Stochastic Decomposition for Two-stage Stochastic Linear Programs with Random Cost Coefficients

Harsha Gangammanavar
Southern Methodist University, harsha@smu.edu

Yifan Liu
84.51°

Suvrajeet Sen
University of Southern California, s.sen@usc.edu

Stochastic decomposition (SD) has been a computationally effective approach to solve large-scale stochastic programming (SP) problems arising in practical applications. By using incremental sampling, this approach is designed to discover an appropriate sample size for a given SP instance, thus precluding the need for either scenario reduction or arbitrary sample sizes to create sample average approximations (SAA). When compared to the solutions obtained using the SAA procedure, SD provides solutions of similar quality in far less computational time using ordinarily available computational resources. However, previous versions of SD were not applicable to problems with randomness in second-stage cost coefficients. In this paper, we extend its capabilities by relaxing this assumption on cost coefficients in the second-stage. In addition to the algorithmic enhancements necessary to achieve this, we also present the details of implementing these extensions which preserve the computational edge of SD. Finally, we illustrate the computational results obtained from the latest implementation of SD on a variety of test instances generated for problems from the literature. We compare these results with those obtained from the regularized L-shaped method applied to the SAA function of these problems with different sample sizes.

Key words: Stochastic programming, stochastic decomposition, sample average approximation, two-stage models with random cost coefficients, sequential sampling.

History:

1. Introduction

The two-stage stochastic linear programming problem (2-SLP) can be stated as:

$$\min \; f(x) := c^\top x + \mathbb{E}[h(x,\tilde{\omega})] \tag{1a}$$
$$\text{s.t. } x \in \mathcal{X} := \{x \mid Ax \leq b\} \subseteq \mathcal{R}^{n_1}$$

where, the recourse function is defined as follows:

$$h(x,\omega) := \min \; d(\omega)^\top y \tag{1b}$$
$$\text{s.t. } D(\omega)y = \xi(\omega) - C(\omega)x$$

$$y \geq 0, y \in \mathcal{R}^{n_2}.$$

The random variable $\tilde{\omega}$ is defined on a probability space $(\Omega, \mathcal{F}, \mathcal{P})$ where the sample space $\Omega \subseteq \mathcal{R}^{r_2}$, and ω denotes a realization of this random variable. The problem statement above allows one or more elements of data (d, D, ξ, C) to depend on the random variable. Moreover, the statement accommodates both continuous as well as discrete random variables. Computational approaches to address this formulation, however, have largely focused on problems with finite support, where uncertainty is represented using a set of realizations $\{\omega^1, \omega^2, \ldots, \omega^N\}$ with their respective probabilities $p^j, j = 1, \ldots, N$ that are assumed to be known. When N is small, a deterministic equivalent problem can be formulated and solved using off-the-shelf LP solvers.

In problems with continuous random variables, calculation of the expectation functional involves a multidimensional integral in high dimension. For such problems, as well as those where the number of possible realizations is astronomically large, sampling-based methods offer a computationally viable means for obtaining a reasonable approximation. When sampling precedes the optimization step (i.e., external sampling), the objective function in (1a) is replaced by its so called sample average approximation (SAA) using a fixed number of outcomes. As an alternative to choosing the sample size, one can allow the algorithm to dynamically determine a sufficiently large sample size during optimization. We refer to such a coupling of sampling and optimization as sequential or internal sampling, and this forms the basis for the stochastic decomposition (SD) algorithm (Higle and Sen (1991, 1994)).

The SD algorithm allows for seamlessly incorporating new outcomes to improve the approximation of the expected recourse function without having to restart the optimization. In order to accomplish this, the algorithm continuously gathers relevant information during the course of its run, and efficiently re-utilizes this information in future iterations. These features have enabled SD to provide high quality solutions in far less computational time using ordinary desktop and laptop computers when compared to the methods that involve using the SAA function to obtain solutions of similar quality. This was demonstrated on a suite of SP instances in Sen and Liu (2016) and on a real scale power system operations planning problem in Gangammanavar et al. (2016).

Nevertheless, previous research on SD, including the most recent work of Sen and Liu (2016), focuses on 2-SLPs where uncertainty affects only the right-hand side elements of

subproblem (1b), i.e., elements of vector ξ and matrix C. In this paper we extend the capability of SD to address 2-SLPs with random cost coefficients making it applicable to problems where parameters (d, ξ, C) are governed by the random variable $\tilde{\omega}$.

In classical approaches to solve 2-SLPs – including the L-shaped method (Van Slyke and Wets (1969)) – the subproblems for all realizations are solved in every iteration and the outer linearization generated in an iteration is based on dual multipliers computed only in that iteration. Hence, when compared to the computational workload for 2-SLPs with deterministic second-stage cost coefficients, an instance with random cost coefficients does not pose any *additional* difficulty for such methods. On the other hand, the reliance on information discovered in earlier iterations to generate SD approximations, and in particular, the repeated use of previously discovered dual vertices can no longer be used without modification when cost coefficients have random elements. In light of this, the main contributions of this work are as follows:

- We show that in the presence of random cost coefficients, dual vertices can be represented by a decomposition consisting of a deterministic representation together with a shift vector which depends on the realization of the random variable. This shift vector can be applied both when the elements of $d(\omega)$ are dependent or independent. Moreover, the dimension of the shift vector depends on the number of random elements in d. For instances in which the number of random elements is small, the dimension of shift vectors is also small, thus allowing SD to retain many of the advantages available in the case of fixed second-stage cost coefficients.

- We design extensions of previous SD computational data structures to accommodate the introduction of random cost coefficients without sacrificing most of the computational advantage of SD. However, the number of random cost coefficients does have a negative impact on the algorithm's performance. In view of this, we also identify optimization model structures which help us to overcome these negative impacts.

- We design new instances with random cost coefficients which can be used to benchmark stochastic programming (SP) algorithms. We also present computational results obtained from the enhanced SD algorithm and compare them to those obtained using the regularized L-shaped method applied to the SAA function of varying sizes.

The rest of the paper is organized as follows. In §2 we will present a succinct review of the SD algorithm focusing on the principal steps involved in computing the approximations

of the first-stage objective function. In §3 we will formalize all of the key algorithmic ideas necessary to enable the SD algorithm to address problems with random cost coefficients. Our discussion will also include notes on challenges associated with implementing these ideas and our approach to address them. This will be followed by a narration of our computational experience in §5 on both small and large scale instances.

2. Background: Stochastic Decomposition

Before we present our work, a brief overview of the SD algorithm is in order. We refer the reader to Higle and Sen (1991), Higle and Sen (1994) and Sen and Liu (2016) for detailed mathematical expositions of this algorithm. Our discussion here will focus on the construction of lower bounding approximations of the expected recourse function in (1a) with an eye towards computer implementation. We will present a high level description of the regularized SD algorithm while ignoring the algorithmic details regarding incumbent-update (used in the proximal term of the regularized version) and stopping rules. In particular, we will focus on the information that is collected during the course of the algorithm, and discuss how this information is utilized to generate/update the approximations in a computationally efficient manner. In the remainder of this paper, the computer programming constructs will be presented in `typewriter` font to distinguish them from the mathematical constructs.

We first state the assumptions for the 2-SLP models as required by previous versions of SD.

A1. The set of first-stage solutions $\mathcal{X}$, and the set of outcomes Ω are compact.

A2. The 2-SLP model satisfies the fixed recourse property; in other words, the recourse matrix D is deterministic.

A3. There exists an $L > -\infty$ such that $L \leq h(x, \tilde{\omega}) < \infty$, almost surely.

A4. The cost coefficient vector d of subproblem (1b) is deterministic.

Note that, $h(x, \tilde{\omega}) < \infty$ under assumption (A3) implies that our 2-SLP model conforms with the relatively complete recourse assumption of SP. Further, a finite lower bound on the recourse value implies that the dual feasible set in non-empty, almost surely. We will denote outcomes of random variables using a superscript over the parameter's name. For instance ξ^j and C^j are equivalent to $\xi(\omega^j)$ and $C(\omega^j)$, respectively. The letter k will be reserved to denote the iteration counter of the algorithm.

2.1. Algorithmic Description

The principal steps of regularized SD are presented in Algorithm 1. We begin our description at iteration k when we have access to a candidate solution x^k, an incumbent solution $\hat{x}^k$ and the current approximation of first-stage objective function $f^k(x)$. The iteration begins by sampling an outcome ω^k of the random variable $\tilde{\omega}$ independently of all prior samples $\{\omega^j\}_{j=1}^{k-1}$. If this outcome is encountered for the first time, it is added to the collection of unique outcomes $\mathcal{O}^{k-1} := \{\omega^1, \omega^2, \ldots\}$ observed during the first $(k-1)$ iterations of the algorithm. The current candidate solution x^k and outcome ω^k are used to evaluate the recourse function $h(x^k, \omega^k)$ by solving the subproblem (1b) to optimality. The resulting optimal dual solution π_k^k is added to the collection of previously discovered dual vertices $\mathcal{V}_0^{k-1}$, i.e., $\mathcal{V}_0^k = \mathcal{V}_0^{k-1} \cup \{\pi_k^k\}$. Using the outcomes in $\mathcal{O}^k$, one may define a SAA function

$$F^k(x) = c^\top x + \frac{1}{k}\sum_{j=1}^{k} h(x, \omega^j) \tag{2}$$

which uses a sample size of k. In iteration k, the SD algorithm creates an approximation $f^k(x)$ which satisfies $f^k(x) \leq F^k(x)$. In order to see how this is accomplished we make the following observations:

1. Under assumptions (A2) – (A4), the dual feasible region $\Pi_0 := \{\pi \mid D^\top \pi \leq d\}$ is a non-empty deterministic set.

2. Linear programming duality ensures that a $\pi \in \mathcal{V}_0^k$ satisfies $\pi^\top[\xi(\tilde{\omega}) - C(\tilde{\omega})x] \leq h(x, \tilde{\omega})$, almost surely for all $x \in \mathcal{X}$.

Thus, we identify a dual vertex in $\mathcal{V}_0^k$ which provides the best lower bounding function for each summand in the SAA function of (2) as follows:

$$\pi_j^k \in \arg\max\{\pi^\top[\xi^j - C^j x^k] \mid \pi \in \mathcal{V}_0^k\} \qquad \forall\ \omega^j \in \mathcal{O}^k, j \neq k. \tag{3}$$

Because of the need to associate the dual vertices with an outcome ω^j and the set $\mathcal{V}_0^k$, we use two indices for dual vertex π_j^k. We reserve the superscript (k) to indicate the iteration of the algorithm when the dual vertex is identified. The subscript (j), on the other hand, points to the outcomes in $\mathcal{O}^k$ for which the dual vertex is identified.

During our presentation of the algorithmic aspects, we use the function `argmax(·)` to represent the procedure in (3). Since π_k^k is the optimal dual solution of the subproblem in (1b) with (x^k, ω^k) as input, it can be used to compute the best lower bound for $h(x, \omega^k)$,

and hence, invoking the `argmax(·)` procedure is not necessary for $j = k$. These dual vertices $\{\pi_j^k\}$ are used to compute the subgradient β_k^k and the intercept term α_k^k of an affine function $\ell_k^k(x)$ as follows:

$$\alpha_k^k = \frac{1}{k}\sum_{j=1}^{k}(\pi_j^k)^\top \xi^j; \qquad \beta_k^k = c - \frac{1}{k}\sum_{j=1}^{k}(C^j)^\top \pi_j^k. \tag{4}$$

Similar calculations with respect to the incumbent solution yields another affine function $\hat{\ell}_k^k(x) = \hat{\alpha}_k^k + (\hat{\beta}_k^k)^\top x$, where $\hat{x}^k$ replaces x^k in (3) and the corresponding dual multipliers are used to obtain $(\hat{\alpha}_k^k, \hat{\beta}_k^k)$ in (4). Both the candidate affine function $\ell_k^k(x)$ and the incumbent affine function $\hat{\ell}_k^k(x)$ provide a lower bounding approximation of the SAA function in (2). Note that the SAA function uses a larger collection of outcomes as the algorithm proceeds, and hence, an affine function ℓ_j^j generated at an earlier iteration $j < k$ provides a lower bounding approximation of the SAA function $F^j(x)$, but not necessarily for $F^k(x)$. Consequently, the affine functions generated in earlier iterations need to be adjusted such that they continue to provide lower bound to the current SAA function $F^k(x)$. Under assumption (A3), the adjustment can be achieved using the following recursive calculations:

$$\alpha_j^k \leftarrow \frac{k-1}{k}\alpha_j^{k-1} + \frac{1}{k}L, \qquad \beta_j^k \leftarrow \frac{k-1}{k}\beta_j^{k-1} \qquad j = 1, \ldots k-1. \tag{7}$$

The most recent piecewise linear approximation $f^k(x)$ is defined as the maximum over the new affine functions ($\ell_k^k(x)$ and $\hat{\ell}_k^k(x)$) and the adjusted old affine functions ($\ell_j^k(x)$ for $j < k$). This update is carried out in Step 11 of Algorithm 1. The next candidate solution is obtained using this updated function approximation. The cut formation is performed in the function `form_cut(·)`, and the updates are carried out in `update_cuts(·)`.

We postpone the discussion on implementational details until §4, and the details regarding subroutines `form_cut(·)`, `update_cuts(·)`, `update_prox(·)` and `argmax(·)` are in **Appendix B**. In this paper, we do not discuss the details regarding incumbent update and stopping rules (`incumbent_update(·)` and `check_optimality(·)`, respectively). These details can be found in earlier works, in particular Higle and Sen (1999) and Sen and Liu (2016). We end this section by drawing attention to two salient features of the SD algorithm.

- *Dynamic Sample Size Selection*: When using the SAA function, a key question is to determine a sample size so that the solution to the sampled instance is an acceptable approximation to the true problem. The works of Shapiro and Homem-de Mello (1998), Kleywegt et al. (2002) offer some guidelines using measure concentration approaches. While the theory

Algorithm 1 Regularized SD Algorithm

Input: Scalar parameters $\sigma > 0$, $q > 0$, and $\tau > 0$.

Initialization: Solve a mean value problem† to obtain $x^1 \in \mathcal{X}$. Initialize $\mathcal{O}^0 = \emptyset$, $\mathcal{V}^0 = \emptyset$, $\hat{x}^0 = x^1$, $\sigma^0 = \sigma$ and $k \leftarrow 0$. Set `optimality_flag` to `FALSE`.

while `optimality_flag = FALSE` **do**

$k \leftarrow k + 1$.

Generate an outcome ω^k by sampling independently from all previous outcomes, and update $\mathcal{O}^k \leftarrow \mathcal{O}^{k-1} \cup \{\omega^k\}$.

Setup and solve linear programs to evaluate $h(x^k, \omega^k)$ and $h(\hat{x}^{k-1}, \omega^k)$.

Obtain optimal dual solutions π_k^k and $\hat{\pi}^k$, and update $\mathcal{V}_0^k \leftarrow \mathcal{V}_0^{k-1} \cup \{\pi_k^k, \hat{\pi}_k^k\}$.

Include new affine functions:

$\ell_k^k(x) =$ `form_cut(`$\mathcal{V}_0^k, x^k, k$`);` $\hat{\ell}_k^k(x) =$ `form_cut(`$\mathcal{V}_0^k, \hat{x}^{k-1}, k$`);`

Update previously generated affine functions $\{\ell_j^k\}_{\forall j} =$ `update_cuts(`$\{\ell_j^{k-1}\}_{\forall j}, k$`);`

Update the first-stage objective function approximation as:

$$f^k(x) = \max_{j=1,\ldots,k-1} \{\ell_j^k(x), \ell_k^k(x), \hat{\ell}_k^k(x)\}. \tag{5}$$

$\hat{x}^k =$`incumbent_update (`$f^k, f^{k-1}, \hat{x}^{k-1}, x^k$`);`

$\sigma^k =$`update_prox(`$\sigma^{k-1}, \|x^k - \hat{x}^{k-1}\|, x^k, \hat{x}^k$`);`

Obtain the next candidate solution by solving the following regularized master problem:

$$x^{k+1} \in \arg\min\{f^k(x) + \frac{\sigma^k}{2}\|x - \hat{x}^k\|^2 \mid x \in \mathcal{X}\}. \tag{6}$$

`optimality_flag = check_optimality(`$\hat{x}^k, \{\ell_j^k(x)\}$`);`

end while

of SAA recommends a sample size for a given level of accuracy, such sample sizes are known to be conservative, thus prompting a manual trial-and-error strategy of using an increasing sequence of sample sizes. It turns out that the computational demands for such a strategy can easily outstrip computational resources. Perhaps, the best known computational study applying the SAA function to some standard test instances was reported in Linderoth et al. (2006) where experiments had to be carried out on a computational grid with hundreds of

nodes. The SD algorithm allows for simulation of a new outcome concurrently with the optimization step. This avoids the need to determine the sample size ahead of the optimization step. Further, the SD stopping criteria include both in-sample and out-of-sample procedures to estimate solution quality. These procedures together allow SD to dynamically determine the statistically adequate sample size which can be sufficiently stringent to avoid premature termination.

- *Variance Reduction*: In any iteration, the SD algorithm builds affine lower bounding functions ℓ_k^k and $\hat{\ell}_k^k$, created at the candidate solution x^k and the incumbent solution $\hat{x}^k$, respectively. Both these functions are created using the same stream of outcomes. This notion of using common random numbers is commonly exploited in simulation studies, and is known to provide variance reduction in function estimation. In addition to that, Sen and Liu (2016) introduced the concept of "compromise decision" within the SD framework which uses multiple replications to prescribe a concordant solution across all replications. This replication process reduces both variance and bias in estimation, and therefore, the compromise decision provides more reliable estimates of the optimal value than solutions obtained in individual replications (see Theorem 2 of Sen and Liu (2016)).

3. SD for Recourse with Random Cost Coefficients

In this section we will remove assumption (A4) and allow the cost coefficients of subproblem (1b) to be uncertain. We will, however, retain assumptions (A1) – (A3). More specifically, we can think of the random variable $\tilde{\omega}$ as inducing a vector $(\xi^j, C^j, d^j)^\top$ associated with each outcome ω^j. As a result, the dual of subproblem (1b) with (x, ω^j) as input is given as

$$\max \ \{\pi^\top[\xi^j - C^j x] \mid D^\top \pi \leq d^j\}. \tag{8}$$

Notice that the dual polyhedron depends on the outcome d^j associated with observation ω^j. This jeopardizes the dual vertex re-utilization scheme adopted by SD to generate the best lower bounding function in (3) as all elements of the set $\mathcal{V}_0^k$ may not be feasible for observation ω^j. The main effort in this section will be devoted to designing a decomposition of dual vectors such that the calculations for obtaining lower bounding approximations and for establishing feasibility of dual vectors can be simplified. To do so, we first leverage (A2) to identify a

† The mean or expected value problem is a deterministic optimization problem obtained by replacing all random variables in (1) by their expected values.

finite collection of basis submatrices of D. Since D is deterministic, the basis submatrices are also deterministic. Secondly, without loss of generality, we represent a random vector as the sum of its deterministic mean and a stochastic shift vector. Consequently, the optimal solution of (8) can be decomposed into a deterministic component and a stochastic shift vector. While our presentation will be focused on general constraint matrix D, certain special structured matrices (e.g., total unimodular, diagonal scaling etc.) provide ample opportunity for specialized implementation. In the following, we will present the general procedure in greater detail.

3.1. Sparsity Preserving Decomposition of Second-Stage Dual Vectors

An important notation used frequently here is the index set of basic variables B^j, and a collection of such index sets $\mathcal{B}^k$. When subproblem (1b) is solved to optimality in iteration k, a corresponding optimal basis is discovered. We record the indices of the basic variables B^k into the collection of previously discovered index sets as: $\mathcal{B}^k = \mathcal{B}^{k-1} \cup B^k$. We will use D_B to denote the submatrix of D indexed by B.

In the k^{th} iteration, the optimal dual solution π_k^k and basis D_{B^k} of the subproblem (1b) with (x^k, ω^k) as input satisfies

$$D_{B^k}^\top \pi_k^k = d_{B^k}^k. \tag{9}$$

Letting $\bar{d} = \mathbb{E}[d(\tilde{\omega})]$, we can express the k^{th} outcome of cost coefficient in terms of the expected value $\bar{d}$ and deviation from the expectation $\delta(\omega^k)$ (or simply δ^k) as

$$d^k = \bar{d} + \delta^k. \tag{10}$$

Using the decomposed representation of cost coefficients (10) for only the basic variables in (9) yields the optimal dual vector as solution which has the following form

$$\pi_k^k = (D_{B^k}^\top)^{-1} \bar{d}_{B^k} + (D_{B^k}^\top)^{-1} \delta_{B^k}^k. \tag{11}$$

This allows us to decompose the dual vector into a two components $\pi_k^k = \nu^k + \theta_k^k$, where

$$\nu^k = (D_{B^k}^\top)^{-1} \bar{d}_{B^k}, \qquad \theta_k^k = (D_{B^k}^\top)^{-1} \delta_{B^k}^k. \tag{12}$$

While the deterministic component ν^k is computed only once for each newly discovered basis D_{B^k}, the stochastic component θ_k^k (shift vector) depends on the basis D_{B^k} as well as observation ω^k. Our motivation to decompose the dual vector[‡] into these two components rests

[‡] The dual vector π_j^i and its components (ν^i, θ_j^i) are indexed by j and i. The subscript j denotes the observation ω^j and the superscript i denotes the index set B^i associated with the dual vector.

on the sparsity of deviation vector $\delta^j_{B^i}$ associated with previous ($j = 1, \ldots, k-1$) and future observations $\omega^j,\ j > i$. Only basic variables with random cost coefficients can potentially have a non-zero element in vector $\delta^j_{B^i}$. When the second-stage has a network structure, the above calculations are far more streamlined because one can take advantage of the tree structure associated with a basis. Such specializations can significantly reduce both computational effort as well as storage.

3.2. Description of the Dual Vertex Set

Let us now apply the above decomposition idea to describe the set of dual vectors associated with an observation ω^j and all the index sets encountered until iteration k. We will denote this set by $\tilde{\mathcal{V}}^k_j$. Since for every basis D_{B^i} the deterministic component ν^i is computed only when it is discovered, our main effort in describing the set of dual vertices is in the computation of the stochastic component θ^i_j (see (12)).

For an index set B^i, let $\tilde{B}^i \subseteq B^i$ index the basic variables whose cost coefficients are random, and e_n denote the unit vector with only the n^{th} element equal to 1 and the remaining elements equal to 0. Using these, we define a submatrix of basis inverse matrix $(D^\top_{B^i})^{-1}$ as follows:

$$\Phi^i = \{\phi^i_n = (D^\top_{B^i})^{-1} e_n \mid n \in \tilde{B}^i\}. \tag{13}$$

In essence, the matrix Φ^i is built using columns of the basis inverse matrix corresponding to basic variables with random cost coefficients. We will refer to these columns as shifting direction vectors. Using these shifting directions, the stochastic component can be computed as:

$$\theta^i_j = \Phi^i \delta^j_{\tilde{B}^i} = \sum_{n \in \tilde{B}^i} \phi^i_n \delta^j_n. \tag{14}$$

With every discovery of a new basis, we will compute not only the deterministic component ν^i, but also the shifting vectors Φ^i. These two elements are sufficient to completely represent $d_{B^i}(\omega^j)$, and consequently the dual vector π^i_j, associated with any observation ω^j. Using these, the dual vector set associated with an observation ω^j can be built as follows:

$$\tilde{\mathcal{V}}^k_j := \{\pi^i_j \mid \pi^i_j = \nu^i + \Phi^i \delta^j_{\tilde{B}^i},\ \forall i \in \mathcal{B}^k\}. \tag{15}$$

With every new observation of $\tilde{\omega}$ encountered, one can use the pre-computed elements ν^i and Φ^i to generate the set of dual vectors. This limits the calculation to computationally manageable sparse matrix multiplication in (14). We will present ideas for efficiently utilizing

the decomposition of dual vectors to directly compute coefficients of affine function $\ell(x)$ later in §4; before that however, we will discuss techniques used to address one additional factor, namely, determining feasibility of dual vectors in $\tilde{\mathcal{V}}_j^k$.

3.3. Feasibility of Dual Vectors

We are interested in building the set $\mathcal{V}_j^k \subseteq \tilde{\mathcal{V}}_j^k$ of feasible dual vectors associated with observation ω^j. When (A4) is in place, the cost coefficients are deterministic and the stochastic shift-vector $\delta^j = 0$ for all observations in $\mathcal{O}^k$. Therefore, every dual vector can be represented using only the deterministic component, i.e., $\pi_j^i = \nu^i$ for all j. In other words, for every element in the set $\mathcal{V}_0^k$ there is a unique basis which yields a dual vector $\nu^i = (D_{B^i}^\top)^{-1}\bar{d}_{B^i} = (D_{B^i}^\top)^{-1}d_{B^i}$ that is feasible for all observations in $\mathcal{O}^k$.

In the absence of (A4), a basis D_{B^i} encountered using an observation $\omega^{j'}$ may fail to yield a feasible dual vector for another observation ω^j for which $\delta^j \neq \delta^{j'}$. Therefore, we need to recognize whether an index set B^i will admit dual feasibility for a given observation ω^j. We formalize the dual feasibility test and necessary computations through the following theorem. In the following, we will use the subscript ω for the dual feasible set and the set of dual vertices (Π_ω and $\mathcal{V}_\omega$, respectively) instead of index j when the corresponding statements are true for any outcome $\omega \in \Omega$, and not just for observations $\omega^j \in \mathcal{O}^k$.

THEOREM 1. *Let $\Pi_0 = \{\pi \mid D^\top\pi \leq \bar{d}\}$ be a nonempty polyhedral set, and $\mathcal{V}_0$ be the set of vertices of Π_0. For a vector valued perturbation $\delta(\omega)$, let $\Pi_\omega = \{\pi \mid D^\top\pi \leq \bar{d}+\delta(\omega)\}$ represent the perturbed polyhedral set and $\mathcal{V}_\omega$ the set of vertices of Π_ω. Given a feasible index set B associated with a vertex of Π_0, the following statements are equivalent:*

1. *A vertex of Π_0, say $\nu \in \mathcal{V}_0$, can be mapped to a vertex of the perturbed polyhedron, Π_ω, as $\nu + \theta_\omega \in \mathcal{V}_\omega$ where $\theta_\omega = (D_B^\top)^{-1}\delta_B(\omega)$ is a parametric vector of ω.*
2. *$G\bar{d} + G\delta(\omega) \geq 0$, where $G = [-D_N^\top(D_B^\top)^{-1}, \boldsymbol{I}]$.*

PROOF. We will begin by first showing that 1 $\implies$ 2.

Since ν is a vertex of the deterministic polyhedron, Π_0, it is the solution of a system of equations given by $D_B^\top\pi = \bar{d}_B$. That is, $\nu = (D_B^\top)^{-1}\bar{d}_B$. Let N denote the index set of non-basic variables, and consequently D_N is the submatrix of the recourse matrix D formed by the columns corresponding to non-basic variables. Since the current basis D_B is also a basis

for perturbated polyhedron, Π_ω, a basic solution can be identified by solving the following system of equations:

$$\begin{aligned} & D_B^\top \pi = \bar{d}_B + \delta_B(\omega) = D_B^\top \nu + \delta_B(\omega), \\ \Leftrightarrow \quad & D_B^\top (\pi - \nu) = \delta_B(\omega), \\ \Leftrightarrow \quad & \pi = \nu + (D_B^\top)^{-1}\delta_B(\omega). \end{aligned}$$

The second equality is due to $\bar{d}_B = D_B^\top \nu$. Defining $\theta_\omega = (D_B^\top)^{-1}\delta_B(\omega)$, we can obtain a basic solution of perturbed polyhedron as $\pi = \nu + \theta_\omega$. This basic solution is feasible, i.e. $\pi \in \mathcal{V}_\omega$, if it also satisfies the following system:

$$\begin{aligned} & D_N^\top \pi \le \bar{d}_N + \delta_N(\omega) \\ \Leftrightarrow \quad & D_N^\top \nu + D_N^\top \theta_\omega \le \bar{d}_N + \delta_N(\omega) \\ \Leftrightarrow \quad & 0 \le [-D_N^\top (D_B^\top)^{-1}\bar{d}_B + \bar{d}_N] + [-D_N^\top (D_B^\top)^{-1}\delta_B(\omega) + \delta_N(\omega)] \\ & \quad = [-D_N^\top (D_B^\top)^{-1}, \boldsymbol{I}][\bar{d}_B^\top, \bar{d}_N^\top]^\top + [-D_N^\top (D_B^\top)^{-1}, \boldsymbol{I}][\delta_B(\omega)^\top, \delta_N(\omega)^\top]^\top \\ & \quad = G\bar{d} + G\delta(\omega) \end{aligned}$$

where, $G = [-D_N^\top (D_B^\top)^{-1}, \boldsymbol{I}]$ with $\boldsymbol{I}$ as an $(n_2 - m_2)$ dimensional identity matrix. Therefore, $\pi = \nu + \theta_\omega \in \mathcal{V}_\omega$ implies that $G\bar{d} + G\delta(\omega) \ge 0$.

In order to show that 2 $\implies$ 1, we can start with the definition $G = [-D_N^\top (D_B^\top)^{-1}, \boldsymbol{I}]$ and the relation $G(\bar{d} + \delta(\omega)) \ge 0$ to obtain the following inequality:

$$[-D_N^\top (D_B^\top)^{-1}, \boldsymbol{I}][\bar{d}_B + \delta_B(\omega), \bar{d}_N + \delta_N(\omega)]^\top \ge 0.$$

By rearranging the terms we get

$$D_N^\top [(D_B^\top)^{-1}\bar{d}_B + (D_B^\top)^{-1}\delta_B(\omega)] \le \bar{d}_N + \delta_N(\omega) = d_N. \tag{16a}$$

Define $\pi = (D_B^\top)^{-1}\bar{d}_B + (D_B^\top)^{-1}\delta_B(\omega)$ and note that

$$D_B^\top \pi = [D_B^\top (D_B^\top)^{-1}]\bar{d}_B + [D_B^\top (D_B^\top)^{-1}]\delta_B(\omega) = \bar{d}_B + \delta_B(\omega) = d_B. \tag{16b}$$

From (16a) and (16b) we can conclude that π is a vertex of Π_ω. Q.E.D.

The implication of the above theorem is that the feasibility of a basis D_B associated with an index set $B \in \mathcal{B}^k$ with respect to an observation $\omega \in \mathcal{O}^k$ can be established by checking if the following inequality is true:

$$[-D_N^\top (D_B^\top)^{-1}, \boldsymbol{I}][\delta_B^\top(\omega), \delta_N^\top(\omega)]^\top \geq \bar{g} \tag{17}$$

where $\bar{g} = [D_N^\top (D_B^\top)^{-1}, -\boldsymbol{I}]\bar{d}$. Once again note that the term on the right-hand side of (17) is a product of a dense matrix and a vector which only needs to be calculated when the basis is discovered for the first time. Moreover, the term $D_N^\top (D_B^\top)^{-1}$ is the submatrix of the tableau formed by the non-basic variables when the simplex algorithm is employed. The left-hand side term of (17) is a product of a dense matrix and a sparse vector which can be carried out efficiently even for a large number of observations. These calculations are further streamlined based on the position of the variable with random cost coefficient in the basis. In this regard, the following remarks are in order.

Remark 1. Suppose that the index set $\tilde{B}^i = \emptyset$, that is, the basic variables associated with index set B^i have deterministic cost coefficients (equivalently, the variables with random cost coefficients are all non-basic), then all the elements of $\delta_B(\omega)$ are zeros and only $\delta_N(\omega)$ has non-zero elements. In such a case, the calculations on left-hand side of (17) for feasibility check yield:

$$[-D_N^\top (D_B^\top)^{-1}, \boldsymbol{I}][\delta_B^\top(\omega), \delta_N^\top(\omega)]^\top = [-D_N^\top (D_B^\top)^{-1}, \boldsymbol{I}][\mathbf{0}^\top, \delta_N^\top(\omega)]^\top = \delta_N(\omega). \tag{18}$$

Consequently, we can verify dual feasibility by checking if $\delta_N(\omega) \geq \bar{g}$. Since any feasible outcome lies inside an orthant, this check can be carried out very efficiently.

Remark 2. If $\tilde{B}^i \neq \emptyset$, i.e., at least one basic variable associated with index set B^i has random cost coefficient, then we need to explicitly check the feasibility of the remapped basic solutions using the inequality (17). Nevertheless, one can take advantage of sparsity of these calculations.

Remark 3. The amount of calculations necessary to compute the dual vector and to establish its feasibility depends on the number of random elements in cost coefficient vector d. When d is fixed, every basis D_{B^i} is associated with a unique dual vector with stochastic component $\theta_i^j = 0$ which remains feasible for any outcome ω^j. In this case, the calculations reduce to those in the original SD algorithm. When the number of random elements of d is small, the dimension of stochastic component is also small. Consequently, calculations in

(12) and (17) impose only a marginal overhead beyond those involved for 2-SLPs with fixed d. Therefore, many of the computational advantages of the original SD are retained.

Remark 4. The advantages resulting from low dimensional randomness in d can also be obtained in some special cases. Suppose that we discover a deterministic matrix Θ (say of dimension $p_2 \times q_2$) such that $p_2 \ll q_2$, and $d(\omega) = \Theta^\top g(\omega)$, with a random vector $g(\omega) \in \mathcal{R}^{p_2}$. Then, we can re-write $d^\top(\omega)y = g(\omega)^\top \Theta y$. Such a structure is characteristic of statistical linear factor models that are used in many financial and economic applications (Tsay (2005)). By appending a second-stage constraint $z - \Theta y = 0$, we can redefine the second-stage problem with an objective function in the form $g(\omega)^\top z$. Here, $g(\omega)$ has only $p_2 \ll q_2$ elements, thus achieving the desired reduction in dimension of random elements in cost coefficients. Since the matrix $(\boldsymbol{I}, -\Theta)$ does not change with $\tilde{\omega}$, this transformation does not affect the fixed recourse structure required for SD. In other words, scalability of our approach may depend on how the model is constructed, and the modeler is well advised to minimize the number of second-stage variables which have random cost coefficients associated with them.

Remark 5. A very special case arises when one random variable may affect cost coefficients of multiple decision variables simultaneously, i.e., $p_2 = 1$. For example, the Federal Reserve's quarterly adjustment of interest rate might impact returns from multiple investments at the same time. Following an earlier remark, if the new decision variable z is non-basic, then the high dimensional orthant reduces to simple one dimensional check to see if the random outcome is within a bound. Alternatively, when the variable z is basic, the feasibility check requires us to verify whether the outcome belongs to a certain interval. Recall that $\phi_j = (D_B)^{-1} e_j$, $j \in \tilde{B}$, and in this special case $|\tilde{B}| = 1$. Hence,

$$[-D_N^\top (D_B^\top)^{-1}, I][\delta_B^\top(\omega), \delta_N^\top(\omega)]^\top = -D_N^\top \phi_j \delta_j(\omega) \geq \bar{g}, \tag{19}$$

which implies that $\delta_{lb} \leq \delta_j(\omega) \leq \delta_{ub}$ with

$$\delta_{lb} = \max_{i \in I_1} \{ \frac{\bar{g}_i}{(-D_N^\top \phi_j)_i} \} \quad \text{where} \quad I_1 = \{ i | (-D_N^\top \phi_j)_i > 0 \}$$
$$\delta_{ub} = \min_{i \in I_2} \{ \frac{\bar{g}_i}{(-D_N^\top \phi_j)_i} \} \quad \text{where} \quad I_2 = \{ i | (-D_N^\top \phi_j)_i < 0 \}.$$

Note that the subscript i in above the equations indicates the ith element (corresponds to a row in basis matrix) of the column vector.

Algorithm 2 SD for problems with random cost coefficients

Steps 1 – 6 in Algorithm 1.

7a: **if** $\omega^k \notin \mathcal{O}^{k-1}$ (observation generated for the first time) **then** initialize the set $\mathcal{V}_k^{k-1}$ as follows:

$$\mathcal{V}_k^{k-1} \leftarrow \{\nu^i + \theta_k^i \mid G^i\delta^k \geq \bar{g}^i, \theta_k^i = \Phi^i\delta^k_{\tilde{B}^i}, i \in \mathcal{B}^{k-1}\}.$$

7b: **end if**

7c: Obtain the optimal index set B^k, and add it to the collection $\mathcal{B}^k \leftarrow \mathcal{B}^{k-1} \cup \{B^k\}$.

7d: **if** $B^k \notin \mathcal{B}^{k-1}$ (basis encountered for the first time) **then** compute ν^k, Φ^k, G^k and $\bar{g}^k$ as:

$$\nu^k = (D_{B^k}^\top)^{-1}\bar{d}_{B^k} \qquad \Phi^k = \{\phi_j^k \mid j \in \tilde{B}^k\}$$
$$G^k = [-D_{N^k}^\top(D_{B^k}^\top)^{-1}, \boldsymbol{I}] \qquad \bar{g}^k = -G^k\bar{d}$$

where ϕ_j^k is defined as in (13).

7e: **for** $\omega^j \in \mathcal{O}^k$ **do** update $\mathcal{V}_j^{k-1}$ as follows:

$$\mathcal{V}_j^k \leftarrow \mathcal{V}_j^{k-1} \cup \{\nu^k + \theta_j^k \mid G^k\delta^j \geq \bar{g}^k, \theta_j^k = \Phi^k\delta^j_{\tilde{B}^k}\}.$$

7f: **end for**

7g: **end if**

8: $\ell_k^k(x) =$ `form_cut(`$\mathcal{B}^k, x, k$`);` $\hat{\ell}_k^k(x) =$ `form_cut(`$\mathcal{B}^k, \hat{x}, k$`);`

Steps 10 – 16 in Algorithm 1.

3.4. Revised Algorithm Description

Before closing this section, we summarize the modifications to the SD algorithm described in §2.1 that are necessary to address 2-SLPs with random cost coefficients. While book-keeping in SD of §2.1 was restricted to dual vertices, the revision proposed in this section mandates that we store the basis generating the dual vectors. Specifically, the modifications affect steps 7-9, and are summarized in Algorithm 2.

At the beginning of the iteration, we have a collection of index sets $\mathcal{B}^{k-1}$ and observations $\mathcal{O}^{k-1}$. Whenever a new observation ω^k is encountered, the set of feasible dual vectors $\mathcal{V}_k^{k-1}$ is initialized by computing the stochastic components θ_k^i using the parameter Φ^i and establishing the feasibility of $\pi_k^i = \nu^i + \theta_k^i$ corresponding to all index sets in $\mathcal{B}^{k-1}$. The feasibility of dual vectors is determined by checking (17), which uses parameters G^i and $\bar{g}^i$. Note that

the parameters used for these computations, viz., Φ^i, ν^i, G^i and $\bar{g}^i$, are precomputed and stored for each index set $B^i \in \mathcal{B}^{k-1}$.

Once the subproblem is solved, we may encounter a new index set B^k. This mandates an update of the sets of feasible dual vectors. In order to carry out these updates, we begin by computing the deterministic parameters ν^k, Φ^k, G^k and $\bar{g}^k$. These parameters are stored for use not only in the current iteration, but also when new observations are encountered in future iterations. For all observations $\omega^j \in \mathcal{O}^k$, the set of feasible dual vectors is updated with information corresponding to the new basis D_{B^k}. This is accomplished by computing the stochastic component θ_j^k, which in turn is used to compute the dual vector $\pi_j^k = v^k + \theta_j^k$, and then we include the dual vector in the set $\mathcal{V}_j^{k-1}$ only if it is feasible. This results in an updated set $\mathcal{V}_j^k$.

For all observations $\omega^j (j \neq k)$, the dual vertex that provides the best lower bounding affine function is identified from the corresponding collection of feasible dual vectors $\mathcal{V}_j^k$. This procedure is completed in `form_cut(·)` which is presented as **Subroutine 4** in **Appendix B**. To give a concrete sense of the calculations being carried out, we present an illustrative example in **Appendix A**. The revised SD algorithm uses the original procedure in Algorithm 1 with steps 7-9 updated as shown in Algorithm 2.

We close this section by noting that the revised algorithm retains the convergence properties of the original SD algorithm with some modifications. Notice that, due to the fixed recourse assumption (A2) there are only finitely many basis matrices $\mathcal{B}$ for the subproblem (1b). Consequently, the set of dual vertices $\tilde{\mathcal{V}}_\omega$ and the set of feasible dual vertices $\mathcal{V}_\omega$ associated with any outcome $\omega \in \Omega$ are also finite. With these observations, the main convergence result for the revised SD algorithm is stated in the following theorem.

THEOREM 2. *For a 2-SLP model that satisfies assumptions* (A1)-(A3) *and* $\sigma^k > 0$*, the revised SD algorithm generates a sequence* $\{\hat{x}^k\}$ *that asymptotically converges to an optimal solution, with probability one.*

PROOF. Under assumption (A3), the dual feasible set Π_ω is non-empty and bounded with probability one. Further, the recourse function is the value of a linear program. These imply that $h(x, \omega)$ is a continuous function in $x \in \mathcal{X}$ and bounded. By construction, the collection of index sets satisfy $\mathcal{B}^k \subseteq \mathcal{B}^{k+1} \subseteq \mathcal{B}$. For a given ω, this set of feasible dual vectors $\mathcal{V}_\omega^k$ and $\mathcal{V}_\omega^{k+1}$ built using collections $\mathcal{B}^k$ and $\mathcal{B}^{k+1}$, respectively, satisfy $\mathcal{V}_\omega^k \subseteq \mathcal{V}_\omega^{k+1} \subseteq \mathcal{V}_\omega$. Here

$\mathcal{V}_\omega$ is the set of vertices of Π_ω. If we define $h^k(x,\omega) = \max\{\pi^\top(\xi(\omega) - C(\omega)x) \mid \pi \in \mathcal{V}_\omega^k\}$, then we have $h^k(x,\omega) \leq h^{k+1}(x,\omega) \leq h(x,\omega)$ for all k and for all $(x,\omega) \in \mathcal{X} \times \Omega$. Observing that $\{h^k\}$ is a monotonically increasing sequence of continuous functions bounded from above by the finite function h, leads us to the same conclusion as Lemma 1 in Higle and Sen (1991). The rest of the proof follows from Theorem 5 in Higle and Sen (1994) and Theorem 1 in Sen and Liu (2016). Q.E.D.

4. Implementational Details

The SD algorithm is designed to effectively record information that is relevant across different realizations of the random variable, and hence, is shareable. This results in significant computational savings for the SD algorithm. However, one cannot completely take advantage of its features without appropriate implementation of this algorithm. Therefore, our presentation of the algorithm would be incomplete without a detailed discussion of the implementation. In this section we present the data structures employed, in our implementation. Detailed description of the principal subroutines associated with checking feasibility of a dual vector, cut generation, cut updates and updating the proximal term are given in **Appendix B** as Subroutines 3-6, respectively. The details presented here expand upon those presented in Sen et al. (1994b) and Chapter 6 of Higle and Sen (1996).

Our implementation relies on decomposing the information discovered during the course of the algorithm into a deterministic component $\bar{(\cdot)}$, and a stochastic component $\Delta_{(\cdot)}$ which captures the deviation from the deterministic component. We discussed the value of this decomposition in our calculations of dual vectors described in §3.1. Here, we will extend this idea to other components of the problem. Note that for most problems the number of parameters that are affected by the random variable are significantly smaller than the dimension of the subproblem. Therefore, our decomposition approach reduces the computations necessary by introducing sparsity. Recall that, the random variable affects the right-hand side ξ, technology matrix C and the cost coefficient vector d. These components can be written as:

$$d(\omega^j) = \texttt{d\_bar} + \texttt{d\_obs[j]}, \quad \xi(\omega^j) = \texttt{xi\_bar} + \texttt{xi\_obs[j]}, \quad C(\omega^j) = \texttt{C\_bar} + \texttt{C\_obs[j]}, \quad (20)$$

where, the deterministic components `d_bar`, `xi_bar` and `C_bar` are set to $\bar{d} = \mathbb{E}[d(\tilde{\omega})]$, $\bar{\xi} = \mathbb{E}[\xi(\tilde{\omega})]$ and $\bar{C} = \mathbb{E}[C(\tilde{\omega})]$, respectively. Note that for deterministic parameters, the stochastic components `*_obs[j]` $= 0$, for all observations ω^j. The strategy to decompose the random

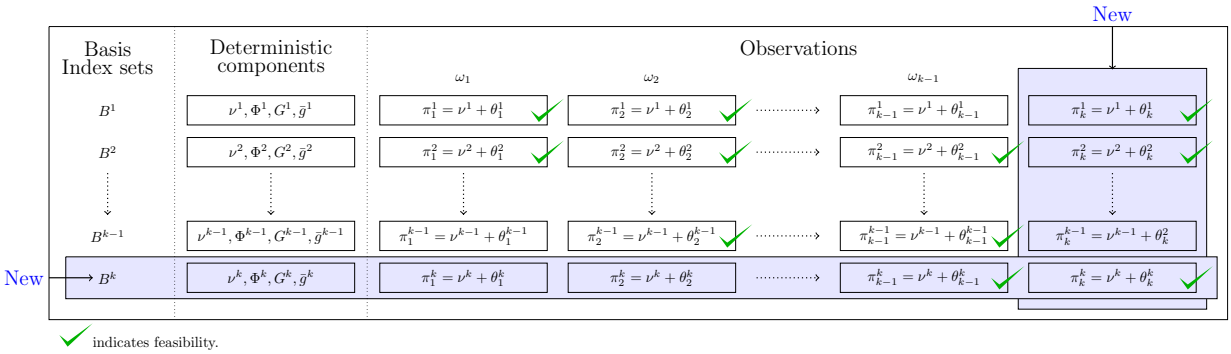


Figure 1 **Illustration of calculations to obtain the set of feasible dual vectors.**

elements into deterministic and stochastic components was also applied in Infanger and Morton (1996) to expedite computation of cut coefficients in a multistage setting. Our choice is driven by the SD data structures for computing coefficients corresponding to newly discovered outcomes (Sen et al. (1994b)), and determining feasibility of newly discovered dual vertices. **Figure 1** illustrates a information matrix that captures the evolution of information over the course of SD iterations. An element of this information matrix corresponds to an observation-index set pair, and contain all terms involving the pair that are necessary for computation during the course of the algorithm. Each iteration of SD may add one observation of $\tilde{\omega}$ and up to two new index sets to the collection $\mathcal{B}^k$ (one from solving subproblem with candidate solution as input, and the other from solving with incumbent solution as input). Calculations associated with each new observation of $\tilde{\omega}$ result in an additional column in the information matrix and discovery of a new basis results in a new row. Every basis is associated with an index set $\tilde{B}^i$ which is stored as `basis_idx[i]`.

4.1. Elements of the Information Matrix

These computations are carried out for the lightly shaded cells in the last column and row of the information matrix. There are two sets of elements that are stored in each cell of the information matrix.

The first set is associated with elements that are necessary to establish feasibility of a basis for a given observation ω^j. When a new basis is observed, these elements are necessary to establish feasibility of the basis not only with respect to observations of $\tilde{\omega}$ already encountered, but also for observations which the algorithm may discover in the future iterations. To see the development of these elements, notice that the inequality used to establish feasibility of a basis D_{B^i} associated with an index set $B^i \in \mathcal{B}^k$ can be rewritten as:

$$G^i\delta - \bar{g}^i = [G^i_{\tilde{B}^i}\delta_{\tilde{B}^i}, \delta_{\tilde{N}^i}]^\top - \bar{g}^i \geq 0.$$

Here, the calculations only involve columns from the constraint matrix corresponding to basic variables with random cost coefficients (indexed by $\tilde{B}^i$), and constant vector $\bar{g}^i$. These elements are stored as:

$$\bar{g}^i = \texttt{sigma\_gbar[i]}; \qquad G^i_{\tilde{B}^i} = \texttt{lambda\_G[i]}. \tag{21}$$

The first term is a floating point vector in $\mathbb{R}^{n_2}$ and the second term is a floating point sparse matrix in $\mathbb{R}^{m_2} \times \mathbb{R}^{q_2}$.

A critical step in computing the lower bounding affine functions in SD is the `argmax(·)` procedure whose mathematical description appears in (3). By introducing the decomposition of subproblem dual vectors into this calculations, we can restate the argument of (3) corresponding to a feasible basis D_{B^i} and observation ω^j as follows:

$$\begin{aligned}(\pi^i_j)^\top(\xi^j - C^j x) = \; & (\nu^i + \Phi^i \delta^j_{\tilde{B}^i})^\top[(\bar{\xi} + \Delta^j_\xi) - (\bar{C} + \Delta^j_C)x] \\ = \; & [(\nu^i)^\top \bar{\xi} - (\nu^i)^\top \bar{C}x] + [(\nu^i)^\top \Delta^j_\xi - (\nu^i)^\top \Delta^j_C x] + \\ & (\bar{\xi}^\top \Phi^i)\delta^j_{\tilde{B}^i} - (\bar{C}^\top \Phi^i)\delta^j_{\tilde{B}^i} x] + [(\Delta^j_\xi)^\top \Phi^i \delta^j_{\tilde{B}^i} - (\Delta^j_C)^\top \Phi^i \delta^j_{\tilde{B}^i} x]. \end{aligned} \tag{22}$$

The elements in the first bracketed term are computed only once when the basis is discovered:

$$(\nu^i)^\top \bar{\xi} = \texttt{sigma\_bbar[i][0]}; \qquad \bar{C}^\top \nu^i = \texttt{sigma\_Cbar[i][0]}. \tag{23}$$

The first term is a floating point scalar, while the second is a sparse vector in $\mathbb{R}^{n_1}$. To facilitate the calculations in the second bracketed term, note that if the structures $\xi(\tilde{\omega})$ and $C(\tilde{\omega})$ are such that there are no random elements in row-m, then the mth element in the term computes to zero. Thus, we use λ^i_0 to denote a vector obtained by extracting only those components of ν^i that correspond to rows with random elements, and naturally stored as a sparse vector `lambda_rho[i][0]`.

Recall that Φ^i is formed by the columns of the basis matrix corresponding to variables with random cost coefficients (see (13)). We treat these columns in a manner similar to the deterministic component ν^i, and therefore, the terms $\bar{\xi}^\top \Phi^i$ and $\bar{C}^\top \Phi^i$ are concatenated to `sigma_bbar` and `sigma_Cbar`, respectively:

$$\bar{\xi}^\top \phi^i_n = \texttt{sigma\_bbar[i][n]}; \qquad \bar{C}^\top \phi^i_n = \texttt{sigma\_Cbar[i][n]}; \quad \forall n \in \tilde{B}^i. \tag{24}$$

This concatenation results in `sigma_bbar[i]` being a floating point vector in $\mathbb{R}^{q_2+1}$ and `sigma_Cbar[i]` a floating point sparse matrix in $\mathbb{R}^{n_2 \times (q_2+1)}$. For the calculations in the

last bracketed term, we use λ_n^i to denote a vector obtained by extracting the components of ϕ_n^i corresponding to rows with random elements, and store it as a sparse vector `lambda_rho[i][n]`. This results in `lambda_rho[i]` to be a sparse matrix in $\mathbb{R}^{q_2 \times (q_2+1)}$.

Given `lambda_rho[i]`, `sigma_bbar[i]` and `sigma_Cbar[i]`, the computation in (22) can be performed as follows:

$$\texttt{delta\_b[i][j]} = \texttt{lambda\_rho[i][0]} \times \texttt{xi\_obs[j]} + \sum_{\texttt{n=1}}^{\texttt{length(basis\_idx[i])}} (\texttt{sigma\_bbar[i][n]} + \texttt{lambda\_rho[i][n]} \times \texttt{xi\_obs[j]}) \times \texttt{d\_obs\_b[i][j]}; \tag{25a}$$

$$\texttt{delta\_C[i][j]} = \texttt{lambda\_rho[i][0]} \times \texttt{C\_obs[j]} + \sum_{\texttt{n=1}}^{\texttt{length(basis\_idx[i])}} (\texttt{sigma\_Cbar[i][n]} + \texttt{lambda\_rho[i][n]} \times \texttt{C\_obs[j]}) \times \texttt{d\_obs\_b[i][j]}; \tag{25b}$$

where, `d_obs_b[i][j]` is defined as a vector with only those elements of `d_obs[j]` indexed by `basis_idx[i]`, i.e. $\tilde{\delta}^j_{\bar{B}_i}$. All matrix multiplications in the above calculation are undertaken in the sparse manner. In the above notation, `delta_b` in (25a) define scalars, while `delta_C` in (25b) define vectors. Note that in a deterministic problem the deviation terms are all zero, and the above calculations reduce to zeros as well. The same will be case if the observation corresponds to the mean value scenario. Therefore, the terms `delta_b` and `delta_C` are necessary to account for variability in random observations.

The choice of the above data structure and calculation is motivated by the fact that `lambda_rho[i]`, `sigma_bbar[i]` and `sigma_Cbar[i]` require less memory than storing entire vectors ν^i and θ^i_j. The reason for this is that in 2-SLPs the size of the second-stage subproblem is significantly larger than the the number of first-stage decisions affecting the subproblem. This, along with the earlier note on the sparsity of deviation of observations from their means, leads to substantial savings in memory usage and required computations. Our implementation takes advantage of further savings achieved by sharing the elements of `lambda_rho[i][n]`, `sigma_bbar[i][n]` and `sigma_Cbar[i][n]` across the indices in $\mathcal{B}^k$ (this is possible when more than one basis matrix have the same column ϕ_n).

In summary, the updates to information matrix are carried out as follows:

1. Column update: Given a new observation ω^k, use (25) to compute the `delta_b[i][k]` and `delta_C[i][k]` for all bases $B^i \in \mathcal{B}^k$.

2. Row update: If a new basis D_{B^k} is seen, then compute the feasibility elements `sigma_gbar[k]` and `lambda_G[k]` in (21). For every element in `basis_idx[k]`, use (23) to compute `sigma_bbar[i][0]` and `sigma_Cbar[i][0]`, and for all $\omega^j \in \mathcal{O}^k$ compute `delta_b[i][j]` and `delta_C[i][j]` using (25).

Due to the dynamic nature of evolution of the information matrix, and therefore, the storage requirement of the SD algorithm, we emphasize the need for careful memory management during implementation of this algorithm.

To conclude, we summarize how the earlier implementation of SD compares with the current version. In 2-SLPs with deterministic cost coefficients, every dual vertex is associated with a unique basis and the index set $\tilde{B}^i = \emptyset$ (`basis_idx` is empty). Therefore, in earlier implementations of the SD algorithm (detailed in Sen et al. (1994b) and Chapter 6 of Higle and Sen (1996)), a cell of the information matrix can be viewed to be associated with a unique dual vertex (instead of a basis) and an observation. Therefore, elements of the cell included `sigma_bbar[i][0]` and `sigma_Cbar[i][0]` in (23), and `delta_b[i][j]` and `delta_C[i][j]` in (25). The dual vertex remains feasible with respect to all observations, and hence, there is no need for the feasibility check procedure. Finally, the calculations in (25) are restricted to only the first terms (without the summation). The original implementation and the updated implementation of the SD algorithm are available under the GNU license from the authors' Github repository as `v1.0` and `v2.0`, respectively USC3DLAB (2019).

5. Computational Experiments

In this section we will present our computational experience with the SD algorithm when applied to instances with random cost coefficients. Our experiments conducted on instances of three problems compare solutions obtained using the SD algorithm with a collection of solutions obtained by optimizing SAA functions with increasing sample size using the regularized L-shaped method. These experiments reveal that the SD algorithm (a) provides a higher quality solution for a given computational budget (time), and (b) takes lower computational time to provide a solution of desired quality. We will provide a brief description of the problems used in the experiments, the algorithm used to solve the SAA instances and the experimental setup before presenting the results in detail.

While there are many challenging instances of 2-SLPs where randomness affects the right-hand side vector ξ, there is a lack of real-scale instances with random cost coefficients.

While the SG-Portfolio problems (Frauendorfer et al. (1996)) have cost coefficients which are modeled as random variables, the available instances have a small set of realizations. Such small sample sizes, do not require the use of sampling-based approaches. Therefore, we modified three larger problems from the literature to include random cost coefficients. These problems are: `ssn_rc` – a telecommunication network resource allocation problem, `scft` – logistics/supply chain planning problem, and `transship` – a multilocation transshipment problem. Further details regarding these problems and their instances considered for our experiments are given in **Appendix C**.

Since 2-SLPs are predominantly solved using the L-shaped method, we use it as a benchmark to present our computational results for SD. We begin this section by briefly describing our implementation of the regularized L-shaped method (see Ruszczyński and Shapiro (2003) for details). The L-shaped method is applicable to problems where uncertainty is characterized by a finite number of realizations (a problem with random variables with discrete distribution or an SAA instance). As before, this set of realizations will be denoted by $\mathcal{O}$. In iteration k, we start with a candidate first-stage solution x^k obtained by solving a master program. We solve subproblems for all possible observations $\omega^j \in \mathcal{O}$ and obtain the optimal dual solution π_j^k. These are used to generate a lower bounding affine function:

$$\alpha_j^k + \beta_j^k x = c^\top x + \sum_{\omega^j \in \mathcal{O}} p^j \cdot (\pi_j^k)^\top [\xi^j - C^j x].$$

In contrast to the coefficients computed in SD (see (4)) which use relative frequency of realizations, the above calculations are based on given probabilities associated with the realizations. In this regard, the affine function always provides a "true" lower bound of the first-stage objective function in (1a), or (2) when a SAA function is employed, and does not warrant the update mechanism in (7). The algorithm is terminated when the gap between in-sample upper and lower bound is within an acceptable tolerance (set to a nominal value of 0.001 in our implementation).

Our implementation of the L-shaped method also provides the option of quadratic regularization in the first-stage. When regularization is used, the first-stage approximate problem includes a proximal term centered around an incumbent solution, and therefore, the candidate solutions are obtained by solving a quadratic program. The incumbent update rules and the stopping criteria are based on Ruszczyński and Shapiro (2003). This implementation is also available on the authors' Github repository USC3DLAB (2019). Both the regularized

L-shaped (RLS) and the SD algorithms are implemented in the C programming language on a 64-bit Intel core i7 − 4770 CPU at 3.4GHz ×8 machine with 32 GB memory. All linear and quadratic programs were solved using CPLEX 12.8 callable subroutines (CPL (2018)).

5.1. Experimental Setup:

We use an experimental setup similar to that introduced by Mak et al. (1999) where lower and upper bounds are estimated through multiple replications. For the SAA procedure, we generate M independent samples $\{\omega^{j,m}\}_{j=1}^{N}$, each of size N. These samples are generated using the standard Monte-Carlo simulation and are used to set up the SAA function which is then solved using the RLS method. Note that variance reduction techniques, such as Latin Hypercube Sampling, can be employed to reduce variance of the SAA estimates (see Linderoth et al. (2006)). The optimal objective function values are denoted as $f_N^{*,m}$ with corresponding solutions as $x_N^{*,m}$. We compute the statistical lower bound to the true optimal value f^* and the $(1-a)$-confidence interval associated with this lower bound as:

$$L_N := \frac{1}{M}\sum_{m=1}^{M} f_N^{*,m}; \qquad CI_L = \left[L_N - \frac{z_{a/2}s_L}{\sqrt{M}}, L_N + \frac{z_{a/2}s_L}{\sqrt{M}}\right]. \tag{26}$$

Here, z_a is the $(1-a)$ quantile level of standard normal distribution and s_L is the sample variance of lower bounds $\{f_N^{*,m}\}$.

An upper bound estimate is computed by evaluating the function at solutions obtained from individual replications, i.e., $x_N^{*,m}$ for $m = 1, \ldots, M$. The functions are evaluated using a different set of samples $\{\hat{\omega}^{j,m}\}_j$, generated independently from those used for optimization as well as those used in evaluating solutions from other replications. The estimate of the function and the variance associated with it is given by:

$$f_{EV,N'}(x) = \frac{1}{N'}\sum_{j=1}^{N'}[c^\top x + h(x,\omega^j)], \qquad s_{EV} = \frac{1}{(N'-1)}\sum_{j=1}^{N'}[c^\top x + h(x,\omega^j) - f_{EV,N'}(x)]^2 \tag{27}$$

The estimation is terminated when s_{EV} is within an acceptable threshold ϵ. The final estimation value is denoted as $f_{EV}(x)$.

The function estimates at solutions obtained by optimizing the SAA function in different replication can in turn be used to obtain an estimate of the upper bound as follows:

$$U_N := \frac{1}{M}\sum_{m=1}^{M} f_{EV}(x_N^{*,m}); \qquad CI_U = \left[U_N - \frac{z_{a/2}s_M}{\sqrt{M}}, U_N + \frac{z_{a/2}s_M}{\sqrt{M}}\right] \tag{28}$$

We also build a confidence interval around this estimation as shown above. Finally we define a pessimistic gap as the difference between the upper limit of CI_U and the lower limit of CI_L. While the pessimistic gap is a reliable indicator of optimality, in some cases it can be too demanding to reduce it beyond a certain level. Nonetheless, we will use this gap as a measure for solution quality – a lower value indicates higher quality solution. For our experiments with the SAA function, we calculated values $f_N^{*,m}$ for $m = 1, \ldots, M$ and varying number of observations N in the SAA function. The function estimation for computing the upper bound is terminated when $s_{EV}/f_{EV,N'} < \epsilon = 0.01$.

The SD algorithm does not require a preset number of observations, but instead dynamically determines the number of outcomes necessary for obtaining statistically optimal solutions based on the progress made. As in the SAA experiments, optimization is performed over M replications. In each replication, statistical optimality of a solution is established using an In-Sample and an Out-of-Sample stopping rule (see Higle and Sen (1999) and Sen and Liu (2016) for details). These rules basically establish whether the approximations have sufficiently stabilized based on a given tolerance level. In our experiments we execute all the instances using three increasingly tighter relative tolerances, i.e., loose (0.01), nominal (0.001), and tight (0.0001).

In our experiments, we set the number of replications $M = 30$ for both the SAA and SD experiments. We use two sets of M random number seeds, the first set is used to generate the samples to be used in the optimization step and the second set is used for evaluation.

5.2. Numerical Results

The results from the SAA experiments with varying sample sizes solved using RLS and the SD experiments with varying tolerance values are reported in **Table 1**, **Table 2**, **Table 3** and **Table 4**. The tables present the average lower bound (26) and upper bound (28) along with half widths of their corresponding 95% confidence intervals (parenthetical values); pessimistic gap as absolute value and as fraction of its respective average lower bound (computed as pessimistic gap over average lower bound); and average computational time with its standard deviation.

The `ssn_rc0` instance is the derivative of SSN which is one of the early 2-SLP models with deterministic cost coefficients. SSN has been used in past to study scalability of sampling-based methods (for example, Linderoth et al. (2006), Sen and Liu (2016)), and therefore, we begin our discussion with results for this model. Firstly, the computational results with RLS

Algorithm	Sample size (std.dev.)	Lower Bound (95% CI)	Upper Bound (95% CI)	Pessimistic Gap	Avg. Time (s) (std.dev.)
RLS-Nominal	50	4.46 (± 0.80)	13.63 (± 0.36)	10.32 (231.24%)	12.54 (4.41)
	100	6.53 (± 0.64)	12.18 (± 0.20)	6.49 (99.52%)	31.54 (8.05)
	500	9.06 (± 0.39)	10.43 (± 0.07)	1.84 (20.30%)	470.34 (208.88)
	1000	9.56 (± 0.25)	10.15 (± 0.04)	0.88 (9.17%)	1698.00 (806.79)
	5000	9.85 (± 0.10)	9.95 (± 0.01)	0.22 (2.20%)	30800.76 (15187.34)
SD-Loose	1567 (± 286)	9.59 (± 0.22)	10.21 (± 0.05)	0.89 (9.27%)	38.91 (17.60)
SD-Nominal	2315 (± 251)	9.72 (± 0.13)	10.14 (± 0.04)	0.59 (6.03%)	103.86 (30.48)
SD-Tight	3318 (± 670)	9.88 (± 0.11)	10.12 (± 0.04)	0.39 (3.98%)	299.02 (177.00)

Table 1 Optimal Objective Value and Optimality Gap Estimates for `ssn_rc0` instances

reveal that the pessimistic gap can be unacceptably high for small sample SAA functions. In fact, the pessimistic gap does not fall below 10% until one uses $N = 1000$. A comparable solution was obtained using SD with loose tolerance within about 2.3% of the computational time used by RLS*.

Across all the instances, the SAA results indicate that the quality of function estimates and solutions improve as the sample size used in creating the SAA function increases. This is clear by noting an overall reducing trend of the pessimistic gap with increasing sample size†. These results computationally verify the theoretical properties of the SAA function. However, as the sample size increases, a larger number of subproblems are solved in the second-stage during the execution of RLS, thus increasing the computational requirement. This is reflected in the increasing computational time with sample size N.

Our SAA results are comparable in quality (in term of bound estimates) to those reported in Linderoth et al. (2006). The use of a computational grid in the earlier study resulted in a wall clock time of $30 - 45$ minutes per replication for SAA instances with $N = 5000$. In comparison, our implementation on a single desktop computer takes in excess of 8.5 hours on an average to complete the replications. These observations underscore the fact that the SAA approach for challenging 2-SLP models requires significant computational time and/or a high-performance computing environment.

Given that the 2-SLPs are computationally very demanding, the re-use of previously discovered structures of an instance in future iterations provides considerable relief. This is the

* In certain cases – e.g., `transship` instances with $N \geq 100$ – a few replications of the RLS experiments take significantly longer than the others. This results in the large standard deviation in the average computational time as seen for $N = 100$ in **Table 4**.

† Since these are stochastic estimates which depend on randomly sampled quantities, one should not expect a strictly monotonic behavior. Moreover, the numbers reported are averages across multiple replications.

Algorithm	Sample size (std.dev.)	Lower Bound (95% CI)	Upper Bound (95% CI)	Pessimistic Gap	Avg. Time (s) (std.dev.)
			ssn_rc1		
RLS-Nominal	50	38.85 (± 8.58)	149.60 (± 5.78)	125.11 (321.99%)	21.82 (5.19)
	100	63.62 (± 6.48)	129.13 (± 3.21)	75.19 (118.20%)	43.69 (5.57)
	500	90.51 (± 4.70)	107.31 (± 0.91)	22.41 (24.76%)	212.49 (14.96)
	1000	93.51 (± 3.06)	104.56 (± 0.40)	14.51 (15.52%)	441.57 (25.41)
	5000	98.35 (± 1.38)	101.70 (± 0.14)	4.86 (4.94%)	2421.76 (123.20)
SD-Loose	1600 (± 266)	95.64 (± 1.95)	104.73 (± 0.56)	11.44 (11.96%)	84.31 (31.39)
SD-Nominal	2352 (± 410)	97.75 (± 1.62)	104.45 (± 0.56)	8.72 (8.92%)	221.41 (95.62)
SD-Tight	3305 (± 614)	99.30 (± 1.55)	103.93 (± 0.65)	6.68 (6.73%)	536.48 (242.50)
			ssn_rc2		
RLS-Nominal	50	19.75 (± 4.07)	73.40 (± 2.59)	60.30 (305.27%)	17.87 (4.40)
	100	30.38 (± 3.21)	62.36 (± 1.47)	36.67 (120.69%)	36.77 (5.79)
	500	44.92 (± 1.88)	51.83 (± 0.31)	9.10 (20.26%)	171.06 (9.36)
	1000	46.77 (± 1.02)	50.91 (± 0.22)	5.38 (11.50%)	359.77 (22.02)
	5000	48.29 (± 0.56)	49.65 (± 0.06)	1.98 (4.09%)	1904.31 (114.96)
SD-Loose	1759 (± 340)	47.54 (± 0.81)	51.24 (± 0.42)	4.83 (10.39%)	131.80 (74.75)
SD-Nominal	2628 (± 423)	48.44 (± 0.86)	50.92 (± 0.32)	3.58 (7.54%)	366.50 (147.59)
SD-Tight	3939 (± 704)	49.02 (± 0.59)	50.76 (± 0.34)	2.58 (5.44%)	1018.46 (476.64)
			ssn_rc3		
RLS-Nominal	50	12.55 (± 2.37)	48.11 (± 2.07)	40.01 (318.85%)	16.47 (2.63)
	100	20.02 (± 2.00)	40.65 (± 1.17)	23.80 (118.85%)	32.95 (4.18)
	500	29.27 (± 1.33)	32.99 (± 0.31)	5.36 (18.33%)	152.07 (9.83)
	1000	29.66 (± 0.93)	31.76 (± 0.11)	3.15 (10.63%)	298.02 (18.52)
	5000	30.55 (± 0.33)	30.95 (± 0.05)	0.77 (2.51%)	1556.83 (87.75)
SD-Loose	1833 (± 334)	31.82 (± 0.68)	32.86 (± 0.36)	2.08 (6.54%)	140.88 (56.61)
SD-Nominal	2648 (± 460)	31.99 (± 0.60)	32.59 (± 0.26)	1.46 (4.58%)	357.93 (152.25)
SD-Tight	3975 (± 662)	32.07 (± 0.52)	32.60 (± 0.28)	1.32 (4.11%)	1033.81 (441.54)
			ssn_rc4		
RLS-Nominal	50	8.67 (± 1.70)	35.36 (± 1.47)	29.87 (344.54%)	14.84 (2.97)
	100	14.86 (± 1.44)	30.25 (± 1.28)	18.11 (121.85%)	29.06 (4.20)
	500	20.64 (± 0.98)	23.89 (± 0.14)	4.37 (21.17%)	133.93 (12.03)
	1000	21.52 (± 0.57)	23.24 (± 0.11)	2.41 (11.18%)	267.68 (16.34)
	5000	22.17 (± 0.21)	22.56 (± 0.04)	0.65 (2.94%)	1337.37 (94.24)
SD-Loose	1797 (± 331)	23.71 (± 0.63)	24.64 (± 0.35)	1.91 (8.07%)	143.08 (69.65)
SD-Nominal	2798 (± 418)	23.99 (± 0.43)	24.56 (± 0.37)	1.37 (5.70%)	417.94 (156.96)
SD-Tight	3928 (± 634)	24.08 (± 0.42)	24.50 (± 0.37)	1.21 (5.01%)	1017.31 (428.80)

Table 2 Optimal Objective Value and Optimality Gap Estimates for ssn_rcG ($G = 1, \ldots, 4$) instances

case in the "sampling-on-the-fly" approach of SD, as the results indicate. The SD results for ssn_rc0 are comparable to those reported in Sen and Liu (2016) which used the previous version of SD implementation. The minor increase in the computational time can be attributed to additional overhead introduced by the data structures necessary to handle the

Algorithm	Sample size (std.dev.)	Lower Bound (95% CI)	Upper Bound (95% CI)	Pessimistic Gap	Avg. Time (s) (std.dev.)
`scft-6`					
RLS-Nominal	50	553806 (± 1900)	553536 (± 250)	1879 (0.34%)	0.14 (0.03)
	100	554129 (± 1659)	553408 (± 226)	1164 (0.21%)	0.25 (0.04)
	500	554032 (± 518)	553393 (± 220)	100 (0.02%)	0.94 (0.09)
	1000	553458 (± 325)	553393 (± 220)	480 (0.09%)	1.83 (0.22)
	5000	553278 (± 134)	553393 (± 220)	469 (0.08%)	8.51 (0.78)
SD-Loose	149 (± 13)	554812 (± 1409)	553598 (± 244)	438 (0.08%)	0.20 (0.02)
SD-Nominal	274 (± 10)	554391 (± 813)	553499 (± 262)	182 (0.03%)	0.40 (0.03)
SD-Tight	526 (± 5)	554239 (± 541)	553658 (± 269)	230 (0.04%)	0.85 (0.06)
`scft-46`					
RLS-Nominal	5	7391427 (± 120741)	7763122 (± 15146)	507582 (6.87%)	3954.39 (905.08)
	10	7486263 (± 72180)	7706001 (± 8586)	300503 (4.01%)	6506.03 (1003.11)
	25	7549045 (± 50789)	7659751 (± 6165)	167661 (2.22%)	10913.06 (1217.61)
	50	7582958 (± 35836)	7642093 (± 3769)	98740 (1.30%)	14268.55 (1298.53)
SD-Loose	288 (± 28)	7339518 (± 17206)	7648999 (± 3509)	330195 (4.50%)	25.24 (4.39)
SD-Nominal	430 (± 44)	7408929 (± 13428)	7647755 (± 3828)	256081 (3.46%)	53.12 (10.41)
SD-Tight	1282 (± 295)	7517501 (± 8061)	7642883 (± 4288)	137731 (1.83%)	373.78 (158.45)

Table 3 Optimal Objective Value and Optimality Gap Estimates for `scft` instances

Algorithm	Sample size (std.dev.)	Lower Bound (95% CI)	Upper Bound (95% CI)	Pessimistic Gap	Avg. Time (s) (std.dev.)
RLS-Nominal	10	2333.13 (± 3.95)	2341.36 (± 1.60)	13.78 (0.59%)	0.20 (0.02)
	20	2333.75 (± 2.99)	2338.99 (± 1.17)	9.40 (0.40%)	0.31 (0.04)
	30	2335.65 (± 2.10)	2337.93 (± 1.01)	5.38 (0.23%)	0.41 (0.04)
	40	2335.82 (± 1.79)	2337.69 (± 1.00)	4.66 (0.20%)	0.52 (0.04)
	50	2336.64 (± 1.64)	2337.38 (± 0.92)	3.30 (0.14%)	3.84 (2.20)
	75	2337.38 (± 1.35)	2337.17 (± 0.89)	2.02 (0.09%)	0.92 (0.07)
	100	2337.59 (± 0.91)	2337.01 (± 0.85)	1.18 (0.05%)	60.55 (164.51)
SD-Loose	168 (± 17)	2336.80 (± 0.85)	2336.88 (± 0.92)	1.86 (0.08%)	0.34 (0.06)
SD-Nominal	292 (± 18)	2336.44 (± 0.83)	2337.02 (± 0.77)	2.18 (0.09%)	0.86 (0.11)
SD-Tight	544 (± 18)	2337.00 (± 0.59)	2337.69 (± 0.88)	2.16 (0.09%)	2.70 (0.23)

Table 4 Optimal Objective Value and Optimality Gap Estimates for `transship` instances

updated information matrix described in §4.1. The increasing tightness in tolerance levels used in experiments with the SD algorithm reflect a more demanding requirement in terms of stability of approximation. This results in a higher computational time as indicated in the tables. An increased tolerance level also results in a more accurate estimation of the objective function, and therefore, the solution obtained from such a setting results in a lower pessimistic gap. In any case, our results show that SD provides high quality solutions in a reasonable amount of time using only ordinary computational resources.

Unlike deterministic optimization, we are more than satisfied to get a solution whose pessimistic gap is within an acceptable limit. This limit is dependent on the instance, for example, a 5% pessimistic gap may be considered acceptable for ill-conditioned problem instances such as `ssn_rcG`. As the results indicate, this is already a very high bar to set for SP algorithms. Thus, whenever we achieve 1% or less in pessimistic gap, we consider the instance as being solved. In light of this, it is only appropriate to compare SP algorithms and sample sizes in terms of the pessimistic gap of the solutions reported.

The computational advantage of SD is evident in the results of `ssn_rcG`, `scft` and `transship` instances. In these instances the results in the last column of the tables show that the time required for SD to generate a solution of comparable quality is significantly lower than the RLS method. For example, the solution obtained using SD with loose tolerance for `ssn_rc0` results in a pessimistic gap of 0.89 which is comparable to the solution obtained using the RLS method applied to a SAA function with sample size of $N = 1000$ (pessimistic gap is 0.88). However, the computational time of SD is lower (by a factor of 43) when compared to the computational time for RLS.

An SAA experiment involves executing multiple optimization steps using RLS, each time increasing the sample size used to build the SAA function if the solution is not acceptable and starting the optimization step from scratch. Therefore, the cumulative experiment time before a solution of desired quality is identified is always significantly higher than the computational time for SD to identify a solution of comparable quality. Moreover, the cumulative time depends on the sequence of sample sizes chosen for experiments (for example, $N = 50, 100, 500, 1000, 5000$ in our experiments for most instances), and there are no clear guidelines for this selection. On the other hand, when SD is executed with a certain tolerance level and if the desired solution quality is not attained, then the tolerance level can be increased and optimization can resume from where it was paused. This means that the additional computational time to achieve a higher quality solution (on average) is simply the difference between the computational times reported for different tolerance levels in our tables. Such "on-demand quality" solution is desired in many practical applications.

6. Conclusions

In this paper, we presented the algorithmic enhancements to SD to address fixed recourse 2-SLPs with random right-hand side components as well as random cost coefficients in the

second-stage. For problems with deterministic cost coefficients, SD re-utilizes previously discovered second-stage dual vertices to compute value function approximations in future iterations. However for problems with random cost coefficients, the feasibility of past dual vertices is not guaranteed. In order to address this issue, we treated the bases which generate the dual vertices as fundamental elements which can be reused in the future. We proposed a sparsity preserving decomposition of dual vectors into deterministic and stochastic components computed using these bases. We presented computationally efficient steps for establishing the feasibility of dual vectors and computing the value function approximations. We also described the data structures employed in our implementation of the algorithm which are designed to be applicable to real-scale problem instances. Our numerical experiments illustrated the computational edge of SD on a variety of test instances when compared to the SAA-based approach.

The multistage stochastic decomposition algorithm was presented in Sen and Zhou (2014) which was the first attempt to extend SD to a multistage setting. While the authors provide algorithmic details supported by strong analytical results, the computational implementation of this extension faces a critical challenge. When sequential sampling is employed in a multistage setting, the approximations of cost-to-go functions are stochastic in nature. Consequently, the optimization problem at a non-terminal stage has an objective function with random cost coefficients. The enhancements presented in this paper for the two-stage problems provide the foundation necessary to build a successful implementation of the multistage stochastic decomposition algorithm. This will be undertaken in our future research endeavors.

The notion of compromise decision was introduced in Sen and Liu (2016) for sampling-based convex optimization algorithms. The principal idea was to use function approximations and solutions generated during different replication of the algorithm to create a "grand-mean" or a compromise problem. The solution obtained by solving this problem, which is termed as the compromise decision, is shown to reduce bias and variance in function estimation, and provide a more reliable performance. They also show that the difference between an average solution (computed across all replications) and a compromise decision provides a natural stopping rule. The idea of compromise decisions are best implemented in a parallel environment. This is because all replications (conducted with different seeds for random number generation) should be continued until the stopping criterion, based on the difference

between average and compromise decision being small enough, is satisfied. Further bounds on objective function estimates can also be obtained as suggested in Deng and Sen (2018). In any event, since our tests were performed on a serial implementation, we did not adopt the compromise solution. Given the relevance of compromise decisions from practitioners point of view, honing the concept of compromise decision for 2-SLPs with random cost coefficients and adapting our implementation for parallel/high performance environment will also be an integral part of our future research.

Acknowledgments

We thank the referees for their careful review of our paper. The research was supported by an AFOSR grant FA 9550-15-1-0267, and an NSF grant 1822327.

References

(2018) *CPLEX Callable Library (C API) Reference Manual*. IBM, 12.8 edition.

Deng Y, Sen S (2018) Learning enabled optimization: Towards a fusion of statistical learning and stochastic programming. Technical report, University of Southern California.

Frauendorfer K, Härtel F, Reiff MF, Schürle M (1996) *SG-Portfolio Test Problems for Stochastic Multistage Linear Programming*, 102–107 (Berlin, Heidelberg: Springer Berlin Heidelberg), ISBN 978-3-642-80117-4, URL `http://dx.doi.org/10.1007/978-3-642-80117-4_18`.

Gangammanavar H, Sen S, Zavala VM (2016) Stochastic optimization of sub-hourly economic dispatch with wind energy. *IEEE Transactions on Power Systems* 31(2):949–959, ISSN 0885-8950, URL `http://dx.doi.org/10.1109/TPWRS.2015.2410301`.

Herer YT, Tzur M, YÃijcesan E (2006) The multilocation transshipment problem. *IIE Transactions* 38(3):185–200, URL `http://dx.doi.org/10.1080/07408170500434539`.

Higle JL, Sen S (1991) Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Mathematics of Operations Research* 16(3):650–669, ISSN 0364765X.

Higle JL, Sen S (1994) Finite master programs in regularized stochastic decomposition. *Mathematical Programming* 67(1-3):143–168, ISSN 0025-5610, URL `http://dx.doi.org/10.1007/BF01582219`.

Higle JL, Sen S (1996) *Stochastic Decomposition: A Statistical Method for Large Scale Stochastic Linear Programming* (Kluwer Academic Publishers, Boston, MA.).

Higle JL, Sen S (1999) Statistical approximations for stochastic linear programming problems. *Annals of Operations Research* 85(0):173–193, ISSN 0254-5330.

Infanger G, Morton DP (1996) Cut sharing for multistage stochastic linear programs with interstage dependency. *Mathematical Programming* 75(2):241–256, ISSN 0025-5610.

Kleywegt AJ, Shapiro S, Homem-de Mello T (2002) The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization* 12(2):479–502, URL http://dx.doi.org/10.1137/S1052623499363220.

Linderoth J, Shapiro A, Wright S (2006) The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research* 142(1):215–241, ISSN 0254-5330, URL http://dx.doi.org/10.1007/s10479-006-6169-8.

Mak W, Morton DP, Wood K (1999) Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters* 24(1):47 – 56, ISSN 0167-6377, URL http://dx.doi.org/https://doi.org/10.1016/S0167-6377(98)00054-6.

Ruszczyński A, Shapiro A (2003) Handbooks in Operations Research and Management Science: Stochastic Programming. *Surveys in Operations Research and Management Science;*, volume 10 (Elsevier).

Sen S, Doverspike RD, Cosares S (1994a) Network planning with random demand. *Telecommunication Systems* 3(1):11–30, ISSN 1018-4864, URL http://dx.doi.org/10.1007/BF02110042.

Sen S, Liu Y (2016) Mitigating uncertainty via compromise decisions in two-stage stochastic linear programming: Variance reduction. *Operations Research* .

Sen S, Mai J, Higle JL (1994b) *Solution of Large Scale Stochastic Programs with Stochastic Decomposition Algorithms*, 388–410 (Boston, MA: Springer US), ISBN 978-1-4613-3632-7, URL http://dx.doi.org/10.1007/978-1-4613-3632-7_19.

Sen S, Zhou Z (2014) Multistage Stochastic Decomposition: A bridge between Stochastic Programming and Approximate Dynamic Programming. *SIAM Journal on Optimization* 24(1):127–153.

Shapiro A, Homem-de Mello T (1998) A simulation-based approach to two-stage stochastic programming with recourse. *Mathematical Programming* 81(3):301–325, ISSN 1436-4646, URL http://dx.doi.org/10.1007/BF01580086.

Tsay R (2005) *Analysis of financial time series*. Wiley series in probability and statistics (Hoboken, NJ: Wiley-Interscience), 2. ed. edition, ISBN 978-0-471-69074-0, URL http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+483463442&sourceid=fbw_bibsonomy.

USC3DLAB (2019) Two-stage stochastic decomposition. https://github.com/USC3DLAB/SD.git.

Van Slyke RM, Wets RJB (1969) L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics* 17(4):638–663, ISSN 00361399.

You F, Wassick JM, Grossmann IE (2009) Risk management for a global supply chain planning under uncertainty: Models and algorithms. *AIChE Journal* 55(4):931–946, URL http://dx.doi.org/10.1002/aic.11721.

Appendix A: An Illustrative Example

We illustrate the random cost developments via an example which we refer to as the "Diamond-shaped" problem. Consider the following problem:

$$\min_{0\le x\le 5} \; f(x) = -0.75x + \mathbb{E}[h(x,\tilde{\omega})], \tag{29}$$

where $h(x,\omega)$ is the value of the following LP:

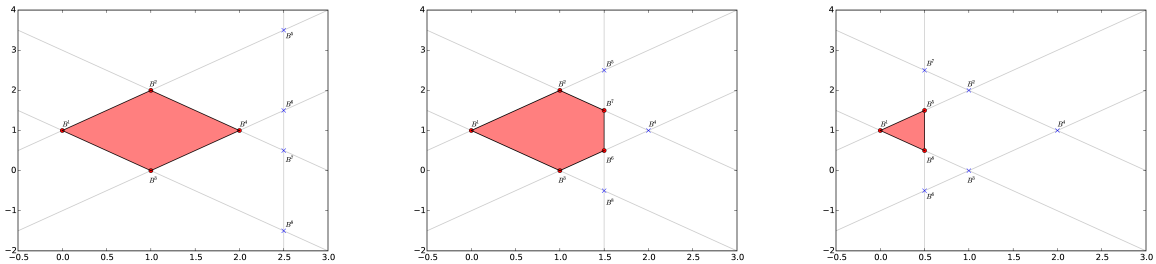
$$\begin{aligned} h(x,\omega) = \min \;& -y_1 + 3y_2 + y_3 + y_4 + \omega_5 y_5 \\ \text{s.t.} \;& -y_1 + y_2 - y_3 + y_4 + y_5 = \omega_6 + \frac{1}{2}x, \;\; -y_1 + y_2 + y_3 - y_4 = \omega_7 + \frac{1}{4}x, \;\; y_i \ge 0, \;\; i = 1,\ldots,5. \end{aligned} \tag{30}$$

The random variables $\tilde{\omega}_5, \tilde{\omega}_6$ and $\tilde{\omega}_7$ are assumed to be independent follow uniform distribution over the intervals $[0,3]$, $[-1,0]$ and $[0,1]$, respectively. The dual LP of the subproblem (30) is given by:

$$\begin{aligned} h(x,\omega) = \max \;& \left(\omega_6 + \frac{1}{2}x\right)\pi_1 + \left(\omega_7 + \frac{1}{4}x\right)\pi_2 \\ \text{s.t.} \;& -\pi_1 - \pi_2 \le -1, \;\; \pi_1 + \pi_2 \le 3, \;\; -\pi_1 + \pi_2 \le 1, \;\; \pi_1 - \pi_2 \le 1, \;\; \pi_1 \le \omega_5. \end{aligned} \tag{31}$$

Notice that the dual feasible region depends on the observation of the random variable $\tilde{\omega}$ through the constraint: $\pi_1 \le \omega_5$. We present all the calculations with respect to three observations of the random variable, viz., $\omega_5^1 = 2.5$, $\omega_5^2 = 1.5$, and $\omega_5^3 = 0.5$. The dual feasible region and the basic solutions of the subproblem dual LP (31) with respect to the three observations are shown in **Figure 2**. For each basis we compute ν^i, Φ^i, G^i and $\bar{g}^i$ as given in Line-3 of Algorithm 2. Recall that these values are computed on-the-fly, as and when the basis are discovered for the first time. We present these values in **Table 5**. In the matrix G^i, the columns associated with the variables with random cost coefficients (boldfaced in the table) are sufficient for the feasibility check (see (17)), and suffices to store only these columns.

When new observations are encountered, the stochastic components required to compute the dual vector are computed and the feasibility of the resultant dual vector is established based on the inequality (17). These calculations are shows in **Table 6**. Note that these computations are based on deviation from the mean of random variable $\delta_5^j = \omega_5^j - \bar{\omega}_5$, that is $\delta_5^1 = 1.0$, $\delta_5^2 = 0.0$ and $\delta_5^3 = -1.0$.



(a) $\omega_5^1 = 2.5$ (b) $\omega_5^2 = 1.5$ (c) $\omega_5^3 = 0.5$

Figure 2 Basic solutions and feasible regions of subproblem dual LP in (31)

$i : B^i/N^i$	ν^i	Φ^i	G^i	$\bar{g}^i$
$1 : \{1,3\}/\{2,4,5\}$	$\begin{pmatrix} 0 \\ 1.0 \end{pmatrix}$	$\emptyset$	$\begin{pmatrix} 1.0 & 0 & 1.0 & 0 & \mathbf{0} \\ 0 & 1.0 & 0 & 1.0 & \mathbf{0} \\ 0.5 & 0.5 & 0 & 0 & \mathbf{1.0} \end{pmatrix}$	$\begin{pmatrix} -2.0 \\ -2.0 \\ -1.5 \end{pmatrix}$
$2 : \{2,3\}/\{1,4,5\}$	$\begin{pmatrix} 1.0 \\ 2.0 \end{pmatrix}$	$\emptyset$	$\begin{pmatrix} 1.0 & 0 & 1.0 & 0 & \mathbf{0} \\ 0 & 1.0 & 0 & 1.0 & \mathbf{0} \\ -0.5 & 0.5 & 0 & 0 & \mathbf{1.0} \end{pmatrix}$	$\begin{pmatrix} -2.0 \\ -2.0 \\ -0.5 \end{pmatrix}$
$3 : \{1,4\}/\{2,3,5\}$	$\begin{pmatrix} 1.0 \\ 0 \end{pmatrix}$	$\emptyset$	$\begin{pmatrix} 1.0 & 0 & 1.0 & 0 & \mathbf{0} \\ 0 & 1.0 & 0 & 1.0 & \mathbf{0} \\ 0.5 & -0.5 & 0 & 0 & \mathbf{1.0} \end{pmatrix}$	$\begin{pmatrix} -2.0 \\ -2.0 \\ -0.5 \end{pmatrix}$
$4 : \{2,4\}/\{1,3,5\}$	$\begin{pmatrix} 2.0 \\ 1.0 \end{pmatrix}$	$\emptyset$	$\begin{pmatrix} 1.0 & 0 & 1.0 & 0 & \mathbf{0} \\ 0 & 1.0 & 0 & 1.0 & \mathbf{0} \\ -0.5 & -0.5 & 0 & 0 & \mathbf{1.0} \end{pmatrix}$	$\begin{pmatrix} -2.0 \\ -2.0 \\ 0.5 \end{pmatrix}$
$5 : \{3,5\}/\{1,2,4\}$	$\begin{pmatrix} 1.5 \\ 2.5 \end{pmatrix}$	$\left\{ \phi_5^5 = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix} \right\}$	$\begin{pmatrix} 1.0 & \mathbf{2.0} & 1.0 & 0 & 0 \\ -1.0 & \mathbf{-2.0} & 0 & 1.0 & 0 \\ 1.0 & \mathbf{0} & 0 & 0 & 1.0 \end{pmatrix}$	$\begin{pmatrix} -3.0 \\ 1.0 \\ -2.0 \end{pmatrix}$
$6 : \{4,5\}/\{1,2,3\}$	$\begin{pmatrix} 1.5 \\ 0.5 \end{pmatrix}$	$\left\{ \phi_5^6 = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix} \right\}$	$\begin{pmatrix} -1.0 & \mathbf{2.0} & 1.0 & 0 & 0 \\ 1.0 & \mathbf{-2.0} & 0 & 1.0 & 0 \\ 1.0 & \mathbf{0} & 0 & 0 & 1.0 \end{pmatrix}$	$\begin{pmatrix} -1.0 \\ -1.0 \\ -2.0 \end{pmatrix}$
$7 : \{2,5\}/\{1,3,4\}$	$\begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$	$\left\{ \phi_5^7 = \begin{pmatrix} 1.0 \\ -1.0 \end{pmatrix} \right\}$	$\begin{pmatrix} 1.0 & \mathbf{0} & 1.0 & 0 & 0 \\ -1.0 & \mathbf{2.0} & 0 & 1.0 & 0 \\ 1.0 & \mathbf{-2.0} & 0 & 0 & 1.0 \end{pmatrix}$	$\begin{pmatrix} -2.0 \\ -1.0 \\ -1.0 \end{pmatrix}$
$8 : \{1,5\}/\{2,3,4\}$	$\begin{pmatrix} 1.5 \\ -0.5 \end{pmatrix}$	$\left\{ \phi_5^8 = \begin{pmatrix} 1.0 \\ -1.0 \end{pmatrix} \right\}$	$\begin{pmatrix} 1.0 & \mathbf{0} & 1.0 & 0 & 0 \\ 1.0 & \mathbf{2.0} & 0 & 1.0 & 0 \\ -1.0 & \mathbf{-2.0} & 0 & 0 & 1.0 \end{pmatrix}$	$\begin{pmatrix} -2.0 \\ -3.0 \\ 1.0 \end{pmatrix}$

Table 5 **Bases and deterministic components used to compute dual vectors of subproblem** (31).

B^i/N_i	Observation ω^1 ($\delta_5^1 = 1.0$)				Observation ω^2 ($\delta_5^2 = 0$)				Observation ω^3 ($\delta_5^3 = -1.0$)			
	π_1^i	$G^i \times \delta^1$	$\bar{g}^i$	Feasibility	π_2^i	$G^i \times \delta^2$	$\bar{g}^i$	Feasibility	π_3^i	$G^i \times \delta^3$	$\bar{g}^i$	Feasibility
B^1/N^1	$\begin{pmatrix}0\\1\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\-1.5\end{pmatrix}$	True	$\begin{pmatrix}0\\1\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\-1.5\end{pmatrix}$	True	$\begin{pmatrix}0\\1\end{pmatrix}$	$\begin{pmatrix}0\\0\\-1\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\-1.5\end{pmatrix}$	True
B^2/N^2	$\begin{pmatrix}1\\2\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\-0.5\end{pmatrix}$	True	$\begin{pmatrix}1\\2\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\-0.5\end{pmatrix}$	True	$\begin{pmatrix}1\\2\end{pmatrix}$	$\begin{pmatrix}0\\0\\-1\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\-0.5\end{pmatrix}$	False
B^3/N^3	$\begin{pmatrix}1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\-0.5\end{pmatrix}$	True	$\begin{pmatrix}1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\-0.5\end{pmatrix}$	True	$\begin{pmatrix}1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\-1\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\-0.5\end{pmatrix}$	False
B^4/N^4	$\begin{pmatrix}2\\1\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\0.5\end{pmatrix}$	True	$\begin{pmatrix}2\\1\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\0.5\end{pmatrix}$	False	$\begin{pmatrix}2\\1\end{pmatrix}$	$\begin{pmatrix}0\\0\\-1\end{pmatrix}$	$\begin{pmatrix}-2\\-2\\0.5\end{pmatrix}$	False
B^5/N^5	$\begin{pmatrix}2.5\\3.5\end{pmatrix}$	$\begin{pmatrix}2\\-2\\0\end{pmatrix}$	$\begin{pmatrix}-3\\1\\-2\end{pmatrix}$	False	$\begin{pmatrix}1.5\\2.5\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}-3\\1\\-2\end{pmatrix}$	False	$\begin{pmatrix}0.5\\1.5\end{pmatrix}$	$\begin{pmatrix}-2\\2\\0\end{pmatrix}$	$\begin{pmatrix}-3\\1\\-2\end{pmatrix}$	True
B^6/N^6	$\begin{pmatrix}2.5\\1.5\end{pmatrix}$	$\begin{pmatrix}2\\-2\\0\end{pmatrix}$	$\begin{pmatrix}-1\\-1\\-2\end{pmatrix}$	False	$\begin{pmatrix}1.5\\0.5\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}-1\\-1\\-2\end{pmatrix}$	True	$\begin{pmatrix}0.5\\-0.5\end{pmatrix}$	$\begin{pmatrix}-2\\2\\0\end{pmatrix}$	$\begin{pmatrix}-1\\-1\\-2\end{pmatrix}$	False
B^7/N^7	$\begin{pmatrix}2.5\\0.5\end{pmatrix}$	$\begin{pmatrix}0\\2\\-2\end{pmatrix}$	$\begin{pmatrix}-2\\-1\\-1\end{pmatrix}$	False	$\begin{pmatrix}1.5\\1.5\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}-2\\-1\\-1\end{pmatrix}$	True	$\begin{pmatrix}0.5\\2.5\end{pmatrix}$	$\begin{pmatrix}0\\-2\\2\end{pmatrix}$	$\begin{pmatrix}-2\\-1\\-1\end{pmatrix}$	False
B^8/N^8	$\begin{pmatrix}2.5\\-1.5\end{pmatrix}$	$\begin{pmatrix}0\\2\\-2\end{pmatrix}$	$\begin{pmatrix}-2\\-3\\1\end{pmatrix}$	False	$\begin{pmatrix}1.5\\-0.5\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}-2\\-3\\1\end{pmatrix}$	False	$\begin{pmatrix}0.5\\0.5\end{pmatrix}$	$\begin{pmatrix}0\\-2\\2\end{pmatrix}$	$\begin{pmatrix}-2\\-3\\1\end{pmatrix}$	True

Table 6 Dual vectors for all bases and their feasibility check for $\omega^1 = 2.5$, $\omega^2 = 1.5$, and $\omega^3 = 0.5$.

Subroutine 3 Subroutine to check feasibility of a dual vector.

```
function check_feasibility(lambda_G, sigma_gbar, d_obs, basis_idx, i, j)
    feasFlag = TRUE; n = 1
    while n <= length(basis_idx) && feasFlag == TRUE do
        if n ∉ basis_idx then
            feasFlag = d_obs[j][n] - sigma_gbar[i][n] >= 0;
        else
            feasFlag = lambda_G[i][n] × d_obs[j][n] - sigma_gbar[i][n] >= 0;
        end if
        n = n + 1;
    end while
    return feasFlag;
end function
```

Appendix B: Principal SD Subroutines

This section presents SD subroutines for checking feasibility, computing cut coefficients, updating the cuts and the proximal parameter.

B.1. Checking Feasibility

Feasibility of a given basis D_{B^i} for an observation ω^j is verified using the inequality in (17). The steps are presented as function `check_feasibility(·)` in **Subroutine 3**. The process of checking feasibility is completed in two steps. The first one is a trivial check for non-basic variables: $\delta^j_{\bar{N}^i} - \bar{g}^i_{\bar{N}^i} \geq 0$. The second step is for basic variables, which involves checking the inequality: $G^i_{\bar{B}^i}\delta_{\bar{B}^i} - \bar{g}^i_{\bar{B}^i} \geq 0$. Once the feasibility check is completed, the results (the return values of `check_feasibility(·)`) are stored in a boolean vector `obsFeasibility[i]`, one for each index set $B^i \in \mathcal{B}^k$. The next set of calculations are carried out only if the basis is found to be feasible. The use of this boolean vector avoids the need to store separate sets of feasible dual vectors V^k_j for each observation ω^j.

B.2. Computing and Updating Cut Coefficients

With the elements of the information matrix defined, we are now in a position to present the calculations involved in computing the coefficients of the lower bounding affine functions. The principal step is the `argmax(·)` procedure presented in **Subroutine 4** which is invoked once for each observation $\omega^j \in \mathcal{O}^k$. The argument for this operation in (3) is computed using elements of the information matrix as:

$$(\pi^i_j)^\top(\xi^j - C^j x) = \texttt{sigma\_bbar[i][0]} + \texttt{delta\_b[i][j]} - (\texttt{sigma\_Cbar[i][0]} + \texttt{delta\_C[i][j]}) \times \texttt{x}.$$

Note that the above calculation is performed only for index set $B^i \in \mathcal{B}^k$ which yields a feasible basis with respect to the observation ω^j. The output of the `argmax` procedure are the coefficients `alpha_star` and `beta_star`. These coefficients correspond to the intercept and subgradient of the affine function which provides the best lower bound to $h(x, \omega^j)$. The `argmax(·)` also returns a pointer to an index set in $\mathcal{B}^k$ which is

Subroutine 4 Subroutines to form cuts.

```
function argmax(bases,x,j)
    maxval = -∞;
    for i = 1...length(bases) do
        if obsFeasibility[i][j] == TRUE then
            val = sigma_bbar[i][0] + delta_b[i][j]
                                    -(sigma_Cbar[i][0] + delta_C[i][j]) × x;
            if maxval < val then
                alpha_star = sigma_bbar[i][0] + delta_b[i][j]
                beta_star = -(sigma_Cbar[i][0] + delta_C[i][j]);
                maxval = val;
                i_star = i;
            end if
        end if
    end for
    return (alpha_star, beta_star, i_star);
end function
```

```
function form_cut(bases,x,k)
    /* alpha, beta, sample_size, i_star and eta_coeff are fields of cut. */
    alpha = 0, beta = [0,...,0], sample_size = k, eta_coeff = 1.0;
    for j = 1,...,k do
        (alpha_star, beta_star, i_star[j]) = argmax(bases,x,j);
        alpha = (j-1)/j × alpha + alpha_star/j;
        beta = (j-1)/j × beta + beta_star/j;
    end for
    return cut;
end function
```

used to compute these coefficients. These pointers are stored in an integer vector called `i_star` for their use in resampling the cut for our statistical optimality tests. We define a `cut` data structure with fields `alpha`, `beta`, `sample_size` and `i_star`, where `sample_size` denote the number of observations which are used to create the cut (this is same as iteration count k, as one sample is added every iteration). The computation of the SD affine functions is shown in function `form_cut(·)` in **Subroutine 4**.

Recall that, as the algorithm progresses the coefficients of the lower bounding affine function are updated as shown in (7). The affine functions are added as constraints in the master problem. The direct implementation

Subroutine 5 Subroutine to update cuts.

```
function update_cut(cuts, k)
    for c = 1 ... length(cuts) do
        eta_coeff[c] = k/(k-1) × eta_coeff[c];
        if L ≠ 0 then
            rhs[c] = rhs[c] + (k-cut_iter[c])/cut_iter[c]) L;
        end if
    end for
    return cuts;
end function
```

Subroutine 6 Subroutine to update proximal parameter

```
/* r1, r2 < 1.0 are given constants, norm(·) computes two-norm. */
function update_prox(prox_param, old_norm, x, hat_x, new_incumb)
    if new_incumb == TRUE then
        new_norm = norm(x, hat_x);
        if new_norm >= r1 × old_norm then
            prox_param = min(max(r2 × prox_param, pow(10,-6)), 1.0);
        end if
    else
        prox_param = prox_param/r2;
    end if
    return prox_param;
end function
```

of this procedure will require updating all the coefficients $(\alpha_j^{k-1}, \beta_j^{k-1})$ of the affine functions. Computationally this may prove to be cumbersome. However, one can view these updates in the mathematically equivalent form for $j = 1, \ldots, k-1$ as:

$$\eta \geq \alpha_j^k + (\beta_j^k)^\top x = \frac{k-1}{k}(\alpha_j^{k-1} + (\beta_j^{k-1})^\top x) = \frac{j}{k}(\alpha_j^j + (\beta_j^j)^\top x) + \frac{k-j}{k} L$$

$$\equiv \quad \left(\frac{k}{j}\right) \eta - (\beta_j^j)^\top x \geq \alpha_j^j + \frac{k-j}{j} L.$$

This representation allows us to retain most of the coefficients at values obtained when they were computed (for example, β_j^j which were computed in iteration j). The updates are restricted only to a single column corresponding to the auxiliary variable η. The right-hand side is updated only if $L \neq 0$. These updates are carried out in the function `update_cut(·)` in **Subroutine 5** where `cut_iter[·]` encodes the iteration when a cut is generated (i.e., j).

B.3. Updating the Proximal Parameter

The SD algorithm uses a proximal term built around the incumbent solution $\hat{x}^k$ to form the regularized master program. Our implementation includes certain enhancements which involve the proximal parameter σ^k and are designed to improve the computational performance of the SD algorithm. Instead of keeping the proximal parameter constant, we allow it to be updated throughout the course of the algorithm (hence, we use iteration index in our notation σ^k). In general, the σ^k is decreased when the incumbent changes and $\|x^{k+1} - \hat{x}^k\|$ increases from one iteration to the next, and increased when the incumbent does not change. This procedure is presented as function `update_prox(·)` in **Subroutine 6** where σ^k is referred as `prox_param`. We note that this procedure does not alter the theoretical properties of the regularized SD algorithm since $\sigma^k > 0$ for all k.

Appendix C: Test Instances

In this section we provide a brief description of the test instances used in our computational study[‡]. The size of the problem instances is summarized in **Table 7**. The fourth column corresponds to the number of second-stage parameters affected by the random variable (RV) presented as number of right-hand side elements and objective cost coefficients. These instances are available in the SMPS format in the authors' Github repository USC3DLAB (2019).

C.1. Sonet Switched Network with Random Cost Coefficients

This problem has its origin in telecommunication network planning and was presented in the original form in Sen et al. (1994a). The problem considers allocation of additional capacity to the links in a network that provides private line telecommunication services. These service requests arise between pairs of nodes in the network. In the original model the service quantity (demand) is the only parameter assumed to depend on the random variable, and the goal is to allocate capacity so as to reduce the total demand lost. Here we extend the model by associating a cost with lost demand which also depends on the random variable. We briefly describe the model here.

We will denote the communication links in the network by $\mathcal{L}$. A service request is characterized by a pair origin-destination nodes which is assumed to be an element of the set $\mathcal{S}$. Each service request (say i) can be satisfied over a set of routes denoted by $\mathcal{R}_i$. If there are no routes with sufficient capacity available to accommodate the request, it cannot be fully served. The existing capacity of the links denoted as e_ℓ needs to be appended by additional capacity x_ℓ which constitutes the first-stage decision variable. The total amount of capacity that can be added is restricted to be with a budget denoted as b. This results in following first-stage optimization problem:

$$\min \left\{ \mathbb{E}[h(x,\tilde{\omega})] \;\middle|\; \sum_{\ell \in \mathcal{L}} x_\ell \leq b,\ x_\ell \geq 0\ \forall \ell \in \mathcal{L} \right\}$$

[‡] The scope of the variables and parameters defined here is restricted to individual subsections.

Instance Name	1^{st} stage variables/constraints	2^{nd} stage variables/constraints	# of RV RHS + Obj.	Magnitude of outcomes
`ssn_rcG`	89/1	709/178	$86 + G$	10^{70}
`SCFT-6`	30/18	86/36	12 + 2	10^{9}
`SCFT-46`	602/174	348/1480	184 + 2	10^{129}
`transship`	7/0	77/35	7 + 7	Normal distribution

Table 7 Details of Problem Instances

Once the capacity decisions have been made, requests for service can be routed in a manner that efficiently utilizes the network resources. To this effect, the function $h(x,\omega)$ captures the cost of unsatisfied service, and is the value of the following LP:

$$\begin{aligned}
\min \ & \sum_{g\in\mathcal{G}} d_g(\omega)\Big[\sum_{i\in\mathcal{S}_g} u_i\Big] \\
\text{s.t.} \ & \sum_{i\in\mathcal{S}}\sum_{i\in\mathcal{R}_i} A_{ir} f_{ir} \le e_\ell + x_\ell \qquad \forall \ell \in \mathcal{L} \\
& \sum_{r\in\mathcal{R}_i} f_{ir} + u_i = \xi_i(\omega) \qquad \forall i \in \mathcal{S}, \\
& f_{ir}, u_i \ge 0 \qquad \forall r \in \mathcal{R}_i, i \in \mathcal{S}.
\end{aligned}$$

Here the decision variable u_i captures the number of units of unsatisfied demand and f_{ir} denotes the amount of service satisfied via route $r \in \mathcal{R}_i$ for every service request $i \in \mathcal{S}$. The term A_{ir} denotes an incidence matrix whose element-ℓ is 1 is link-ℓ belongs to route $r \in \mathcal{R}_i$, and 0 otherwise. While the first set of constraints ensure that the total flow on each link in the network does not exceed the sum of existing and newly installed capacity, the second set of constraints account for the total amount of service requested by $i \in \mathcal{S}$ as either satisfied over all the associated routes or as unsatisfied. The right-hand side of the second set of constraint, representing the total service requested, depends on the random variable. We distinguish the service requests by clustering them into groups denoted by the set $\mathcal{G}$, and penalize the unsatisfied service request differently across these groups. Moreover the penalty cost for group g, denoted by d_g, also depends on the random variable. Note that the original model can be recovered by setting $|\mathcal{G}| = 1$ and $d_g(\tilde{\omega}) = 1$, almost surely.

To create the instances of this model we assume that the demand and the cost coefficients are independent of one another. We use the original stochastic information for the service demand data, and use a discrete distribution to model the demand random variable. We created different instances by varying the number of groups. These instances are denoted as `ssn_rcG` where $G = 1, \ldots, 4$.

C.2. Supply Chain - Freight Transportation Optimization

This problem is aimed at mid-term planning of a global multi-product chemical supply chain and appeared first in You et al. (2009). A set of products are manufactured and distributed through a given global supply chain that includes a large number of world wide customers and a number of geographically distributed plants and distribution centers. The facilities can hold inventory and are connected to each other by transportation links. The problem is to determine the monthly production and inventory levels of each facility, and the monthly shipping quantities between network nodes such that the total expected cost of operating the global

supply chain is minimized while satisfying customer demands over the specified planning horizon. Due to involved notations in the model, we restrict to a verbal presentation of the model, and refer the reader to the original publication for the mathematical formulation.

The problem involves decisions made over a finite horizon at multiple discrete time epochs. The inventory, transshipment and production decisions are made at the beginning of the time period. These decisions are subject to mass balance constraints at the plants, distribution centers and customer locations. These mass balance equation take into account the transportation time during shipping process such that the freight can arrive at the destination on time. The initial inventory required for mass balance at all facilities is assumed to be known, and an explicit constraint ensures that the inventory level is maintained above a certain minimum requirement. In addition to these constraints, the decision variables are bounded by their respective capacities. The decisions for the first time period are made in a here-and-now manner, while the decisions for the remaining horizon are adaptive in nature and depend on the realization of random variables corresponding to customer demands and freight rates. The objective function is to minimize the total expected cost which includes the inventory holding costs at facilities, freight and facility throughput costs for shipment between facilities (including customer locations), and penalty costs for unmet demand.

Unfortunately, the data used in the original publication is not publicly available. Therefore, we have generated two instances of the model using synthetic data. The first instance (`SCFT-6`) is for a supply chain network with 2 plants, 3 distribution centers and 6 customer locations which are connected by 14 transportation links. The second more realistic instance is comparable in size to the to that used in You et al. (2009). This instance has 5 plants, 13 distribution centers and 46 customer locations. These facilities are connected by 131 transportation links. In our instances the model horizon is set to three months – the first month is considered to be the the here-and-now stage and the remaining two months constitute the second-stage. Further, in our instances 10% of the freight rates are affected by randomness and are assumed to follow a discrete distribution. The size of the two problem instances is summarized in **Table 7**.

C.3. Multilocation Transshipment Problem

This problem, adopted from Herer et al. (2006), involves several nonidentical retailers and one supplier who coordinate their replenishment strategies and transshipments to minimize the long-run expected cost. The system inventory is periodically reviewed, and the replenishment orders are placed with the supplier. At the beginning of a period, replenishment orders (placed in previous time period) arrive which are used to satisfy any outstanding backlog and to increase the inventory level. Following this, the demand is realized, transshipments are completed immediately which are used to satisfy the demand, the unsatisfied demand is backlogged, and the inventory quantities are updated. Finally, based on the inventory and backlog quantities, new replenishment decisions are placed with the supplier. The system costs are determined by transshipment costs, inventory holding cost and backlog penalties. In the original work, the authors assume that only the demand at each retailer in a period is random and stationary over time. We extend their model by also allowing the transshipment costs to be random.

Under non-shortage inducing policy assumption, Herer et al. (2006) show that there exists an optimal order-up-to policy. The order upto quantities are estimated using a sample-path optimization method such as

Infinitesimal Perturbation Analysis. Alternatively, one can formulate this problem as a 2-SLP with the first-stage tasked with identifying the order-up-to quantities, and the second-stage is associated with replenishment orders and transshipments. To present the 2-SLP formulation, we will use N to denote the number of suppliers, s_i and $d_i(\tilde{\omega})$ to denote the first-stage decision representing the order-up-to quantity and random variable dependent demand at retailer i. The second-stage decisions are the ending inventory e_i, stock used to satisfy demand f_i, inventory increased through replenishment q_i and amount of backlog met after replenishment r_i at retailer i, and stock at retailer i used to meet demand at retailer j using transshipment t_{ij}. The unit holding and backlog penalty costs at retailer i, unit transshipment cost from retailer i to j are denoted as h_i, p_i and $c_{ij}(\tilde{\omega})$, respectively. Given an order-up-to policy for the replenishment quantities, Herer et al. (2006) suggest a linear cost network flow model to identify the optimal transshipment quantities. We will use the same approach to state the 2-SLP as follows:

$$\min_{s_i \geq 0,\ i=1,\ldots,N} \mathbb{E}[h((s_i)_i, \tilde{\omega})] \tag{32a}$$

where,

$$\begin{aligned} h((s_i)_i, \tilde{\omega}) = \min\ & \sum_{i=1}^{N}(h_i e_i + p_i r_i) + \sum_{i=1, i\neq j}^{N} c_{ij}(\tilde{\omega}) t_{ij} \\ \text{s.t. } & f_i + \sum_{j=1, j\neq i}^{N} t_{ij} + e_i = s_i \quad i = 1,\ldots,N \\ & f_i + \sum_{j=1, j\neq i}^{N} t_{ij} + r_i = d_i(\tilde{\omega}) \quad i = 1,\ldots,N \\ & \sum_{i=1}^{N}(r_i + q_i) = \sum_{i=1}^{N} d_i(\tilde{\omega}) \\ & e_i + q_i = s_i \quad i = 1,\ldots,N \\ & e_i, f_i, q_i, r_i, t_{ij} \geq 0 \quad i, j = 1,\ldots,N. \end{aligned} \tag{32b}$$

To build our instances with random cost coefficients, we have used the demand distribution information presented in Table 3 of Herer et al. (2006). The cost coefficients are nonidentical and built as described in §4.5 of Herer et al. (2006). We assume that the transshipment costs follow a normal distribution with mean equal to the deterministic quantities from the reference and standard deviation set to 20% of the mean. This instance is named as `transship` in our database.