SIAM J. COMPUT.
Vol. 40, No. 5, pp. 1361–1401

SAMPLING AND COST-SHARING: APPROXIMATION ALGORITHMS FOR STOCHASTIC OPTIMIZATION PROBLEMS*

ANUPAM GUPTA†, MARTIN PÁL‡, R. RAVI§, AND AMITABH SINHA¶

Abstract. We consider two- and multistage versions of stochastic combinatorial optimization problems with recourse: in this framework, the instance for the combinatorial optimization problem is drawn from a known probability distribution π and is only revealed to the algorithm over two (or multiple) stages. At each stage, on receiving some more information about the instance, the algorithm is allowed to build some partial solution. Since the costs of elements increase with each passing stage, there is a natural tension between waiting for later stages, to gain more information about the instance, and purchasing elements in earlier stages, to take advantages of lower costs. We provide approximation algorithms for stochastic combinatorial optimization problems (such as the Steiner tree problem, the Steiner network problem, and the vertex cover problem) by means of a simple sampling-based algorithm. In every stage, our algorithm samples the probability distribution of the requirements and constructs a partial solution to serve the resulting sample. We show that if one can construct cost-sharing functions associated with the algorithms used to construct these partial solutions, then this strategy results in provable approximation guarantees for the overall stochastic optimization problem. We also extend this approach to provide an approximation algorithm for the stochastic version of the uncapacitated facility location problem, a problem that does not fit into the simpler framework of our main model.

Key words. stochastic optimization, approximation algorithms, combinatorial optimization, cost-sharing functions

AMS subject classifications. 68W25, 90C15, 90C27

DOI. 10.1137/080732250

1. Introduction.

Infrastructure planning and installation problems often have to deal with the uncertainty caused by demand which evolves over time, without deterministic knowledge characterizing this demand. For example, communication network infrastructure has to be installed before the actual usage pattern is known, albeit forecasts and estimates of the demand may be available. While there is a tremendous amount of literature on deterministic optimization models, the preferable (and indeed, more accurate) method for tackling such problems in the presence of partial probabilistic information is the field of stochastic optimization.

In this paper, we consider several canonical combinatorial optimization problems in a stochastic setting, such as Steiner tree and facility location. Observe that the deterministic versions of these problems are themselves NP-hard, so that one must

*Received by the editors August 5, 2008; accepted for publication (in revised form) May 13, 2011; published electronically September 27, 2011. Preliminary versions of the content of this paper appeared in [18] and [19]. The work of the third and fourth authors was supported in part by NSF grant CCR-0105548 and by ITR grant CCR-0122581 (the ALADDIN project).

http://www.siam.org/journals/sicomp/40-5/73225.html

†Department of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 (anupamg@cs.cmu.edu). This author's work was supported in part by NSF CAREER award CCF-0448095 and by an Alfred P. Sloan Fellowship.

‡Google, Inc., 76 9th Avenue, 4th Floor, New York, NY 10011 (martin@palenica.com). This author's work was supported in part by ONR grant N00014-98-1-0589 and by NSF grant EIA 02-05116.

§Tepper School of Business, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 (ravi@cmu.edu).

¶Ross School of Business, University of Michigan, 701 Tappan Avenue, Ann Arbor, MI 48109 (amitabh@umich.edu).

sacrifice either generality and speed (if one uses exact algorithms such as integer programming) or accuracy (if one uses fast heuristics and approximation algorithms).

Traditional stochastic optimization models [28, 4, 27] have chosen the former approach. These models suffer from the well-known problem of the curse of dimensionality when applied to NP-hard problems such as the ones considered in this paper. Given that the deterministic versions of these problems are already hard to solve exactly, exact algorithms applied to the stochastic extensions often become insurmountably slow due to this combinatorial explosion.

In contrast, the field of theoretical computer science has often used the framework of approximation algorithms [47] to provide fast algorithms with provable guarantees on the quality of the solution. However, barring a few exceptions (detailed in section 1.1), very few approximation algorithms have been proposed for stochastic optimization problems. In this paper, we provide a general framework for converting approximation algorithms for deterministic problems into those for their stochastic counterparts.

The added ingredient allowing us to extend deterministic approximation algorithms to the stochastic setting is the idea of cost-sharing functions [48]. These functions allocate the overall cost of the solution to the clients receiving service. We show that if the deterministic approximation algorithms can be associated with cost-sharing functions which satisfy certain properties, then a very simple sampling-based algorithm can be proved to be a good approximation.

Our algorithm relies on the simple but fundamental idea of sampling. Suppose one could draw samples from an oracle of some kind which perfectly mimics the process by which the actual demand is generated. One naïve strategy would be to draw a sufficient number of samples and construct partial solutions based purely on these samples. In this paper, we show that this strategy in fact works! In the presence of algorithms with associated cost-sharing functions, one can construct the partial solutions of earlier stages by simply drawing a certain number of samples and using the deterministic algorithm to construct a solution for the resulting clients. Although we focus on problems where the deterministic version is NP-hard, our approach may also be applied to problems where the deterministic version is solvable in polynomial time but the stochastic version is NP-hard (such as the shortest path problem [39]).

In the remainder of this section, we briefly review the literature and place our results in the context of the existing work. In section 2, we describe our stochastic optimization model in detail, while section 3 contains our definition of a cost-sharing function along with the properties it needs to satisfy. Section 4 contains the main set of theorems that describe how arbitrary stochastic optimization problems can be well-approximated if approximation algorithms with cost-sharing functions are available. In section 5, we construct such approximation algorithms and cost-sharing functions for some classical combinatorial optimization problems. Although the uncapacitated facility location problem does not fit directly into our main model, our overall approach of approximation using sampling does in fact work with some modifications; the corresponding results are provided in section 6. Our results are summarized in Table 1.1, with the precise definition of the model and cost-sharing functions appearing in section 2.

1.1. Literature review. The study of stochastic optimization dates back to the initial postwar days of linear programming, with early papers by Dantzig [7] and Beale [3]. The field is now fairly well developed, and the interested reader is referred to comprehensive texts by Birge and Louveaux [4], Kall and Wallace [27],

TABLE 1.1

Summary of results. All our results are obtained in the "black box" model (section 2), and the strictness definitions are provided in section 3. The approximation ratio of the first-stage algorithm used is denoted α. The "Other work" column lists the best known approximation ratios for some versions of these problems, with the following abbreviations used to identify the model/algorithm: SAA for two-stage sample average approximation, ID for two-stage independent decisions (section 4.1.2), FS for two-stage finite scenario, and kSAA for k-stage sample average approximation. Hardness results are for the deterministic versions.

Problem	Strictness theorem	Stoch. model	This paper	Hardness	Other work
Steiner tree	$\alpha = 1.55$, 2-strict	Two-stage	3.55	$\frac{96}{95}$ [6]	12.6 (no root) [17]
	$\alpha = 2$, 1-superstrict cross-monotone	k-stage	$2k$		6 (no root) [11]
Steiner net.	$\alpha = 4$, 4-unistrict	ID	8	$\frac{96}{95}$ [6]	5 (ID) [11]
Vertex cover	$\alpha = 2$, 1-unistrict	ID	3	1.16 [21]	4 (SAA) [42], 2 (FS) [39] $2+\epsilon$ (kSAA) [44]
	$\alpha = 2$, 2-strict 3-superstrict	Two-stage	4		
Uncap. facility location	$\alpha = 3$, 5.45-strict	Two-stage	8.45	1.46 [15]	2.369 (FS) [44], $1.7k - 0.2$ (kSAA) [45]

and Klein Haneveld and van der Vlerk [28]. While substantial progress has been made on stochastic extensions of linear programming, only moderate progress has been reported for general integer programs even for the two-stage case [41, 29].

In the field of scheduling, stochastic problems have been studied extensively in the literature, since often jobs have to be scheduled without complete information about their processing times or arrival times. Pinedo [37] is a recent text covering some of this work, while other approximation algorithms for stochastic scheduling have been provided for various types of problems by Kleinberg, Rabani, and Tardos [30], Goel and Indyk [13], Möhring, Schulz, and Uetz [34], and Skutella and Uetz [43].

Within the past decade, there has been a surge of interest in approximation algorithms for stochastic optimization problems. The earliest such work that we are aware of is that of Dye, Stougie, and Tomasgard [9]. Initially, much of this work was for two-stage models, including that of Ravi and Sinha [39], Immorlica et al. [23], Shmoys and Swamy [42], and Gupta, Ravi, and Sinha [20]. Much of this work is in the *finite scenario* model, wherein the second stage is specified by a list of possible scenarios, each scenario completely specified by the requirements, costs, and probability of occurrence. Since the papers above (with the exception of [42]) consider each scenario explicitly, the running time is dependent on the number of scenarios.

In contrast, this paper assumes no restrictions on the number of scenarios but relies on *sampling access* to the probability distribution of the subsequent stages. That is, one can at any time efficiently generate a sample scenario for which the probability of the scenario occurring is the same as the probability of generating that scenario in our sample. Our algorithms require only sampling access to generate solutions whose expected value is bounded compared to the expected value of the optimal solution.

There is another stream of literature which also uses only sampling access, using the *sample average approximation* (SAA) algorithm to construct solutions. Concurrent with our work, Shmoys and Swamy [42] obtained approximation algorithms for the two-stage model with sampling access (i.e., the black box model), and extended it in [45] to multistage stochastic optimization problems, using the SAA method. Their algorithm proceeds by sampling a large number of scenarios and constructing

an integer programming (IP) formulation for the multistage problem that includes all sampled scenarios. They show that an approximate solution to the linear relaxation of this integer program can be computed efficiently, and if the underlying deterministic problem admits an LP-rounding approximation algorithm, then the fractional solution to the multistage stochastic problem can also be rounded to an integer solution with bounded cost.

An important ingredient in our algorithms is the usage of cost-sharing functions to bound the costs of the solutions constructed. The notion of cost allocation is an important concept in economics (surveyed, for example, by Young [48]), and lately this idea has been making an appearance in approximation algorithms. Cost-sharing functions have been used to study the relationship between games and algorithms, as well as to devise better approximation algorithms, for both deterministic and stochastic optimization problems. Some of the early work in this arena includes that of Jain and Vazirani [25, 26] and Moulin and Shenker [35]. Many combinatorial optimization problems can be formulated as integer linear programs, and cost shares can sometimes be interpreted as feasible dual solutions to the linear relaxations of these formulations [16, 26]. In this paper, although we do not define cost shares in terms of dual solutions, our cost-sharing functions for some problems are derived from such duals.

Gupta et al. [16] introduced the notion of *strictness* in cost-sharing functions, which essentially ensures that the cost of the solution is split somewhat proportionately among the clients requiring service. Strictness turns out to be a very powerful concept, and Gupta et al. [16] use it to provide approximation algorithms for various network design problems, including multicommodity rent-or-buy and Steiner tree. Our work builds on [16]; we appropriately modify the definition of strictness and show that strict cost-sharing functions can also be used for approximation algorithms for stochastic optimization. Pál and Tardos [36] also use cost-sharing functions to develop approximation algorithms for uncapacitated facility location, which we also extend to provide an approximation algorithm for the stochastic version of uncapacitated facility location in our model.

Preliminary versions of the ideas in this paper appeared in [18] and [19]. Subsequent to that, Hayrapetyan, Swamy, and Tardos [22] considered a variant of multistage stochastic optimization for network design, and their proofs use some of the cost-sharing ideas similar to the cost-shares we define for the stochastic Steiner tree problem. Our version of the stochastic Steiner tree problem assumes that a root vertex that is guaranteed to be a terminal in all scenarios is known in the first stage. Gupta and Pál [17] and Fleischer et al. [11] considered stochastic optimization versions of the Steiner tree problem (among other problems) without requiring a known root vertex and were able to use cost-sharing functions to provide approximation algorithms with approximation ratios of 12.6 and 6.

The literature on approximation algorithms for stochastic optimization problems has also spawned other research with more complex models that combine features of stochastic optimization and online algorithms. For instance, Anthony and Gupta [2] consider a multistage environment where at every stage some demands need to be served, but elements can be leased for fixed lengths as an alternative to purchasing them for the entire horizon or renting them for a single stage. They reduce the problems to appropriate stochastic optimization problems and also provide approximation algorithms for other problems in their framework. Garg et al. [12] study online algorithms where the input arises from a known probability distribution (as opposed to adversarially, which is the dominant model in research on online algorithms) and

provide approximability results for some such problems.

We conclude this section with a brief discussion on inapproximability and lower bounds. The survey by Swamy and Shmoys [46] describes some problems for which approximating the multistage stochastic version is provably harder than for the deterministic counterpart. For instance, the shortest-paths problem is solvable in polynomial time in the deterministic version, but its stochastic version is NP-hard. The Steiner tree problem is approximable to a constant factor in the deterministic setting, but the stochastic problem with general cost inflation is as hard to approximate as the group Steiner tree problem. Linear programs are #P-hard to solve in the black-box model [10], but the deterministic versions are in P. Of course, existing hardness results for the deterministic versions automatically imply that the same hardness results hold for the stochastic versions; these are listed in Table 1.1. Since our theorems bound the cost for each stage, individually, by the optimal cost, a lower bound of k for k-stage problems holds for algorithms proceeding by our technique of specifying a deterministic algorithm and a cost-sharing function.

There are also some lower bounds on how good these cost-sharing functions can get: [11] provides a lower bound for cost-shares for the Steiner tree problem arising from primal-dual-type algorithms, and [24] shows that, for the uncapacitated facility location problem, cross-monotonic cost-shares that recover more than 1/3 of the cost of a solution do not exist. Könemann et al. [32] prove that no cross-monotonic cost-shares exist for the Steiner tree problem that recover more than 1/2 of the cost of a solution. Note that these bounds on limitations of cost-sharing schemes provide bounds on the applicability of our algorithms, but they do not necessarily prove inapproximability of the stochastic problems themselves.

2. Model description. We begin by defining an abstract combinatorial optimization problem Π, first in a deterministic setting. Let U be a universe of *clients* or *demands* that need to be served, and X be the set of *elements* we can purchase to serve a given set of clients. Each element $e \in X$ has a nonnegative cost, given by $c(e) \in \mathbb{R}^+$. Given a set of clients $S \subseteq U$, let $\mathsf{Sols}(S) \subseteq 2^X$ denote the set of *feasible solutions* of S, or one that *satisfies* or meets the requirement of each client in S. For an element set $F \subseteq X$, let $c(F) = \sum_{e \in F} c(e)$ denote the cost of F.

The deterministic combinatorial optimization problem is therefore specified by the sets U, X, S, the cost function $c(.)$, and the set of feasible solutions $\mathsf{Sols}(S)$. The objective of the minimization problem is to find a minimum cost feasible solution; that is, find $F \in \mathsf{Sols}(S)$ which minimizes $c(F)$. We let $\mathsf{Det}(\Pi)$ denote this deterministic problem.

To illustrate this abstraction, consider the rooted Steiner tree problem on a graph $G = (V, E)$. In this problem, we are given a root node $r \in V$, a set of terminals $R \subseteq V$, and a cost function on the edges. The objective is to find a minimum cost tree which spans $r \cup R$. Here the universe of possible clients is $U = V$, the element set X is the set of edges E, and the client set S is the set of terminals R. A client $v \in R$ is satisfied by a set of edges $F \subseteq E$ if and only if F contains a path from v to r. The set of feasible solutions $\mathsf{Sols}(S)$ is therefore the set of all subgraphs of G which contain a connected component which includes all vertices in $r \cup R$.

We restrict our attention to problems which satisfy certain "well-behavedness" properties. We call such problems *subadditive*, and define the requirements as follows.

Definition 2.1. *A problem* Π *is* subadditive *if all of the following conditions hold:*

SA1. *If* S *and* S' *are two legal client sets for* Π*, then so is* $S \cup S'$.

SA2. *If* $F \in \mathsf{Sols}(S)$ *and* $F \subseteq F' \subseteq X$, *then* $F' \in \mathsf{Sols}(S)$.

SA3. *If* S *and* S' *are two legal client sets for* Π *and* $F \in \mathsf{Sols}(S)$ *and* $F' \in \mathsf{Sols}(S')$, *then* $F \cup F' \in \mathsf{Sols}(S \cup S')$.

In the Steiner tree problem described above, subadditivity is ensured by the presence of the root: if $T_i = (V, E_i)$ is a subgraph spanning $S_i \cup \{r\}$, then $(V, \cup_i E_i)$ is a subgraph that spans $\cup_i S_i \cup \{r\}$. As in some previous papers which give approximation algorithms for two-stage stochastic optimization problems [23, 39], we restrict our attention to subadditive problems. Subadditivity is not essential for constructing approximation algorithms for stochastic problems in general; for example, [17] and [11] provide constant-factor approximation algorithms for the (nonsubadditive) stochastic Steiner tree problem without a root vertex. In our work, subadditivity plays two important roles: it ensures feasibility of the strategy of constructing partial solutions for sampled client sets, and it helps bound the cost of the solution.

Often, we will denote a problem instance by simply specifying the element set X and the client set S, where the other components of the problem (the universe U, cost function $c()$, and set of feasible solutions) are clear from the context. Given such a problem (X, S), we let $\mathsf{OPT}(X, S)$ denote the cost of an optimal solution to the problem; that is, $\mathsf{OPT}(X, S) = \min_{F \in \mathsf{Sols}(S)} c(F)$. If an algorithm $\mathcal{A}$ is used to compute a feasible solution for the problem (X, S), the cost of this solution is denoted $c(\mathcal{A}(X, S))$.

2.1. Stochastic versions of deterministic problems. The stochastic version of a combinatorial optimization problem is obtained when we don't know the client set explicitly, and some set of decisions have to be made knowing only a distribution π that the client sets are drawn from. In particular, we consider the model of *k-stage stochastic optimization with recourse*, which we describe in the following. Initially, only the universe U, the element set X, and the cost function $c()$ are known, while a probability distribution π gives us some information about the client set S. In each of k stages, the probability distribution gets more refined, until the final stage when all uncertainty is removed and the actual client set S is revealed. We can "purchase" elements of X at any stage in this process, though the elements get progressively costlier as time passes. The objective is to compute a minimum cost solution which satisfies S. However, since this is a stochastic problem and only probabilistic information is known in the preliminary stages, the best one can hope for is a strategy for purchasing elements that minimizes the *expected cost* of the solution while constructing a feasible solution for S.

2.1.1. The two-stage problem. The two-stage problem $\mathsf{Stoc}(\Pi)$ corresponding to the problem Π is particularly easy to describe: in the first stage, we assume that the universe U, the element set X, the cost function $c()$, and the probability distribution π are known, as is the *inflation factor* $\sigma \geq 1$. The probability distribution $\pi : 2^X \to [0, 1]$ is over the possible client set $S \subseteq V$ that can appear in the second stage. Given this information, we are allowed to purchase a first-stage set $F_0 \subseteq U$ at cost $c(F_0)$.

In the second stage, a demand set $\mathbf{S}$ is drawn from the distribution π and revealed to us, whence we can buy some more elements $F_{\mathbf{S}} \subseteq U$, at cost $c(F_{\mathbf{S}})$, such that $F_0 \cup F_{\mathbf{S}} \in \mathsf{Sols}(\mathbf{S})$. (We will use boldface to denote that $\mathbf{S}$ is a random variable.) Any costs incurred in the second stage are inflated by a factor σ compared to those same costs incurred in the first stage. The objective is to minimize the expected cost shown below, where the expectation is taken over the distribution π:

$$Z = c(F_0) + \mathbf{E}[\sigma\, c(F_{\mathbf{S}})]. \tag{2.1}$$

1. The universe of clients U, set of elements X, cost function $c()$, and inflation factors $\{\sigma_i\}$ are specified. We are also given the set of feasible solutions $\mathsf{Sols}(S)$ for any client set $S \in 2^U$.
2. The first $k-1$ stages occur, with the following occurring in stage i:
   (a) The signal $\mathbf{s}_i$ is revealed; let the observed signal be s_i. The vector of signals received so far is given by $s = (s_1, s_2, \ldots, s_i)$, and the probability of the final client set being S is given by the conditional distribution $\pi[S|s]$.
   (b) We purchase a set of elements F_i at a cost of $c(F_i)\prod_{j=1}^{i}\sigma_i$. Elements purchased in previous stages are retained and cannot be discarded.
3. The final stage k occurs: in this stage, the actual client set S is revealed, according to the probability distribution $\pi[S|s]$. We purchase a final set of elements F_k so that $\cup_{j=1}^{k} F_j \in \mathsf{Sols}(S)$. That is, the union of elements purchased in stages 1 through k must be a feasible solution for S.

FIG. 2.1. *Sequence of events for multistage stochastic optimization.*

We will later show how to relax the assumption of a *constant* inflation factor σ: we will show that we can obtain similar results even if the inflation factor σ is a random variable arbitrarily correlated with the random client set $\mathbf{S}$. Details on this are given in section 4.3.

2.1.2. The general *k*-stage problem. The description of the k-stage problem requires us to specify how we are given progressively more information about the final client set S. We use the notion of *signals* to capture the information revealed in each stage. Let $\mathbf{s}_i$ be a random variable denoting the signal received in stage i, and s_i denote the actual signal received in stage i. These signals are correlated with the final client set S in that, at any stage, the probability that S will be the client set revealed at the end is a function of the signals received so far.

Furthermore, we now have an inflation parameter associated with *each stage*. At each stage i, the cost function is multiplied by another factor of σ_i. Therefore the cost of purchasing an element set F to serve a client set S in stage i is $c(F) \times \prod_{j=1}^{i}\sigma_i$. We will assume that these multipliers σ_i are also known up-front. We assume that costs are nondecreasing, which corresponds to $\sigma_i \geq 1$ for all i.

The precise sequence of events is laid out in Figure 2.1. We want an algorithm which computes the set F_i in each stage so that the overall expected cost of the solution is minimized. Formally, the objective function is $Z = \mathbf{E}[\sum_{i=1}^{k}(\prod_{j=1}^{i}\sigma_i)c(F_i)]$.

Note that the expectation is taken over the probability distribution π. For the problem (and any algorithm) to make sense, we must have the conditional distributions consistent with each other. This is specified as follows when the space of possible signals is a discrete set; the analogous specification holds for continuous sets. Let $p(t|s)$ denote the probability of receiving signal t, given that the vector s of signals has been received so far, and let $s+t$ denote the vector obtained by concatenating the signal t to the end of the vector s. The probability distributions and signals are consistent if the following holds for any vector of signals s and client set S: $\pi(S|s) = \sum_t \pi(S|s+t)p(t|s)$.

It is also apparent that the first signal $\mathbf{s}_1$ and first inflation multiplier σ_1 play no

meaningful role. Nevertheless, we retain them for notational ease. We normalize them so that $\mathbf{s}_1$ has unit support (it always takes the same value), and $\sigma_1 = 1$. Furthermore, the only information revealed in the last stage is the set of realized clients S. This can also be modeled as a random variable $\mathbf{S}$. For notational convenience, we sometimes use the k-stage signal $\mathbf{s}_k$ to refer to the random variable $\mathbf{S}$ and vice versa.

An alternate view of the k-stage process sometimes seen in the literature (e.g., [45]) is by means of a "scenario tree." In this, the stochastic process is modeled as a rooted tree with k levels of nodes, where a level indicates the number of edges from the root. There is a single node at level 1, and each level i corresponds to stage i of the stochastic process. From the unique level-1 node, traversing to one of its children represents one realization of the second stage, so each second stage scenario is represented by a distinct level-2 node. Continuing in this fashion, each level-i node represents a stage-i scenario, the nodes in the path from the root to it indicate the past history of realized scenarios, and the subtree rooted at this node represents the conditional probability distribution given that we are in the scenario represented by the node. A full evolution of the stochastic process is now interpreted as traversing a unique root-leaf path. We mention this description to aid the reader, but we retain our description because it allows for the possibility that the inflation factors σ_i have continuous support, as will be the case in section 4.3.

2.2. Sampling model. While the sequence of events is laid out in Figure 2.1, there is one additional channel of information available to our algorithm. We assume that the conditional distribution $\pi(.|.)$ can be *sampled efficiently* at any point of time. That is, at any point of time, where s is the vector of signals received so far, we can access an *oracle* which returns a client set S, where the probability of returning S is precisely $\pi(S|s)$. In other words, *the oracle perfectly mimics the real probability distribution* $\pi(.|.)$ and can be called upon to return independent samples from this distribution as often as necessary. Furthermore, the time taken by this oracle is polynomial in the size of the input. This oracle access is all that is required by our algorithm; the probability distribution $\pi(.|.)$ itself is never required. One may observe that in Figure 2.1 the probability distribution $\pi(.|.)$ is never revealed. In our analysis, we will use only the fact that the oracle follows the distribution $\pi(.|.)$.

The oracle is sometimes called a *black box* in the literature, and we use the two terms interchangeably in this paper. We remark briefly on the implications of having such oracle access. Note that the success of any stochastic optimization algorithm critically depends on the accuracy of the information available. If one is optimizing in the face of an uncertain future, the best one can do is limited by the availability of forecasts or probability distributions about the future. That said, oracle access is the minimum one can ask for in terms of how the forecasts are made available.

This is in contrast to previous work in the finite scenario model in the literature, where one requires an explicit listing of all possible scenarios, along with their probabilities. If the number of scenarios is too large, the computational complexity of the resulting algorithms becomes prohibitive, since it may not even be possible to list all the scenarios succinctly. The other stream of work in the stochastic optimization literature relies on explicit knowledge of the probability distribution $\pi(.|.)$ at all points of time. The oracle model, on the other hand, can handle an exponential number of scenarios with arbitrarily complicated probability distributions *so long as they can be sampled efficiently.*

This being said, there has been some recent exciting work on *scenario reduction*: taking a distribution π given as a black box and obtaining a "small" number M of

explicit scenarios such that any solution to the stochastic problem for these scenarios is a good approximation to the original problem; see, e.g., [31, 5, 45] for some work along these lines.

3. Cost-sharing functions. The primary analytic concept used in our analysis is that of *cost-sharing functions*. We define these functions and elucidate their properties in this section. Loosely speaking, a cost-sharing function allocates the cost of providing service to the recipients of the service. If we can compute the "willingness to pay" of clients and use it as a lower bound, then the cost-sharing function allows us to bound the overall cost of the solution.

Cost-sharing functions have long been used in game theory [48]; lately, they have also been used in approximation algorithms. We will use a slight variant of a cost-sharing function defined first by Gupta et al. [16]: in contrast to previous cost-sharing functions, these are defined relative to a fixed approximation algorithm for the problem Π.

DEFINITION 3.1 (cost-sharing function). *A cost-sharing function ξ assigns a nonnegative real number denoted $\xi(X, S, j)$ to each client j in a subset of clients S when elements from X must be purchased to serve S.*

We use $\xi(X, S, S')$ to denote $\sum_{j \in S'} \xi(X, S, j)$.

DEFINITION 3.2 (cost-sharing algorithm). *A cost-sharing algorithm $\mathcal{A}$ takes an instance (X, S) of the problem Π and returns a feasible solution $\mathcal{A}(X, S) \subseteq \mathsf{Sols}(S)$ and a cost-sharing function ξ.*

Clients which do not require service (i.e., are not in S) should not incur any cost. In other words, only clients in S should be assigned nonnegative cost shares; $\xi(X, S, j)$ must be zero if $j \notin S$. We require this for all our algorithms, and for the rest of the paper we will work with this requirement without necessarily explicitly stating or verifying it.

DEFINITION 3.3 (competitiveness). *A cost-sharing function ξ is* competitive *if for every problem instance (X, S) we have $\xi(X, S, S) \leq \mathsf{OPT}(X, S)$.*

A competitive cost-sharing function thus serves as a lower bound on the cost of the optimal solution. If we bound the cost of the solution found by an algorithm by some multiple of the cost-sharing function, we obtain an approximation algorithm. In the literature, competitiveness is sometimes called *fairness*. We also require all our cost-sharing functions to be competitive, and this is assumed throughout the paper without our explicitly mentioning it.

Recall that $\mathcal{A}$ is an *α-approximation algorithm* if $c(\mathcal{A}(X, S)) \leq \alpha\, \mathsf{OPT}(X, S)$. For the multistage problems, we will need the following stronger guarantee, which can be thought of as being complementary to competitiveness.

DEFINITION 3.4 (approximation w.r.t. ξ). *A cost-sharing algorithm $\mathcal{A}$ (with cost-sharing function ξ) is* an α-approximation algorithm w.r.t. ξ *if*

$$c(\mathcal{A}(X, S)) \leq \alpha\, \xi(X, S, S). \tag{3.1}$$

In the literature (e.g., [8, 24]), the related definition of α-budget balance is sometimes used when the cost-sharing function is such that there exists a solution $F(S)$ with $c(F(S)) \leq \alpha \xi(X, S, S)$. Definition 3.4 requires the specification of an algorithm $\mathcal{A}$, and we use this definition in the rest of this paper. Note that if ξ is competitive, then $\xi(X, S, S) \leq \mathsf{OPT}(X, S)$, and thus an α-approximation algorithm w.r.t. ξ is also simply an α-approximation algorithm.

3.1. Cross-monotonicity. Cost-sharing functions can be restricted to satisfy several other properties, which aid in the analysis of some of the algorithms in this

paper. We now define the relevant properties. Among the properties described below, the specific properties required by each algorithm will be listed when the algorithm is described.

DEFINITION 3.5 (cross-monotone). *A cost-sharing function ξ is* cross-monotone *if for every pair of client sets S and T such that $S \subseteq T$ and every client $j \in S$ we have $\xi(X, T, j) \leq \xi(X, S, j)$.*

Cross-monotonicity models the fact that if the client set increases, the cost shares of clients in the original client set should not go up. In other words, clients should not be penalized if more clients need to be served; instead, since there are more clients to share the cost of service, the cost to any individual client should possibly go down. Cross-monotonicity is a property that we will not always be able to satisfy, but we obtain interesting implications when cross-monotonicity is satisfied.

3.2. Strictness of cost-sharing functions. We now come to a notion that will be crucial to our analyses; this is a slight extension of the notion of "strictness" defined earlier in [16].[1]

DEFINITION 3.6 (strict). *A cost-sharing algorithm $\mathcal{A}$ (and its associated cost-sharing function ξ) is called* β-strict *if for any sets of clients S and $T \subseteq U \setminus S$ there exists a solution $F_T \subseteq X$ constructable in polynomial time such that $\mathcal{A}(X, S) \cup F_T \in \mathsf{Sols}(S \cup T)$ and*

$$c(F_T) \leq \beta \times \xi(X, S \cup T, T). \tag{3.2}$$

Loosely speaking, one should think of T as being the set of clients coming in a second stage, and S as being some set of clients in the first stage, and the notion of β-strictness as relating the cost of the recourse solution constructed to the share of the cost to be borne by T had the clients in T been present in the first stage itself (which is given by $\xi(X, S \cup T, T)$). A formalization of this notion will appear in the proof of Theorem 4.1. Usually, we will also specify an algorithm to compute the solution F_T guaranteed in the above theorem. This algorithm will be called the *augmenting algorithm* and will be denoted by $\mathsf{Aug}_{\mathcal{A}}$.

For the case of independent decisions, we do not require the full power of strictness, and the following definition will suffice.

DEFINITION 3.7 (unistrict). *A cost-sharing algorithm $\mathcal{A}$ (and its associated cost-sharing function ξ) is called* β-unistrict *if for any set of clients S and singleton client $j \in U \setminus S$ there exists a solution $F_j \subseteq X$ constructable in polynomial time such that $\mathcal{A}(X, S) \cup F_j \in \mathsf{Sols}(S \cup \{j\})$ and $c(F_j) \leq \beta \times \xi(X, S \cup \{j\}, j)$.*

Unistrictness is a weaker requirement than strictness, since it is equivalent to strictness restricted so that only a single client is added at a time.

On the other hand, k-stage stochastic problems will require us to use the following (slightly different) notion of strictness. Given a set $X' \subseteq X$ with costs for each element $e \in X$, let X/X' denote the same element set X with the costs of the elements in X' set to zero (i.e., with the elements in X' "contracted").

DEFINITION 3.8 (superstrict). *A cost-sharing algorithm $\mathcal{A}$ (and its associated cost-sharing function ξ) is called* β-superstrict *if for any sets of clients S and $T \subseteq U \setminus S$ we have*

$$\xi(X/\mathcal{A}(X, S), T, T) \leq \beta \times \xi(X, S \cup T, T). \tag{3.3}$$

[1] The notion of strictness defined in [16] was the same as our definition restricted to the case when $|T| = 1$. We use this notion in some cases as well, but call it "unistrictness"; see Definition 3.7.

Let us quickly contrast superstrictness (Definition 3.8) to strictness (Definition 3.6). If $F = \mathcal{A}(X, S)$ is the solution output by $\mathcal{A}$ in the set S, and F_T is the solution constructed to serve the new second-stage clients $T \setminus S$, then strictness required us (a) to efficiently find a recourse solution F_T to augment F and get a solution for $S \cup T$, and (b) to charge the cost of F_T to $\xi(X, S \cup T, T)$ (up to a factor of β). Under the new definition of superstrictness, we require that (a′) the solution F_T be obtained by setting the costs of elements in F to zero and running the *same* algorithm $\mathcal{A}$ on the resulting "reduced" set X/F (i.e., the augmenting algorithm is defined as first setting the costs of elements in F to zero and then running $\mathcal{A}$), and (b′) the *cost shares* of T in this run $\xi(X/F, T, T)$ be charged to cost shares $\xi(X, S \cup T, T)$ of T in the original run (up to a factor of β).

4. The transfer theorems. In this section, we show how we can obtain approximation algorithms for stochastic optimization problems from approximation algorithms for their nonstochastic counterparts, using the essential ingredients mentioned above—sampling, signals, and cost-sharing algorithms. We will begin with the special case of the two-stage model, giving our results both for general distributions in the black-box setting (in section 4.1.1) and for the case of independent decisions (in section 4.1.2). We then move on to the general k-stage case (in section 4.2) by appropriately generalizing and extending the two-stage framework.

Our results in this section are described in terms of an abstract optimization problem Π, for which we have an approximation algorithm $\mathcal{A}$ as well as an associated cost-sharing function ξ and an augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$. We give details of the specific algorithms and cost-shares for various problems in section 5.

4.1. Transfer theorems for two-stage problems. The simplest form of stochastic optimization is over $k = 2$ stages. Recall the discussion in section 2, and section 2.1.1 in particular. Since the only signal received is the null signal $\mathbf{s}_1$, it plays no role in the two-stage model and is not discussed any further in this section. The probability distribution π is also no longer a conditional distribution. Also, since the only meaningful inflation factor is σ_2, we just use σ to represent the second-stage inflation factor.

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

A naïve attempt would be to sample once from the distribution and use the set obtained as our prediction for the future: however, this is not aggressive enough in that it ignores the fact that the future is more expensive by a factor of σ. In fact, as $\sigma \to \infty$, the optimal solution would be to assume that every client in U will be realized and must be accounted for in the first stage itself. Motivated by these concerns, the algorithm for the problem $\mathsf{Stoc}(\Pi)$ is stated in Figure 4.1, in terms of the α-approximation algorithm $\mathcal{A}$ (and its associated β-strict cost-sharing function ξ and an augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$).

Our algorithm requires a number of samples that is linear in σ. Indeed, if all we have is sampling access to the distribution π, it is not hard to see that $\Omega(\sigma)$ samples are needed.

The algorithm (as well as the sequence of events) is described in Figure 4.1. We name the algorithm Boost-and-Sample for obvious reasons: we build our first-stage solution purely on the basis of a sample, except that the sample is "boosted" in the

Algorithm Boost-and-Sample (Π).
Given: Problem Π, defined by universe of clients U, probability distribution π, universe of elements X, cost function $c()$, and inflation factor σ.
1. Draw $\lfloor\sigma\rfloor$ independent samples $S_1, S_2, \ldots, S_{\lfloor\sigma\rfloor}$ of clients by sampling from the distribution π. Let $D = \cup_i S_i$.
2. Using the algorithm $\mathcal{A}$, construct a first-stage solution $F_1 \in \mathsf{Sols}(D)$. Algorithm $\mathcal{A}$ must be an α-approximation which includes a β-strict cost-sharing function ξ and augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$.

Output: The first-stage solution F_1.
3. The second stage occurs, and the client set S is realized. Using the augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ on the client set $S \setminus D$, compute the second-stage solution F_2 such that $F_2 \in \mathsf{Sols}(S \setminus D)$. By subadditivity, we will have $F_1 \cup F_2 \in \mathsf{Sols}(S)$.

Output: The second-stage solution F_2.

FIG. 4.1. *Algorithm Boost-and-Sample for two-stage stochastic optimization.*

sense that the number of client sets we obtain is equal to (the greatest integer no more than) the inflation factor σ.

As stated in the description of the algorithm above, we require that $\mathcal{A}$ be an α-approximation algorithm for the (deterministic) problem $\mathsf{Det}(\Pi)$. We also require that ξ be β-strict and that there exist an algorithm $\mathsf{Aug}_{\mathcal{A}}$ to compute the augmenting solution F_T. Note that, for the two-stage case, we do not require ξ to be cross-monotone or superstrict.

4.1.1. Analysis of Boost-and-Sample for two-stage problems. We now prove the first of our main results: the simple sampling-based algorithm Boost-and-Sample provides an approximation algorithm for the stochastic optimization problem.

THEOREM 4.1 (two-stage general theorem). *Consider a subadditive combinatorial optimization problem* Π. *Let* $\mathcal{A}$ *be a cost-sharing algorithm for* Π *such that the following hold:* (i) $\mathcal{A}$ *is an* α*-approximation algorithm for* $\mathsf{Det}(\Pi)$*;* (ii) $\mathcal{A}$ *is associated with a* β*-strict competitive cost-sharing function* ξ*; and* (iii) $(\mathcal{A},\xi)$ *is associated with an augmenting algorithm* $\mathsf{Aug}_{\mathcal{A}}$. *Then Algorithm* Boost-and-Sample *is an* $(\alpha+\beta)$*-approximation algorithm for the two-stage problem* $\mathsf{Stoc}(\Pi)$.

Proof. We will bound the expected costs of our first- and second-stage solutions separately. The first-stage cost will be bounded by comparison with the optimal cost directly, without using cost-shares. The cost-sharing function will be useful in bounding the cost of the second-stage solution, by allowing us to compare the expected cost of the augmentation with the cost of the optimal solution.

Let us fix an optimal solution, which buys the elements of F_1^* in the first stage and buys the elements of $F_2^*(S)$ in the second stage when $\mathbf{S} = S$ (i.e., the set of realized clients is S). Note that $F_2^*(S)$ needs to serve only the clients in $S \setminus S_1^*$. Therefore, the optimal cost is given by

$$Z^* = c(F_1^*) + \sum_{S \in 2^U} \pi(S)\sigma c(F_2^*(S)). \tag{4.1}$$

Let us define $Z_1^* = c(F_1^*)$ and $Z_2^* = \sum_{S\in 2^U} \pi(S)\sigma c(F_2^*(S))$. To simplify the notation, we shall assume that the inflation factor σ is an integer, that is, $\sigma = \lfloor\sigma\rfloor$; since the

algorithm takes the floor of σ at the very outset, this is without loss of generality.

First stage. Recall that we take σ samples $D_1, D_2, \ldots, D_\sigma$ in the first stage and build a solution on $D = \cup_{i=1}^{\sigma} D_i$. We claim that there exists a set of elements $\widetilde{F}_1 \in \mathsf{Sols}(D)$ with expected cost $\mathbf{E}[\, c(\widetilde{F}_1)\,] \leq Z^*$. Indeed, define $\widetilde{F}_1 = F_1^* \cup F_2^*(D_1) \cup F_2^*(D_2) \cup \cdots \cup F_2^*(D_\sigma)$. By the definition of the optimal solution, $F_1^* \cup F_2^*(D_i)$ must be a solution to D_i, and hence the subadditivity of Π implies that $\widetilde{F}_1 \in \mathsf{Sols}(D)$. In the following, we treat the samples D_i as random variables and compute the expectation over them:

$$\begin{aligned}\mathbf{E}[\, c(\widetilde{F}_1)\,] &\leq c(F_1^*) + \mathbf{E}\left[\sum_{i=1}^{\sigma} c(F_2^*(D_i))\right] \\ &= c(F_1^*) + \sum_{i=1}^{\sigma} \mathbf{E}[\, c(F_2^*(D_i))\,] \\ &= c(F_1^*) + \sigma \sum_{S \in 2^U} \pi(S) c(F_2^*(S)) \;= Z^*,\end{aligned}$$

where the penultimate equality follows from the fact that each of the D_i's is chosen from the probability distribution π. Since $\mathcal{A}$ is an α-approximation for $\mathsf{Det}(\Pi)$, and we know that the above solution $\widetilde{F_1}$ had expected cost at most Z^*, our solution F_1 (output by Boost-and-Sample in the first stage) satisfies $\mathbf{E}[\, c(F_1)\,] \leq \alpha \mathbf{E}[\, c(\widetilde{F}_1)\,] \leq \alpha Z^*$, thus bounding our expected first-stage cost by αZ^*.

Second stage. Let S be the set of realized clients, and let $F_2(S)$ be the result of algorithm $\mathsf{Aug}_{\mathcal{A}}$ on the client set $S \setminus D$ such that $F_1 \cup F_2(S) \in \mathsf{Sols}(S)$. We need to bound our expected second-stage cost $\sigma \mathbf{E}[\, c(F_2(S))\,]$; note the factor of σ due to second-stage inflation.

Recall the definition of β-strictness; our hypothesis that the algorithm $\mathcal{A}$ is associated with a β-strict cost-sharing function ξ implies that

$$c(F_2(S)) \leq \beta\, \xi(X, D \cup S, S \setminus D). \tag{4.2}$$

In the rest of the proof, we will bound from above the expected value of the cost-share on the right-hand side in the above expression.

To this end, consider the following alternative probabilistic process to generate the sets D_i and the set S. Draw $\sigma+1$ independent samples $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_{\sigma+1}$ from the distribution π. Now choose a random value K uniformly at random from $\{1, 2, \ldots, \sigma+1\}$, and let $S = \hat{D}_K$ and $D = \cup_{i \neq K} \hat{D}_i$. This process is distributed identically to the original process, since we assume that our black box outputs an independent random sample from π. Let $\hat{D}$ be the union of all D_i's, and let $\hat{D}_{-i}$ be the union $\cup_{l \neq i} \hat{D}_l$ of all the sets except $\hat{D}_i$. Since the cost sharing function ξ is competitive, we have

$$\sum_{i=1}^{\sigma+1} \xi(X, \hat{D}, \hat{D}_i \setminus \hat{D}_{-i}) \leq \mathsf{OPT}(\hat{D}). \tag{4.3}$$

Moreover, since K is chosen uniformly at random from a set of size $\sigma + 1$, we get

$$\mathbf{E}[\, \xi(X, \hat{D}, \hat{D}_K \setminus \hat{D}_{-K})\,] \leq \frac{1}{\sigma+1} \mathsf{OPT}(\hat{D}). \tag{4.4}$$

Finally, since the alternate process is probabilistically identical to the one we used to

pick D and S,

$$\begin{aligned}\mathbf{E}[\,\xi(X, D \cup S, S \setminus D)\,] &= \mathbf{E}[\,\xi(X, \hat{D}, \hat{D}_K \setminus \hat{D}_{-K})\,] \\ &\le \frac{1}{\sigma+1}\mathbf{E}[\,\mathsf{OPT}(\hat{D})\,]. \qquad (4.5)\end{aligned}$$

To complete the argument, we now show that $\mathbf{E}[\,\mathsf{OPT}(\hat{D})\,] \le \frac{\sigma+1}{\sigma} Z^*$. To derive a feasible solution to $\hat{D}$, define $\widetilde{F}_2 = F_1^* \cup F_2^*(\hat{D}_1) \cup F_2^*(\hat{D}_2) \cup \cdots \cup F_2^*(\hat{D}_{\sigma+1})$. Again, the fact that $\tilde{F}_2 \in \mathsf{Sols}(\hat{D})$ follows from the subadditivity of Π. Thus we have

$$\begin{aligned}\mathbf{E}[\,\mathsf{OPT}(\hat{D})\,] &\le c(F_1^*) + \sum_{i=1}^{\sigma+1} \mathbf{E}[\,c(F_2^*(\hat{D}_i))\,] \\ &\le Z_1^* + \frac{(\sigma+1)Z_2^*}{\sigma} \\ &\le \frac{\sigma+1}{\sigma}\,(Z_1^* + Z_2^*) = \frac{\sigma+1}{\sigma}\,Z^*. \qquad (4.6)\end{aligned}$$

Finally, chaining together (4.2), (4.5), and (4.6), we get the following: $\mathbf{E}[\,c(F_2)\,] \le \beta\mathbf{E}[\,\xi(X, D \cup S, S \setminus D)\,] \le \frac{\beta}{\sigma} Z^*$. Since each element is costlier in the second stage by a factor of σ, we infer that our expected second-stage cost is $\mathbf{E}[\,\sigma c(F_2)\,] \le \beta Z^*$. Now putting together the first- and second-stage costs gives the bound claimed in the theorem. □

It is natural to conjecture that tighter results may be obtained by choosing some number other than $\lfloor\sigma\rfloor$ for the number of samples used in constructing the first-stage solution. We believe that while small and problem-specific improvements might be possible, sampling $\lfloor\sigma\rfloor$ times is the best possible for Theorem 4.1. Sampling fewer than $\lfloor\sigma\rfloor$ times does not improve the bound on the first-stage cost since the optimal solution might have a large first-stage component; it hurts the second-stage cost since the proportionate cost of the second stage is now much larger. Sampling greater than σ times clearly hurts the first-stage cost while offering no improvement in the second-stage cost guarantee (for example, when all scenarios require disjoint solutions).

We also point out that, in general, Theorem 4.1 cannot yield an approximation ratio of better than $\min\{\alpha+\beta, 2\alpha\}$, as the following example shows. Consider an instance of the stochastic Steiner tree problem on the following graph and probability distribution. The graph G consists of several subgraphs, $H_1, H_2, \ldots, H_n$, none of whom share a vertex, where $n \gg 1$. There is also a distinct root vertex r. The shortest path between any two subgraphs H_i and H_j goes through the root. Each scenario consists of a set of terminals all contained within a single subgraph H_i. Let $\sigma = 1$. The optimal solution, therefore, is to wait for the second stage to realize, and then use an existing Steiner tree approximation algorithm to construct a solution for the realized subgraph (scenario). Our algorithm will first sample, which yields a single H_i, and construct an approximate Steiner tree for this first-stage sampled scenario. Then, in the second stage, with high probability a distinct subgraph H_j will be the realized scenario. Our algorithm will then construct a second approximate Steiner tree for H_j. Thus, our algorithm approximately constructs two distinct Steiner trees, each with approximation ratio α, resulting in an overall approximation ratio of 2α. If a cost-sharing algorithm had been used in the second stage, the overall approximation ratio would be $\alpha+\beta$. In general, there is no opportunity to save costs between the solutions of the two stages—as is seen in the proof of Theorem 4.1, the cost of each stage is bounded by the overall optimal cost, and there are very few inequalities that

offer any opportunity for additional savings. Similar examples can be constructed for all the stochastic models that we consider in the paper, so the approximation ratios proved in this paper do not offer much opportunity for improvement unless the underlying algorithm is significantly changed.

4.1.2. The special case of independent decisions. Within the context of two-stage stochastic optimization, we now consider a specific model for the second stage, which we call the *independent-decisions* model. In this model, each client $j \in U$ has a probability π_j of requiring service *independent* of all other clients.

For this special case, we show that unistrict cost-sharing functions are sufficient to obtain algorithms for stochastic problems. Recall from Definition 3.7 that unistrictness is a weaker condition than the requirement of strictness that was used in Theorem 4.1. Unistrictness allows us to obtain approximation algorithms for a wider set of problems, while also obtaining stronger results than can be obtained via strictness for some problems. Given a problem Π, we use $\mathsf{Ind}(\Pi)$ to denote the stochastic extension of Π in this independent-decisions model.

Our algorithm for this special case is a minor modification from Figure 4.1. The sample D is generated by selecting each element $j \in U$ with probability $\min\{1, \sigma\pi_j\}$. We assume that we have a cost-sharing algorithm $\mathcal{A}$, with an associated β-*unistrict* cost-sharing function ξ and augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$.

We refer to this algorithm also as Boost-and-Sample, since we discuss it only in this section and the fundamental idea of sampling more often remains the crux of the algorithm. Note that the algorithm can be implemented in polynomial time regardless of the magnitude of σ. We prove an analogous version of Theorem 4.1 for the independent-decisions model. We note that the argument used to bound the cost of the second-stage solution bears some resemblance to the proof of the performance guarantee for the multicommodity rent-or-buy problem studied in [16]. While the problem studied in [16] is a single-stage problem, the analysis technique extends to our stochastic setting and allows for generalization in that if appropriate cost-sharing functions can be found for any other problem, then our result allows its stochastic version to be well-approximated.

THEOREM 4.2 (two-stage independent decisions). *Consider a subadditive combinatorial optimization problem* Π. *Let* $\mathcal{A}$ *be a cost-sharing algorithm for* Π *such that the following hold:* (i) $\mathcal{A}$ *is an* α*-approximation algorithm for* $\mathsf{Det}(\Pi)$*;* (ii) $\mathcal{A}$ *is associated with a* β*-unistrict competitive cost-sharing function* ξ*; and* (iii) $(\mathcal{A},\xi)$ *is associated with an augmenting algorithm* $\mathsf{Aug}_{\mathcal{A}}$. *Then Algorithm* Boost-and-Sample *is an* $(\alpha+\beta)$*-approximation algorithm for the two-stage problem* $\mathsf{Stoc}(\Pi)$.

Proof. While it is possible to prove this result by closely following the lines of the proof for Theorem 4.1, we give a slightly different proof here. First, some notation: let $\pi(S) = \prod_{j\in S}\pi_j \prod_{j\notin S}(1-\pi_j)$. Let F_1^* be the first-stage component of the optimal solution, with S_1^* defined as before as the set of clients chosen for service in the first stage of the optimal solution. Let $F_2^*(S)$ be the second-stage component if the set of realized clients is S, and let Z^* be defined as in (4.1).

First stage. Again, we claim that there is $\widehat{F}_1 \in \mathsf{Sols}(D)$ such that $\mathbf{E}[\, c(\widehat{F}_1)\,] \le Z^*$; the proof in this case is by a slightly simpler "coupling" argument. As a thought experiment, let us throw elements of D into σ sets $D_1, \ldots, D_\sigma$ independently and uniformly at random. Now, since D was picked by sampling each element $j \in U$ at rate $\min\{1, \sigma\,\pi_j\}$, each D_i is distributed as though we sampled element $j \in U$ with probability at most π_j. (The contents of different D_i's are correlated negatively, but we will use only linearity of expectations.)

Define $\widehat{F}_2 = F_1^* \cup F_2^*(D_1) \cup F_2^*(D_2) \cup \cdots \cup F_2^*(D_\sigma)$. Again, $\widehat{F}_2 \in \mathsf{Sols}(D)$ from subadditivity, and

$$\begin{aligned}\mathbf{E}[\, c(\widehat{F}_2)\,] &\le c(F_1^*) + \mathbf{E}\left[\sum_{i=1}^{\sigma} c(F_2^*(D_i))\right] \\ &= c(F_1^*) + \sum_{i=1}^{\sigma} \mathbf{E}[\, c(F_2^*(D_i))\,] \\ &\le c(F_1^*) + \sigma \sum_S \pi(S)\, c(F_2^*(S)) = Z^*.\end{aligned}$$

Now an α-approximation algorithm for $\mathsf{Det}(\Pi)$ gives us a solution F_1 with $\mathbf{E}[\, c(F_1)\,] \le \alpha c(\widehat{F}_2) \le \alpha\, Z^*$, bounding our first-stage costs.

Second stage. Let S be the set of realized clients, and let $F_2(S) = \bigcup_{j \in S \setminus D} F_2(\{j\})$, where $F_2(\{j\})$ is the result of our augmentation algorithm $\mathsf{Aug}_{\mathcal{A}}$ on the individual client j. Note that for all $j \in S \setminus D$ we have $F_1 \cup F_2(\{j\}) \in \mathsf{Sols}(D \cup \{j\})$; thus by subadditivity, $F_1 \cup F_2(S) \in \mathsf{Sols}(S)$. We need to bound our expected second-stage cost, which is $\sigma\, \mathbf{E}[\, c(F_2(S))\,]$, which we will bound by the expected *first-stage* cost.

Define ϕ_j for an element j to be the random variable $\phi_j = \xi(X, D, j)$ if $j \in D$, and 0 otherwise. Likewise, define ψ_j for $j \in S \setminus D$ to be the cost $c(F_2(\{j\}))$ of augmenting a solution for D to include j as well; otherwise, $\psi_j = 0$. Let $X_j = \sigma\psi_j - \beta\, \phi_j$. Now let us condition on all the first-stage coin-tosses $\mathcal{T}$ in U except for j's toss. That is, we condition on the random variables governing whether each of the clients in $U \setminus \{j\}$ is selected in the first stage or not. Let $D_{\mathcal{T}}$ be all the clients picked according to $\mathcal{T}$ (which does not include j), and consider the expected value of X_j over the first-stage toss for j and the tosses of the realized set S:

$$\mathbf{E}[\, \sigma\, \psi_j |\ \mathcal{T}\ ] = \sigma \times \pi_j \times\ c(F_2(\{j\})) \times (1 - \min\{1, \sigma\pi_j\}) \quad \text{and} \tag{4.7}$$

$$\mathbf{E}[\, \beta\, \phi_j |\ \mathcal{T}\ ] = \beta \times \min\{1, \sigma\pi_j\} \times \xi(X, D_{\mathcal{T}} \uplus \{j\}, j). \tag{4.8}$$

By unistrictness of $\mathcal{A}$, it follows that (4.8) is at least (4.7), and hence $\mathbf{E}[\, X_j |\ \mathcal{T}\ ] \le 0$. Since this holds for all $\mathcal{T}$, we have that $\mathbf{E}[\, X_j\,] \le 0$ unconditionally, and thus

$$\mathbf{E}[\, \psi_j\,] \le \frac{\beta}{\sigma}\, \mathbf{E}[\, \phi_j\,]. \tag{4.9}$$

Using the fact that positive cost shares are only assigned to clients requiring service and that the cost-sharing function is competitive, we have

$$\sum_{j \in U} \mathbf{E}[\, \phi_j\,] = \sum_{j \in U} \mathbf{E}[\, \xi(X, D, j)\,] = \mathbf{E}\left[\sum_{j \in D} \xi(X, D, j)\right] \tag{4.10}$$

$$\le \mathbf{E}[\, \mathsf{OPT}(D)\,] \le Z^*. \tag{4.11}$$

Furthermore, $\mathbf{E}[\, c(F_2(S))\,] \le \sum_j \mathbf{E}[\, \psi_j\,]$ by subadditivity; using this, (4.9), and (4.10), we get that the expected second-stage cost $\sigma\mathbf{E}[\, c(F_2(S))\,] \le \beta\, Z^*$, thus proving the result. □

4.2. Transfer theorem for multistage problems. We now consider the case of multistage stochastic optimization problems. Recall the set-up and sequence of events from section 2.1 and Figure 2.1. Briefly, there are k stages labeled 1 through k;

Procedure Recur-Sample (Π, stage i, vector of signals s).

1. *(Base case)* If $i = k$, draw one sample set of clients S_k from the conditional distribution $\pi(.|s)$. Return the set S_k.
2. *(New samples)* If $i < k$, draw $\lfloor \sigma_{i+1} \rfloor$ samples of the signal $\mathbf{s}_{i+1}$ from the conditional distribution $p(.|s)$. Let $s^1, s^2, \ldots, s^n$ be the sampled signals, where $n = \lfloor \sigma_{i+1} \rfloor$.
3. *(Recursive calls)* For each sample signal s^j, recursively call Recur-Sample (Π, $i + 1, s + s^j$) to obtain a sample set of clients S^j.
4. Return the set $S_i = \cup_{j=1}^{n} S^j$.

FIG. 4.2. *The recursive sampling procedure for multistage stochastic optimization.*

the final client set is revealed only in stage k, while in each stage we get a signal s_i and elements are costlier by a factor σ_i compared to the previous stage. The conditional distribution $\pi(.|s)$ can be sampled from at any point, where s is the vector of signals received so far.

Once again, we show how to extend an algorithm for the deterministic problem Det(Π) to handle the multistage stochastic version of Π. As before, we require an α-approximation algorithm $\mathcal{A}$ with respect to a cost-sharing function ξ for the problem Det(Π). We also require that ξ be β-superstrict and cross-monotone. We also need an augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$; since ξ is superstrict, the augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ is the original algorithm $\mathcal{A}$, where the costs of elements already purchased is set to zero. Note that the requirements of superstrictness and cross-monotonicity are stronger than the requirements for two-stage problems.

Let us first outline the key idea of the algorithm, which is a natural extension of the two-stage algorithm. In the two-stage algorithm outlined in Figure 4.1, the first stage involved simulating $\sigma = \sigma_1$ independent runs of the second stage and building a solution that satisfied the union of these simulations. We use the same basic idea for the k-stage problem: in each stage i, we sample σ_i "copies" of the remaining $(k-i)$-stage stochastic process. Each copy provides us with a random set of clients to satisfy, and we build a solution that satisfies the union of all these clients. The "base case" of this recursive idea is the final stage, where we get a set S of clients and just build the required solution for S. The recursive sampling procedure is described in Figure 4.2.

Our algorithm, in each stage i (except the final, kth stage), uses the recursive sampling procedure described in Figure 4.2 to emulate σ_{i+1} copies of itself on the remaining $k - i$ stages. Having obtained a collection of sampled sets of clients, it then augments the current partial solution to a feasible solution for these sampled sets. Finally, in the kth stage it performs the ultimate augmentation to obtain a feasible solution for the revealed sets of demands. The expected number of calls to the sampling black box in stage i is $\prod_{j=i+1}^{k} \sigma_j$.

The new algorithm is called Multi-Boost-and-Sample and is described in Figure 4.3. Formally, in the first stage, we call Algorithm Multi-Boost-and-Sample (Det(Π), 1, $\emptyset, \emptyset, \emptyset$). This returns a set of elements F_1, which are purchased as the first-stage partial solution. In stage i, having already observed the signal-vector s so far and having purchased elements B_i to serve clients S_i, we call Algorithm Multi-Boost-and-Sample (Det(Π), i, s, B_i, S_i) and purchase the set of elements F_i returned by the algorithm. In

the final stage k, the algorithm automatically purchases a set of elements F_k generated by the cost-sharing algorithm $\mathcal{A}$ to serve any new unserved elements $S \setminus S_k$.

Algorithm Multi-Boost-and-Sample (Π, i, s, B_i, S_i).
Given: Problem Π, defined by universe of clients U, probability distribution π, universe of elements X, cost function $c()$, and inflation vector $\sigma = (\sigma_1 = 1, \sigma_2, \ldots, \sigma_k)$.
Stage i.
Set of signals s received so far, which allows the construction of the conditional distribution $\pi(.|s)$.
Set of elements B_i purchased so far.
Set of clients S_i served so far.
Input: If the current stage $i < k$, observe the signal s_i, and let $s = s + s_i$. If this is the final stage $i = k$, observe the required set of clients S instead.
1. If $i < k$, use Procedure Recur-Sample (Π, i, s) to obtain a sample set of clients D_i. Else if $i = k$, let $D_k = S$.
2. Set the cost of elements $e \in B_i$ to zero. Using algorithm $\mathcal{A}$, find a set of elements $F_i \subseteq X \setminus B_i$ to buy so that $F_i \in \mathsf{Sols}(D_i \setminus S_i)$.
3. Update $B_{i+1} = B_i \cup F_i$ and $S_{i+1} = S_i \cup D_i$.
Output: The i-stage solution F_i and set of new clients served $D_i \setminus S_i$.

FIG. 4.3. *Algorithm Multi-Boost-and-Sample for multistage stochastic optimization.*

As a side note, recall that in the two-stage model, we had distinct algorithms for the first and second stages ($\mathcal{A}$ and $\mathsf{Aug}_{\mathcal{A}}$, respectively). If, in the two-stage model, the augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ was identical to running $\mathcal{A}$ with the costs of the first-stage elements set to zero, then the resulting two-stage algorithm is identical to running Multi-Boost-and-Sample with $k = 2$.

4.2.1. Analysis of Multi-Boost-and-Sample for multistage problems. We will now show that Algorithm Multi-Boost-and-Sample can be used to translate an approximation algorithm $\mathcal{A}$ for the deterministic version $\mathsf{Det}(\Pi)$ of a problem Π into its k-stage stochastic version $\mathsf{Stoc_k}(\Pi)$. The quality of this translation depends on the approximation guarantee of $\mathcal{A}$ with respect to some superstrict cost-sharing function. The main result of this section is the following.

THEOREM 4.3 (k-stage general theorem). *Given a problem Π, if $\mathcal{A}$ is an α-approximation algorithm w.r.t. a β-superstrict competitive cross-monotone cost-sharing function ξ, then* Multi-Boost-and-Sample *is an $\alpha \cdot \sum_{i=0}^{k-1} \beta^i$-approximation algorithm for the k-stage stochastic problem $\mathsf{Stoc_k}(\Pi)$.*

Before we prove Theorem 4.3, we set the stage for the proof by providing a brief overview of the proof technique and proving a couple of lemmas which provide useful bounds. A naïve attempt to prove this result along the lines of Theorem 4.1 does not succeed, since we have to move back and forth between the cost-shares and the actual cost of the solutions F_i, which causes us to lose factors of $\approx \alpha$ at each step. Instead, in our proof below, we bound all costs incurred in terms of the ξ's: we first argue that the expected sum of cost-shares from the first stage is no more than the optimum total expected cost Z^*. We then bound the expected sum of cost-shares in each consecutive stage in terms of the expected cost-shares from the previous stage (with a loss of a factor of β at each stage). Finally, we bound the actual cost of the

partial solution constructed at stage i by α times the expected cost-shares for that stage, which gives us the geometric sum claimed in the theorem.

Thus, in the k-stage stochastic problem, we are placing a stronger requirement on the algorithm $\mathcal{A}$ by requiring it to be an α-approximation with respect to the cost-sharing function ξ; the two-stage algorithm required $\mathcal{A}$ to be an α-approximation, but not necessarily w.r.t. ξ. Indeed, the stronger requirement here leads to a weaker bound: running Multi-Boost-and-Sample with $k = 2$ yields an $\alpha(1+\beta)$-approximation, which is (weakly) worse than the $(\alpha + \beta)$-approximation obtained for the two-stage problem (Theorem 4.1).

Let F^* be an optimal solution to the given instance of $\mathsf{Stoc_k}(\Pi)$. We denote by F_i^* the partial solution built in stage i; recall that $F_i^* = F_i^*(s_1, s_2, \ldots, s_i)$ is a function of the set of all possible i-tuples of signals that could be observed before stage i. The expected cost of this solution can be expressed as

(4.12)
$$Z^* = \sigma_1 \mathbf{E}[\, c(F_1^*(\mathbf{s}_1))\,] + \sigma_1\sigma_2 \mathbf{E}[\, c(F_2^*(\mathbf{s}_1, \mathbf{s}_2))\,] + \cdots + \sigma_1 \cdots \sigma_n \mathbf{E}[\, c(F_k^*(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k))\,].$$

For the proof of this theorem, we will assume that all the inflation factors σ_i are positive integers.

The first ingredient of the proof is to show that the cost-shares from stage 1 are not too large; this is similar to the beginning of the proof of Theorem 4.1, except that we are interested in a bound on the cost-shares and not the actual cost of the solution.

LEMMA 4.4. *The expected cost-share* $\mathbf{E}[\, \xi(X, D_1, D_1)\,]$ *is at most the total optimum cost* Z^*.

Proof. Consider D_1, the sample set of clients returned by Recur-Sample $(\Pi, 1, s_1)$. We claim there is a solution $\widehat{F}(D_1)$ such that $\mathbf{E}[\, c(\widehat{F}(D_1))\,] \le Z^*$ (the expectation is over all sample paths in the execution of the procedure Recur-Sample). To construct the solution $\widehat{F}(D_1)$, we consider the tree of recursive calls of the procedure Recur-Sample. For each recursive call Recur-Sample (Π, i, s), we add the set of elements $F_i^*(s)$ to $\widehat{F}(D_1)$. Using subadditivity, we establish that (1) $\widehat{F}(D_1)$ is a feasible solution for the set D_1 and (2) the expected cost $\mathbf{E}[\, c(\widehat{F}(D_1))\,] \le \mathbf{E}[\, \sum_{i=1}^{k} \big(\prod_{j \le i} \sigma_j\big) c(F_i^*)\,] = Z^*$. Since the expected cost of a feasible solution for D_1 is bounded above by Z^*, the competitiveness of ξ implies that this bound must hold for the sum of cost-shares as well. □

We now show that the total cost-shares from the $(i+1)$th stage can be related to the cost shares from the ith stage.

LEMMA 4.5. *Let* $\widehat{F} = F_1 \cup \cdots \cup F_{i-1}$ *be the solution constructed in a particular execution of the first* $i-1$ *stages, and let* s_i *be the signal observed in stage* i. *Let* D_i *and* D_{i+1} *be the random variables denoting the samples returned by the procedure* Recur-Sample *in Stages* i *and* $i+1$, *and let* F_i *be the (random) solution constructed by* $\mathcal{A}$ *for the set of clients* $D_i \setminus S_i$, *where* S_i *is the set of clients served in stages prior to* i *(therefore,* $S_{i+1} = S_i \cup D_i$*). Then,*

(4.13)
$$\mathbf{E}[\, \xi(X/(\widehat{F} \cup F_i), D_{i+1} \setminus S_{i+1}, D_{i+1} \setminus S_{i+1})\,] \le \frac{\beta}{\sigma_{i+1}} \cdot \mathbf{E}[\, \xi(X/\widehat{F}, D_i \setminus S_i, D_i \setminus S_i)\,].$$

Proof. Recall that the sampling procedure Recur-Sample (Π, i, s) gets $\ell = \lfloor \sigma_{i+1} \rfloor$ independent samples $s^1, s^2, \ldots, s^\ell$ of the signal $\mathbf{s}_{i+1}$ from the distribution π conditioned on $s = s_1, \ldots, s_i$, and then for each sampled signal calls itself recursively to

obtain the ℓ sets $S^1, \ldots, S^\ell$. Note that the set $D_i = \bigcup_{j=1}^{\ell} S^j$ is simply the union of these ℓ sets. On the other hand, the set D_{i+1} is obtained by observing the signal s_{i+1} (which is assumed to come from the same distribution $\pi(.|s)$) and then calling Recur-Sample with the observed value of s_{i+1}.

We now consider an alternate, probabilistically equivalent, view of this process, very similar to the proof of Theorem 4.1. We first take $\ell+1$ samples $s^1, \ldots, s^{\ell+1}$ of the signal $\mathbf{s}_{i+1}$ from the distribution $\pi(.|s)$. Call the procedure Recur-Sample (Π, $i+1$, $s + s^j$) for each s^j to obtain sets $S^1, \ldots, S^{\ell+1}$. Pick an index j uniformly at random from the set of integers $1, \ldots, \ell+1$. Let $D_{i+1} = S^j$, and let D_i be the union of the remaining ℓ sets. This process of randomly constructing a pair of sets (D_i, D_{i+1}) is clearly equivalent to the original process. Note that $D_i \cup D_{i+1} = \bigcup_{l=1}^{\ell+1} S^l$.

To simplify notation, let us denote the set of elements $X/\widehat{F}$ by $\widehat{X}$. By the definition of β-superstrictness and the fact that F_i is the solution constructed for the set $D_i \setminus S_i$, we first get

$$\xi(\widehat{X}/F_i, D_{i+1} \setminus S_{i+1}, D_{i+1} \setminus S_{i+1}) \leq \beta \cdot \xi(\widehat{X}, (D_i \cup D_{i+1}) \setminus S_i, D_{i+1} \setminus S_{i+1}). \tag{4.14}$$

By definition, we have $\xi(\widehat{X}, (D_i \cup D_{i+1}) \setminus S_i, (D_i \cup D_{i+1}) \setminus S_i)) = \xi(\widehat{X}, (D_i \cup D_{i+1}) \setminus S_i, D_i \setminus S_i)) + \xi(\widehat{X}, (D_i \cup D_{i+1}) \setminus S_i, D_{i+1} \setminus S_{i+1})$; this equation also holds in expectation. Now we use the facts that the actual set of clients D_{i+1} has the same distribution as the set S^j which is sampled uniformly from $\ell+1$ choices in the equivalent process above, and that D_i is the union of the remaining ℓ sets. This gives us that

$$\begin{aligned} &\mathbf{E}[\, \xi(\widehat{X}, (D_i \cup D_{i+1}) \setminus S_i, D_{i+1} \setminus S_{i+1}) \,] \\ &\qquad \leq \mathbf{E}\left[\frac{1}{\ell+1} \times \xi(\widehat{X}, (D_i \cup D_{i+1}) \setminus S_i, (D_i \cup D_{i+1}) \setminus S_i) \right]. \end{aligned} \tag{4.15}$$

The two inequalities above yield the following:

$$\mathbf{E}[\, \xi(\widehat{X}, (D_i \cup D_{i+1}) \setminus S_i, D_{i+1} \setminus S_{i+1}) \,] \leq \mathbf{E}\left[\frac{1}{\ell} \times \xi(\widehat{X}, (D_i \cup D_{i+1}) \setminus S_i, D_i \setminus S_i) \right]. \tag{4.16}$$

Finally, we use cross-monotonicity of the cost-shares ξ, which says that the cost-share of clients in $D_i \setminus S_i$ does not increase when the elements of $D_{i+1} \setminus S_i$ join the fray. This implies that

$$\mathbf{E}[\, \xi(\widehat{X}, (D_i \cup D_{i+1}) \setminus S_i, D_i \setminus S_i) \,] \leq \mathbf{E}[\, \xi(\widehat{X}, D_i \setminus S_i, D_i \setminus S_i) \,]. \tag{4.17}$$

Chaining the above inequalities (4.14), (4.16), and (4.17) proves the lemma. ∎

Having proved the main supporting lemmas (that the cost-shares of the first stage are at most Z^*, and that the total cost-shares of each subsequent stage can be related to those from the previous stage), we now turn to bounding the approximation ratio for Multi-Boost-and-Sample.

Proof of Theorem 4.3. Recall that the expected cost of the solution given by Algorithm Multi-Boost-and-Sample (Π) is

$$\mathbf{E}[\, Z \,] = \mathbf{E}\left[\sum_{i=1}^{k} \left(\prod_{j=1}^{i} \sigma_j \right) c(F_i) \right]. \tag{4.18}$$

In the above expression, recall that the ith-stage solution $F_i = \mathcal{A}(X/B_i, D_i \setminus S_i)$, where $B_i = \cup_{j<i} F_j$ is the set of elements already purchased, D_i is the sampled set of demands, and S_i is the set of clients already served. Since $\mathcal{A}$ is an α-approximation algorithm w.r.t. the β-superstrict cost-sharing function ξ, we get that

$$c(F_i) \leq \alpha\, \xi(X/B_i, D_i \setminus S_i, D_i \setminus S_i). \tag{4.19}$$

Now using Lemma 4.5 inductively on $\xi(X/B_i, D_i \setminus S_i, D_i \setminus S_i)$, we find that

$$\mathbf{E}[\,\xi(X/B_i, D_i \setminus S_i, D_i \setminus S_i)\,] \leq \mathbf{E}\left[\frac{\beta^{i-1}}{\prod_{j=1}^{i} \sigma_j}\, \xi(X, D_1, D_1)\right]. \tag{4.20}$$

Using this inequality with (4.18) and (4.19) yields

$$\mathbf{E}[\,Z\,] \leq \alpha\, \mathbf{E}\left[\sum_{i=1}^{k} \beta^{i-1}\, \xi(X, D_1, D_1)\right]. \tag{4.21}$$

Lemma 4.4 bounds $\mathbf{E}[\,\xi(X, D_1, D_1)\,]$ from above by Z^*. Using this bound in inequality (4.21) above completes the proof of Theorem 4.3. ∎

4.3. Correlated inflation factors. In the model and the results proved above, we have assumed that the inflation factors are fixed constants specified in advance to the algorithm. This is a fairly restrictive assumption in general, since the inflation parameters are very likely to be correlated with the set of clients that materializes in the second stage. Indeed, one may imagine that a large set of clients may be related to a booming economy and thus to higher costs, etc. In this section, we show that the Boost-and-Sample algorithm for the two-stage problem can easily be extended to situations where the inflation factor is correlated to the client set.

4.3.1. The model with correlated costs. To model correlated costs, we assume that our probability distribution $\pi : \mathbb{R}_{\geq 1} \times 2^V \to [0, 1]$ is not merely over client sets but over tuples *(inflation, client set)*; here $\pi(\sigma, S)$ is the probability that the client set S arrives with an inflation factor of σ. Again, we assume that we are given the distribution π as a black box from which we can draw independent samples. Moreover, we need one more assumption: we need an upper bound M on the value of the random variable $\boldsymbol{\sigma}$. This need not be a *tight* upper bound: choosing a pessimistically high value of M will only increase the running time of the algorithm but will not affect the approximation ratio.

A naïve attempt to extend Algorithm Boost-and-Sample in the two-stage case would be to obtain $\mathbf{E}[\boldsymbol{\sigma}]$ samples to compute the first-stage solution. Unfortunately, this is clearly a poor algorithm, as the following easy example shows. Consider the case where with probability 1/2, inflation factor $\sigma = M \gg 1$, and $S = \emptyset$, and there are several other nonempty client sets S, each with $\sigma \approx 1$ and whose probabilities of realization sum to 1/2. The expected value of $\boldsymbol{\sigma}$ is approximately $M/2$, and hence Algorithm Boost-and-Sample will almost surely obtain a large number of nonempty samples and construct an expensive first-stage solution, whereas the optimal solution is to defer purchasing elements to the second stage.

4.3.2. The algorithm for correlated costs. The algorithm Boost-and-Sample-Correlated with correlated inflation factors is presented in Figure 4.4. Loosely, the algorithm takes a suitably large number of samples of (inflation factor, client set)

Algorithm Boost-and-Sample-Correlated (Π).

Given: Problem Π, defined by universe of clients U, probability distribution π', universe of elements X, cost function $c()$, and upper bound M on inflation factors.

1. Draw M independent samples from the joint distribution π' of $(\boldsymbol{\sigma}, \mathbf{S})$. Let $(\sigma_1, S_1), (\sigma_2, S_2), \ldots, (\sigma_M, S_M)$ denote this collection of samples.
2. *(Rejection sampling)* For $i = 1, \ldots, M$, *accept* sample S_i with probability σ_i/M. Let $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ be the accepted samples, and let $D = \cup_j S_{i_j}$.
3. Using the algorithm $\mathcal{A}$, construct a first-stage solution $F_1 \in \mathsf{Sols}(D)$. Algorithm $\mathcal{A}$ must be an α-approximation algorithm which includes a β-strict cost-sharing function ξ and augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$.

Output: The first-stage solution F_1 and the set of clients served D.

4. The second stage occurs, and the client set S is realized. Using the augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$, compute the second-stage solution F_2 such that $F_2 \in \mathsf{Sols}(S \setminus D)$.

Output: The second-stage solution F_2.

FIG. 4.4. *Algorithm Boost-and-Sample-Correlated for two-stage stochastic optimization with random inflation factors.*

tuples, but rejects samples having smaller inflation factors with a larger probability. To get some intuition for the new step of rejection sampling, observe that if we sample a pair (σ_i, S_i) where the sampled inflation factor σ_i is large, the cost of waiting to serve the set S_i in the second stage is higher, and we ought to handle the associated S_i in the first stage—and indeed, the sample is more likely to be accepted. On the other hand, if the inflation factor σ_i is low, there is smaller penalty for waiting until the second stage, and thus we reject the sample with a larger probability.

Note that Algorithm Boost-and-Sample-Correlated is identical to Algorithm Boost-and-Sample if the inflation factor is deterministic (i.e., if the random variable $\boldsymbol{\sigma}$ takes on value σ with probability 1). Observe also that the expected number of times a scenario (σ, S) is sampled is $\sigma\pi'_{\sigma,S}$, which is the factor that multiplies the second-stage cost incurred if this scenario occurs. This mirrors Algorithm Boost-and-Sample, and therefore a similar analysis allows us to bound the cost of the solution.

In order to obtain an approximation algorithm for the two-stage stochastic version of our routine, the requirements are the same as in Algorithm Boost-and-Sample: an α-approximation algorithm with a β-strict cost-sharing function and an augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$. The following theorem shows that the rejection probabilities chosen for the algorithm are the correct quantitative choices, and establishes the main result with correlated inflation factors.

THEOREM 4.6 (two-stage general theorem with correlated costs). *Consider a subadditive combinatorial optimization problem* Π. *Let* $\mathcal{A}$ *be a cost-sharing algorithm for* Π *such that the following hold:* (i) $\mathcal{A}$ *is an* α*-approximation algorithm for* $\mathsf{Det}(\Pi)$*;* (ii) $\mathcal{A}$ *is associated with a* β*-strict competitive cost-sharing function* ξ*; and* (iii) $(\mathcal{A}, \xi)$ *is associated with an augmenting algorithm* $\mathsf{Aug}_{\mathcal{A}}$. *Then Algorithm* Boost-and-Sample-Correlated *is an* $(\alpha + \beta)$*-approximation algorithm for the two-stage problem*

Stoc(Π).

Proof. Let us transform the "random inflation" stochastic problem instance (X, π') to one with a fixed inflation factor as follows: the distribution $\hat{\pi}(\sigma, S) = \pi'(\sigma, S) \times (\sigma/M)$; note that this ensures that $\sum_{\sigma,S} \hat{\pi}(\sigma, S) \leq 1$, and hence we can increase the probability $\hat{\pi}(1, \emptyset)$ so that the sum becomes exactly 1 and $\hat{\pi}$ is a well-defined probability distribution. The inflation factor for this new instance is set to M, and hence the σ output by $\hat{\pi}$ is only for expositional ease.

Now the objective function for this new problem is to minimize the expected cost under this new distribution, which is $c(F_1) + \sum_{\sigma,S} \hat{\pi}(\sigma, S) M c(F_{\sigma,S}) = c(F_1, D) + \sum_{\sigma,S} \pi'(\sigma, S)(\sigma/M) M c(F_{\sigma,S})$, which is the same as the original objective function. Hence the two problems are identical, and running Boost-and-Sample on the new distribution $\hat{\pi}$ with inflation parameter M would give us an $(\alpha + \beta)$-approximation using Theorem 4.1.

Finally, note that one can implement the distribution $\hat{\pi}$, given sampling access to the distribution π', by just rejecting any sample (σ, S) with probability σ/M. This is precisely what is done in the rejection sampling stage of Algorithm Boost-and-Sample-Correlated and is the link between this algorithm and Algorithm Boost-and-Sample. This completes the proof of the theorem. □

While allowing the inflation factor $\boldsymbol{\sigma}$ to be a random variable generalizes our model somewhat, we still require that the *relative* costs of elements remain the same in every scenario. A more general model would be one where the costs of individual elements change arbitrarily in different scenarios, and costs of elements in different scenarios are not necessarily proportional. Naturally, this creates some additional complications, in some cases making the problem much harder; e.g., one can show that the Steiner tree problem with nonproportional inflation parameters is at least as hard to approximate as the group Steiner tree problem [39, Theorem 2].

In principle, it ought to be possible to construct an algorithm using the ingredients in this paper for a multistage stochastic problem with correlated costs. The key would be to construct a sampling procedure that balances the probabilities of scenarios with their corresponding cost inflation factors, as is done in the two-stage case with the rejection sampling step. Whether or not this can be done, and what conditions may be required for it, is left open for future research.

5. Strictness theorems. We now show the applicability of our framework by considering some concrete combinatorial optimization problems. In this section, we consider three problems: Steiner tree, Steiner network, and vertex cover. For each problem, we consider various stochastic models (e.g., two-stage, multistage, independent-decisions, etc.) and show the existence of appropriate approximation algorithms and cost-sharing functions that allow us to use the results of the previous section to obtain approximation algorithms for the stochastic versions.

In the next section, we also consider the uncapacitated facility location problem. As discussed there, the problem does not fit naturally into our model of section 2. Nevertheless, the main ideas of our algorithm can be applied, and we analyze this in detail in section 6.

5.1. Steiner tree. In the classical (deterministic) Steiner tree problem, we are given a complete graph $G = (V, E)$, where the edge costs c_e satisfy the triangle inequality. (This assumption is without loss of generality, since we can take the metric completion of the graph instead.) We are also given a set of *terminals* $R \subseteq V$; we say that a set of edges E' *spans* R if all the terminals in R lie in the same connected component of E'. The goal of the Steiner tree problem is to find a minimum cost set

of edges that spans R. Let us cast this problem in our framework: the universe of clients U is simply the vertex set V, while the set of elements X which provide service is the edge set E. A client set S now corresponds to a set R of terminals, since the terminals are those vertices which need to be connected. The set $\mathsf{Sols}(S)$ is simply the set of all subgraphs which span S (or R).

Observe that, as specified above, the Steiner tree problem is *not subadditive*. Indeed, consider two disjoint sets of terminals R_1 and R_2: while their union $R_1 \cup R_2$ is a valid set of terminals, the union of two Steiner trees T_1 and T_2 (which span R_1 and R_2, respectively) may not form a single connected component and hence may not be a solution for the union of the terminals. This violates condition (SA2) of Definition 2.1. In order to make the problem subadditive, we restrict ourselves to the *rooted* version of the Steiner tree problem. In this version, a vertex $r \in V$ is designated to be the *root*, and the set of terminals R is restricted to those subsets of V which include r. It is easy to see that this makes the problem subadditive. In the deterministic version of the problem, the rooted Steiner tree problem is identical to the unrooted one, but the same is not true for the stochastic version of the problem. And indeed, the subadditivity allows us to use our framework to obtain approximation algorithms for the stochastic versions of the Steiner tree problem.

Formally, the deterministic Steiner tree problem $\mathsf{Det}(\mathsf{ST})$ that we consider is defined according to the specification in section 2 as follows. We are given a complete graph $G = (V, E)$, a designated root vertex $r \in V$, and an edge cost function c.

- Universe $U := V \setminus \{r\}$.
- Element set $X := E$.
- Cost function $c(e) = c_e$.
- Given client set S, the set of feasible solutions $\mathsf{Sols}(S)$ is the set of all subgraphs that contain a path between each vertex in S and r.

5.1.1. Two-stage problem: Strictness theorem. The two-stage stochastic optimization version of the Steiner tree problem is described fairly simply. In the first stage, given the graph $G = (V, E)$, the edge costs c_e, the root $r \in V$, and the inflation factor σ, we want to purchase some set of edges F_1 and choose to serve the set D of all nodes which have a path in F_1 from the root r. Then in the second stage, the actual set of terminals $S \subseteq V$ is revealed. We now need to purchase a (possibly empty) set of additional edges F_2 such that $F_1 \cup F_2$ spans the set of vertices $S \cup \{r\}$. However, the cost of purchasing an edge e in the second stage (i.e., as a part of F_2) is σc_e. Hence, the problem is to compute a set of edges F_1 to purchase in the first stage which minimizes the expected total cost, given by $c(F_1) + \sigma \mathbf{E}[\, c(F_2)\,]$. As noted in section 2.1, the signal plays no role in a two-stage model and is ignored.

In order to approximate the two-stage stochastic Steiner tree problem using Theorem 4.1, we need to specify a first-stage algorithm $\mathcal{A}$, a cost-sharing function ξ, and an associated augmentation algorithm $\mathsf{Aug}_{\mathcal{A}}$. These are specified below, following which we prove the competitiveness and strictness of the cost-sharing function.

The algorithm $\mathcal{A}$: The approximation algorithm $\mathcal{A}$ for Steiner tree that we use is the minimum spanning tree (MST) heuristic; given a set of terminals D, it ignores the vertices not in $D \cup \{r\}$ and builds a minimum spanning tree on $D \cup \{r\}$. (We can use any MST algorithm here, e.g., Prim's algorithm [38].) It is well known that the cost $c(\mathcal{A}(D \cup \{r\}))$ of any MST is within a factor of 2 of the cost of the optimal Steiner tree $\mathsf{OPT}(D \cup \{r\})$ [47]; hence $\mathcal{A}$ is an approximation algorithm with $\alpha = 2$.

The cost-sharing function ξ: Given an MST $\mathcal{A}(D)$ on the set of terminals $D \cup \{r\}$, root it at r and define the *parent edge* of any vertex $j \in D$ (where $j \neq r$) to be the edge incident on j lying on the unique simple path from j to r in the tree $\mathcal{A}(D)$. We denote the parent edge of $j \neq r$ by $p(j)$ and define $\xi_{Prim}(E, D, j) = \frac{1}{2}c(p(j))$: the cost-share of vertex j is half the cost of its parent edge in the tree $\mathcal{A}(D)$. The cost-share of the root r is $\xi_{Prim}(E, D, r) = 0$. (These cost-shares are often referred to as *Prim cost-shares*.)

The augmentation algorithm $\mathsf{Aug}_{\mathcal{A}}$: The augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ merely zeros out the cost of the edges in $\mathcal{A}(D \cup \{r\})$ (essentially contracting them) and runs the MST heuristic on the resulting graph.

LEMMA 5.1. *The cost-shares ξ_{Prim} are competitive and are 2-strict for the Steiner tree algorithm used in the augmentation, $\mathsf{Aug}_{\mathcal{A}}$.*

Proof. By definition, $\sum_{j \in D} \xi_{Prim}(E, D, j) = \frac{1}{2}c(\mathcal{A}(D))$; since the MST is a 2-approximation to the Steiner tree problem, we have $\frac{1}{2}c(\mathcal{A}(D)) \leq c(\mathsf{OPT}(D))$, proving the competitiveness of ξ_{Prim}.

To prove 2-strictness, consider two disjoint sets of terminals S and T, and let $D' = S \cup T$. If $p'(j)$ is the parent edge of node $j \in D'$ in the minimum spanning tree $\mathrm{MST}(D' \cup \{r\})$, then recall that $\xi_{Prim}(E, D', T) = \sum_{j \in T} c(p'(j))$. Note that if we consider the graph after we collapse the nodes of S to r, then adding the parent edges in $\cup_{j \in T} p'(j)$ would create a spanning tree for the set $T \cup \{r\}$ in this contracted graph. However, we defined the augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ to find a spanning tree *of minimum cost* in this contracted graph, and hence the cost of the augmentation is at most the cost of $\cup_{j \in T} p'(j)$. This is at most $2 \sum_{j \in T} \xi_{Prim}(E, D', j)$, which proves the 2-strictness. □

THEOREM 5.2. *The two-stage stochastic Steiner tree problem can be approximated to a factor of* 4.

Proof. Lemma 5.1 shows that the cost-sharing function is 2-strict and is associated with the first-stage algorithm $\mathcal{A}$ and augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$. Furthermore, the discussion preceding the theorem indicates that $\mathcal{A}$ is a 2-approximation for the problem. Hence, Theorem 4.1 implies that Algorithm Boost-and-Sample results in a 4-approximation for the two-stage stochastic Steiner tree problem. □

The best approximation for the Steiner tree problem is a 1.55 approximation due to Robins and Zelikovsky [40]; setting $\mathcal{A}$ to be this algorithm results in the following corollary.

COROLLARY 5.3. *There is a 1.55-approximation algorithm $\mathcal{A}$ for the Steiner tree problem along with a cost-sharing function ξ_{Prim} that is 2-strict with respect to $\mathcal{A}$. Hence, there is a 3.55-approximation algorithm for the two-stage stochastic version of the Steiner tree problem.*

Clearly, the above corollary also implies a 3.55-approximation to the problem Ind(Steiner tree) with independent coin-tosses. We do not know whether we can obtain a better approximation for this case by finding a *unistrict* cost-sharing function with strictness better than 2. (For some problems such as vertex cover, discussed in section 5.3, we will indeed be able to show unistrict cost-sharing functions with strictness strictly better than for the general strict cost-sharing functions, and hence will obtain tighter guarantees for the independent coin-flips version of such problems.)

5.1.2. Multistage stochastic Steiner tree: Using superstrictness. We now go on to consider a multistage stochastic version of the Steiner tree problem. Recall the k-stage stochastic model described in section 2.1 (as well as briefly in section 4.2). Initially, only the graph G and edge costs $c : E \to \mathbb{R}$ are known,

along with a probability distribution π about the eventual set of terminals S. This probability distribution is specified as a black box which we will use to generate our samples. In each stage, we observe a signal which refines the probability distribution of the terminal set. Furthermore, in each stage, we can purchase some edges. The cost of purchasing edges in stage i is $c(e)\Pi_{j=1}^{i}\sigma_j$, where the multipliers σ_j are also known in advance.

Recall the Multi-Boost-and-Sample algorithm described in Figure 4.3, which works as follows. In each stage $i = 1, 2, \ldots, k-1$, we generate a sample set of terminals D_i by calling the recursive sampling procedure Recur-Sample, described in Figure 4.2. Note that Recur-Sample must also be fed the vector of all signals received so far $(s_1, s_2, \ldots, s_i)$. Once we have the set of terminals D_i, we use the algorithm $\mathcal{A}$ to extend the partial solution constructed so far (if any) to also serve the clients in D_i. Once the final set of terminals D_k is revealed (in stage k), we again use $\mathcal{A}$ to extend the partial solution to cover any uncovered elements in D_k.

Observe that in multistage problems the algorithms used in all stages are the same: there is no distinction between an initial algorithm $\mathcal{A}$ and an augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$. Indeed, $\mathsf{Aug}_{\mathcal{A}}$ is obtained by contracting the edges bought in the previous stages (which is equivalent to zeroing out the costs of these edges) and running $\mathcal{A}$ on the resulting instance. Hence the cost-sharing function ξ (as defined in Definition 3.8) and the transfer theorem (Theorem 4.3) are in terms of a single algorithm $\mathcal{A}$ and the superstrictness of this cost-sharing function ξ.

It remains to specify an algorithm $\mathcal{A}$ and an associated cost-sharing function ξ that meet the requirements of Theorem 4.3. For the case of the Steiner tree problem, we can indeed find a cost-sharing function which is provably superstrict and cross-monotone, so that we can use Theorem 4.3 to obtain our approximation guarantee.

The algorithm $\mathcal{A}$: Once again, the approximation algorithm $\mathcal{A}$ that we use is the MST heuristic. As discussed in section 5.1.1, this is a 2-approximation algorithm for the Steiner tree problem. In fact, the MST heuristic returns a solution that is a 2-approximation w.r.t. the cost-sharing function defined below, as shown in [25], a fact we need in order to apply Theorem 4.3.

The cost-sharing function ξ: Since it is known that the Prim cost-shares ξ_{Prim} used in section 5.1.1 are not cross-monotone, we have to use a more sophisticated construction than that in section 5.1.1. The cost-sharing function we use in this section is one given by Jain and Vazirani [25]. Briefly, it can be described as follows. The underlying algorithm to generate the cost-sharing functions is a primal-dual algorithm for Steiner tree [1, 14], which we only sketch here for brevity. Given a set of terminals D, one may view the primal-dual algorithm as a continuous-time process which grows dual "moats" around each node in $D \cup \{r\}$ (as is indeed done in [1, 14, 25]); as these moats collide, they merge to form a new moat that keeps growing. Let t_j denote the time at which vertex j lies in the same moat as the root. At any time $t < t_j$, let $n_j(t)$ be the number of vertices sharing a moat with j. The cost-sharing function is then defined as follows: $\xi(E, D, j) = \int_0^{t_j} \frac{1}{n_j(t)} dt$. This dual process has an associated primal process as well, which builds a forest: it maintains the invariant that the terminals in each moat lie in a single connected component, and hence whenever two moats collide and merge, the primal process buys edges to connect the trees associated with these moats. Interestingly enough, this primal process when run on a set D of terminals, along with the root r, terminates with the MST on the set $D \cup \{r\}$, provided the edge lengths satisfy the triangle inequality.

In the moat-growing process above observe that, at any time t that is at most the stopping time of vertex j in the run with a larger terminal set, the moat of j contains more terminals than in the run with a smaller terminal set. Therefore, increasing the terminal set would only cause j to share a moat with the root sooner, and cross-monotonicity of the cost-sharing function follows. Moreover, one can view the moats as feasible duals for an LP relaxation of the Steiner tree problem, which proves that the cost-shares are competitive. There is one small difference between our moat-growing process and that in [25]: we grow a moat around the root node as well, whereas this is not necessary in [25]. Nevertheless, the following result from [25] holds.

LEMMA 5.4 (Jain and Vazirani [25, Theorem 3 and Corollary 4]). *The cost-sharing function ξ is cross-monotone and competitive.*

It remains to verify the superstrictness of ξ.

LEMMA 5.5. *The cost-sharing function ξ defined above is* 1*-superstrict w.r.t. the algorithm $\mathcal{A}$.*

Proof. In order to verify the 1-superstrictness, we consider two sets of clients S and T and compare two different values: the cost-shares $\xi(E, S \cup T, T \setminus S)$ allocated to $T \setminus S$ in the original graph (V, E) when both S and T need to be connected (we call this run R1 of the algorithm), and the cost-shares $\xi(E/\mathcal{A}(S), T \setminus S, T \setminus S)$ allocated to T when a solution to the terminals in S has already been built and can be used for no extra charge (run R2). It will be most convenient to view all the edges of $\mathcal{A}(S)$ as having been contracted, and hence all the vertices in $\mathcal{A}(S)$ are co-located with the root.

To compare these two values, consider any terminal $j \in T \setminus S$, and let t_j^a and t_j^b be the times when j belongs to the root's moat in R1 and R2, respectively. First, observe that $t_j^a \geq t_j^b$; i.e., the t_j value in R2 is no more than in R1, for the following reason. In R2, a tree connecting S to the root has already been built, and t_j^b records the time when j collides with a moat containing the root via some node in $\mathcal{A}(S)$. In contrast, t_j^a records the time a moat containing j collides with a moat containing $\{r\}$, possibly by first sharing moats with some nodes in S which then collide with the root's moat.

Second, we claim that for any $t < t_j^b$ the value of $n_j(t)$ is equal in both executions: this is because for such time t the terminals in j's moat consist only of terminals in $T \setminus S$, and hence j's moat in both runs is the same at time t. From the definition of ξ, it now follows that $\xi(E/\mathcal{A}(S), T \setminus S, j) \leq \xi(E, S \cup T, j)$ for any $j \in T \setminus S$. Summing over all $j \in T \setminus S$ yields the lemma. ∎

This allows us to prove the following theorem about the k-stage stochastic optimization version of the Steiner tree problem.

THEOREM 5.6. *Algorithm* Multi-Boost-and-Sample *is a $2k$-approximation algorithm for the k-stage stochastic optimization version of the Steiner tree problem.*

Proof. This follows from Lemmas 5.4 and 5.5 and Theorem 4.3. ∎

5.2. Steiner network. The Steiner network problem is a generalization of the Steiner tree problem and is defined over an edge-weighted graph as in section 5.1. Formally, given a graph $G = (V, E)$ with edge weights c_e, the deterministic problem Det(SN) is defined as follows:

- Universe $U := \{V' \subseteq V : |V'| = 2\}$.
- Element set $X := E$.
- Cost function $c(e) = c_e$.
- Given client set S, the set of feasible solutions Sols(S) is the set of all sub-

graphs such that there is a path between the two vertices in every vertex-pair in S.

The problem is easily verified to be subadditive. The following result is due to Gupta et al. [16] and allows us to use Theorem 4.2 for the two-stage independent coin-flips stochastic version of the problem.

THEOREM 5.7 (see [16]). *There is a 4-approximation algorithm for the Steiner network problem which admits a 4-unistrict cost-sharing function.*

THEOREM 5.8. *The two-stage stochastic Steiner network problem under the independent-decisions model can be approximated to within a factor of 8.*

Proof. This follows from Theorems 5.7 and 4.2 above. □

5.3. Vertex cover. In the vertex cover problem, we are given a graph $G = (V, E)$ with costs c_v on vertices. The clients are the edges, and our goal is to choose a subset V' of the vertices so that each edge is covered; i.e., at least one of its adjacent vertices is chosen. Formally, the deterministic vertex cover problem $\mathsf{Det}(\mathsf{VC})$ is defined as follows, given a graph $G = (V, E)$ and a vertex cost function c:

- Universe $U := E$.
- Element set $X := V$.
- Cost function $c(v) = c_v$.
- Given client set S, the set of feasible solutions $\mathsf{Sols}(S)$ is the set of all vertex subsets V' such that at least one end-point of each edge in S is included in V'.

In the stochastic version, as before, the actual set of clients that need coverage is revealed only in the last stage, while vertex costs increase by σ_i in stage i. We will show in this section that the vertex cover problem has a 2-approximation algorithm with respect to an associated 2-strict cost-sharing function ξ. We will also show that the cost-sharing function is 3-superstrict, thus allowing us to approximate the multistage stochastic version of the problem as well.

We do this by considering a different version of $\mathsf{Stoc}(\mathrm{VC})$, which we call the *relaxed stochastic vertex cover* problem. In this relaxed version, we are allowed to make payments to a vertex in all the stages. In particular, the k-stage relaxed $\mathsf{Stoc}(\mathrm{VC})$ problem is defined as follows. The universe, element set, and cost function remain the same as in $\mathsf{Det}(\mathsf{VC})$ defined above. There is also a payment vector $p : 1, 2, \ldots, k \times V \to \mathbb{R}^+$, where $p^i(v)$ specifies the payment made to vertex v in stage i. At any stage j, a vertex v is considered "purchased" if the inflation-adjusted partial payments made to vertex v so far equal (or exceed) its cost: $\sum_{i=1}^{j} p^i(v)/(\prod_{i=1}^{j} \sigma_i) \geq c(v)$. The algorithm is thus required to define the payment vector p^j, once stage j is realized. After the last-stage payment vector p^k has been defined, we can construct the set of purchased vertices V_k; this is required to be a feasible vertex cover for the final client set S_k. Note that we are relaxing the way the k-stage problem is defined; such a relaxation is valid for any problem $\mathsf{Det}(\Pi)$ considered in our formulation.

In this analysis, we restrict our attention to a two-stage model, since that is sufficient for defining the cost-sharing function. As stated above, let $p^1(v)$ and $p^2(v)$ be the payments made in the first and second stages, respectively, and as in the two-stage models, we use σ to represent the factor by which costs are inflated in the second stage. Now, vertex v is chosen (purchased) if and only if $p^1(v) + p^2(v)/\sigma \geq c_v$. Again, given a set of realized edges S, the set of chosen vertices must form a feasible vertex cover for S. The cost of our solution in this two-stage model is defined to be just the sum of payments, i.e., $\sum_{v \in V} p^1(v) + p^2(v)$, and the goal is to minimize the expected cost.

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

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

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

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

Clearly, the algorithm $\mathcal{A}$ is a 2-approximation for the vertex cover problem w.r.t. the cost-sharing function ξ. We will prove below that ξ is a 2-strict cost-sharing function. Let T be another subset of edges, and we specify below how $\mathcal{A}(X, S)$ can be augmented using $\mathsf{Aug}_{\mathcal{A}}$ with F_T so that $\mathcal{A}(X, S) \cup F_T \in \mathsf{Sols}(S \cup T)$.

The augmentation algorithm $\mathsf{Aug}_{\mathcal{A}}$: Define the reduced cost of each vertex v as $c'_v = c_v - p(v)$. First run the original algorithm $\mathcal{A}$ on the edge set $S \cup T$ with costs $\{c_v\}$; let $p^1(v)$ be the payment made to vertex v in this run. Define $\tilde{p}(v) = \max\{0, p^1(v) - p(v)\}$. Purchase all vertices where $\tilde{p}(v) = c'_v$. Note that a vertex is considered purchased if the payment made in this stage, $\tilde{p}(v)$, is sufficient to pay for the reduced cost of the vertex, c'_v. Therefore, the total cost of the augmentation stage is defined as $c(F_T) = \sum_{v \in F_T} \tilde{p}(v)$.

LEMMA 5.9. *The augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ described above produces a feasible solution for the second stage in the relaxed vertex cover problem.*

Proof. This follows from the observation that for any vertex $v \in V$ we have $c_v - p(v) - \tilde{p}(v) \leq c_v - p^1(v)$, and since p^1 is feasible for the instance with edge set $S \cup T$ and costs c, we also have p^1 feasible for the edge set T with costs c. ∎

In order to prove strictness with $\beta = 2$, we need to show that $c(F_T) \leq 2\xi(V, S \cup T, T)$. To this end, we compare several runs of $\mathcal{A}$ on different related inputs:

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

function computed by $\mathcal{A}$. Furthermore, this is the run that computes the cost-shares $\xi(V, S \cup T, T)$, wherein $\xi(V, S \cup T, e) = y^1_e$ for each $e \in T$.

- **Run** R_S: The run R_S is the run of $\mathcal{A}$ on the set of edges S, but with costs $c^S = c - p^1_T$ (i.e., reduced by the payments of T in R_1). The corresponding duals and payments are labeled y^S_e and $p^S(v)$, respectively.
- **Run** R_T: Similar to R_S, the run R_T is on the edges T, with reduced costs $c^T = c - p^1_S$. The corresponding duals and payments are labeled y^T_e and $p^T(v)$, respectively.
- **Run** R_2: This is the run of Algorithm $\mathcal{A}$ on the edge set S, with original costs c, and hence corresponds to the actual run of the first stage. Let y^2_e be the duals and $p^2(v)$ the payments computed.
- **Run** R_3: This is the run of Algorithm $\mathcal{A}$ on the edge set T, with reduced costs $c^3 = c - p^2$. The corresponding payments are labeled p^3; this run corresponds to the actual augmentation after run R_2 constructs a first-stage solution.

By the definition of R_S, the freezing time of all edges $e \in S$ in the two runs R_S and R_1 is the same; hence the dual y^S is just the dual y^1 restricted to the set S, and $p^1_S = p^S$. Similarly, the dual y^T from the run R_T is identical to the dual y^1 restricted to T, and $p^1_T = p^T$. Before we prove the 2-strictness of ξ, we prove a technical lemma to aid in the analysis.

LEMMA 5.10 (Lipschitz continuity). *Consider two runs R and $\hat{R}$ of $\mathcal{A}$ with the same edge set S on two different cost vectors c and $\hat{c}$, and let p and $\hat{p}$ be the two vectors of payments computed. If we define Δ so that $(p - \hat{p}) = (c - \hat{c}) + \Delta$, then $||\Delta||_1 \le ||c - \hat{c}||_1$.*

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

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

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

LEMMA 5.11. *If F_T is the second-stage solution produced by $\mathsf{Aug}_{\mathcal{A}}$ as described above and ξ is the cost-sharing function defined by R_1, then $c(F_T) \le 2\xi(V, S \cup T, T)$.*

Proof. We begin by comparing the runs R_S and R_2 described above: define Δ_1 so that $p^2 - p^1_S = c - (c - p^1_T) + \Delta_1 = p^1_T + \Delta_1$. Lemma 5.10 now implies that $||\Delta_1||_1 \le ||p^1_T||_1$. Now observe that $c(F_T) = ||\tilde{p}||_1 \le ||\Delta_1||_1$. According to the description of the run R_1 above, we also have $||p^1_T||_1 = 2\xi(V, S \cup T, T)$. The lemma follows by combining these three inequalities. ∎

We now have our main result for the relaxed vertex cover problem.

THEOREM 5.12. *The relaxed vertex cover problem admits a* 2*-approximation algorithm w.r.t. an associated* 2*-strict cost-sharing function.*

Proof. The primal-dual algorithm is well known to be a 2-approximation [47], and Lemma 5.11 proves the 2-strictness of the cost-sharing function ξ described in Run R_1. □

THEOREM 5.13. *The two-stage stochastic vertex cover problem can be approximated to within a factor of* 4*.*

Proof. This follows from Theorems 5.12 and 4.1 above. □

In order to obtain an approximation guarantee for the multistage stochastic version of the vertex cover problem via Theorem 4.3, we need to show that the cost-sharing function is superstrict and cross-monotone. We are able to prove superstrictness below. Unfortunately, as [24] showed, there does not exist a cost-sharing function for the vertex cover problem that is cross-monotone, competitive, and recovers more than $O(n^{-1/3})$ of the cost, thus rendering Theorem 4.3 inapplicable.

In [19], it was claimed that the k-stage problem can be approximated even in the absence of cross-monotone cost-sharing functions. As stated in the Acknowledgments section of this paper, that result is incorrect. At this point, it is an open question whether superstrict cost-sharing functions which are not cross-monotone can be used to approximate general multistage stochastic problems under our model, or even to approximate just the multistage version of vertex cover. Nevertheless, although approximating multistage stochastic vertex cover remains open, we provide the lemma proving superstrictness below because it may be of independent interest and also perhaps helpful in settling the question of approximability of multistage stochastic vertex cover.

LEMMA 5.14. *The cost-sharing function ξ is* 3*-superstrict.*

Proof. We prove superstrictness by comparing runs R_3 and R_T. Observe that the payments in these two runs are p^3 and p^1_T, respectively, while the costs are $c - p^2$ and $c - p^1_S$. Define Δ_2 to be such that $p^3 - p^1_T = (c - p^2) - (c - p^1_S) + \Delta_2$. As in Lemma 5.11, we have $p^1_S - p^2 = -(p^1_T + \Delta_1)$, and this yields

$$p^3 - p^1_T = (-p^1_T + \Delta_1) + \Delta_2. \tag{5.1}$$

Using Lemma 5.10, we now have $||\Delta_2||_1 \le ||p^1_T + \Delta_1||_1 \le ||p^1_T||_1 + ||\Delta_1||_1 \le 2||p^1_T||_1$, where the last inequality was established in Lemma 5.11. Equation (5.1) also implies that $p^3 = \Delta_1 + \Delta_2$, so that $||p^3||_1 \le ||\Delta_1||_1 + ||\Delta_2||_1 \le 3||p^1_T||$. Observe now that $\xi(X/\mathcal{A}(X,S),T,T) = \frac{1}{2}||p^3||_1$ and $\xi(X, S \cup T, T) = \frac{1}{2}||p^1_T||_1$, and the lemma is proved. □

5.3.1. Unistrict cost-sharing function for vertex cover. The same constructions and algorithms also provide a stronger unistrict cost-sharing function. Once again, we consider relaxed vertex cover without loss of generality.

THEOREM 5.15. *There is a* 2*-approximation algorithm $\mathcal{A}$ for relaxed vertex cover that admits a* 1*-unistrict cost-sharing function ξ. Hence, the two-stage stochastic vertex cover problem under the independent-decisions model admits a* 3*-approximation.*

Proof. The algorithm $\mathcal{A}$, as well as the cost-shares ξ, are the same as for Theorem 5.12. To augment a solution $\mathcal{A}(S)$ on the addition of the edge $e = (u, v)$, the augmentation procedure $\mathsf{Aug}_{\mathcal{A}}$ opens the end-point whose reduced cost is less. That is, if the payments in $\mathcal{A}(S)$ are denoted by p, we pay $\delta = \min(c_u - p(u), c_v - p(v))$ to the vertex from $\{u, v\}$ that achieves this minimum and purchase it. Proving unistrictness is now equivalent to proving that $\delta \le \xi(V, S \cup \{e\}, e)$.

Indeed, consider the runs $\mathcal{A}(S)$ and $\mathcal{A}(S \cup \{e\})$. Both runs behave identically until

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

Finally, applying Theorem 4.2 with $\alpha = 2$ and $\beta = 1$ yields the 3-approximation and completes the proof. □

6. Extension: Uncapacitated facility location. The uncapacitated facility location problem (UFL) is one of the most well-studied combinatorial optimization problems, and one that lends itself naturally to being considered in a multistage stochastic context. However, the problem does not fit into the class of combinatorial optimization problems defined in section 2. As a resultant, the results of section 4 cannot be used to construct approximation algorithms for Stoc(UFL). Nevertheless, it is possible to define a stochastic version of UFL and to construct an approximation algorithm for it using the main ideas of this paper: using sampling to generate client sets and using approximation algorithms to construct partial solutions for them where the costs of the partial solutions are bounded by cost shares. In this section, we provide an approximation algorithm for a two-stage stochastic version of UFL, by suitably adapting some of the results above.

We begin by defining the deterministic problem Det(UFL). An instance of Det(UFL) is given by a set of facilities F and a set of clients U. The distances $c_{(i,j)}$ between any pair of points i, j from $F \cup U$ form a metric. Each facility p has an opening cost f_p; the goal is to open a subset of facilities F' to minimize the opening costs plus the sum of distances from each client to its open facility. That is, if we let $c(j, F')$ represent the distance from client j to the nearest facility in F', then the objective is to minimize

$$\sum_{p \in F'} f_p + \sum_{j \in U} c(j, F'). \tag{6.1}$$

Notice that the cost function has two components. The first component, $\sum_{p \in F'} f_p$, is the cost of the facilities opened, while the second, $\sum_{j \in S} c(j, F')$, is the cost of serving clients from opened facilities. Let us examine how this relates to the model framework described in section 2. In order to fit Det(UFL) into that model, we must define the set of elements X that can be purchased to serve clients, wherein a solution consists of a subset of elements that satisfy the client set chosen. Therefore, the element set must consist of both the facility set F and the set of all possible facility-client connections $F \times U$. That is, $X = F \cup (F \times U)$. In such a case, if we used Algorithm Boost-and-Sample, a first-stage solution would be required to not only open some facilities, but also buy some edges connecting these facilities to some potential clients. If we consider the practical context of the facility location problem, such a requirement is a little unrealistic—while it does make sense to preemptively build facilities in a stochastic context in anticipation of future demand, it often does not make sense to pay for service costs in advance for clients who may not even materialize.

Consequently, the literature on stochastic versions of the UFL [45, 42, 39] has focused on a different stochastic model. In this model, in the first stage, only some facilities are purchased (call this set F_1). In the second stage, after the clients have been realized, a second set of facilities (F_2) may be purchased (at an inflated cost), and the service cost is incurred for serving each client from its nearest facility in the

set $F_1 \cup F_2$. That is, the service cost is incurred only in the second stage and only for realized clients. We adopt this definition of Stoc(UFL) for the rest of this section. We will also modify the cost-sharing function: ξ will now help to pay a portion of the client's connection cost to the facility serving it, plus possibly a portion of the cost of the facility. We will not define strictness for the cost-sharing function; instead, we will directly use its construction to bound the expected second-stage cost.

Because Stoc(UFL) now does not fit into the model of 2, we will suitably adapt the notation and terminology in such a way as to reuse results from section 4 where possible, and prove other necessary results here. This will enable us to provide a constant factor approximation algorithm for the two-stage version of Stoc(UFL) using the same techniques as are used in the rest of this paper.

Formally, the two-stage stochastic UFL Stoc(UFL) is defined as follows. As in Det(UFL) above, we are given a set of facilities F, a set of clients U, a distance metric $c : F \times U \to \mathbb{R}^+$, and a facility cost function $f : F \to \mathbb{R}^+$. Additionally, we have a known cost inflation factor σ, and the known probability distribution $\pi : 2^S \to [0, 1]$ that defines the probability of client sets to realize in the second stage. As in Det(UFL), let $c(j, F')$ be defined as $\min_{i \in F'} c(i, j)$ for any $F' \subseteq F$.

In the first stage, we are required to purchase a set of facilities $F_1 \subseteq F$. In the second stage, after the client set $\mathbf{S}$ is revealed to be S, a second set of facilities $F_2(S)$ is purchased. Additionally, a service cost of $c_s(S, F_1 \cup F_2(S)) = \sum_{j \in S} c(j, F_1 \cup F_2(S))$ is incurred. The objective is then to minimize the total expected cost, where the expectation is taken over π:

$$Z = \sum_{i \in F_1} f_i + \mathbf{E}\left[\sigma \sum_{i \in F_2(\mathbf{S})} f_i + c(F_1 \cup F_2(\mathbf{S}), \mathbf{S})\right]. \tag{6.2}$$

As before, in order to provide an approximation algorithm for Stoc(UFL), we will begin with an approximation algorithm $\mathcal{A}$ for Det(UFL), along with an associated cost-sharing function ξ and an augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$. Given these, the corresponding boosted sampling algorithm for Stoc(UFL) is described in Figure 6.1 and labeled SUFL-Boost-and-Sample. There are two key differences from Boost-and-Sample: (i) SUFL-Boost-and-Sample does not construct a partial solution in the first stage—it only purchases a set of facilities—and (ii) SUFL-Boost-and-Sample pays for the service costs of all clients in the second stage, instead of only the new clients considered by Boost-and-Sample.

We now describe the first-stage algorithm $\mathcal{A}$, the augmentation algorithm $\mathsf{Aug}_{\mathcal{A}}$, and the cost-sharing function. Subsequently, we will prove the approximation ratio for SUFL-Boost-and-Sample directly, without invoking Theorem 4.1. As will be seen, the key ideas of the proof are the same as in Theorem 4.1; we only have to make appropriate modifications to allow for the service cost component that renders direct applicability of Theorem 4.1 impossible.

Neither our algorithm nor our cost-sharing function is new; however, the fact that the cost-shares can be used to bound the cost of the solution constructed by the algorithm is new and hence proved here.

The algorithm $\mathcal{A}$: Our first-stage algorithm $\mathcal{A}$ is a slight modification of the one developed by Mettu and Plaxton [33], which we sometimes refer to as the "MP" algorithm. Intuitively, all clients are progressively asked to pay increasing amounts towards constructing a solution. Facilities are considered for opening if the clients they serve are able to pay enough to cover the cost

Algorithm SUFL-Boost-and-Sample.

Given: An instance of Stoc(UFL), defined by universe of clients U, probability distribution π, facility set F, cost functions f and c, and inflation factor σ.

1. Draw $\lfloor\sigma\rfloor$ independent samples $D_1, D_2, \ldots, D_{\lfloor\sigma\rfloor}$ of clients by sampling from the distribution π. Let $D = \cup_i D_i$.
2. Using the algorithm $\mathcal{A}$, purchase a first-stage set of facilities $F_1 \subseteq F$. Algorithm $\mathcal{A}$ is an α-approximation which includes a cost-sharing function ξ and augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$.

Output: The first-stage set of facilities F_1.

3. The second stage occurs, and the client set S is realized. Using the augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ on the client set S, compute the second-stage set of facilities $F_2(S)$. Each client $j \in S$ is assigned to their nearest facility from the set $F_1 \cup F_2(S)$.

Output: The second-stage solution $F_2(S)$.

FIG. 6.1. *Algorithm SUFL-Boost-and-Sample for two-stage* Stoc(UFL).

of the facility as well as the sum of distances from the facility to all the clients it serves. Complications arise because the same client may be asked to pay for multiple facilities, and this is what results in the algorithm being an approximation (as opposed to an exact one), as will become clear when the details are perused. The algorithm proceeds by considering facilities in the order in which they can be paid for, provided the clients are able to pay for a sufficient fraction of the facilities. Other clients are connected to their nearest open facility, even if they haven't paid for the cost of opening it. The details are provided in the rest of this section.

The augmentation algorithm $\mathsf{Aug}_{\mathcal{A}}$: In the algorithm $\mathcal{A}$, open facilities get funding from clients in $S \cap D$ (realized clients already sampled in the first stage) as well as $T = S \setminus D$ (new clients that arrive in the second stage). If a sufficient fraction of the funding is from clients in T, the facility is called T-heavy. The augmentation algorithm proceeds in a fashion similar to that of the original algorithm: it progressively considers T-heavy facilities in the order in which they can be paid for, and opens them if the clients in S are able to pay a significant fraction of the facility cost. Remaining clients are again simply assigned to their nearest open facility.

The cost-sharing function ξ: The cost-sharing function is that developed by Pál and Tardos [36]. It allocates a portion of the total (facility opening and service) cost of the solution to the clients. In terms of the algorithm $\mathcal{A}$, the cost-share of a client is either the amount it was asked to pay to open a facility or the distance to a facility serving it if the client did not pay towards opening any facility.

We now provide the technical details of the algorithms and cost-sharing function. For a facility p, let $B(p, \tau)$ be a ball with center p and radius τ. We define the *opening time* $t_p(S)$ of a facility p w.r.t. the client set S to be the unique radius τ such that

$$f_p = \sum_{j \in B(p,\tau) \cap S} (\tau - c(j,p)). \tag{6.3}$$

Let the set $C_p(S) = \{j \in S : c(j,p) < t_p(S)\}$ be called the *contributing set* for p. Note that if we charge each client in $C_p(S)$ the amount $t_p(S)$, we exactly recover the facility cost of p plus the cost of assigning clients in $C_p(S)$ to p. We drop the set of clients from the notation and say F_2, t_p, and C_p instead of $F_2(S)$, $t_p(S)$, and $C_p(S)$, respectively, when there is no danger of confusion. The cost-shares of clients are then defined as

$$\xi(F,S,j) = \min_{p \in F}\{\max(t_p(S), c(j,p))\}. \tag{6.4}$$

This is the same cost-sharing function defined in [36], where competitiveness was also proved (Theorem 2.2). Intuitively, the contribution of user j towards the facility p should be either t_p if $j \in C_p$ or the connection cost $c(j,p)$ if $j \notin C_p$. The client can (and does) choose to contribute only to the least demanding facility; the facility p for which this minimum is attained is called the *primary* facility of j (in the run on S). A facility p is said to be *well-funded* if $\xi(F,S,j) \geq t_p(S)/3$ for all $j \in C_p$.

The algorithm $\mathcal{A}$ that we use is, superficially, a slight modification of the general MP algorithm; given a set of clients S, the algorithm $\mathcal{A}$ considers all the well-funded facilities p in order of increasing opening time $t_p(S)$. For each such (well-funded) facility p, the algorithm declares it *open* if there are no previously opened facilities within a radius $2t_p(S)$ around p. (The general MP algorithm considers all facilities p as candidates for opening, but ends up opening only those that are well-funded.)

For each open facility p, the algorithm $\mathcal{A}$ assigns all clients in C_p to p. (By construction, the sets C_p for open facilities p are disjoint.) It then assigns each client not lying in any C_p to its closest open facility. The following facts can be derived from arguments in [33] and [36]:

1. For each open facility p, the cost-shares $\xi(F,S,C_p)$ of the clients in C_p pay $1/3$ of f_p plus their assignment cost. (See [36, Lemma 2.4] and the preceding discussion therein.)
2. For each facility p, there exists a well-funded facility q (possibly $p = q$) such that $c(p,q) \leq 2(t_p - t_q)$. (Note that it must be that $t_q \leq t_p$.) That is, either p itself is well-funded, or there is a well-funded facility q fairly close to p.
3. For each facility p, there exists an *open* facility q within a distance of $2t_p$. Hence, if p is a primary facility for some client j, then $c(j,q) \leq 3\xi(F,S,j)$.

The following theorem was proved in [33, 36].

THEOREM 6.1. *The algorithm $\mathcal{A}$ is a 3-approximation for UFL.*

We now proceed to define the augmentation algorithm and bound the cost of the overall solution using the cost-sharing function. To this end, consider a set of new clients T with $D \cap T = \emptyset$. In the following, let $C_p = C_p(D \cup T)$ denote the contributor set of a facility p in the run $\mathcal{A}(D)$. Similarly, when we say a facility p is well-funded, we mean that p is well-funded in the run $\mathcal{A}(D)$. A facility p is called *T-heavy* if $|C_p \cap T| \geq b|C_p|$ (where the parameter $b \in (0,1)$ will be specified later), and is called *T-light* otherwise. Note that a T-light facility must have $|C_p \cap D| \geq (1-b)|C_p|$.

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

of the analysis of the algorithm, clients in $S \cap D = S \setminus T$ are assigned to the same facilities that they were assigned to in the first stage. Of course, these clients can be reassigned to their nearest open facilities after the second stage, resulting only in a decrease in costs.

Before we proceed any further, we prove a technical lemma.

LEMMA 6.2. *If p is a T-light facility, then*

$$t_p(D) \le \frac{1}{1-b} t_p(D \cup T) - \frac{1}{|C_p \cap D|} \sum_{j \in C_p \cap T} c(j,p). \tag{6.5}$$

Proof. Consider the set $C_p = \{j \in D \cup T : c(j,p) < t_p\}$; by definition (6.3),

$$f_p + \sum_{j \in C_p} c(j,p) = |C_p| t_p(D \cup T). \tag{6.6}$$

Since p is T-light, $|C_p \cap D| \ge (1-b)|C_p|$. Therefore,

$$\begin{aligned} &f_p + \sum_{j \in C_p \cap D} c(j,p) \\ &\quad = |C_p| t_p(D \cup T) - \sum_{j \in C_p \cap T} c(j,p) \le |C_p \cap D| \frac{t_p(D \cup T)}{1-b} - \sum_{j \in C_p \cap T} c(j,p). \end{aligned} \tag{6.7}$$

Also observe that $t_p(D \cup T)/(1-b) - \sum_{j \in C_p \cap T} c(j,p)/|C_p \cap D| \ge t_p(D \cup T)$, so all clients in $C_p \cap D$ are still contributing to the facility p at time $t_p(D \cup T)/(1-b) - \sum_{j \in C_p \cap T} c(j,p)/|C_p \cap D|$. So, at time $t_p(D \cup T)/(1-b) - \sum_{j \in C_p \cap T} c(j,p)/|C_p \cap D|$, facility p was already paid for in the run $\mathcal{A}(D)$, proving the lemma. ∎

We now prove that Algorithm SUFL-Boost-and-Sample results in a solution of cost no more than 8.45 times the optimum. Consider an optimal solution, where F_1^* is the set of facilities opened in the first stage and $F_2^*(S)$ is the set of facilities opened in the second stage if client set S is realized. Any optimal solution always serves all clients using their nearest open facility. Therefore, the optimum cost Z^* can be written as

$$Z^* = c(F_1^*) + \sum_{S \in 2^U} \pi(S) \Big[\sigma c(F_2^*(S)) + c_s(S, F_1^* \cup F_2^*(S)) \Big]. \tag{6.8}$$

In order to bound the cost of the solution produced by Algorithm SUFL-Boost-and-Sample, we proceed as follows. Recall that facilities are purchased in both stages, while the service cost is incurred only in the second stage. In the following, we will first bound the cost of the first-stage facilities plus the service cost of all realized clients who also appeared in the first-stage sample. Next, we will bound the cost of second-stage facilities in addition to the service costs of all new clients. That is, for the purpose of bounding the service costs, we will partition the realized client set into two sets: those that appeared in the first-stage sample and those that are new. The service cost of the former will be bounded along with the first-stage facility costs, while the service cost of the latter will be bounded along with the second-stage facility costs.

Recall that D is the set of clients sampled in the first stage. So, the expected first-stage cost is given by $\mathbf{E}[\, c(F_1) + c_s(D \cap S, F_1)\,]$. We bound this quantity with respect to Z^* below, along the same lines as the bound on the first-stage costs in Theorem 4.1.

LEMMA 6.3. $\mathbf{E}[\, c(F_1) + c_s(D \cap S, F_1)\,] \le 3Z^*$.

Proof. Recall that the optimal cost is given by

$$Z^* = c(F_1^*) + \sum_{S \in 2^U} \pi(S)\Big[\sigma c(F_S^*) + c_s(S, F_1^* \cup F_S^*)\Big]. \tag{6.9}$$

As in the proof of Theorem 4.1, we assume that σ is an integer. Given the set of clients D sampled in the first stage, we define $\tilde{F}_1 = F_1^* \cup F_2^*(D_1) \cup F_2^*(D_2) \cup \cdots \cup F_2^*(D_\sigma)$. The following inequality then follows exactly as in the proof of Theorem 4.1:

$$\mathbf{E}[\, c(\tilde{F}_1) + c_s(D, \tilde{F}_1)\,] \le Z^*. \tag{6.10}$$

Observe now that $\tilde{F}_1$ is a feasible solution for the Det(UFL) problem, given client set D. The lemma now follows because Algorithm $\mathcal{A}$ is a 3-approximation for the Det(UFL) problem. ∎

We now consider the facilities opened in the second stage, as well as the service costs for clients who were not sampled in the first stage. Recall that T denotes the set of new clients revealed in the second stage. Let $b \in (0,1)$ be a constant whose value will be specified later.

LEMMA 6.4. *The service cost of the augmentation is bounded as follows:* $c_s(T, F_1 \cup F_2) \le (1 + \frac{2}{1-b})\xi(F, D \cup T, T)$.

Proof. First, consider any well-funded T-heavy facility p. Since p is T-heavy, the share of each client in $C_p \cap T$ can pay for its own connection cost. Hence we must consider clients in j whose primary facility p is either not well-funded or not T-heavy. We claim that in both cases there must be a facility close to p opened either by $\mathcal{A}(D)$ (the first stage) or in the augmenting stage. Note that, by the properties of the algorithm $\mathcal{A}$, there is a well-funded facility q such that $t_q(D \cup T) \le t_p(D \cup T)$ and $c(p,q) \le 2(t_p(D \cup T) - t_q(D \cup T))$.

Now, if q is T-heavy, by the properties of our augmentation procedure, there must be a facility r that was open in the augmentation step such that $c(q,r) \le 2t_q(D \cup T)$. On the other hand, if q is T-light, we have that $t_q(D) \le t_q(D \cup T)/(1-b)$ by Lemma 6.2 above. Thus, in the run $\mathcal{A}(D)$, there must be an open facility r such that $c(q,r) \le 2t_q(D) \le (2/(1-b))t_q(D \cup T)$.

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

$$\begin{aligned} c(j,r) &\le c(j,p) + c(p,q) + c(q,r) \\ &\le c(j,p) + 2(t_p(D \cup T) - t_q(D \cup T)) + (2/(1-b))t_q(D \cup T) \\ &\le c(j,p) + (2/(1-b))t_p(D \cup T) \\ &\le (1 + 2/(1-b))\xi(F, D \cup T, j). \end{aligned} \tag{6.11}$$

Summing over all clients $j \in T$ yields the lemma. ∎

We can now bound the expected service cost of the new clients as follows.

LEMMA 6.5. *The expected service cost of the augmentation is bounded as follows:* $\mathbf{E}[\, c_s(T, F_1 \cup F_2)\,] \le (1 + \frac{2}{1-b})Z^*$.

Proof. The lemma follows from the competitiveness of the cost-sharing function ξ and by taking expectations on both sides of the inequality proved in Lemma 6.4. ∎

All that remains is to bound the expected facility costs of the new facilities, $\mathbf{E}[\, c(F_2(\mathbf{S}))\,]$.

LEMMA 6.6. *The expected facility installation cost of the augmentation is bounded as follows:* $\sigma\mathbf{E}[\, c(F_2)\,] \le \frac{3}{b}Z^*$.

Proof. Because the augmentation algorithm opens only well-funded T-heavy facilities, we have the following: $c(F_2) \leq \frac{3}{b}\xi(F, S, T)$. However, facility costs are inflated by a factor of σ in the second stage, so the actual cost incurred by our algorithm is $\sigma c(F_2)$. To bound this cost, we appeal to an argument similar to that of Theorem 4.1.

As in Theorem 4.1, consider an alternate probabilistic process to generate the sets D_i and the set S: draw $\sigma + 1$ independent samples $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_{\sigma+1}$ from the distribution π. Now choose K uniformly at random from $\{1, 2, \ldots, \sigma + 1\}$, and let $S = \hat{D}_K$ and $D = \cup_{i \neq K} \hat{D}_i$. This process is distributed identically to the original process, as before. Let $\hat{D} = \cup_{i=1}^{\sigma+1} \hat{D}_i$ and $\hat{D}_{-i} = \hat{D} \setminus \hat{D}_i$. It follows from the definition of this alternate probabilistic process that

$$\mathbf{E}[\,\xi(F, \hat{D}, \hat{D}_K \setminus \hat{D}_{-K})\,] \leq \frac{1}{\sigma}\mathbf{E}[\,\xi(F, \hat{D}, \hat{D}_{-K})\,]. \tag{6.12}$$

Since $\hat{D}, \hat{D}_{-K}$, and $\hat{D}_K \setminus \hat{D}_{-K}$ are identically distributed as $D \cup T, D$, and T, respectively, we have

$$\mathbf{E}[\,\xi(F, D \cup T, T)\,] \leq \frac{1}{\sigma}\mathbf{E}[\,\xi(F, D \cup T, D)\,]. \tag{6.13}$$

We note that the cost-sharing function ξ is cross-monotone, because for any client sets S and T and any facility p it is always the case that $t_p(S \cup T) \leq t_p(S)$. This results in $\xi(F, D \cup T, D) \leq \xi(F, D, D)$. Next, competitiveness of the cost-sharing function yields $\xi(F, D, D) \leq c(\mathsf{OPT}(D))$. As in the proof of Lemma 6.3, we also have $c(\mathsf{OPT}(D)) \leq Z^*$. Chaining these inequalities and applying expectations, we conclude that $\mathbf{E}[\,\sigma c(F_2)\,] \leq \frac{3}{b}Z^*$. ∎

We now have our main result for this section.

THEOREM 6.7. *The two-stage stochastic UFL can be approximated to a factor of* 8.45.

Proof. Lemma 6.3 bounds the expected cost of the first-stage solution by $3Z^*$. Lemmas 6.5 and 6.6 bound the second-stage service and facility costs, respectively, by $(1 + \frac{2}{1-b})Z^*$ and $\frac{3}{b}Z^*$. Their sum is minimized at $b = 3 - \sqrt{6}$, yielding a second-stage approximation ratio of $3 + \sqrt{6}$ and an overall approximation ratio of $6 + \sqrt{6} \approx 8.45$. ∎

We conjecture that the independent-decisions and multistage versions of stochastic UFL can also be approximated by techniques similar to those in this paper. We do not pursue those results here, because they would add tedium while not providing much in the way of insight. We do point out, however, that the stochastic UFL result demonstrates the applicability of our algorithm and its analysis to problems that do not necessarily fall naturally into the framework of section 2.

7. Conclusion. In this paper, we have provided a general framework for converting approximation algorithms for deterministic problems into those for k-stage stochastic versions. Our algorithms rely on the presence of cost-sharing functions with strictness properties, but we demonstrate the existence of such cost-sharing functions for several canonical combinatorial optimization problems.

Despite our progress, several open questions in this area remain. There are several deterministic combinatorial optimization problems for which cost-sharing functions are not readily available. Apart from obtaining these for other problems, one is also interested in characterization of the types of problems for which one may (or may not) be able to construct cost-sharing functions.

Our results for k-stage stochastic optimization are also somewhat weak, because they require the cost-sharing function to be cross-monotone and superstrict, properties

we are able to establish only for the Steiner tree problem. It would be of interest to construct approximation algorithms without such strong requirements on the cost-sharing functions, or provide superstrict cross-monotone cost-sharing functions for other problems. Another natural question is whether we can avoid the linear loss in the number of stages and hence develop $o(k)$-approximation algorithms for k-stage stochastic optimization.

Acknowledgments. Preliminary versions of the results in this manuscript appeared in the extended abstracts [18] and [19]. In the process of preparing this manuscript, we discovered an error in [19]: Theorem 2 (proving the approximability of multistage stochastic optimization problems in the absence of cross-monotone cost-sharing functions) is incorrect, and therefore withdrawn.

We thank the referees of the conferences as well as of this journal, and the associate editor, for their enlightening feedback during preparation of this manuscript. In particular, the error in [19] was discovered during the review process of this manuscript, and we thank the referee for identifying the problem. The referees' feedback has greatly improved this manuscript.

REFERENCES

[1] A. Agrawal, P. Klein, and R. Ravi, *When trees collide: An approximation algorithm for the generalized Steiner problem on networks*, SIAM J. Comput., 24 (1995), pp. 440–456.

[2] B. M. Anthony and A. Gupta, *Infrastructure leasing problems*, in Proceedings of the 12th Integer Programming and Combinatorial Optimization Conference (IPCO), Ithaca, NY, 2007, Lecture Notes in Comput. Sci. 4513, Springer, New York, 2007, pp. 424–438.

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

[4] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, Springer-Verlag, New York, Berlin, 1997.

[5] M. Charikar, C. Chekuri, and M. Pál, *Sampling bounds for stochastic optimization*, in Approximation, Randomization and Combinatorial Optimization, Lecture Notes in Comput. Sci. 3624, Springer, Berlin, 2005, pp. 257–269.

[6] M. Chlebík and J. Chlebíková, *Approximation hardness of the Steiner tree problem on graphs*, in Proceedings of the 8th Scandinavian Workshop on Algorithm Theory, Turku, Finland, 2002, Lecture Notes in Comput. Sci. 2368, Springer, New York, 2002, pp. 95–99.

[7] G. B. Dantzig, *Linear programming under uncertainty*, Management Sci., 1 (1955), pp. 197–206.

[8] N. R. Devanur, M. Mihail, and V. V. Vazirani, *Strategyproof cost-sharing mechanisms for set cover and facility location games*, Decision Support Syst., 39 (2005), pp. 11–22.

[9] S. Dye, L. Stougie, and A. Tomasgard, *The stochastic single resource service-provision problem*, Naval Res. Logistics, 50 (2003), pp. 869–887.

[10] M. Dyer and L. Stougie, *Computational complexity of stochastic programming problems*, Math. Program., 106 (2006), pp. 423–432.

[11] L. Fleischer, J. Könemann, S. Leonardi, and G. Schäfer, *Strict cost sharing schemes for Steiner forest*, SIAM J. Comput., 39 (2010), pp. 3616–3632.

[12] N. Garg, A. Gupta, S. Leonardi, and P. Sankowski, *Stochastic analyses for online combinatorial optimization problems*, in Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2008, pp. 942–951.

[13] A. Goel and P. Indyk, *Stochastic load balancing and related problems*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, New York, 1999, IEEE Computer Society Press, Piscataway, NJ, 1999, pp. 579–586.

[14] M. X. Goemans and D. P. Williamson, *A general approximation technique for constrained forest problems*, SIAM J. Comput., 24 (1995), pp. 296–317.

[15] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, J. Algorithms, 31 (1999), pp. 228–248.

[16] A. Gupta, A. Kumar, M. Pál, and T. Roughgarden, *Approximations via cost-sharing: Simpler and better approximation algorithms for network design*, J. ACM, 54 (2007), pp. 1–38.

[17] A. Gupta and M. Pál, *Stochastic Steiner trees without a root*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 3580, Springer, New York, 2005, pp. 1051–1063.

[18] A. Gupta, M. Pál, R. Ravi, and A. Sinha, *Boosted sampling: Approximation algorithms for stochastic optimization*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, ACM, New York, 2004, pp. 417–425.

[19] A. Gupta, M. Pál, R. Ravi, and A. Sinha, *What about Wednesday? Approximation algorithms for multistage stochastic optimization*, in Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), Berkeley, CA, 2005, Lecture Notes in Comput. Sci. 3624, Springer, New York, 2005, pp. 86–98.

[20] A. Gupta, R. Ravi, and A. Sinha, *LP rounding approximation algorithms for stochastic network design*, Math. Oper. Res., 32 (2007), pp. 345–364.

[21] J. Hastad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.

[22] A. Hayrapetyan, C. Swamy, and E. Tardos, *Network design for information networks*, in Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Vancouver, BC, 2005, pp. 933–942.

[23] N. Immorlica, D. Karger, M. Minkoff, and V. 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, New Orleans, LA, 2004, pp. 684–693.

[24] N. Immorlica, M. Mahdian, and V. Mirrokni, *Limitations of cross-monotonic cost-sharing schemes*, ACM Trans. Algorithms, 4 (2008), 24.

[25] K. Jain and V. Vazirani, *Applications of approximation algorithms to cooperative games*, in Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (STOC), Heraklion, Greece, 2001, ACM, New York, pp. 364–372.

[26] K. Jain and V. V. Vazirani, *Equitable cost allocations via primal–dual-type algorithms*, SIAM J. Comput., 38 (2008), pp. 241–256.

[27] P. Kall and S. W. Wallace, *Stochastic Programming*, John Wiley & Sons, New York, 1994.

[28] W. K. Klein Haneveld and M. H. van der Vlerk, *Stochastic integer programming: General models and algorithms*, Ann. Oper. Res., 85 (1999), pp. 39–57.

[29] W. K. Klein Haneveld and M. H. van der Vlerk, *Stochastic Programming*, unpublished textbook, Department of Econometrics and Operations Research, University of Groningen, The Netherlands, 2003.

[30] J. Kleinberg, Y. Rabani, and É. Tardos, *Allocating bandwidth for bursty connections*, SIAM J. Comput., 30 (2000), pp. 191–217.

[31] A. J. Kleywegt, A. Shapiro, and T. Homem-de-Mello, *The sample average approximation method for stochastic discrete optimization*, SIAM J. Optim., 12 (2001), pp. 479–502.

[32] J. Könemann, S. Leonardi, G. Schäfer, and S. H. M. van Zwam, *A group-strategyproof cost sharing mechanism for the Steiner forest game*, SIAM J. Comput., 37 (2008), pp. 1319–1341.

[33] R. R. Mettu and C. G. Plaxton, *The online median problem*, SIAM J. Comput., 32 (2003), pp. 816–832.

[34] R. H. Möhring, A. S. Schulz, and M. Uetz, *Approximation in stochastic scheduling: The power of LP-based priority policies*, J. ACM, 46 (1999), pp. 924–942.

[35] H. Moulin and S. Shenker, *Strategyproof sharing of submodular costs: Budget balance versus efficiency*, Econom. Theory, 18 (2001), pp. 511–533.

[36] M. Pál and E. Tardos, *Group strategyproof mechanisms via primal-dual algorithms*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, Cambridge, MA, 2003, IEEE Computer Society Press, Piscataway, NJ, 2003, pp. 584–593.

[37] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice–Hall, Englewood Cliffs, NJ, 1995.

[38] R. C. Prim, *Shortest interconnection networks and some generalizations*, Bell System Tech. J., 36 (1957), pp. 1389–1401.

[39] R. Ravi and A. Sinha, *Hedging uncertainty: Approximation algorithms for stochastic optimization problems*, Math. Program., 108 (2006), pp. 97–114.

[40] G. Robins and A. Zelikovsky, *Improved Steiner tree approximation in graphs*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 770–779.

[41] R. Schultz, L. Stougie, and M. H. van der Vlerk, *Two-stage stochastic integer programming: A survey*, Statist. Neerlandica, 50 (1996), pp. 404–416.

[42] D. Shmoys and C. Swamy, *An approximation scheme for stochastic linear programming and its application to stochastic integer programs*, J. ACM, 53 (2006), pp. 978–1012.

[43] M. Skutella and M. Uetz, *Stochastic machine scheduling with precedence constraints*, SIAM J. Comput., 34 (2005), pp. 788–802.
[44] A. Srinivasan, *Approximation algorithms for stochastic and risk-averse optimization*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 2007, pp. 1305–1313.
[45] C. Swamy and D. Shmoys, *Sampling-based approximation algorithms for multi-stage stochastic optimization*, in Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, PA, 2005, IEEE Computer Society Press, Piscataway, NJ, 2005, pp. 357–366.
[46] C. Swamy and D. Shmoys, *Approximation algorithms for 2-stage stochastic optimization problems*, ACM SIGACT News, 37 (2006), pp. 33–46.
[47] V. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, Berlin, 2001.
[48] H. P. Young, *Cost allocation*, in Handbook of Game Theory, R. J. Aumann and S. Hart, eds., North–Holland, Amsterdam, 1994, Vol. 2, Chapter 34, pp. 1193–1235.