. Also let $Y^i = \sum_{v \in V} X_v^i \cdot \min(S_v, 2^i)$ be the random variable denoting the sum of truncated sizes of type-i jobs. By definition of type-i, we have $Y^i \leq 2 \cdot 2^i$ with probability one, and hence $\mathbb{E}[Y^i] \leq 2^{i+1}$. For ease of notation let $q_v = \mathbb{P}[X_v^i = 1]$ for all $v \in V$. We now have,

$$\begin{aligned} 2^{i+1} \;\geq\; \mathbb{E}[Y^i] &= \sum_{v \in V} q_v \cdot \mathbb{E}[\min(S_v, 2^i) \mid X_v^i = 1] \\ &= \sum_{v \in V} q_v \cdot \mathbb{E}[\min(S_v, 2^i)] \;=\; \sum_{v \in V} q_v \cdot s_i(v) \end{aligned} \tag{5.5}$$

Now consider the expected reward $\mathsf{OPT}(i)$. We can write:

$$\mathsf{OPT}(i) = \sum_{v \in V} \sum_{\ell \in [B]} \sum_{k=2^i-1}^{2^{i+1}-2} \mathbb{P}[\mathbf{1}_{v,\mathsf{dist}=l,\mathsf{size}=k}] \cdot R_{v,\ell+k} \quad \leq \quad \sum_{v \in V} \sum_{\ell \in [B]} \mathbb{P}[\mathbf{1}_{v,\mathsf{type}=i,\mathsf{dist}=\ell}] \cdot R_{v,\ell+2^i-1} \tag{5.6}$$

where $\mathbf{1}_{v,\mathsf{dist}=l,\mathsf{size}=k}$ is the indicator for whether OPT visits v with $\mathsf{dist} = l$ and $\mathsf{size} = k$, and $\mathbf{1}_{v,\mathsf{type}=i,\mathsf{dist}=\ell}$ is the indicator for whether OPT visits v as type-i with $\mathsf{dist} = \ell$. Now going back to the metric, let $\mathcal{P}$ denote the set of all possible rooted paths traced by $\mathsf{OPT}(i)$ in the metric (V, d). Now for each path $P \in \mathcal{P}$, define the following quantities.

1. $\beta(P)$ is the probability that $\mathsf{OPT}(i)$ traces P.
2. For each vertex $v \in P$, $d_v(P)$ is the *travel time* (i.e. dist) incurred in P prior to reaching v. Note that the actual time at which v is visited is $\mathsf{dist} + \mathsf{size}$, which is in general larger than $d_v(P)$.
3. $w_\lambda(P) = \sum_{v\in P} \left[\frac{1}{2} \cdot R_{v,d_v(P)+2^i-1} - \lambda \cdot s_i(v)\right]$.

Moreover, for each $v \in P$, let $\ell_v(P) = \min\{\ell \in I_v^i \mid \ell \geq d_v(P)\}$; recall the definition I_v^i using Claim 5.9.1, and since $B \in I_v^i$, the quantity $\ell_v(P)$ is well-defined.

For any path $P \in \mathcal{P}$, consider P as a solution to $\mathcal{I}_i(\lambda)$ that visits the copies $\{\langle v, \ell_v(P)\rangle : v \in P\}$ within their deadlines. It is feasible for the instance $\mathcal{I}_i(\lambda)$ because for each vertex in P, we defined $\ell_v(P) \geq d_v(P)$ and therefore we would reach the chosen copy within its deadline. Moreover, the objective value of P is precisely

$$\sum_{v\in P} \hat{r}_i(v, \ell_v(P), \lambda) \;=\; \sum_{v\in P} \left[R_{v,\ell_v(P)+2^i-1} - \lambda \cdot s_i(v)\right] \;\geq\; \sum_{v\in P} \left[\frac{1}{2} \cdot R_{v,d_v(P)+2^i-1} - \lambda \cdot s_i(v)\right] \;=\; w_\lambda(P)$$

where the inequality above uses the definition of $\ell_v(P)$ and Claim 5.9.1. Now,

$$\begin{aligned}\mathsf{OPT}(\mathcal{I}_i(\lambda)) \;&\geq\; \max_{P\in\mathcal{P}} w_\lambda(P) \;\geq\; \sum_{P\in\mathcal{P}} \beta(P) \cdot w_\lambda(P) \;=\; \sum_{P\in\mathcal{P}} \beta(P) \cdot \sum_{v\in P} \left[\frac{1}{2} \cdot R_{v,d_v(P)+2^i-1} - \lambda \cdot s_i(v)\right] \\ &\geq\; \frac{1}{2} \sum_{v\in V} \sum_{\ell\in[B]} \mathbb{P}[\mathbf{1}_{v,\mathsf{type}=i,\mathsf{dist}=l}] \cdot R_{v,\ell+2^i-1} \;-\; \lambda \cdot \sum_{v\in V} \mathbb{P}[X_v^i] \cdot s_i(v) \;\geq\; \frac{\mathsf{OPT}(i)}{2} \;-\; \lambda \cdot 2^{i+1}\end{aligned}$$

Above the second-to-last inequality is by interchanging summations and splitting the two terms from the previous expression, the first term in the final inequality comes from (5.6), and the second term comes from (5.5) and the fact that $q_v = \mathbb{P}[X_v^i] = \mathbb{P}[v \text{ visited as type-}i]$. ■

Now let AlgDO denote an α-approximation algorithm for the $\mathsf{DeadlineOrient}$ problem. (We abuse notation and use $\mathsf{AlgDO}(\mathcal{I}_i(\lambda))$ to denote both the α-approximate solution on instance $\mathcal{I}_i(\lambda)$ as well as its value.) We focus on the "right" setting of λ defined thus:

$$\lambda_i^* \quad := \quad \max\left\{\lambda \;:\; \mathsf{AlgDO}(\mathcal{I}_i(\lambda)) \geq \frac{2^i \cdot \lambda}{\alpha}\right\} \tag{5.7}$$

Lemma 5.9.4 *For any $i \in [\log_2 B]$, we get $\lambda_i^* \geq \mathsf{OPT}(i)/2^{i+3}$, and hence $\mathsf{AlgDO}(\mathcal{I}_i(\lambda_i^*)) \geq \mathsf{OPT}(i)/8\alpha$.*

Proof: Consider the setting $\widehat{\lambda} = \mathsf{OPT}(i)/2^{i+3}$; by Lemma 5.9.3, the optimal solution to the $\mathsf{DeadlineOrient}$ instance $\mathcal{I}_i(\widehat{\lambda})$ has value at least $\mathsf{OPT}(i)/4 \geq 2^i \cdot \widehat{\lambda}$. Since AlgDO is an α-approximation algorithm for $\mathsf{DeadlineOrient}$, it follows that $\mathsf{AlgDO}(\mathcal{I}_i(\widehat{\lambda})) \geq \mathsf{OPT}(\mathcal{I}_i(\widehat{\lambda}))/\alpha \geq 2^i \cdot \widehat{\lambda}/\alpha$. So $\lambda_i^* \geq \widehat{\lambda} \geq \mathsf{OPT}(i)/2^{i+3}$. ■

Obtaining CorrOrient **solution from** AlgDO(λ_i^*)

It just remains to show that the solution output by the approximation algorithm for DeadlineOrient on the instance $\mathcal{I}_i(\lambda_i^*)$ yields a good non-adaptive solution to the original CorrOrient instance. Let $\sigma = \mathsf{AlgDO}(\lambda_i^*)$ be this solution—hence σ is a rooted path that visits some set $P \subseteq N_i$ of nodes within their respective deadlines. The algorithm below gives a subset $Q \subseteq P$ of nodes that we will visit in the non-adaptive solution; this is similar to the algorithm for KnapOrient in Section 5.5.

Algorithm 17 Algorithm A_i for CorrOrient given a solution for $\mathcal{I}_i(\lambda_i^*)$ characterized by a path P.

```
1: let $y = \left(\sum_{v \in P} s_i(v)\right)/2^i$.
2: partition the vertices of $P$ into $c = \max(1, \lfloor 2y \rfloor)$ parts $P_1, \ldots, P_c$ s.t $\sum_{v \in P_j} s_i(v) \le 2^i$ for
   all $1 \le j \le c$.
3: set $Q \leftarrow P_k$ where $k = \arg\max_{j=1}^{c} \sum_{\langle v,\ell \rangle \in P_j} r_i(v, \ell)$.
4: for each $v \in V$, define $d_v := \min\{\ell : \langle v, \ell \rangle \in Q\}$
5: let $\overline{Q} := \{v \in V : d_v < \infty\}$ be the vertices with at least one copy in $Q$.
6: sample each vertex in $\overline{Q}$ independently w.p. $1/2$, and visit these sampled vertices in order
   given by $P$.
```

At a high level, the algorithm partitions the vertices in σ into groups, where each group obeys the size budget of 2^i in expectation. It then picks the most profitable group of them. The main issue with Q chosen in Step 3 is that it may include multiple copies of the same vertex. But because of the way we constructed the sets I_i^v (based on Claim 5.9.1), we can simply pick the copy which corresponds to the earliest deadline, and by discarding all the other copies, we only lose out on a constant fraction of the reward $r_i(Q)$. Our first claim bounds the total (potential) reward of the set Q we select in Step 3.

Claim 5.9.5 *The sum* $r_i(Q) = \sum_{\langle v,\ell \rangle \in Q} r_i(v, \ell)$ *is at least* $\mathsf{OPT}(i)/(8\alpha)$.

Proof: Consider the following chain of inequalities:

$$\begin{aligned} r_i(Q) \;&\ge\; \frac{r_i(P)}{c} \;\ge\; \frac{\lambda_i^*\, y 2^i + \lambda_i^*\, 2^i/\alpha}{c} \;=\; \lambda_i^*\, 2^i \cdot \left(\frac{y + 1/\alpha}{c}\right) \\ &\ge\; \lambda_i^*\, 2^i \cdot \min\left\{ y + \frac{1}{\alpha}, \frac{1}{2} + \frac{1}{2\alpha y} \right\} \;\ge\; \frac{\lambda_i^*\, 2^i}{\alpha} \end{aligned}$$

The second inequality is due the the fact that $2^i\, \lambda_i^*/\alpha \le \mathsf{AlgDO}(\lambda_i^*) = \hat{r}_i(P) = r_i(P) - \lambda_i^* \cdot s_i(P)$ due to the choice of λ_i^* in equation (5.7); the third inequality is by $c \le \max\{1, 2y\}$; and the last inequality uses $\alpha \ge 2$. To conclude we simply use Lemma 5.9.4. ■

Next, we show that we do not lose much of the total reward by discarding duplicate copies of vertices in Q.

Claim 5.9.6 $\sum_{v \in \overline{Q}} R_{v, d_v + 2^i - 1} \ge \mathsf{OPT}(i) \,/\, 16\alpha$.

Proof: For each $u \in \overline{Q}$, let $Q_u = Q \bigcap \{\langle u, \ell \rangle\}_{\ell \in [I^i_u]}$ denote all copies of u in Q. Now by the definition of d_u we have $\ell \geq d_u$ for all $\langle u, \ell \rangle \in Q_u$. So for any $u \in \overline{Q}$,

$$\sum_{\langle u,\ell \rangle \in Q_u} R_{u,\ell+2^i-1} \quad \leq \quad \sum_{\ell \in I^i_u : \ell \geq d_u} R_{u,\ell+2^i-1} \quad \leq \quad 2 \cdot R_{u,d_u+2^i-1}$$

Above, the last inequality uses the definition of I^i_u as given by Claim 5.9.1. Adding over all $u \in \overline{Q}$,

$$\sum_{u \in \overline{Q}} R_{u,d_u+2^i-1} \quad \geq \quad \frac{1}{2} \sum_{u \in \overline{Q}} \sum_{\langle u,\ell \rangle \in Q_u} R_{u,\ell+2^i-1} \quad = \quad \frac{1}{2} \sum_{\langle v,\ell \rangle \in Q} r_i(\langle v, \ell \rangle) \quad \geq \quad \frac{\mathsf{OPT}(i)}{16\alpha}$$

Here, the last inequality above uses Claim 5.9.5. This completes the proof. ■

We now argue that the algorithm A_i reaches any vertex $v \in \overline{Q}$ before the time $d_v + 2^i - 1$ with constant probability.

Claim 5.9.7 *For any vertex $v \in \overline{Q}$, it holds that $\mathbb{P}[A_i$ reaches job v by time $d_v + 2^i - 1] \geq \frac{1}{2}$.*

Proof: We know that because P is a feasible solution for the DeadlineOrient instance, the distance traveled before reaching the copy $\langle v, d_v \rangle$ is at most d_v. Therefore in what remains, we show that with probability $1/2$, the total size of previous vertices is at most $2^i - 1$. To this end, let $\overline{Q}'$ denote the set of vertices sampled in Step 6. We then say that the *bad event* occurs if $\sum_{u \in \overline{Q}' \setminus v} \min(S_u, 2^i) \geq 2^i$. Indeed if $\sum_{u \in \overline{Q}' \setminus v} \min(S_u, 2^i) < 2^i$, then certainly we will reach v by time $d_v + 2^i - 1$.

We now bound the probability of the bad event. For this purpose, observe that

$$\mathbb{E}\left[\sum_{u \in \overline{Q}' \setminus v} \min(S_u, 2^i)\right] \quad \leq \quad \frac{1}{2} \sum_{u \in \overline{Q}} \mathbb{E}\left[\min(S_u, 2^i)\right] \ ,$$

by linearity of expectation and the fact that each $u \in \overline{Q}$ is sampled into $\overline{Q}'$ with probability $1/2$. Now the latter sum is at most $(1/2) \sum_{u \in \overline{Q}} s_i(u) \leq 2^{i-1}$, by the partitioning in Step 2. Hence, the probability of the bad event is at most $1/2$ by Markov's inequality. ■

Theorem 5.9.8 *The expected reward of the algorithm A_i is at least $\mathsf{OPT}(i)/64\alpha$.*

Proof: We know from the above claim that for each vertex $v \in \overline{Q}$, A_i reaches v by time $d_v + 2^i - 1$ with probability at least $1/2$. In this event, we sample v with probability $1/2$. Therefore, the expected reward collected from v is at least $(1/4) R_{v,d_v+2^i-1}$. The proof is complete by using linearity of expectation and then Claim 5.9.6. ■

5.9.2 Evidence of hardness for CorrOrient

Our approximation algorithm for CorrOrient can be viewed as a reduction to DeadlineOrient, at the loss of an $O(\log B)$ factor. We now provide a reduction in the reverse direction: namely, a β-approximation algorithm for CorrOrient implies a $(\beta - o(1))$-approximation algorithm for

DeadlineOrient. In particular this means that a sub-logarithmic approximation ratio for CorrOrient would also improve the best known approximation ratio for DeadlineOrient.

Consider any instance $\mathcal{I}$ of DeadlineOrient on metric (V, d) with root $\rho \in V$ and deadlines $\{t_v\}_{v \in V}$; the goal is to compute a path originating at ρ that visits the maximum number of vertices before their deadlines. We now define an instance $\mathcal{J}$ of CorrOrient on the same metric (V, d) with root ρ. Fix parameter $0 < p \ll \frac{1}{n^2}$. The job at each $v \in V$ has the following distribution: size $B - t_v$ (and reward $1/p$) with probability p, and size zero (and reward 0) with probability $1 - p$. To complete the reduction from DeadlineOrient to CorrOrient we will show that:

$$(1 - o(1)) \cdot \mathsf{OPT}(\mathcal{I}) \quad \le \quad \mathsf{OPT}(\mathcal{J}) \quad \le \quad (1 + o(1)) \cdot \mathsf{OPT}(\mathcal{I}).$$

Let τ be any solution to $\mathcal{I}$ that visits subset $S \subseteq V$ of vertices within their deadline; so the objective value of τ is $|S|$. This also corresponds to a (non-adaptive) solution to $\mathcal{J}$. For any vertex $v \in S$, the probability that zero processing time has been spent prior to v is at least $(1 - p)^n$; in this case, the start time of job v is at most t_v (recall that τ visits $v \in S$ by time t_v) and hence the conditional expected reward from v is $p \cdot \frac{1}{p} = 1$ (since v has size $B - t_v$ and reward $1/p$ with probability p). It follows that the expected reward of τ as a solution to $\mathcal{J}$ is at least $\sum_{v \in S} (1 - p)^n \ge |S| \cdot (1 - np) = (1 - o(1)) \cdot |S|$. Choosing τ to be the optimal solution to $\mathcal{I}$, we have $(1 - o(1)) \cdot \mathsf{OPT}(\mathcal{I}) \le \mathsf{OPT}(\mathcal{J})$.

Consider now any adaptive policy σ for $\mathcal{J}$, with expected reward $R(\sigma)$. Define path σ_0 as one starting from the root of σ that always follows the branch corresponding to size zero instantiation. Consider σ_0 as a feasible solution to $\mathcal{I}$; let $S_0 \subseteq V$ denote the vertices on path σ_0 that are visited prior to their respective deadlines. Clearly $\mathsf{OPT}(\mathcal{I}) \ge |S_0|$. When policy σ is run, every size zero instantiation gives zero reward; so if positive reward is obtained, the sample path must diverge from σ_0. Moreover, if there is positive reward, the sample path must have positive size instantiation at some vertex in S_0: this is because a positive size instantiation at any $(V \setminus S_0)$-vertex violates the bound B (by definition of sizes and set S_0). Hence,

$$\mathbb{P}\left[\sigma \text{ gets positive reward}\right] \quad \le \quad p \cdot |S_0| \tag{5.8}$$

Moreover, since the reward is always an integral multiple of $1/p$,

$$\begin{aligned} R(\sigma) \quad &= \quad \frac{1}{p} \cdot \sum_{i=1}^{n} \mathbb{P}\left[\sigma \text{ gets reward at least } i/p\right] \\ &= \quad \frac{1}{p} \cdot \mathbb{P}\left[\sigma \text{ gets positive reward}\right] \; + \; \frac{1}{p} \cdot \sum_{i=2}^{n} \mathbb{P}\left[\sigma \text{ gets reward at least } i/p\right] \end{aligned} \tag{5.9}$$

Furthermore, for any $i \ge 2$, we have:

$$\mathbb{P}\left[\sigma \text{ gets reward at least } i/p\right] \le \mathbb{P}\left[\text{at least } i \text{ jobs instantiate to positive size}\right] \le \binom{n}{i} \cdot p^i \le (np)^i.$$

It follows that the second term in (5.9) can be upper bounded by $\frac{1}{p} \cdot \sum_{i=2}^{n} (np)^i \le 2n^2 p$ since $np < \frac{1}{2}$. Combining this with (5.8) and (5.9), we obtain that $R(\sigma) \le |S_0| + 2n^2 p = |S_0| + o(1)$ since $n^2 p \ll 1$. Since this holds for any adaptive policy σ for $\mathcal{J}$, we get $\mathsf{OPT}(\mathcal{I}) \ge (1 - o(1)) \cdot \mathsf{OPT}(\mathcal{J})$.

5.10 Conclusions

In this chapter, we presented constant-factor approximation algorithms for the non-adaptive version of the StocOrient problem, and also an $O(\log\log B)$-factor adaptivity gap for the adaptive version. We believe that the $O(\log\log B)$ loss in the adaptivity gap is purely an artifact of our analysis and that the gap is indeed a constant factor. Following the results of this thesis, we also considered several variants of this problem such as the *sequence orienteering problem*, where the algorithm *must visit* a pre-specified set of vertices (say depots) in a fixed order. These results appear in [71].

Chapter 6

Conclusion

The main focus of this thesis was on the approximability of combinatorial optimization problems from a stochastic setting, i.e., the algorithm gets to know exact input only over several stages, but the input is chosen from a known distribution (unlike online algorithms where the input is adversarially chosen). Mor specifically, we studied the following problems: (a) the *Two-stage Stochastic Survivable Network Design* problem where the collection of (source,sink) pairs is drawn randomly from a known distribution, (b) the *Stochastic Knapsack* problem with random (and potentially correlated) sizes/rewards for jobs, (c) the *Multi-Armed Bandits* problem, where the individual Markov Chains make random transitions, and finally (d) the *Stochastic Orienteering* problem, where the random tasks/jobs are located at different vertices on a metric. We presented different fairly general techniques (and consequently, algorithms) for solving these problems with near-optimal approximation guarantees.

We now present some of the main open problems which arise from the works in this thesis. Please also refer to the conclusion of each chapter for some specific open questions relating to the contents of the chapter.

6.1 Stochastic Knapsack with Correlated Jobs

In this section, we consider the StocK problem, but where the sizes may be correlated *across jobs*. Recall that all the results of this thesis work only for the distributions of each job being independent of the others. Indeed, for a motivating example of such a correlated setting, consider the following Map-Reduce example.

Problem Definition: Two-Wise Correlated Knapsack

There are a collection of n pairs of task at the server, which also has a time budget of B. Each pair $(i, m(i))$ of tasks comes with a *joint distribution* over sizes and rewards. Above, $m(i)$ is the mate job of i. The goal, as always, is to schedule these jobs adaptively so as to maximize the expected total reward of jobs completed within the budget of B.

The reader can think of one of the jobs of a given pair as the map task and its corresponding mate as the reduce task. Notice that the running time of a map task can be highly correlated with that of the corresponding reduce task, since they essentially operate on the same input data[1].

From a technical point, the problem becomes more challenging than the basic StocK problem because of the presence of a large adaptivity gap. Indeed, the following example will illustrate this phenomenon.

Example 6.1.1 *There are n pairs of jobs indexed by $\{(s_i, l_i)\, 1 \leq i \leq n\}$. A job pair (s_i, l_i) has the following size distribution: with ptobability $1/n$, s_i has size 1 and l_i has size n. With probability $1 - 1/n$, s_i has size 2 and l_i has size $3n$. The budget of the knapsack is $3n$. All small jobs s_i have 0 reward while the large jobs l_i have unit reward.*

It is easy to construct an adaptive solution with expected $\Omega(1)$ reward — try all small jobs, and if any of them completed at size 1, then run the corresponding large job. The total space occupied by all the small jobs is at most $2n$, and with probability at least $1-(1-1/n)^n \geq 1-1/e$, at least one of the small jobs will complete at size 1: in this case, the corresponding large has size n and will fit inside the knapsack and fetch us unit reward. However, since a non-adaptive strategy must specify an ordering of jobs up front, it does not have the luxury of deciding which large job to try and extract reward from. Therefore, it can effectively try at most three large job, and will fetch expected reward $O(1/n)$.

In a more general problem, we can have polynomially many scenarios of job sizes: each scenario specifies a particular size of a job, and the goal is to maximize the total expected reward of jobs which fit into the knapsack. In this problem, even though there are polynomially many scenarios, the algorithm faces the daunting exploration-exploitation task: Indeed, the algorithm must decide between running the jobs whose size it knows but which provide little information about the actual scenario that has arisen, and running those jobs which help "isolating" the scenario but don't provide as much reward themselves.

6.2 SNDP with Vertex Connectivity Requirements

While we have a very good understanding of the edge-connectivity versions of the SNDP problem, the vertex-connectivity versions still pose many unresolved challenges. For example, all known vertex-connectivity SNDP algorithms (with any non-trivial guarantees) are highly global in their function: this immediately means that we don't know how to make these algorithms handle input uncertainty. It is an interesting challenge to design more "local" algorithms that are amenable to ones which can handle input uncertainty. In particular, the following is a concrete open problem.

Problem Definition: Online 2-VCND

We are given a graph with non-negative edge-costs, and a specified root vertex r. A sequence of demand vertices arrives online with vertex v_i arriving on the i^{th} day. The goal is to buy a set of edges E_i on the i^{th} day such that $\cup_{j=1}^{i} E_j$ forms a feasible solution in that v_i is 2-vertex-connected to the root r. The performance of the online algorithm is, as always, measured by its competitive ratio.

[1]While it is the case that in actual Map-Reduce systems, execution of the map task must preceed that of the reduce task, we omit this requirement for simplicity of presentation

The main reason why our techniques from Chapter 2 do not extend to the case of vertex-connectivity is that edge-connectivity is transitive, i.e., if u is 2-edge-connected to v and v is 2-edge-connected to w, then u is 2-edge-connected to w as well. Our algorithm heavily relies on this feature of edge-connectivity.

6.3 Stochastic Routing Problems

In the stochastic network design problems often studied in literature (see Chapter refchap:sndp for references), edges effectively have infinite capacity. That is, once an edge is bought, it can provide connectivity to all the demands which use it. An interesting future direction to study is one where edges also have capacities. Indeed, the following is a question which formalizes this notion in one particular manner.

Problem Definition: Two-Stage Stochastic Routing

In this problem, we are given a graph $G = (V, E)$ and a distribution over demand vertices and their desired connectivity (i.e., flow) requirement to a pre-specified root vertex r. Now, in the first stage, the algorithm can reserve some capacity on each edge $e \in E$, and it incurs a concave cost depending $f_e(c)$ for reserving c units of capacity on edge e. Then in the second stage, the actual demands are realized, and the algorithm can augment capacity on the edges so that the resultant edge capacities can support the realized demand values. However, the augmentation can only be done at an inflated price — i.e., if the capacity on edge e is augmented from c to c', the algorithm incurs a cost of $\sigma(f_e(c') - f_e(c))$. The goal is then to devise a strategy which minimizes the total expected cost.

Bibliography

[1] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *48th FOCS*, pages 781–790, 2008. 2.4.3, 2.4.3

[2] Micah Adler and Brent Heeringa. Approximating Optimal Binary Decision Trees. In *APPROX*, pages 1–9, 2008. 5.1.1

[3] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995. ISSN 0097-5397. (Preliminary version in *23rd STOC*, 1991). 2.2, 2.7

[4] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Seffi Naor. The online set cover problem. In *35th STOC*, pages 100–105, 2003. 2.2, 2.4.4, 2.4.7

[5] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Seffi Naor. A general approach to online network optimization problems. In *15th SODA*, pages 577–586, 2004. ISBN 0-89871-558-X. 2.2, 2.4.2

[6] Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized Steiner problem. *Theoret. Comput. Sci.*, 324(2-3):313–324, 2004. ISSN 0304-3975. 2.2

[7] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *STOC*, pages 166–174. 5.1, 5.1.1, 5.9

[8] Nikhil Bansal, Don Coppersmith, and Maxim Sviridenko. Improved approximation algorithms for broadcast scheduling. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 344–353, New York, 2006. ACM. 1.2

[9] Nikhil Bansal, Rohit Khandekar, and Viswanath Nagarajan. Additive guarantees for degree bounded directed network design. In *40th STOC*, pages 769–778, 2008. 2.2

[10] Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra. When lp is the cure for your matching woes: Improved bounds for stochastic matchings. In *ESA (2)*, pages 218–229, 2010. 1.5.3, 5.1.1

[11] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th FOCS*, pages 184–193, 1996. 2.4.3

[12] E. M. L. Beale. On minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society. Series B (Methodological)*, 17(2):pp. 173–184, 1955. ISSN 00359246. URL `http://www.jstor.org/stable/2983952`. 1.4.3

[13] Piotr Berman and Chris Coulston. On-line algorithms for steiner tree problems (extended

abstract). In *29th STOC*, pages 344–353, 1997. 2.2

[14] Dimitri P. Bertsekas. *Dynamic programming and optimal control.* Athena Scientific, Belmont, MA, third edition, 2005. ISBN 1-886529-26-4. 4.1

[15] Dimitris Bertsimas and Adam J. Mersereau. A learning approach for interactive marketing to a customer segment. *Operations Research*, 55(6):1120–1135, 2007. 4.1

[16] Dimitris Bertsimas and Jose Nino-Mora. Conservation laws, extended polymatroids and multiarmed bandit problems; a polyhedral approach to indexable systems. *Mathematics of Operations Research*, 21(2):257–306, 1996. 1.5.3, 5.1.1

[17] Anand Bhalgat. A $2+\varepsilon$-approximation algorithm for the stochastic knapsack problem. In *Manuscript*, 2011. 3.2, 3.9

[18] Anand Bhalgat. A $(2+\varepsilon)$-approximation algorithm for the stochastic knapsack problem. At `http://www.seas.upenn.edu/~bhalgat/2-approx-stochastic.pdf`, 2011. 3, 5, 5.1.1

[19] Anand Bhalgat, Ashish Goel, and Sanjeev Khanna. Improved approximation results for stochastic knapsack problems. In *SODA*, 2011. 1.6.2, 3, 3.2, 5.1.1

[20] John R. Birge and François Louveaux. *Introduction to stochastic programming.* Springer Series in Operations Research. Springer-Verlag, 1997. ISBN 0-387-98217-5. 1.4.1, 1.4.3, 3.2

[21] Avrim Blum, Shuchi Chawla, David R. Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM J. Comput.*, 37(2):653–670, 2007. 5, 5.1.1

[22] Niv Buchbinder and Joseph (Seffi) Naor. Improved bounds for online routing and packing via a primal-dual approach. In *46th FOCS*, pages 293–304, 2006. 2.2

[23] Ann Melissa Campbell, Michel Gendreau, and Barrett W. Thomas. The orienteering problem with stochastic travel and service times. *Annals OR*, 186(1):61–81, 2011. 5.1.1

[24] Deepayan Chakrabarti, Ravi Kumar, Filip Radlinski, and Eli Upfal. Mortal multi-armed bandits. In *NIPS*, pages 273–280, 2008. 4.1

[25] Tanmoy Chakraborty, Julia Chuzhoy, and Sanjeev Khanna. Network design for vertex connectivity. In *40th STOC*, pages 167–176, 2008. 2.2

[26] Yuk Hei Chan, Wai Shing Fung, Lap Chi Lau, and Chun Kong Yung. Degree bounded network design with metric costs. In *48th FOCS*, 2008. 2.2

[27] Shuchi Chawla and Tim Roughgarden. Single-source stochastic routing. In *Proceedings of APPROX*, pages 82–94. 2006. 3.2

[28] Chandra Chekuri and Nitish Korula. Single-sink network design with vertex connectivity requirements. In *FST&TCS*, 2008. 2.2

[29] Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *APPROX-RANDOM*, pages 72–83, 2004. 5.1.1

[30] Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *FOCS*, pages 245–253, 2005. 5.1.1

[31] Chandra Chekuri, Nitish Korula, and Martin Pál. Improved algorithms for orienteering and related problems. In *SODA*, pages 661–670. 5, 5.1.1, 5.5

[32] Chandra Chekuri, Anupam Gupta, Nitish Korula, Ravishankar Krishnaswamy, and Amit Kumar. Cost-sharing for subgraph k-edge-connectivity. unpublished, July 2008. 2.2

[33] Ke Chen and Sariel Har-Peled. The Euclidean orienteering problem revisited. *SIAM J. Comput.*, 38(1):385–397, 2008. ISSN 0097-5397. doi: 10.1137/060667839. URL `http://dx.doi.org/10.1137/060667839`. 5.1.1

[34] Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *ICALP (1)*. 5.1.1

[35] Joseph Cheriyan and Adrian Vetta. Approximation algorithms for network design with metric costs. In *37th STOC*, pages 167–175, 2005. 2.2, 2.6, 2.7

[36] Joseph Cheriyan, Santosh Vempala, and Adrian Vetta. An approximation algorithm for the minimum-cost k-vertex connected subgraph. *SIAM J. Comput.*, 32(4):1050–1055, 2003. 2.2

[37] Julia Chuzhoy and Sanjeev Khanna. Algorithms for single-source vertex connectivity. In *48th FOCS*, pages 105–114, 2008. 2.2

[38] Jr. Coffman, E. G., L. Flatto, M. R. Garey, and R. R. Weber. Minimizing expected makespans on uniform processor systems. *Adv. Appl. Prob.*, 19(1):pp. 177–201, 1987. ISSN 00018678. 3.2

[39] George B. Dantzig. Linear programming under uncertainty. *Manage. Sci.*, 50: 1764–1769, December 2004. ISSN 0025-1909. doi: 10.1287/mnsc.1040.0261. URL `http://dl.acm.org/citation.cfm?id=1245920.1245922`. 1.4.3

[40] Brian C. Dean. *Approximation Algorithms for Stochastic Scheduling Problems.* PhD thesis, MIT, 2005. 3.2, 3.4, 3.4.1

[41] Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *SODA*, pages 395–404, 2005. 1.6.2, 3, 3.2, 3.5, 3.5.2

[42] Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Math. Oper. Res.*, 33(4):945–964, 2008. 1.5.3, 3.2, 3.4, 3.4.1, 5, 5.1.1, 5.3, 5.6

[43] Shane Dye, Leen Stougie, and Asgeir Tomasgard. The stochastic single resource service-provision problem. *Naval Research Logistics (NRL)*, 50(8):869–887, 2003. ISSN 1520-6750. doi: 10.1002/nav.10092. URL `http://dx.doi.org/10.1002/nav.10092`. 1.4.3

[44] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. In *37th STOC*, pages 494–503, 2005. 2.4.3, 2.4.3

[45] Jittat Fakcharoenphol and Bundit Laekhanukit. An O($\log^2$ k)-approximation algorithm for the k-vertex connected spanning subgraph problem. In *40th STOC*, pages 153–158, 2008. 2.2

[46] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. System Sci.*, 69(3):485–497, 2004. ISSN 0022-0000. 2.4.3

[47] Vivek F. Farias and Ritesh Madan. The irrevocable multiarmed bandit problem. *Oper. Res.*, 59(2):383–399, 2011. 4.1

[48] Lisa Fleischer, Kamal Jain, and David P. Williamson. Iterative rounding 2-approximation algorithms for minimum-cost vertex connectivity problems. *J. Comput. System Sci.*, 72(5):

838–867, 2006. ISSN 0022-0000. 2.2

[49] Lisa Fleischer, Jochen Könemann, Stefano Leonardi, and Guido Schäfer. Simple cost sharing schemes for multicommodity rent-or-buy and stochastic steiner tree. In *38th STOC*, pages 663–670, 2006. ISBN 1-59593-134-1. doi: http://doi.acm.org/10.1145/1132516.1132609. 2.7

[50] David A. Freedman. On tail probabilities for martingales. *Annals of Probability*, 3:100–118, 1975. doi: 10.1214/aop/1176996452. 5.7.8

[51] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979. ISBN 0-7167-1044-7. 3

[52] J. C. Gittins. *Multi-armed bandit allocation indices.* Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons Ltd., Chichester, 1989. ISBN 0-471-92059-2. With a foreword by Peter Whittle. 4.1

[53] Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *40th Annual Symposium on Foundations of Computer Science (New York, 1999)*, pages 579–586. IEEE Computer Soc., Los Alamitos, CA, 1999. 3.2

[54] Ashish Goel, Sudipto Guha, and Kamesh Munagala. Asking the right questions: model-driven optimization using probes. In *PODS*, pages 203–212, 2006. 4.1

[55] Ashish Goel, Sanjeev Khanna, and Brad Null. The ratio index for budgeted learning, with applications. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 18–27, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics. 1.6.3, 1.6.3, 4.1, 4.9

[56] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. ISSN 0097-5397. 2.2

[57] Michel X. Goemans, Andrew V. Goldberg, Serge Plotkin, David B. Shmoys, Éva Tardos, and David P. Williamson. Improved approximation algorithms for network design problems. In *5th SODA*, pages 223–232, 1994. 2, 2.2

[58] Sudipto Guha and Kamesh Munagala. Model-driven optimization using adaptive probes. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 308–317, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5. Full version as *Adaptive Uncertainty Resolution in Bayesian Combinatorial Optimization Problems*, `http://arxiv.org/abs/0812.1012v1`. 4.1

[59] Sudipto Guha and Kamesh Munagala. Approximation algorithms for budgeted learning problems. In *STOC*, pages 104–113. 2007. 1.5.3, 4.1, 4.3, 5.1.1

[60] Sudipto Guha and Kamesh Munagala. Multi-armed bandits with metric switching costs. In *ICALP*, pages 496–507, 2009. 1.6.3, 1.6.3, 4.1, 4.3, 4.9, 5.1.1, 5.5, 5.9.1

[61] Sudipto Guha, Kamesh Munagala, and Peng Shi. On index policies for restless bandit problems. *CoRR*, abs/0711.3861, 2007. `http://arxiv.org/abs/0711.3861`. Full version of *Approximation algorithms for partial-information based stochastic control with Markovian rewards* (FOCS'07), and *Approximation algorithms for restless bandit problems*, (SODA'09). 3.6, 4.1

[62] Sudipto Guha, Kamesh Munagala, and Martin Pal. Iterated allocations with delayed feedback. *ArXiv*, arxiv:abs/1011.1161, 2011. 4.1, 4.9

[63] Anupam Gupta and Jochen Konemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16 (1):3 – 20, 2011. ISSN 1876-7354. doi: DOI:10.1016/j.sorms.2010.06.001. URL `http://www.sciencedirect.com/science/article/B984X-50HX8RK-1/2/913c930f4e4fc84ff180` 2

[64] Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. Boosted sampling: Approximation algorithms for stochastic optimization problems. In *36th STOC*, pages 417–426, 2004. 1.4.3, 1.6.1, 2.1, 2.2, 2.5, 2.5

[65] Anupam Gupta, Amit Kumar, Martin Pál, and Tim Roughgarden. Approximation via cost sharing: simpler and better approximation algorithms for network design. *J. ACM*, 54(3): Art. 11, 38 pp. (electronic), 2007. ISSN 0004-5411. 2.2

[66] Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Online and stochastic survivable network design. In *41st STOC*, 2009. 1.9

[67] Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Approximation algorithms for optimal decision trees and adaptive tsp problems. In *ICALP (1)*, pages 690–701, 2010. doi: http://dx.doi.org/10.1007/978-3-642-14165-2_58. 5.1.1

[68] Anupam Gupta, Ravishankar Krishnaswamy, Marco Molinaro, and R. Ravi. Approximation algorithms for correlated knapsacks and non-martingale bandits. *To appear in FOCS 2011*, arxiv: abs/1102.3749, 2011. 5.9

[69] Anupam Gupta, Ravishankar Krishnaswamy, Marco Molinaro, and R. Ravi. Approximation algorithms for correlated knapsack and non-martingale bandits. In *51st STOC*, 2011. 1.9

[70] Anupam Gupta, Ravishankar Krishnaswamy, Marco Molinaro, and R. Ravi. Approximation algorithms for correlated knapsacks and non-martingale bandits. *CoRR*, abs/1102.3749, 2011. 4.4

[71] Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, and R. Ravi. Approximation algorithms for stochastic orienteering. In *Manuscript*, 2012. 5.10

[72] Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, and R. Ravi. Approximation algorithms for stochastic orienteering. In *23rd SODA*, 2012. 1.9

[73] Dorit s. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, June 1988. ISSN 0097-5397. doi: 10.1137/0217033. URL `http://dx.doi.org/10.1137/0217033`. 1.1

[74] Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991. ISSN 0895-4801. 2.1, 2.2, 2.4.2, 2.6, 2.6

[75] Nicole Immorlica, David Karger, Maria Minkoff, and Vahab Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *15th SODA*, pages 684–693, 2004. 2.2

[76] Nicole Immorlica, David Karger, Maria Minkoff, and Vahab S. Mirrokni. On the

costs and benefits of procrastination: approximation algorithms for stochastic combinatorial optimization problems. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04, pages 691–700, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics. ISBN 0-89871-558-X. URL `http://dl.acm.org/citation.cfm?id=982792.982898`. 1.4.3

[77] Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001. (Preliminary version in *39th FOCS*, pages 448–457, 1998). 2, 2.2

[78] E.G. Coffman Jr. and I. Mitrani. A characterization of waiting time performance realizable by single-server queues. *Operations Research*, 28(3):810–821, 1980. 1.5.3, 5.1.1

[79] Samir Khuller and Balaji Raghavachari. Improved approximation algorithms for uniform connectivity problems. *J. Algorithms*, 21(2):434–450, 1996. 2.2

[80] Philip N. Klein and R. Ravi. When cycles collapse: A general approximation technique for constrained two-connectivity problems. In *3rd IPCO*, pages 39–56, 1993. 2.2

[81] Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM J. Comput.*, 30(1):191–217, 2000. ISSN 1095-7111. 3.2

[82] Anton J. Kleywegt and Jason D. Papastavrou. The dynamic and stochastic knapsack problem with random sized items. *Oper. Res.*, 49:26–41, January 2001. ISSN 0030-364X. 3.2

[83] Guy Kortsarz and Zeev Nutov. Approximating node connectivity problems via set covers. *Algorithmica*, 37(2):75–92, 2003. 2.2

[84] Guy Kortsarz, Robert Krauthgamer, and James R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM J. Comput.*, 33(3):704–720, 2004. ISSN 0097-5397. 2.2

[85] S. Rao Kosaraju, Teresa M. Przytycka, and Ryan S. Borgstrom. On an Optimal Split Tree Problem. In *WADS*, pages 157–168, 1999. 5.1.1

[86] Lap Chi Lau and Mohit Singh. Additive approximation for bounded degree survivable network design. In *40th STOC*, pages 759–768, 2008. 2.2

[87] Lap Chi Lau, Joseph Naor, Mohammad R. Salavatipour, and Mohit Singh. Survivable network design with degree or order constraints. In *39th STOC*, pages 651–660, 2007. 2.2

[88] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46(3):259–271, February 1990. ISSN 0025-5610. doi: 10.1007/BF01585745. URL `http://dx.doi.org/10.1007/BF01585745`. 1.1

[89] Rolf H. Möhring, Andreas S. Schulz, and Marc Uetz. Approximation in stochastic scheduling: the power of lp-based priority policies. *J. ACM*, 46(6):924–942, 1999. 1.5.3, 5.1.1

[90] Rolf H. Möhring, Andreas S. Schulz, and Marc Uetz. Approximation in stochastic scheduling: the power of LP-based priority policies. *Journal of the ACM (JACM)*, 46(6):924–942, 1999. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/331524.331530. 3.2

[91] Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted steiner tree and related problems. In *FOCS*, pages 210–219, 2011. 2.8

[92] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, 1995. 3.2

[93] R. Ravi and Amitabh Sinha. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. In *10th IPCO*, pages 101–115, 2004. 2.2

[94] R. Ravi and Amitabh Sinha. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. *Math. Program.*, 108:97–114, August 2006. ISSN 0025-5610. doi: 10.1007/s10107-005-0673-5. URL `http://dl.acm.org/citation.cfm?id=1132713.1132716`. 1.4.3

[95] David B. Shmoys and Chaitanya Swamy. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *J. ACM*, 53:978–1012, November 2006. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/1217856.1217860. URL `http://doi.acm.org/10.1145/1217856.1217860`. 1.4.3

[96] David B. Shmoys and Chaitanya Swamy. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *J. ACM*, 53(6):978–1012, 2006. ISSN 0004-5411. 2.2

[97] Martin Skutella and Marc Uetz. Scheduling precedence-constrained jobs with stochastic processing times on parallel machines. In *12th SODA*, pages 589–590. Society for Industrial and Applied Mathematics, 2001. ISBN 0-89871-490-7. 3.2

[98] David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15 (3):435–454, 1995. ISSN 0209-9683. 2.2