MULTISTAGE STOCHASTIC DECOMPOSITION AND ITS APPLICATIONS

by

Zhihong Zhou

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF SYSTEMS AND INDUSTRIAL ENGINEERING

In Partial Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2012

THE UNIVERSITY OF ARIZONA

GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Zhihong Zhou
entitled Multistage Stochastic Decomposition and Its Applications
and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

____ Date: April 6, 2012
Suvrajeet Sen

____ Date: April 6, 2012
Guzin Bayraksan

____ Date: April 6, 2012
Young Jun Son

____ Date: April 6, 2012
Wei Hua Lin

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the dissertation final copies to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

____ Date: April 6, 2012
Dissertation Director: Suvrajeet Sen

____ Date: April 6, 2012
Dissertation Director: Guzin Bayraksan

STATEMENT BY AUTHOR

ACKNOWLEDGEMENTS

I would like to sincerely thank those who helped me in my research and doctoral study.

First and foremost, I must give my high, respectful gratitude to my advisor, Prof. Suvrajeet Sen for his patience, guidance, understanding and friendship during my graduate studies at University of Arizona and Ohio State University. I would also appreciate that Prof. Sen picked up a very interesting and important problem for me. Moreover, I would say his mentoring to me is far more than my academic research. More importantly, when a lot of unexpected things happened to me, Prof. Sen is always trying his best to help me especially when I was in my difficult time. Meanwhile, Prof. Sen provides me the unique opportunity to gain a wide breadth of experience in working under different environments/labs. I am very sure Prof. Sen's mentorship is my best treasure and I feel very lucky to have the opportunity to work under his advise. In other aspects, Prof. Sen encouraged me to not only grow up as an independent researcher but also to be brave under different circumstances. Without Prof. Sen's kind, persistent help and guidance, this dissertation would not have been completed.

I would like to express my appreciation to my committee members, Dr. Guzin Bayraksan, Dr. Young Jun Son, Dr. Wei Hua Lin, Dr. Daniel Zeng and Dr. Ferenc Szidarovszky for their kind guidance and valuable suggestions in my dissertation as well as my graduate studies. I would also like to thank Dr. Larry Head and Linda Cramer for their generous help during my PhD study.

Finally, I would like to thank my parents, Qilan Zhang and Weiqing Zhou, and my wife, Lu Chen, for their supports, encouragements and loves, which motivate me to work and study. Especially, I really appreciate my wife for her love and accompany. I would also like to thank my friends and classmates at both University of Arizona and Ohio State University, Bingyuan Chen, Yang Yuan, Yunwei Qi and Shuguang Kang. It was always pleasant to work together in the lab.

TABLE OF CONTENTS

LIST OF FIGURES........ 8

LIST OF TABLES........ 9

LIST OF ABBREVIATIONS........ 10

ABSTRACT........ 11

CHAPTER 1: INTRODUCTION........ 13

1.1 Literature review of stochastic programming........ 13

1.2 Multistage stochastic programming and challenges........ 16

1.3 Applications of stochastic programming........ 18

1.4 Contributions........ 18

1.5 Dissertation outline........ 20

CHAPTER 2: TWO-STAGE SD ALGORITHM........ 21

2.1 Introduction........ 21

2.2 Stochastic decomposition........ 21

2.3 Convergence results of SD algorithm........ 28

CHAPTER 3: ENHANCEMENTS OF TWO-STAGE SD ALGORITHM........ 39

3.1 Introduction of two extensions........ 39

3.2 The unique limit for incumbent sequence........ 39

3.3 SD with re-sampling........ 41

3.4 Computational results........ 45

CHAPTER 4: MULTISTAGE STOCHASTIC DECOMPOSITION........ 54

4.1 Introduction........ 54

TABLE OF CONTENTS -*Continued*

4.2 Multistage stochastic decision models.. 55

4.3 Multistage stochastic decomposition algorithm................................. 60

4.3.1 Simulate a sample path... 64

4.3.2 Solve nodal dual approximations.. 66

4.3.3 Collect information and update approximations.................... 68

4.3.4 Update incumbent solutions.. 71

4.4 Asymptotic convergence of multistage SD... 72

4.5 Remarks on the asymptotic analysis... 82

4.6 A production and inventory application.. 83

4.7 Computational results... 84

CHAPTER 5: CONCLUSIONS.. 87

REFERENCES.. 90

LIST OF FIGURES

Figure 1: Solution times of regularized SD for CEP1: with and without re-sampling........ 47

Figure 2: Comparison of solution quality for CEP1: SD VS SDR........ 47

Figure 3: Solution times of regularized SD for PGP2: with and without re-sampling........ 48

Figure 4: Comparison of solution quality for PGP2: SD VS SDR........ 48

Figure 5: Solution times of regularized SD for 20TERM: with and without re-sampling........ 49

Figure 6: Comparison of solution quality for 20TERM: SD VS SDR........ 50

Figure 7: Comparison of solution quality for SSN: SD VS SDR........ 52

Figure 8: Comparison of solution quality for STORM: SD VS SDR........ 52

Figure 9: Caricature of visits to nodes within one iteration........ 63

Figure 10: Sequence of first-stage solutions for a small production inventory instance (81 scenarios)........ 85

Figure 11: Sequence of first-stage solutions for a large production-inventory instance (32^5scenarios)........ 86

LIST OF TABLES

Table 1: Comparison of CPU time and objective value for SD and SDR in large scale instances with 20 replications: SSN and STORM............. 53

Table 2: Results from Replications of the Multistage Stochastic Decomposition Algorithm.. 85

LIST OF ABBREVIATIONS

ADP: Approximate Dynamic Programming.

CCA: Cut Calculation Algorithm.

CI: Confidence Interval.

DOASA: Dynamic Outer Approximation Sampling Algorithms.

DP: Dynamic Programming.

LP: Linear Programming.

MSD: Multistage Stochastic Decomposition.

MSLP: Multistage Stochastic Linear Programs.

MSP: Multistage Stochastic Programs.

NDA: Nodal Dual Approximation.

NDS: Nodal Decision Simulation.

NSMA: Nodal Sample Mean Approximation.

RSD: Regularized Stochastic Decomposition.

SD: Stochastic Decomposition.

SDDP: Stochastic Dual Dynamic Programming.

SDR: Stochastic Decomposition with Re-sampling.

SLP: Stochastic Linear Programs.

SP: Stochastic Programming.

ABSTRACT

In this dissertation, we focus on developing sampling-based algorithms for solving stochastic linear programs. The work covers both two stage and multistage versions of stochastic linear programs. In particular, we first study the two stage stochastic decomposition (SD) algorithm and present some extensions associated with SD. Specifically, we study two issues: a) are there conditions under which the regularized version of SD generates a unique solution? and b) in cases where a user is willing to sacrifice optimality, is there a way to modify the SD algorithm so that a user can trade-off solution times with solution quality? Moreover, we present our preliminary approach to address these questions.

Secondly, we investigate the multistage stochastic linear programs and propose a new approach to solving multistage stochastic decision models in the presence of constraints. The motivation for proposing the multistage stochastic decomposition algorithm is to handle large scale multistage stochastic linear programs. In our setting, the deterministic equivalent problems of the multistage stochastic linear program are too large to be solved exactly. Therefore, we seek an asymptotically optimum solution by simulating the SD algorithmic process, which was originally designed for two-stage stochastic linear programs (SLPs). More importantly, when SD is implemented in a time-staged manner, the algorithm begins to take the flavor of a simulation leading to what we refer to as optimization simulation.

As for multistage stochastic decomposition, there are a couple of advantages that deserve mention. One of the benefits is that it can work directly with sample paths,

and this feature makes the new algorithm much easier to be integrated within a simulation. Moreover, compared with other sampling-based algorithms for multistage stochastic programming, we also overcome certain limitations, such as a stage-wise independence assumption.

CHAPTER 1: INTRODUCTION

1.1 Literature review of stochastic programming

Stochastic optimization handles a class of optimization problems where uncertainty appears in the parameters of objective functions or constraints. Usually, uncertainty is described by random variables with known probability distributions. Stochastic programming models are especially useful in the case that one needs to make the decisions before the entire data information are observed in real time simulation optimizations. In the area of stochastic optimization, stochastic programming proposes a framework to incorporate randomness in both linear and nonlinear models ([15], [3] and [12]). Stochastic programming is intimately related with other paradigms in decision making under uncertainty. For example, it has ties with (approximate) dynamic programming, stochastic control, statistical decision theory, etc. Although all of these model decision making under uncertainty, they have their own specifications. In terms of algorithms, decision analytic approaches for decision making under uncertainty problems allow the decision maker to apply very common preference functions and both single and multiple objectives can be embedded in the decision analytic framework ([5]). However, in decision analytic approaches, the need to incorporate all the outcomes of random variables restricts these approaches to be suitable for small to moderate size of the decision problems. On the other hand, in dynamic programming, due to Markovian assumptions, the optimal policies can be obtained for infinite horizon problems ([6]). However, the path dependence characteristics in a variety of applications bring substantial difficulties to dynamic programming models.

In general, the infrastructure of stochastic programming models allows path dependence in the optimization model. Meanwhile, infinite states and constraints at every state “can be” embedded in the stochastic programming model, which makes stochastic programming very effective in handling large scale problems. Compared to dynamic programming, one advantage of the stochastic programming is that it alleviates the strong mathematical assumptions required in dynamic programming (e.g. Markovian assumptions). Therefore, stochastic programming brings a rich class of models and is more useful in handling real applications. In general, large scale stochastic optimization problems in both discrete and continuous variants are difficult to solve, which is mainly due to the presence of multi-dimensional integration in the expectation function. Moreover, it is difficult to build manageable/efficient algorithms to handle large scale stochastic optimization problems. This is particularly so for multistage stochastic programming problems, for which the number of scenarios may be too large to be enumerated even when the random variables are discrete.

In such situations, one might choose appropriate approximations. In most cases, approximations happen in two ways, data aggregation and data selecting. The approximation algorithms using data aggregation lead to successive approximation methods ([22], [18] and [23]), while most of the data selecting approximation algorithms refer to sampling-based algorithms. Since approximation algorithms solve stochastic programming approximately, it is essential to examine the solution quality by performing some statistical tests ([55] and [64]). In order to get asymptotic optimal solutions, [67] recommended increasing the sample sizes at every iterations. However, accordingly, the computational burden rises significantly. Stochastic decomposition algorithm ([28], [31] and [32]) for two-stage stochastic programming embeds sampling in the decomposition framework, which modifies approximations in previous iterations. However, sampling-based algorithms for multistage stochastic programming are not many. Moreover, assessing the solution quality for stochastic programming is another important issue for sampling based algorithms. [1] proposed the Monte Carlo sampling-based procedures to evaluate the solution quality of the stochastic programs via optimality gap. And more recently, [2] developed a sequential sampling procedure to calculate the optimality gap using known candidate solutions of the stochastic programming. Moreover, the candidate solution is (near) optimal if the optimality gap is sufficiently small.

Most of the approximation algorithms are closely linked to the deterministic algorithms, such as the L-shaped method of [69] and Benders' decomposition ([4]) for two-stage problems. Moreover, [7] extends the Benders' decomposition to the multistage version. However, these algorithms are effective in solving small scale stochastic programming. When solving large scale stochastic programs, one may refer to approximation algorithms. On the other hand, another type of algorithms have advantages on parallel computing ([44]), such as progressive hedging ([56]) and diagonal quadratic approximation ([42]). The common feature of this type of algorithms is to relax the nonanticipativity constraints. Since these algorithms have specific structure (network structure), they are very suitable to be parallelized.

1.2 Multistage stochastic programming and challenges

Multistage stochastic programs (MSP) pose some of the more challenging optimization problems. Usually, this class of problems is computationally intractable even when the random variables in the MSP have finite supports. Although MSP may be reformulated as linear programs, the number of constraints grows exponentially as the number of stages in MSP increases, which make the problem extremely difficult to be solved using deterministic equivalent paradigm.

In general, most applications of multistage stochastic linear programs (MSLP) have been addressed via algorithms that use deterministic approximations of the value function (e.g., [7], [24], [56] and [42]). The underlying uncertainty and sequential evolution of data in these multistage sequential decision problems lead to a sequential optimization under uncertainty model. Unfortunately, as the stochastic process governing uncertainty becomes complicated (e.g., as in the case of correlated exchange rates in financial models [70]), deterministic approximations for this class of problems can become unwieldy because of the need to represent uncertainty via a scenario tree. In order to avoid such an explosion, [25] has proposed effective ways to reduce the size of the scenario tree based on some predetermined limits on computational resources.

Another approach which is applicable to certain classes of MSLP (e.g., problems with randomness only in the right-hand side or the objective function) is provided in [11], where the approximations provide decisions which achieve a prescribed tolerance from optimality. For more general problems though, sampling-based approximations provide the main route to scalable algorithms for stochastic programming. It is therefore important to design algorithms that can provide approximations which, in the long run, yield solutions that are arbitrarily close to an optimum.

Among sampling-based approaches for MSLP, the most popular ones are based on traveling a sampled subtree in any given iteration. These approaches trace back to the work of [47] and more recently [16]. The starting point of these methods is the deterministic nested Benders' decomposition approach, and sampling is used to create a subtree that is traversed to develop approximations. Basically, these approaches ([47] and [16]) extend the two-stage benders decomposition algorithm to a multistage setting based on the sampled scenario tree. These methods can be classified as the stochastic dual dynamic programming (SDDP) algorithms. [49] provides a comprehensive summary of the SDDP algorithms, which include [13], [16], [34], [17], and [53]. The common characteristics of SDDP is to embed the nested benders decomposition infrastructure into the sampled sub-tree. Moreover, [49] defines another category of algorithms for multistage stochastic linear programs, Dynamic Outer Approximation Sampling Algorithms (DOASA). In DOASA, Cut Calculation Algorithm (CCA) is applied to generate the inexact optimality cuts in the backward pass. In fact, SDDP applies deterministic approximations (Nested Benders Decomposition) for the given sampled scenario tree, while the multistage SD incorporates stochastic approximations for the sample mean cost function. As a result, the convergence of SDDP for a given sampled scenario tree follows the same arguments of finite dual extreme points as in Nested Benders Decomposition. Although the convergence with probability one can be proved for SDDP and DOASA, the stagewise independence assumption of these algorithms limit their abilities to model complex, path dependent stochastic processes.

1.3 Applications of stochastic programming

In recent years, several applications such as financial planning ([43]), production systems ([48]), power systems operations ([45]), supply chain management ([27]) and others have benefited from multistage stochastic linear programming models. The key ingredient that makes MSLP attractive for these applications is the ability to accommodate uncertainty in a manner in which data evolves over time.

For instance, investment decisions are made prior to obtaining all the information in portfolio optimization problems. In power generation, since the electricity cannot be stored, it is important to determine the optimal power generation policy to avoid losses. In particular, randomness can happen in many ways in power generation systems, such as stochastic demand and random electricity generators failures. In such cases, a stochastic programming model is essential to describe the system complexity and randomness as well as generating optimal policies. Some recent surveys have addressed these issues, such as the recent books by [36] , [8], and [54], and some articles, e.g., [9] and [32].

1.4 Contributions

Algorithms referred to "Stochastic Decomposition" (SD) constitute a class of decomposition methods that combine traditional deterministic decomposition (e.g. Benders' decomposition) with statistical sampling methods ([28], [30], and [32]). Asymptotically optimal solutions are obtained using these methods. More importantly, SD algorithms have been successful in providing solutions using workstation class computers where other methods have resorted to high performance computing to solve the same instances (e.g. [61]). Nevertheless, there are issues that call for further investigation.

In this dissertation, first, the specific extensions of two-stage SD (regularized SD) that we study cover both convergence theory and algorithmic considerations. In the area of convergence theory, we study the unique limit property of the sequence of incumbent solutions generated by the regularized version of SD. On the algorithmic side, we address the issue of scalability. We show that without loss of asymptotic properties, it is possible to let a user speed-up calculations in any iteration by avoiding the need to use every previously generated outcome in the "argmax" process of cut generation. The results associated with these extensions are provided. Secondly, we propose a statistically motivated sequential sampling method that is applicable to multistage stochastic linear programs, and we refer to it as the multistage stochastic decomposition (MSD) algorithm.

Moreover, we present the convergence properties as well as preliminary evidence of computational possibilities for multistage stochastic decomposition algorithm. As with earlier SD methods for two-stage stochastic linear programs, this approach preserves one of the most attractive features of SD: asymptotic convergence of the solutions can be proven (with probability one) without any iteration requiring more than a small sample size. This data-driven approach also allows us to sequentially update value function approximations, and the computations themselves can be organized in a manner that decomposes the scenario generation (stochastic) process from the optimization computations. In addition to these algorithmic properties, we provide preliminary computational results using a production/inventory instance.

1.5 Dissertation outline

The dissertation is organized as follows: Chapter 2 introduces and reviews the sampling-based algorithms for two-stage stochastic linear programs. In particular, we review the two-stage SD algorithm as well as its asymptotic convergence properties, which is the basis of the multistage stochastic decomposition algorithm. In Chapter 3, we study some extensions of two stage stochastic decomposition algorithm, which has potential to speed up the two-stage stochastic decomposition algorithm. In Chapter 4, we propose the multistage stochastic decomposition algorithm for multistage stochastic linear programs, which alleviates the computation complexities in MSLP algorithms. Moreover, in solving the MSLP, the multistage stochastic decomposition algorithm does not require to traverse the entire scenario tree to reach the asymptotic optimal solutions. Chapter 4 also provides some preliminary computational results of multistage stochastic decomposition algorithm by applying the algorithm to the production and inventory problem with uncertain demands. Finally, in Chapter 5, we conclude our research and discuss some possible future research directions.

CHAPTER 2: TWO-STAGE SD ALGORITHM

2.1 Introduction

Over the past couple of decades, it has become apparent that stochastic programming problems are best solved by using decomposition coordination methods ([8]). There have been several attempts at designing deterministic decomposition methods, including the L-shaped method and its extensions ([69] and [7]), scenario aggregation algorithm ([56]) and diagonal quadratic approximation methods ([42]). On the other hand, some algorithms are based on purely statistical approximations ([59] and [50]). Alternatively, [28] and [14] present sequential sampling methods which combine traditional deterministic decomposition (e.g. Benders' decomposition) with statistical sampling methods.

One class of sampling based approaches is "Stochastic Decomposition" (SD) which has been shown to provide asymptotically optimal solutions ([28] and [30]). Moreover, SD algorithms are promising in parallel computing where other sampling methods have been relied on high performance computing to solve the same cases (e.g. [61]). Nevertheless, there are some instances in which memory and computational requirements can become burdensome. In next section, we introduce the stochastic decomposition algorithm.

2.2 Stochastic decomposition

In general, the two stage stochastic linear programming model has the following form:

$$\underset{x \in \Re^{n_1}}{\text{Min}} \quad f(x) := c^\top x + E[h(x, \tilde{\omega})] \tag{1a}$$

$$\text{s.t.} \qquad Ax \ \leq \ b, \tag{1b}$$

where, A is $m_1 \times n_1$, b is $m_1 \times 1$, $\tilde{\omega}$ is a random variable defined on the probability space $(\Omega, \mathcal{A}, \mathcal{F})$, and

$$h(x, \omega) = \underset{y \in \Re^{n_2}}{\text{Min}} \quad g^\top y \tag{2a}$$

$$\text{s.t.} \quad Wy = r(\omega) - T(\omega)x, \tag{2b}$$

$$y \geq 0 \tag{2c}$$

In this model, we have restricted (2) to a fixed recourse version in which random variables do not appear in the data elements g and W.

In essence, SD is a stochastic version of Benders' decomposition ([4]), which has been recognized as the L-shaped method in the SP literature ([69]). The main idea of these decomposition methods is to construct a piecewise linear approximation to the recourse function and update the approximation during each iteration of the algorithm. The major difference between SD and deterministic decomposition methods (Benders' decomposition, L-shaped method) is the computational effort to create a new cut in the approximation. In traditional deterministic approximations, the recourse function needs to be evaluated for every outcome of the random variable, which requires the solution of as many second stage problems (2) as there are realizations ω of random variables $\tilde{\omega}$. Consequently, deterministic decomposition algorithms are computationally feasible for only those instances in which a few outcomes are sufficient to model randomness. Moreover, deterministic decomposition methods are restricted to only those instances in which the random variables are discrete. In contrast, SD successfully overcomes these limitations by combining sampling with sequential approximations in such a manner as to reduce the computational effort in generating a new cut (a new piecewise linear approximation) in each iteration.

In SD, a new sampled outcome (ω^k) is obtained independently of all previous outcomes, $\{\omega^1, \ldots, \omega^{k-1}\}$, at iteration k. Hence, this corresponds to the i.i.d. samples. In fact, the SD algorithm constructs a *lower bounding affine approximation* of the sample mean function

$$H_k(x) = \frac{1}{k}\sum_{t=1}^{k} h(x, \omega^t). \tag{3}$$

In order to approximate the above function by a lower bound, one might adopt the ideas underlying the Benders' cut. However, such a strategy would require us to solve k linear programs to obtain a subgradient for each outcome $h(x, \omega^t)$, $t = 1, \ldots, k$ (at the point x^k). As a matter of fact, SD theory suggests that asymptotic convergence can be achieved even without solving all these LPs. Instead, it is sufficient to solve only one second-stage LP (2) in any iteration. Furthermore, we incorporate previously obtained data (on the optimal dual solutions for (2)) to define a lower bounding approximation of $h(x, \omega^t)$, for $t < k$. The details are as follows.

For the most recent outcome denoted as ω^k, we evaluate $h(x^k, \omega^k)$, which includes solving (2) with (x^k, ω^k) as the inputs. Let π_k^k denote the dual optimum to this problem. Then, the data collection process within the SD algorithm accumulates this optimal dual vector into a set of previously discovered optimal dual vectors denoted V_{k-1}. As a result, the updated set of the optimal dual vectors at iteration k is denoted V_k (i.e. $V_k = V_{k-1} \cup \pi_k^k$). Thus the set V_k is a collection of the optimal dual vector discovered during the construction of $h(x^t, \omega^t)$, $t = 1, \dots, k$, in each iteration. Next, a lower bounding function for the k^{th} sample mean function (3) is generated by assigning a dual feasible solution for each previously observed outcome $\{\omega^t\}$, $t < k$. To see this, note that LP duality ensures that for $\pi \in V_k$

$$\pi^\top [r(\omega^t) - T(\omega^t)x] \leq h(x, \omega^t) \ \forall x. \tag{4}$$

Thus, in iteration k, the dual vector in V_k that provides the best lower bounding approximation at $\{h(x^k, \omega^t)\}$, for $t < k$, is given by the dual vector for which

$$\pi_t^k(x) \in \operatorname{argmax}\{\pi^\top [r(\omega^t) - T(\omega^t)x] \mid \pi \in V_k\}. \tag{5}$$

When the above operation is undertaken with appropriate data structures ([31]), it is computationally faster than solving a linear program from scratch. In any event, it follows that

$$H_k(x) \geq \frac{1}{k} \sum_{t=1}^{k} (\pi_t^k)^\top [r(\omega^t) - T(\omega^t)x]. \tag{6}$$

Using the lower bound in (6), SD applies a procedure similar to Benders' decomposition, in that the right-hand-side of (6) is added as a new cut of the piecewise linear approximation of $E[h(x,\tilde{\omega})]$.

Finally, we note one more distinction between deterministic and stochastic decomposition. Since the number of outcomes increases as the number of iterations grows, different number of sample sizes is used in generating the approximate cuts. Therefore, in iteration t of SD, one uses a cut that approximates $H_t(x)$, not $H_k(x)$, for $t < k$. As a result, the former cuts need to be re-adjusted to be guaranteed to be the lower bounds for $H_k(x)$. Without loss of generality, we can assume that $h(x,\omega) \geq L$ almost surely. With this assumption, it is clear that $H_k(x) \geq \frac{t}{k} H_t(x) + \frac{k-t}{k} L$. Hence, by multiplying each previously generated cut $(t = 1, \ldots, k-1)$ by the multiplier $\frac{t}{k}$, all previously generated cuts provide a lower bound for the sample mean approximation $H_k(x)$. In any event, the approximation for the first-stage objective function at iteration k is then given by

$$f_k(x) := c^\top x + \operatorname*{Max}_{t=1,\ldots,k} \left\{ \frac{t}{k} \times \frac{1}{t} \sum_{j=1}^{t} (\pi_j^t)^\top [r(\omega^j) - T(\omega^j)x] \right\}.$$

Since these approximations are generated in a recursive manner, it is best to consult [31] regarding the data structures and updates to be used for efficient implementations.

The most basic version of SD uses the sequence $\{x^k\}$ such that

$$x^{k+1} \in \operatorname{argmin}\{f_k(x) \mid x \in X\},$$

where $X = \{x \mid Ax \leq b\}$ denotes the first-stage feasible region. However, to improve the algorithmic properties of SD, we recommend the use of a regularized approximations as defined in [30]. Denoting an incumbent at iteration k as $\hat{x}^k$, we recommend the following

$$x^{k+1} \in \operatorname{argmin}\{f_k(x) + \frac{1}{2}\|x - \hat{x}^k\|^2 \mid Ax \leq b\}. \tag{7}$$

In the following, we will use $\rho_k(x) = \frac{1}{2}\|x - \hat{x}^k\|^2$. The condition to update the incumbent is whether the (sample mean) point estimate of the objective value at x^{k+1} is better than the point estimate of the objective value of the incumbent $\hat{x}^k$. If it is the case, then $\hat{x}^{k+1} = x^{k+1}$; else, $\hat{x}^{k+1} = \hat{x}^k$. This procedure is referred to as the Regularized Stochastic Decomposition (RSD) method which is used for our computational study.

In the last part of this summary, we should point out one aspect in computing implementations. We note that as k changes, so does $H_k(x)$. However, all but one of the observations used in defining $H_k(x)$ are used in $H_{k-1}(x)$. Hence, as k becomes large, the use of common random numbers will reduce variance.

Before introducing the SD algorithm, we also provide the following notations and assumptions in the problem:

x_k: The candidate solution of the k^{th} iteration.

$\hat{x}_k$: The incumbent solution of the k^{th} iteration.

i_k: The iteration at which $\hat{x}_k$ is identified.

J_k: The index set of cuts in iteration k.

$(A1)$ The set X is a nonempty compact polyhedron.

$(A2)$ The feasibility set of second stage problem (2), $\Pi = \{\pi | \pi W \leq g\}$, is not empty.

$(A3)$ For all $x \in X$, $h(x, \tilde{\omega}) < \infty$ with probability one.

$(A4)$ The random matrix $T(\tilde{\omega})$ satisfies $E[\| T(\tilde{\omega}) \|] < \infty$.

$(A5)$ In the probability space $(\Omega, \mathcal{A}, P)$, Ω is a compact set.

Moreover, we also define $\gamma_k = f_{k-1}(x^k) - f_{k-1}(\hat{x}^{k-1}),\ r^t = r(\omega^t),\ T^t = T(\omega^t)$. The procedure of SD algorithm is summarized as follows:

Algorithm 1 Stochastic decomposition algorithm

```
1: Step 0: Initialization. Let $k = 0$, $V_0 = \emptyset$, $J_0 = \emptyset$, $\eta_0(x) \equiv -\infty$. Let $x^1 \in X$,
   $L \leq h(x, \omega)$ for all $(x, \omega) \in X \times \Omega$ and $q \in (0, 1)$ be given. Let $\hat{x}^0 = x^1$, and
   $i_0 = 0$.
2: Step 1: Generate sample $\omega^k$. Let $k = k + 1$. Randomly generate a sample of $\tilde{\omega}$,
   $\omega^k$, independently of any previously generated samples.
3: Step 2: Update $V_k$ and $J_k$. (See Algorithm 2 in next page.)
4: Step 3: Test the incumbent. If we have
       $f_k(x^k) - f_k(\hat{x}^{k-1}) < q[f_{k-1}(x^k) - f_{k-1}(\hat{x}^{k-1})]$,        (8)
   then $\hat{x}^k = x^k$, $i_k = k$. Otherwise $\hat{x}^k = \hat{x}^{k-1}$, $i_k = i_{k-1}$.
5: Step 4: Solve the master problem:
       Min  $c^T x + \eta + \rho_k(x)$
       s.t. $\eta \geq \alpha_t^k + (c + \beta_t^k)x \quad \forall t \in J_k$        (9)
            $Ax \leq b$.
   Let $x^{k+1}$ be one optimal solution of (9). Go back to Step 1.
```

Algorithm 2 Subroutine: Step 2

```
1: Update $V_k$. Let $V_k = V_{k-1} \cup \{\pi(x^k, \omega^{t_k}), \pi(\hat{x}^{k-1}, \omega^{t_k})\}$, where $\pi(x, \omega^t) \in \text{argmax}\{\pi(r^t - T^t x) | \pi \in \Pi\}$.
2: Update $J_k$. Let $J_k = J_{k-1} \cup \{k\}$.
3: Construct the coefficients of the $k^{th}$ cut to be added to the master problem. Compute the coefficients for the $k^{th}$ cut as follows:
       $$\alpha_k^k + \beta_k^k x \equiv \tfrac{1}{k}\textstyle\sum_{t=1}^k (\alpha_t^k + \beta_t^k x), \tag{10}$$
   where
       $$\alpha_t^k + \beta_t^k x = \pi_t^k (r^t - T^t x) \tag{11}$$
   and $\pi_t^k \in \text{argmax}\{\pi(r^t - T^t x^k) | \pi \in V_k\}$.
4: Construct the coefficients of the $(i_{k-1})^{th}$ cut to be added to the master problem. Compute the coefficients for the $(i_{k-1})^{th}$ cut as follows:
       $$\alpha_{i_{k-1}}^k + \beta_{i_{k-1}}^k x \equiv \tfrac{1}{k}\textstyle\sum_{t=1}^k \hat{\pi}_t^k (r^t - T^t x), \tag{12}$$
   where $\hat{\pi}_t^k \in \text{argmax}\{\pi(r^t - T^t \hat{x}^{k-1}) | \pi \in V_k\}$.
5: Update the remaining cuts. For $t \in J_k \backslash \{k, i_{k-1}\}$, let
       $$\begin{array}{rcl} \alpha_t^k & = & \frac{k-1}{k}\alpha_t^{k-1} + \frac{1}{k}L \\ \beta_t^k & = & \frac{k-1}{k}\beta_t^{k-1}. \end{array} \tag{13}$$
```

2.3 Convergence results of SD algorithm

In this section, we review the asymptotic convergence results of the SD algorithm, which form the theoretical basis of asymptotic convergence property in our multistage stochastic decomposition algorithm. We start from the following theorem:

Theorem 1 *Suppose assumptions (A1-A5) hold. Assume $\{z^k\}_{k \in \mathcal{K}} \subset X$, where $\mathcal{K}$ is an infinite subsequence of the natural numbers. We define*

$$G_k(x) = \tfrac{1}{k}\textstyle\sum_{t=1}^k h(x, \omega^t), \tag{14}$$

where ω^t is among samples, $\omega^1, \cdots, \omega^k$. Then with probability one

$$\{z^k\}_{k \in \mathcal{K}} \to \hat{z} \quad \textit{implies} \quad \{G_k(z^k)\}_{k \in \mathcal{K}} \to H(\hat{z}) \tag{15}$$

Proof: See [31] for details. ■

Next, we show that the approximation of cost function converges to a continuous function.

Lemma 1 *Suppose assumptions (A1-A5) hold. Define*

$$\eta_k(x) = \text{Max}\{\alpha_t^k + \beta_t^k x | t \in J_k\} \tag{16}$$

where $\alpha_t^k + \beta_t^k x$ is the t^{th} cut in the SD algorithm. Then the algorithm guarantees that $\eta_k(x) \leq G_k(x)$ for all $x \in X$. In other words, for all $1 \leq t \leq k$ and $x \in X$, we have $\alpha_t^k + \beta_t^k x \leq G_k(x)$.

Proof: Note $\pi_t^k = \text{argmax}\{\pi(r^t - T^t x^k) | \pi \in V_k\}$. Define $\Pi = \{\pi | \pi W \leq g\}$ as the dual feasible region of the second stage problem (2). Since $V_k \subset \Pi$, it is clear that

$$h(x, \omega^t) \quad \geq \pi_t^k (r^t - T^t x) \quad \forall x,$$

and

$$\begin{aligned} G_k(x) = \frac{1}{k}\sum_{t=1}^{k} h(x,\omega^t) &\geq \frac{1}{k}\sum_{t=1}^{k} \pi_t^k(r^t - T^t x) \\ &= \alpha_k^k + \beta_k^k x. \end{aligned}$$

Notice that when $x = x^k$, the equality $G_k(x^k) = \alpha_k^k + \beta_k^k x^k$ may not hold. Similarly, we also have that $\alpha_{i_{k-1}}^k + \beta_{i_{k-1}}^k x \leq G_k(x)$.

For any other cut, $n \in J_k \backslash \{k, i_{k-1}\}$, according to the construction in SD algorithm, for any $1 \leq n \leq k$,

$$\alpha_n^k + \beta_n^k x = (\frac{n}{k}\alpha_n^n + \frac{k-n}{k}) + (\frac{n}{k}\beta_n^n)x. \tag{17}$$

And the results follows,

$$\begin{aligned} \alpha_n^k + \beta_n^k x &= \frac{n}{k}(\alpha_n^n + \beta_n^n x) + \frac{k-n}{k} \\ &\leq \frac{1}{k}\sum_{t=1}^{k} h(x,\omega^t) \\ &= G_k(x). \end{aligned}$$

■

Lemma 2 *Define*

$$h_k(x,\omega) = \text{Max}\{\pi(r(\omega) - T(\omega)x) | \pi \in V_k\}. \tag{18}$$

Suppose assumptions $(A1 - A5)$ hold, then there exists a continuous function φ such that the sequence of functions $\{h_k\}_{k=1}^{\infty}$ uniformly converges to φ on $X \times \Omega$.

Proof: Note that $V_k \subset V_{k+1} \subset V$, where V is the set of vertices of the dual feasible solutions of the problem. This implies that $h_k(x,\omega) \leq h_{k+1}(x,\omega) \leq h(x,\omega)$ for all k and (x,ω). By monotone convergence theorem, for each point $(x,\omega) \in X \times \Omega$, the sequence $\{h_k(x,\omega)\}_{k=1}^{\infty}$ converges. Therefore, we can define a function $\varphi(x,\omega) = \lim_{k\to\infty} h_k(x,\omega)$ on $X \times \Omega$. On the other hand, $V_k \subset V_{k+1} \subset V$ and the finiteness of V imply that there exists a finite set $\bar{V} \subset V$ such that $\lim_{k\to\infty} V_k = \bar{V}$. Therefore,

$$\begin{aligned} \varphi(x,\omega) &= \lim_{k\to\infty} h_k(x,\omega) \\ &= \lim_{k\to\infty} \text{Max}\{\pi(r(\omega) - T(\omega)x) | \pi \in V_k\} \\ &= \text{Max}\{\pi(r(\omega) - T(\omega)x) | \pi \in \bar{V}\}, \end{aligned} \tag{19}$$

which implies that φ is a continuous function. Now we can conclude that $\{h_k\}_{k=1}^{\infty}$ is a sequence of functions that monotonely converges to a continuous function φ on a compact set $X \times \Omega$. Therefore $\{h_k\}_{k=1}^{\infty}$ converges to φ uniformly ([30]). ■

Lemma 3 *Suppose assumptions $(A1 - A5)$ hold. Let $\{x^k\}_{k\in\mathcal{K}}$ be an infinite subsequence of the candidate solutions $\{x^k\}$ in the SD algorithm. When $\{x_k\}_{k\in\mathcal{K}} \to \bar{x}$, $\varphi(\bar{x},\omega^t) = h(\bar{x},\omega^t)$ for all t with probability one.*

Proof: See [30] for details. ■

Theorem 2 *Suppose assumptions* $(A1 - A5)$ *hold. Let* $\{x^k\}_{k\in\mathcal{K}}$ *be an infinite subsequence of the candidate solutions* $\{x^k\}$ *in the SD algorithm. When* $\{x_k\}_{k\in\mathcal{K}} \to \bar{x}$*,*

$$\lim_{k\in\mathcal{K}} \eta_k(x^k) = H(\bar{x}), \tag{20}$$

with probability one.

Proof: Let $\mathcal{K} = \{k_n\}_{n=1}^{\infty}$. By construction,

$$\eta_k(x^k) \geq \alpha_k^k + \beta_k^k x^k = \tfrac{1}{k}\textstyle\sum_{t=1}^{k} h_k(x^k, \omega^t).$$

Notice $\{h_k\}_{k=1}^{\infty}$ converges uniformly to φ on $X \times \Omega$. So $\forall \epsilon > 0$, there exists n_1 such that for all $n \geq n_1$, $|h_{k_n}(x, \omega) - \varphi(x, \omega)| \leq \epsilon$ for all $(x, \omega) \in X \times \Omega$. Therefore, for all x^{k_n} and ω^t, $|h_{k_n}(x^{k_n}, \omega^t) - \varphi(x^{k_n}, \omega^t)| \leq \epsilon$. And we can conclude that

$$\frac{1}{k_n}\sum_{t=1}^{k_n}[h_{k_n}(x^{k_n}, \omega^t) - \varphi(x^{k_n}, \omega^t)] \to 0.$$

Also, we claim that

$$\frac{1}{k_n}\sum_{t=1}^{k_n}[\varphi(x^{k_n}, \omega^t) - \varphi(\bar{x}, \omega^t)] \to 0.$$

Indeed, since $\varphi(x, \omega)$ is continuous on a compact set $X \times \Omega$, $\varphi(x, \omega)$ is also uniformly continuous on $X \times \Omega$. Thus the above limit holds. Note that $h_{k_n}(x^{k_n}, \omega^t) - \varphi(\bar{x}, \omega^t) =$

$[h_{k_n}(x^{k_n},\omega^t)-\varphi(x^{k_n},\omega^t)]+[\varphi(x^{k_n},\omega^t)-\varphi(\bar{x},\omega^t)]$, which leads to the following limit

$$\begin{aligned} & \tfrac{1}{k_n}\textstyle\sum_{t=1}^{k_n}[h_{k_n}(x^{k_n},\omega^t)-\varphi(\bar{x},\omega^t)] \\ = & \tfrac{1}{k_n}\textstyle\sum_{t=1}^{k_n}[h_{k_n}(x^{k_n},\omega^t)-\varphi(x^{k_n},\omega^t)] \\ & +\tfrac{1}{k_n}\textstyle\sum_{t=1}^{k_n}[\varphi(x^{k_n},\omega^t)-\varphi(\bar{x},\omega^t)] \\ \to & \ 0. \end{aligned}$$

On the other hand, according to the strong law of large numbers, with probability one,

$$\lim_{n\to\infty}\tfrac{1}{k_n}\textstyle\sum_{t=1}^{k_n}h(\bar{x},\omega^t)=E[h(\bar{x},\tilde{\omega})]=H(\bar{x}).$$

Therefore, we have

$$\begin{aligned} & \lim_{n\to\infty}\tfrac{1}{k_n}\textstyle\sum_{t=1}^{k_n}h_{k_n}(x^{k_n},\omega^t) \\ = & \lim_{n\to\infty}\tfrac{1}{k_n}\textstyle\sum_{t=1}^{k_n}\varphi(\bar{x},\omega^t) \\ = & \lim_{n\to\infty}\tfrac{1}{k_n}\textstyle\sum_{t=1}^{k_n}h(\bar{x},\omega^t)=H(\bar{x}). \end{aligned}$$

Finally, the results follow by,

$$\eta_k(x^{k_n})\leq\tfrac{1}{k_n}\textstyle\sum_{t=1}^{k_n}h(x^{k_n},\omega^t)\to H(\bar{x}).$$

■

Lemma 4 *Suppose assumptions $(A1-A5)$ hold. Let $f_k(x) = cx + \eta_k(x)$ be the objective function of (9), and $\{x_k\}_{k\in\mathbb{N}}$ denote the sequence of candidate solutions generated by the SD algorithm. Then $\liminf_{k\to\infty}[f_k(x^k) - f_{k-1}(x^k)] = 0$ with probability one.*

Proof: See [30] for details. ■

Theorem 3 *Suppose assumptions $(A1 - A5)$ hold. Let $\{x^k\}_{k\in\mathbb{N}}$ be the sequence of candidate solutions generated by the SD algorithm. There exists an infinite subsequence $\{x^k\}_{k\in\mathcal{K}}$ such that every accumulation point of this subsequence is an optimal solution of* (1)*.*

Proof: We have $\liminf_{k\to\infty}[f_k(x^k) - f_{k-1}(x^k)] = 0$ with probability one. Therefore, from the definition of "$\liminf$", there exists an infinite subsequence $\{x^k\}_{k\in\mathcal{K}}$ such that $\lim_{k\in\mathcal{K}}[f_k(x^k) - f_{k-1}(x^k)] = 0$. By construction in the SD algorithm, for all k and $x \in X$,

$$f_{k-1}(x^k) = cx^k + \eta_{k-1}(x^k) \le f_{k-1}(x) \le cx + \tfrac{1}{k-1}\textstyle\sum_{t=1}^{k-1} h(x,\omega^t). \tag{21}$$

Let $\{x^k\}_{k\in\mathcal{K}'}$ be any convergent subsequence of $\{x^k\}_{k\in\mathcal{K}}$, and $\{x^k\}_{k\in\mathcal{K}'} \to \bar{x}$. From Theorem 1, $\lim_{k\in\mathcal{K}'} f_k(x^k) = c\bar{x} + E[h(\bar{x},\tilde{\omega})]$. On the other hand, from (21), we have $\lim_{k\in\mathcal{K}'} f_{k-1}(x^k) \le cx + E[h(x,\tilde{\omega})]$ for any fixed $x \in X$. Note $\lim_{k\in\mathcal{K}'} f_k(x^k) = \lim_{k\in\mathcal{K}'} f_{k-1}(x^k)$. So $c\bar{x} + E[h(\bar{x},\tilde{\omega})] \le cx + E[h(x,\tilde{\omega})]$. The conclusion follows. ■

Theorem 4 *Suppose assumptions* $(A1 - A5)$ *hold. Let* $\{\hat{x}^k\}_{k\in\mathbb{N}}$ *be the sequence of incumbent solutions generated by the SD algorithm. Let* $\{\hat{x}^{k_n}\}_{n\in\mathbb{N}}$ *be a convergent infinite subsequence of* $\{\hat{x}^k\}_{k\in\mathbb{N}}$ *such that* $\{\hat{x}^{k_n}\}_{n\in\mathbb{N}} \to \hat{x}$*. The following limits hold:*

$$\lim_{n\to\infty} f_{k_n}(\hat{x}^{k_n}) = \lim_{n\to\infty} f_{k_n+1}(\hat{x}^{k_n}) = f(\hat{x}) \tag{22}$$

Proof: See [30] for details. ■

Lemma 5 *Suppose assumptions* $(A1 - A5)$ *hold. Let* $\{\hat{x}^k\}_{k\in\mathbb{N}}$ *be the sequence of incumbent solutions generated by the SD algorithm. Let* $\{k_n\}_{n\in N}$ *represent the sequence of iterations at which the incumbent solution is changed. If* N *is finite, then* $\limsup_{k\to\infty} \gamma^k = 0$ *with probability one. Otherwise,* $\limsup_{m\to\infty} \frac{1}{m}\sum_{n=1}^{m} \gamma^{k_n} = 0$ *with probability one.*

Proof: We discuss two cases in the proof. In the first case, N is finite. So there exist $\hat{x}$ and $K < \infty$ such that $\hat{x}^k = \hat{x}$ for all $k \geq K$ and in Step 3 of Algorithm 1,

$$f_k(x^k) - f_k(\hat{x}) \geq q[f_{k-1}(x^k) - f_{k-1}(\hat{x})] = q\gamma^k.$$

According to Theorem 3, there exists an infinite subsequence $\{x^k\}_{k\in\mathcal{K}}$ of candidate solutions such that every accumulation point of this subsequence is an optimal solution of (1). Without loss of generality, we can assume that $\{x^k\}_{k\in\mathcal{K}} \to x^* \in X^*$. So we have that $\lim_{k\in\mathcal{K}} f_k(x^k) = f(x^*)$ and $\lim_{k\in\mathcal{K}} f_{k-1}(x^k) = f(x^*)$ with probability one. Also, according to Theorem 4, the following limits hold with probability one: $\lim_{k\in\mathcal{K}} f_k(\hat{x}) = f(\hat{x})$ and $\lim_{k\in\mathcal{K}} f_{k-1}(\hat{x}) = f(\hat{x})$. Thus,

$$\begin{aligned} & f(x^*) - f(\hat{x}) \\ = \; & \lim_{k\in\mathcal{K}}[f_k(x^k) - f_k(\hat{x}^{k-1})] \\ \geq \; & q \lim_{k\in\mathcal{K}}[f_{k-1}(x^k) - f_{k-1}(\hat{x}^{k-1})] \\ = \; & q[f(x^*) - f(\hat{x})] = q \lim_{k\in\mathcal{K}} \gamma^k, \end{aligned}$$

with probability one. Since $q \in (0,1)$ and $x^* \in X^*$, $f(x^*) = f(\hat{x})$. And $\lim_{k\in\mathcal{K}} \gamma^k = 0$. Therefore, $0 = \lim_{k\in\mathcal{K}} \gamma^k \leq \limsup_{k\to\infty} \gamma^k \leq 0$ with probability one.

In the second case, N is an infinite set. So in Step 3 of Algorithm 1,

$$f_{k_n}(x^{k_n}) - f_{k_n}(\hat{x}^{k_n-1}) < q[f_{k_n-1}(x^{k_n}) - f_{k_n-1}(\hat{x}^{k_n-1})] = q\gamma^{k_n} \leq 0.$$

By definition of k_n, we notice that $\hat{x}^{k_n-1} = \hat{x}^{k_{n-1}}$. So $f_{k_n}(x^{k_n}) - f_{k_n}(\hat{x}^{k_n-1}) = f_{k_n}(\hat{x}^{k_n}) - f_{k_n}(\hat{x}^{k_{n-1}}) \leq q\gamma^{k_n} \leq 0$. And

$$\tfrac{1}{m}\textstyle\sum_{n=1}^{m}[f_{k_n}(\hat{x}^{k_n}) - f_{k_n}(\hat{x}^{k_{n-1}})] \leq \tfrac{q}{m}\textstyle\sum_{n=1}^{m}\gamma^{k_n} \leq 0, \forall m,$$

which implies that

$$\tfrac{1}{m}\textstyle\sum_{n=1}^{m-1}[f_{k_n}(\hat{x}^{k_n}) - f_{k_{n+1}}(\hat{x}^{k_n})] + \tfrac{1}{m}[f_{k_m}(\hat{x}^{k_m}) - f_{k_1}(\hat{x}^{k_0})] \leq \tfrac{q}{m}\textstyle\sum_{n=1}^{m}\gamma^{k_n} \leq 0,$$

for all m. Suppose the left hand side of the above inequality converges to zero when

m goes to infinity, with proabbility one. Then

$$\lim_{m\to\infty} \frac{1}{m} \sum_{n=1}^{m} \gamma^{k_n} = 0,$$

with probability one. ■

Theorem 5 *Suppose assumptions $(A1 - A5)$ hold. Let $\{\hat{x}^k\}_{k\in\mathbb{N}}$ be the sequence of incumbent solutions generated by the SD algorithm. There exists an infinite subsequence $\{\hat{x}^k\}_{k\in\mathcal{K}}$ such that every accumulation point of this subsequence is an optimal solution of* (1)*, with probability one.*

Proof: Let $\{k_n\}_{n\in N}$ represent the sequence of iterations at which the incumbent solution is changed. According to Lemma 5, when N is finite, $\limsup_{k\to\infty} \gamma^k = 0$; when N is infinite, $\lim_{m\to\infty} \frac{1}{m} \sum_{n=1}^{m} \gamma^{k_n} = 0$, which implies that $0 = \lim_{m\to\infty} \sum_{n=1}^{m} \gamma^{k_n} \leq \limsup_{n\to\infty} \gamma^{k_n} \leq \limsup_{k\to\infty} \gamma^k \leq 0$. Therefore, there always exists a subsequence indexed by $\mathcal{K}$ such that

$$\lim_{k\in\mathcal{K}} \gamma^{k+1} = 0,$$

with probability one. Let $x^* \in X^*$ be an optimal solution. Notice that $\gamma^{k+1} = f_k(x^{k+1}) - f_k(\hat{x}^k) \leq f_k(x^*) - f_k(\hat{x}^k)$ for all $k \in \mathcal{K}$. Let $\hat{x}$ be an accumulation point of

$\{\hat{x}^k\}_{k\in\mathcal{K}}$. Assume that $\mathcal{K}' \subset \mathcal{K}$ and $\{\hat{x}^k\}_{k\in\mathcal{K}'} \to \hat{x}$. Therefore, we have

$$\begin{aligned}
& \lim_{k\in\mathcal{K}'}[\gamma^{k+1} + f_k(\hat{x}^k)] \\
= \quad & f(\hat{x}) \\
\leq \quad & \limsup_{k\in\mathcal{K}'} f_k(x^*) \\
\leq \quad & cx^* + \tfrac{1}{k}\textstyle\sum_{t=1}^{k} h(x^*, \omega^t) \\
= \quad & f(x^*),
\end{aligned}$$

with probability one. Therefore, $\hat{x} \in X^*$ with probability one. ■

CHAPTER 3: ENHANCEMENTS OF TWO-STAGE SD ALGORITHM

3.1 Introduction of two extensions

In this chapter, we present two extensions in regularized stochastic decomposition algorithm. The extensions cover both theoretical and computational issues. In particular, we analyze two aspects. First, we examine conditions under which the regularized SD algorithm produces a sequence of incumbent solutions that converge to a unique limit. Next, we discuss a computational enhancement that allows the formations of cuts using re-sampling. Thus, the user can trade-off solution times with solution quality and further speed up the computations in the regularized SD algorithm.

Moreover, we show that without loss of asymptotic properties, it is possible to let a user speed up calculations in any iteration by avoiding the need to use every previously generated outcome in the "argmax" process of cut generation. We compare the computational results of SD algorithm with and without re-sampling using several instances.

3.2 The unique limit for incumbent sequence

In this section, we study the conditions that the regularized SD algorithm will converge to a unique limit. Theorem 5 provides the results, where the regularized SD algorithm converges to an optimal solution with probability one.

Moreover, [32] shows how one may test for a convergence of subsequence by combining the definition of K^* with stopping rules for regularized SD. However, these results do not guarantee that the sequence has a unique limit. The following theorem provides this result.

Theorem 6 *Let $\{\hat{x}^k\}$ denote the entire sequence of incumbent solutions generated by regularized SD algorithm. Let f be defined as in* (1)*, then $\{\hat{x}^k\} \to x^*$ with probability one.*

Proof: By considering whether the incumbent sequence changes infinitely many times, we divide the proof into two cases. If the sequence of incumbent solutions changes only finitely many times, then it is obviously true that the sequence of incumbent solution has a unique limit x^*.

Next, we consider the case that the sequence of incumbent solution changes infinitely many times. By the results of equation 2.6 on page 115 in [31], we have

$$f_k(x^{k+1}) - f_k(\hat{x}^k) \leq -||x^{k+1} - \hat{x}^k||^2 \leq 0. \tag{23}$$

Let K_0 denote the iterations at which the incumbent solution changes and consider the m successive indices in K_0, we have

$$\frac{1}{m}\sum_l [f_{k_l}(x^{k_l+1}) - f_{k_l}(\hat{x}^{k_l})] \leq -\frac{1}{m}\sum_l ||\hat{x}^{k_l+1} - \hat{x}^{k_l}||^2 \leq 0. \tag{24}$$

Since $k_l \in K_0$, $\hat{x}^{k_{l+1}} = x^{k_l+1}$ holds. Moreover, the left hand side of (24) can be written as:

$$\Delta_m = \frac{1}{m}[f_{k_m}(\hat{x}^{k_{m+1}}) - f_{k_0}(\hat{x}^{k_0})] + \frac{1}{m}\sum_{l=0}^{m-1}[f_{k_l}(\hat{x}^{k_{l+1}}) - f_{k_{l+1}}(\hat{x}^{k_{l+1}})]. \tag{25}$$

Based on the results in Lemma 5, the second term (summation term) in (25) converges to 0 and the first term is bounded (the approximation of f_k has a lower and upper bound). Hence, $\lim_{m\to\infty}\Delta_m \to 0$. Based on (24), we have $\lim_{m\to\infty}\frac{1}{m}\sum_l ||\hat{x}^{k_{l+1}} - \hat{x}^{k_l}||^2 \to 0$. Therefore, the entire sequence of incumbent solutions converge and the uniqueness of the limit follows. ■

3.3 SD with re-sampling

In this section, we discuss the potential reduction of computational effort by using re-sampling, which is also known as bootstrapping ([19]). Generally speaking, in stochastic decomposition, there are two sources of errors; one is the statistical approximation error and the other error results from linearization using cuts.

Although the regularized SD algorithm can achieve asymptotic optimality with finite master programs, asymptotic convergence (with probability one) requires that the number of observations in any cut grow indefinitely. The issue can be explained as follows: in every iteration, the stochastic decomposition algorithm requires all previously generated samples to form a new cut. Therefore, as the number of iterations increases, the computational effort in cut generation also grows. However, it may not be necessary to make full use of all the generated samples to construct a new cut. In addition, some of the samples may not be very useful in forming a new cut, which motivates us to apply the concept of re-sampling in stochastic decomposition.

The idea of re-sampling is straightforward: at iteration k, instead of using all the generated samples (k) to construct a new cut, we only use a relatively small number of samples (L_k and $L_k < k$) to form a new cut. When k is large, we can reduce the computational effort significantly by choosing $L_k \ll k$. With this modification, one only needs to carry out the comparison in (5) for L_k elements. Moreover, the convergence results of SD carry over into the modified SD with re-sampling. Before providing the convergence results of SD with re-sampling, we briefly summarize the bootstrap method.

Let $\{\omega_1, ..., \omega_k\}$ be a random sample of size k of a random variable $\tilde{\omega}$ with distribution F. Let F_k be the empirical distribution of $\{\omega_1, ..., \omega_k\}$. Define a random variable $T(\omega_1, ..., \omega_k; F)$, which depends upon distribution F. The bootstrap method provides a way to approximate the distribution of $T(\omega_1, ..., \omega_k; F)$ by $T(\theta_1, ..., \theta_k; F_k)$, where $\{\theta_1, ..., \theta_k\}$ denotes a random sample of size k under distribution F_k. A justification for bootstrapping is provided by the following theorem in [68], where the sample mean is approximated by bootstrapping, that is: $T(\omega_1, ..., \omega_k; F) = 1/k \sum_{i=1}^{k} \omega_i$, and the sup-norm ($||G||_\infty = sup_{s \in R}|G(s)|$) is used to study convergence.

Lemma 6 *Let $\bar{\theta}_k = 1/k \sum_{i=1}^{k} \theta_i$, $\bar{\omega}_k = 1/k \sum_{i=1}^{k} \omega_i$. Let $\mu = E_F(\tilde{\omega})$ denote the expectation of $\tilde{\omega}$, and let $E_F(\tilde{\omega})^2 < \infty$. Let P and P_k denote the probabilities under*

F and F_k respectively. Then

$$||P\{k^{1/2}(\bar{\omega}_k - \mu) \leq s\} - P_k\{k^{1/2}(\bar{\theta}_k - \bar{\omega}_k) \leq s\}||_\infty \rightarrow 0, \quad a.s.$$

Proof: See [68]. ■

Basically, Lemma 6 studies the uniform convergence of the discrepancy between distribution of the quantity $k^{1/2}(\bar{\omega}_k - \mu)$ and its bootstrapped approximation. Motivated by this result, we present asymptotic properties of stochastic decomposition with re-sampling, where an approximation of the sample mean function $H_k(x^k)$ is of primary interests. Therefore, we provide the following theorem.

Theorem 7 a) *Assume $E(h(x^k, \tilde{\omega}))^2 < \infty$ for all k. Let $\mu_k = E_F(h(x^k, \tilde{\omega}))$, $\bar{h}_k(x^k) = 1/k \sum_{i=1}^k h(x^k, \omega_i)$, $\hat{h}_k(x^k) = (1/\sum_{i=1}^k I_i) \sum_{i=1}^k I_i h(x^k, \omega_i)$, where I_i is a realization of a Bernoulli random variable $I \sim$ Bernoulli(p). Let P and P_k denote the probabilities under F and F_k respectively. Then*

$$||P\{k^{1/2}(\bar{h}_k(x^k) - \mu_k) \leq s\} - P_k\{k^{1/2}(\hat{h}_k(x^k) - \bar{h}_k(x^k)) \leq s\}||_\infty \rightarrow 0, \quad a.s.$$

b) *Assume $E(\Theta(x^k, \tilde{\omega}))^2 < \infty$ for all k. Let any cut defining $f_k(x)$ be denoted $\Theta(x, \{\omega_n\}_{n=1}^k) = c^\top x + \text{Max}_{t=1,\dots,k} \left\{ \frac{t}{k} \times \frac{1}{t} \sum_{\ell=1}^t (\pi_\ell^t)^\top [r(\omega_\ell) - T(\omega_\ell) x] \right\}$ and the coefficients of the subgradient associated with the cut be denoted by $\Theta_j(x, \{\omega_n\}_{n=1}^k), j = 1, ..., m$. Let $\mu_{\Theta_j(x^k)} = E_F(\Theta_j(x^k, \tilde{\omega}))$, $\bar{\Theta}_{jk}(x^k) = 1/k \sum_{i=1}^k \Theta_j(x^k, \{\omega_n\}_{n=1}^i)$, $\hat{\Theta}_{jk}(x^k) = (1/\sum_{i=1}^k I_i) \sum_{i=1}^k I_i \Theta_j(x^k, \{\omega_n\}_{n=1}^i)$, where I_i is a realization of a Bernoulli random* *variable $I \sim Bernoulli(p)$. Let P and P_k denote the probabilities under F and F_k respectively. Assume that dual vectors defining the cuts are chosen as in SD, but with an additional requirement that among alternative dual optima in the "argmax" process, we choose the one which was discovered first during the algorithmic process. Then,*

$$||P\{k^{1/2}(\bar{\Theta}_{jk}(x^k) - \mu_{\Theta_j(x^k)}) \leq s\} - P_k\{k^{1/2}(\hat{\Theta}_{jk}(x^k) - \bar{\Theta}_{jk}(x^k)) \leq s\}||_\infty \to 0, \quad a.s.$$

Proof: a): This is simply an application of Lemma 6.

b): First note that as $k \to \infty$ the set $V_k \to V$, and note that one can impose an order on this set based on the sequence in which they were discovered during the algorithm. Moreover, in the "argmax" operation, a dual vertex beats another only if its objective value is strictly larger than one of the previously tested vertices. Since the ordering of dual vertices remains fixed, it follows that for any subsequence $\{x^k\}_K \to x^*$, the dual vertex that provides the value $\{h(x^k, \omega^t)\}_{k\in K}$, remains fixed (for a given ω^t), as $k \to \infty, k \in K$. Hence the cut coefficients generated by the "argmax" process are continuous over x for a given ω^t. The consistency of the re-sampled cut now follows using the continuity of $h(x, \tilde{\omega})$. ■

For readers familiar with implementing SD, we observe that one can implement this re-sampling version without changing the code a great deal. One relatively straightforward way to include re-sampling within the cut generation process is to accept/reject an outcome within a sample, based on a Bernoulli random variable. Thus, if p ($p > 0$) is the acceptance (success) probability, then we generate a uniform random variate for each previously generated outcome, and use only those outcomes ($\omega^t, t < k$) for which the random number is less than p. Clearly, as p increases, the cut generation process tends to use more outcomes in the approximation.

3.4 Computational results

In this section, we provide computational results comparing both versions of regularized SD, with and without re-sampling.

• *Instances.* We demonstrate the effectiveness of re-sampling by solving both small scale instances and larger ones. Among the smaller instances, we use CEP1, which is a machine capacity planning problem, and PGP2 which is a power generation planning problem ([30]). Among the larger problems, we test our methodology with 20TERM ([41]), STORM ([42]), both of which arose in logistics planning, and SSN which is a telecommunication planning problem ([61]).

• *Environment.* The LP solver in our computation uses the ILOG CPLEX callable library, version 10.0, and all programs were compiled and run in a Unix environment on a Sun workstation (Sun Fire V440).

• *Procedure.* We start the experiments by first running SD using a stopping tolerance of $\epsilon = 0.0001$ (see [32]). Given the number of iterations obtained in this manner, we resolve the instances using re-sampling with an acceptance probability p of 0.6.

We begin by reporting computational results for the small instances, namely, CEP1 and PGP2. In Figures 1-4, we compare solution times and objective values for both versions of regularized SD (i.e. with and without re-sampling). Figures 1 and 2 focus on CEP1, for which we run the first 30 iterations without re-sampling (in either version). For the version with re-sampling, we start the re-sampling process at iteration 30. As shown in Figure 1, the total time (as well as the time per iteration) are reduced by using re-sampling. The important question however is whether the re-sampling process generates solutions that are comparable in quality. To answer this question we present Figure 2 which depicts the objective function values of points visited by the re-sampling process. These values were obtained by running an out-of-sample evaluator that samples the objective function, given a first stage solution used in a particular iteration.

As shown in Figure 2, the objective values associated with these points (from the re-sampled algorithm) are higher, but as suggested by Theorem 3, the difference between the objective values diminishes as the iterations proceed. And after about 80 iterations, the objective function values associated with both algorithms are very close, although the amount of time taken by the re-sampled scheme is lower. The same general trend is observed for PGP2, for which we started the re-sampling process at iteration 60, and ran this instance until iteration 160. As shown in Figures 3 and 4, we observe that the solutions provided by both methods are comparable (446.0231 vs 445.8673), while reducing computational time via re-sampling.

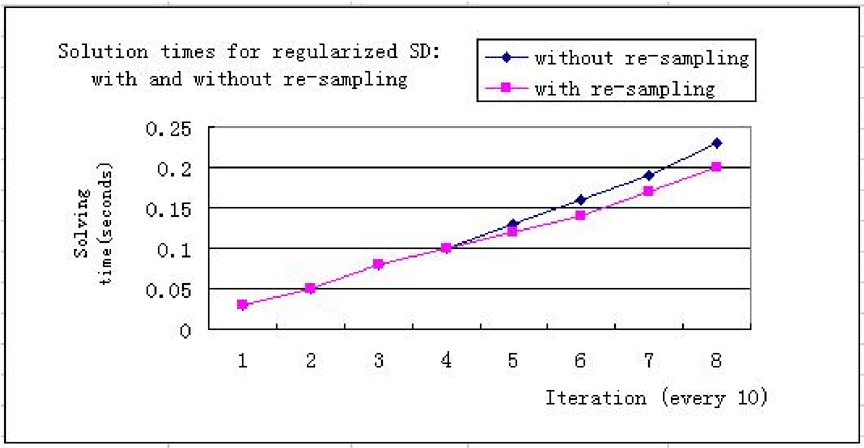

Figure 1: Solution times of Regularized SD for CEP1: with and without re-sampling

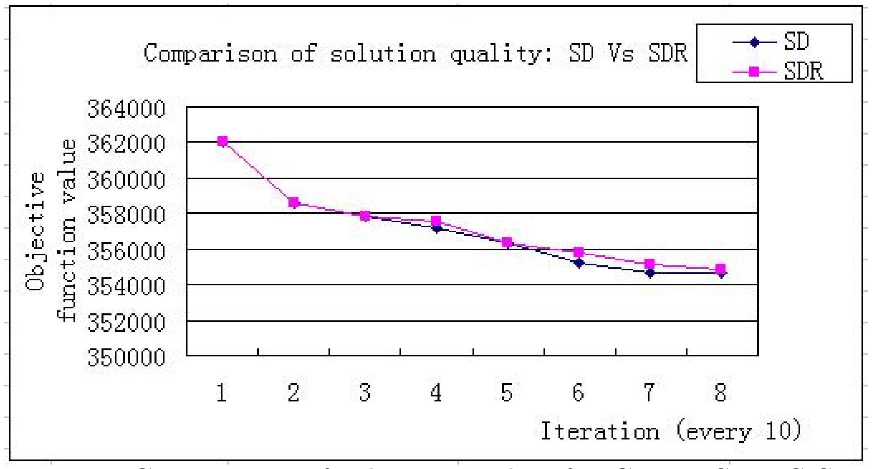

Figure 2: Comparison of solution quality for CEP1: SD VS SDR

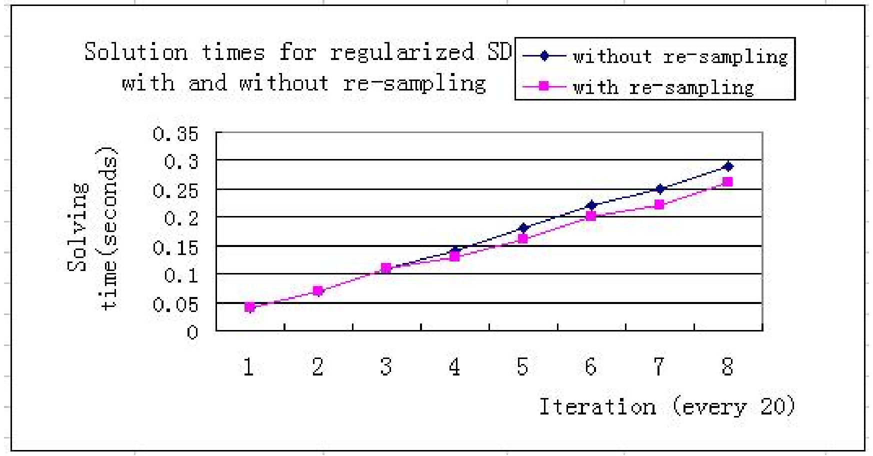

Figure 3: Solution times of Regularized SD for PGP2: with and without re-sampling

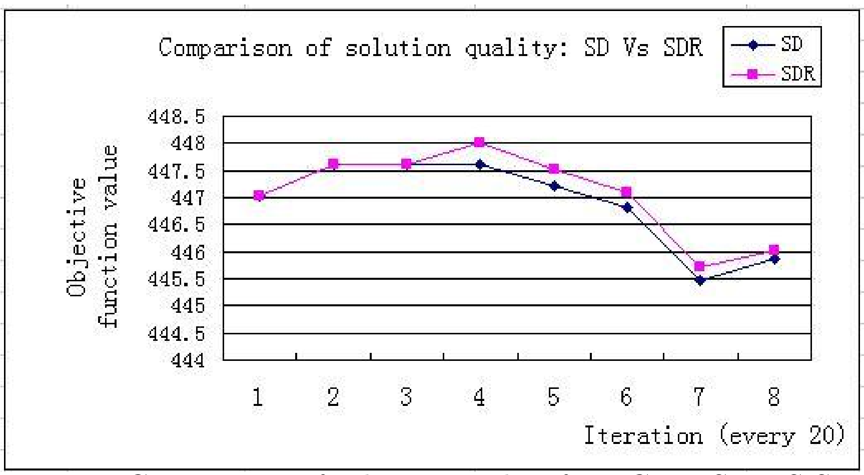

Figure 4: Comparison of solution quality for PGP2: SD VS SDR

Next, we compare both versions of regularized SD by solving some larger instances: 20TERM, STORM and SSN. For each of these instances, we start the re-sampling process after 300 iterations although the total number of iterations vary: 800 for 20TERM, 2400 for SSN and 1500 for STORM. Recall that these iteration counts were determined by running the regularized version of SD without re-sampling, and using the stopping rule that is designed for SD ([32]).

Basically, Figure 5 reports the solution times of 20TERM for the two versions of regularized SD we are comparing. We record the CPU time (in seconds) every 100 iterations. As illustrated in Figure 5, there is no difference between the two versions for the first 300 iterations because the re-sampling process was started only after 300 iterations. However, after 300 iterations, the re-sampled version gains speed as iterations proceed. Moreover, at iteration 800, the re-sampled version takes 62 seconds compared to 84.5 seconds for regularized SD without re-sampling, which results in a reduction of 26.7% in computational time.

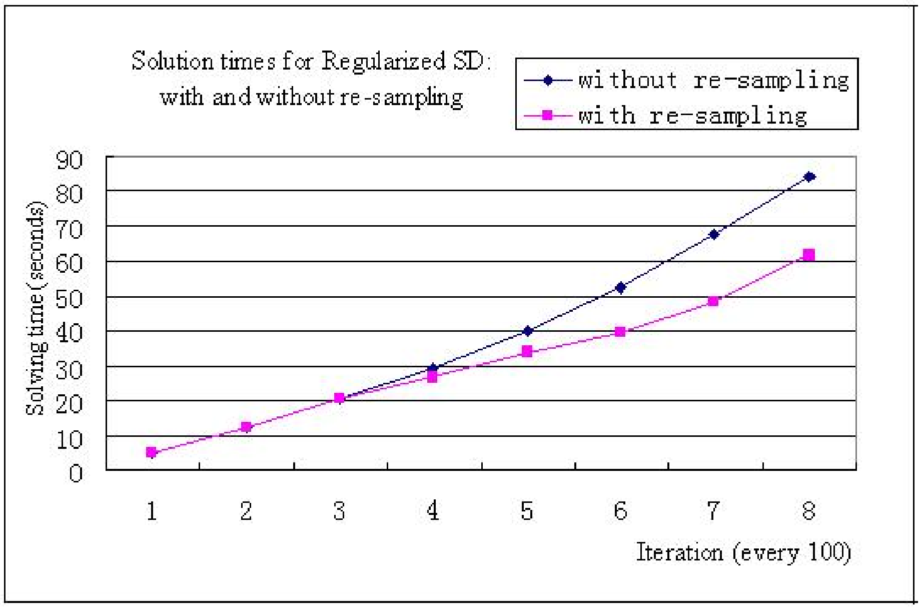

Figure 5: Solution times of regularized SD for 20TERM: with and without re-sampling

Figure 6 demonstrates the solution quality of 20TERM obtained by both versions of regularized SD. As expected, there is no difference between these versions of SD for the first 300 iterations. At iteration 400, there is a jump in objective function value due to re-sampling. Once again, the conclusion of Theorem 7 is demonstrated in Figure 6, where the objective values obtained by the re-sampled version are not as good in the early iterations of re-sampling, but the two versions begin to converge to the same value as iterations proceed. As a matter of fact, at iteration 800, one observes scant difference in objective function value between the two versions, with the original regularized SD version yielding 254561.8310 and the re-sampled version providing a slightly higher value at 254572.1976.

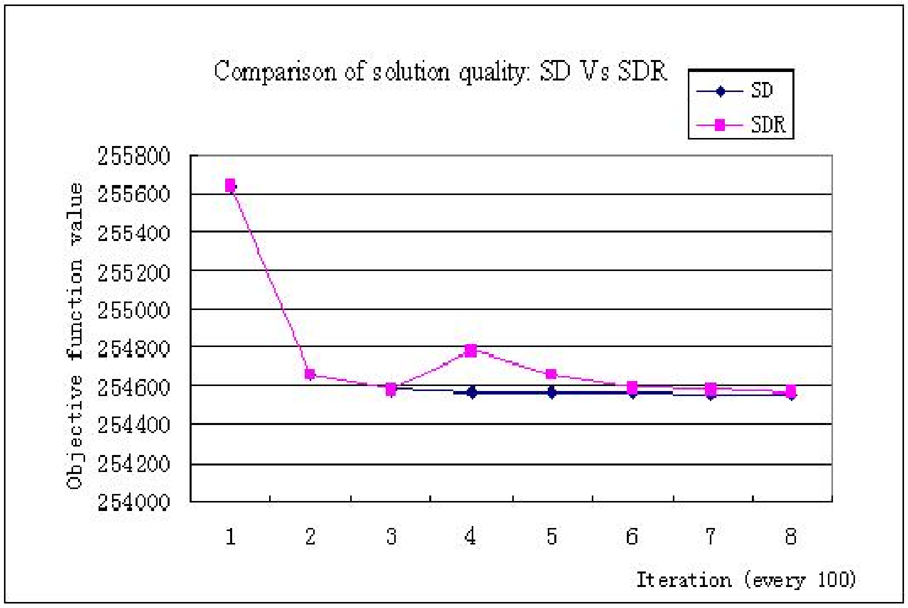

Figure 6: Comparison of solution quality for 20TERM: SD VS SDR

While the computations reported above indicate the benefits from re-sampling, they really do not convey the full extent of savings that can arise from this enhancement. In order to give the reader a more complete sense of the advantages, we now present computations associated with two of the most challenging instances (SSN and STORM) in the SLP literature. Because of the challenging nature of these problems, [39] have reported computational results by solving these problems using grid computing. Moreover, because of the need to replicate the runs (using independent seeds), savings due to the re-sampling approach can become additively substantial.

We demonstrate this advantage below.

Table 1 reports the computational times and objective values for SSN and STORM using 20 replications for each. From the results in Table 1, we obtain the total computational times for each method. For SSN, the total running time for 20 replications without re-sampling is 28104.16 secs (approximately 7.8 hours) on a Sun Fire V440. Upon re-sampling, the total computational time was 20510.593 (approximately 5.7 hours), representing a reduction of about 27% (2.1 hours). We note that the best value obtained in these replications is approximately 10.0358, whereas the best value from the re-sampled version is approximately 10.0949, which is also demonstrated in Figure 7.

As for STORM, the total computational time (over 20 replications) was reduced by 28%, and Figure 8 shows the objective values obtained by the two methods. At first glance, the approximations in Figure 8 might appear to be poor. However, notice that the entire range of objective values $[1.55 \times 10^7, 1.565 \times 10^7]$ is less than 1% of the smallest value. Hence, this variability is marginal.

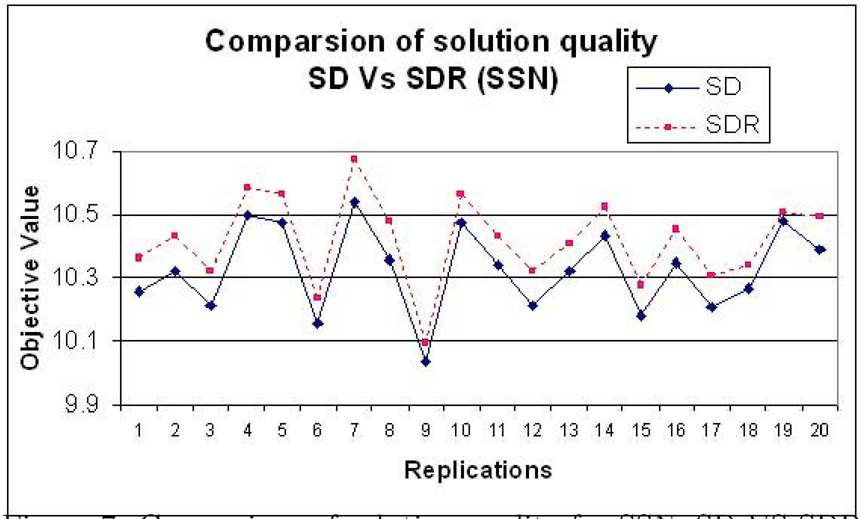

Figure 7: Comparison of solution quality for SSN: SD VS SDR

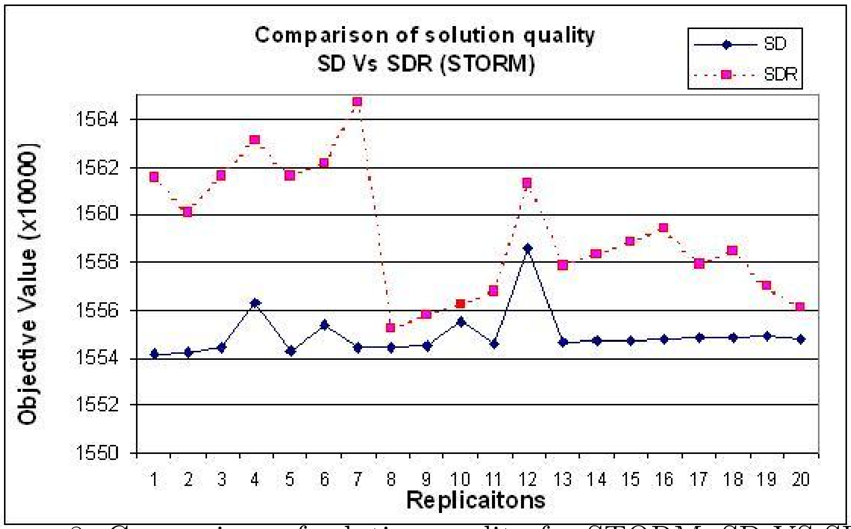

Figure 8: Comparison of solution quality for STORM: SD VS SDR

Table 1: Comparison of CPU time and estimated objective value for SD and SDR in large scale instances with 20 replications: SSN and STORM

	SSN(2400)	SSN	STORM(1500)	STORM
	CPU Time(secs)	Obj. Value	CPU Time(secs)	Obj. Value
SD	1307.35	10.2566	1620.54	15541695
SDR	923.13	10.3624	1163.53	15615752
SD	1233.43	10.3248	1641.28	15542102
SDR	904.45	10.4331	1148.46	15600974
SD	1445.67	10.2111	1662.02	15544510
SDR	1046.92	10.3215	1133.39	15616195
SD	1532.62	10.4976	1682.76	15562917
SDR	1174.28	10.5843	1158.32	15631417
SD	1301.78	10.4745	1703.5	15543325
SDR	936.17	10.5639	1193.25	15616638
SD	1490.58	10.1578	1564.24	15553733
SDR	1025.76	10.2392	1088.18	15621860
SD	1579.39	10.5371	1744.98	15544140
SDR	1175.35	10.6747	1273.11	15647081
SD	1168.95	10.3574	1665.72	15544548
SDR	864.91	10.4808	1158.04	15552303
SD	1457.23	10.0358	1786.46	15544956
SDR	1054.53	10.0949	1242.97	15557525
SD	1145.06	10.4751	1607.2	15555363
SDR	834.13	10.5629	1227.9	15562746
SD	1534.61	10.3443	1727.94	15545771
SDR	1183.77	10.4331	1212.83	15567968
SD	1223.45	10.2136	1648.68	15586178
SDR	923.32	10.3219	1197.76	15613189
SD	1312.22	10.3229	1769.42	15546586
SDR	942.91	10.4071	1282.69	15578411
SD	1401.05	10.4322	1690.16	15546994
SDR	1022.53	10.5235	1267.62	15583633
SD	1489.83	10.1815	1710.9	15547401
SDR	1042.95	10.2775	1252.55	15588854
SD	1578.35	10.3458	1831.64	15547809
SDR	1161.87	10.4533	1337.48	15594076
SD	1167.44	10.2061	1652.38	15548217
SDR	851.79	10.3087	1222.41	15579297
SD	1756.25	10.2664	1773.12	15548624
SDR	1210.81	10.3441	1307.34	15584519
SD	1545.05	10.4787	1693.86	15549032
SDR	1170.463	10.5095	1192.27	15569741
SD	1433.85	10.3888	1689.32	15547513
SDR	1060.55	10.4912	1207.34	15561221

CHAPTER 4: MULTISTAGE STOCHASTIC DECOMPOSITION

4.1 Introduction

Solution of constrained stochastic decision models remains the important branch of optimization. Despite decades of research on optimization models and algorithms, our ability to solve constrained stochastic optimization problems remains a continuing challenge. One might lay the blame on the well known curse of dimensionality associated with Dynamic Programming (DP, e.g. [52]). However, the difficulties go beyond DP. Other approaches to stochastic optimization problems (e.g. [65]a) also present a rather bleak outlook for constrained stochastic optimization.

Despite these difficulties, practitioners, and specialists in the art of modeling do not have the luxury of setting these problems aside. On a daily basis, decisions are made in the face of uncertainty while accommodating physical and financial constraints. Moreover, these systems are inherently dynamic, with the simplest models resulting in two-stage problems. We will focus on multistage problems, and in this setting, the demands on optimization algorithms rise to a whole new level. In this chapter, we introduce the notion of optimization by simulating the algorithmic process, and hence we refer to this process as "Optimization Simulation".

Some algorithms that use deterministic approximations and sampling approximations have been mentioned in chapter 1. Although the most popular sampling-based methods (e.g. [47] and [16]) provide practical approaches for approximate solutions of MSLP with large complicated scenario trees, there are computational bottlenecks: a) even with discrete valued stochastic processes, it is unclear how one might interface with a simulation that might have the ability to generate a sample path, b) asymptotic convergence proofs require iterations which traverse the entire scenario tree at some steps of the method ([40]), and c) stage-wise independence of the stochastic process appears to be critical ([66]).

In contrast, the multistage SD method proposed in this chapter provides revised updates as more sample paths are observed sequentially. Indeed, as with its two-stage predecessor ([30] and [31]), the multistage SD method learns to approximate value functions, as well as decisions in a sequential manner. Moreover, this sequential process is shown to provide asymptotic optimality without requiring us to traverse the entire tree in any particular iteration.

The plan of this chapter is as follows. Section 4.2 begins with a multistage stochastic linear programming problem setting. The multistage stochastic decomposition algorithmic schema, as well as a discussion comparing MSD and other sampling methods are presented in section 4.3. In sections 4.4 and 4.5, we prove its asymptotic convergence properties. Finally, sections 4.6 and 4.7 study a production and inventory instance and provide the preliminary computational results using multistage stochastic decomposition algorithm.

4.2 Multistage stochastic decision models

In our formulation, we consider a $T+1$ stage multistage stochastic linear programs. In addition, we assume that time will march ahead from now ($t = 0$) to the end of horizon ($t = T$), where T is a given positive integer. With this notation, there will be

$T+1$ stages in the formulation. Thus, as a special instance, the two stage formulation will have stages indexed by $t = 0$ and $T = 1$.

Let $(\Omega, \mathcal{F}, \Gamma)$ denote the filtered probability space (i.e. $\mathcal{F}_t \in \mathcal{F}$, for $t = 1, ..., T$ and $\mathcal{F}_{t_1} \in \mathcal{F}_{t_2}$, for $t_1 < t_2$). Usually, $\Omega = \Omega_1 \times ... \times \Omega_T$ ($\Omega_t \in \mathcal{R}^{v_t}$ with v_t a positive integer). One generated scenario/path is denoted as $\underline{\omega}_t = (\omega_1, ..., \omega_t)$, which consists of t elements of the process. In addition, the corresponding random variable is denoted as $\underline{\tilde{\omega}}_t$.

In the filtered probability space, $\mathcal{F}_t$, the σ-algebra denotes the collection of available data information to the decision maker at current time period t. In particular, for simplicity, we restrict the random variables $\underline{\tilde{\omega}}_t$ to be finite for all t. Moreover, $\mathcal{F}_t$ is generated by a finite partition $\{\Theta_t^l\}$ of Ω and $\mathcal{F}_{t+1}$ is finer than $\mathcal{F}_t$ (i.e. for any set $\bar{\Theta} \in \{\Theta_t^l\}, \exists$ a collection of sets in $\{\Theta_{t+1}^l\}$, indexed by $\mathbb{C}(\bar{\Theta})$, say, such that $\bar{\Theta} = \cup_{l \in \mathbb{C}(\bar{\Theta})} \Theta_{t+1}^l$).

The relationships between the sets in the partition Θ_t^l and their children $\mathbb{C}(\{\Theta_t^l\})$ can be encoded in the form of a tree which is termed as a scenario tree in stochastic programming. A node in period t represents a subset of paths (such as $\bar{\Theta}$), which have the same events in the first t periods, and record some new event in period $t+1$. Since algorithms will work by using the tree as a road-map, it will be convenient to index nodes on the scenario tree by n, and denote the (unconditional) probability of reaching node n by p_n.

Moreover, $n+$ is the children node of node n, which is $n+ \in \mathbb{C}(n)$. And $\tilde{n}+$ represents any random children node of n. By the same token, $n-$ denotes the parent of node n. Finally, letting $p_{(n+|n)}$ denote the probability of reaching node $n+$, given that the process has arrived at node n. Therefore, we can generate a sample path through the scenario tree from the root node to terminal node. On the other hand, if $\underline{\tilde{\omega}}_t$ is a continuous stochastic process, we need to take great care of measurability issues ([11]).

The multistage SD method will work with sample paths which is generated from the scenario tree. We use the notation N to represent the set of nodes in the scenario tree. In the following statement of the sequential decision model, we associate a time index $t(n)$ with each node n. We will use the following convention in the model stated below.

a) The root node is indexed by node 0.

b) If node n belongs to the last stage, the expected value/recourse function of $n+$ (the future) is 0.

c) Finally, we define composite state variables $\underline{s}_n = (x_n, \underline{\omega}_n)$. Note that the composite state $\underline{s}_n$ used here traces the entire history of the data state $\underline{\omega}_n$. Thus by using the scenario tree, SP allows us to capture the history of the process. However, the state x_n is only defined at node n, and is data and decision dependent, prompting us to refer to them as the "3d" states.

d) The initial state x_0 is given.

In the following, we introduce the mathematical formulation of the multistage stochastic linear program model:

$$\text{Min}\{d_0^T u_0 + E[h_{0+}(\tilde{\underline{s}}_{0+})] : u_0 \in U_0, x_{0+} = a_{0+} + A_{0+}x_0 + B_{0+}u_0 \; a.s.\}, \tag{26}$$

where h_n are defined recursively for $n \geq 1$ as

$$h_n(\underline{s}_n) = c_n^T x_n + \text{Min}\{d_n^T u_n + E[h_{n+}(\tilde{\underline{s}}_{n+})] : u_n \in U_n(x_n), \tag{27}$$

$$U_n(x_n) = \{u_n | D_{t(n)} u_n \leq b_n - C_n x_n\} \; a.s.\} \tag{28}$$

and

$$x_{n+} = a_{n+} + A_{n+}x_n + B_{n+}u_n, \;\; a.s. \tag{29}$$

Since the dependence of the value function on data can be captured via the node index n, without loss of generality, we drop the dependence on $(\underline{\omega}_n)$, and simply write $(c_n, d_n, A_n, B_n, C_n, D_n, a_n, b_n)$ as data for node n. Moreover, the constraints on the decisions at node n are written as $u_n \in U_n(\underline{s}_n)$ a.s. However, in order to appeal to the ideas of two-stage SD ([28]), and to ease some of the computational burden, we assume $D_n = D_{t(n)}$, that is, these instances satisfy a multistage version of the fixed recourse assumption for two-stage problems. All other data elements are allowed to depend on the state of the stochastic process at node n.

The parallels between the two-stage and the multistage stochastic program should be clear. As usual, the above recursive statement is such that the multistage model can be conceptually interpreted both as a specialization as well as a generalization of a two stage model. From a computational point of view, however, the presence of a nested collection of conditional expectation functions, poses far more serious computational challenges for the multistage case.

There are at least four advantages to the above formulation of the multistage SP model: a) the value function is stated in terms of the (state) variables $\underline{s}_n$ that couple successive stages in the model, and since these are usually in a lower dimensional space than the decision variables (u_n), the approximations are more manageable and scalable; b) in the course of the algorithmic development, it will become clear that SD provides a bridge between SP and ADP; c) the above notation is standard in dynamic systems theory and software (such as Matlab), and finally, d) our approach actually extends asymptotic convergence properties of both SP and ADP.

Item a) above has also been observed in [52] who suggests that in resource allocation applications, most models have far fewer coupling (state) variables leading to approximations in lower dimensional spaces. Other applications, such as financial models, also satisfy such properties because the number of stocks that determine the set of decision variables is often much larger than the portfolio or the class of investments tracked over time. As for items b) and c), we note the role of SD as unifying algorithm between SP and ADP. In this sense, the notion of approximations has been a central focus of stochastic programming algorithms which allow both path dependent stochastic processes, as well as, constraints in the stochastic decision model. Finally recall that the implications of item d) have already been discussed in the introductory section.

4.3 Multistage stochastic decomposition algorithm

In the following, we will use certain extensions of the regularized version of the two-stage SD algorithm (See chapter 2). In the multistage case, each non-terminal node $n \in \mathcal{N}$ will be endowed with a mechanism to initialize and update incumbent decisions, denoted $\hat{u}_n^{k-1}$ in iteration k. As with two-stage SD, $\hat{u}_0^{k-1}$ will represent a solution that is estimated to be the best decision observed prior to iteration k (for stage 0). The incumbent decisions for the subsequent nodes are noted to be those future decisions that support the choice $\hat{u}_0^{k-1}$. Candidate decisions, denoted u_0^k, will refer to those solutions that are obtained by solving nodal decision simulations described below. They are referred to as candidates because they may replace incumbent decisions. Because we will traverse only one path in any iteration, the algorithm will not visit other nodes that are not on the traversed path. Accordingly, candidate decisions will be generated for only a subset of nodes of the observed scenario tree, and these may become incumbent decisions if certain criterions are satisfied. Note that nodes that are not on the sample path for iteration k will not change their incumbent decisions. The states x_n generated using incumbent decisions will be referred to as incumbent states, and those associated with candidate decisions will be referred to as candidate states.

We begin by presenting a summary of the multistage stochastic decomposition algorithm which is discussed in greater depth subsequently.

1. Simulate a sample path $\underline{\omega}_T^k$ *and for all nodes of the scenario tree, update counts as well as frequencies reflecting the number of visits, and the fraction of visits. We will refer to nodes along the sample path by the notation* $n \in \underline{\omega}_t^k$ *(If one initializes all counts to be zero, then one can simply update counts (of visits to nodes) on the sample path observed in the current iteration.)*

1.1 (No nodes on the path are new.) If the current sample path, denoted $\underline{\omega}_n^k$*, has been revealed in some prior iteration, then assume that approximations* f_n^{k-1} *are available for all* $n \in \underline{\omega}_t^k$*, as well as incumbent states and decisions for all* $n \in \underline{\omega}_t^k$*. Starting with* $n = 0$*, we will optimize* f_n^{k-1} *for a given state* x_0*, denote the solution as* u_n^k *and then identify a candidate state* $x_{n+}^k = a_{n+} + A_{n+}x_n^k + B_{n+}u_n^k$*. Using the latter state, we obtain a candidate decision* u_{n+}^k *by solving a nodal decision simulation (NDS, see (30)). This process is repeated for all non-terminal nodes on the sample path.*

1.2 If some nodes $n \in \underline{\omega}_T^k$ *have not been visited in previous iterations, then perform operations of step 1.1 until no such nodes are available, and then, solve two scenario LPs associated with the remainder of the path. One LP is initialized with the incumbent decision, and the other with the candidate decision. The resulting solutions yield the incumbent and candidate sequences for nodes on the entire path* $\underline{\omega}^k$*.*

2. Solve the nodal dual approximation (NDA see (32)). If control to this step is passed from step 1, then, update counts as well as frequencies reflecting the number of visits, the empirical conditional probability of visits, and use the terminal node for the sample path as node n; otherwise use n as dictated by step 3 below. Solve $NDA(n)$

using the incumbent state $\hat{\underline{s}}_n^{k-1}$*, as well as the candidate state* $\underline{s}_n^k$*, and proceed to step 3 using node* $n \leftarrow n-$. *Note that this step is only performed for non-root nodes.*

3. Update approximations. Using n *provided by step 2, collect information regarding sub-gradients for* h_{n+}^k *(see(32)), and form two affine lower bounding approximations for* h_n^k *(One is the approximations obtained for the incumbent state* $\hat{\underline{s}}_n^{k-1}$*, while the other is for the candidate state* $\underline{s}_n^k$*). These updates provide functions* f_n^k *(see (35)). If* $n = 0$*, we proceed to step 4; otherwise, we return to step 2 with the current node* n*. (This step is only performed for non-root or non-terminal nodes).*

4. Update the incumbent. Let $q \in (0,1), \bar{\sigma} \geq \underline{\sigma} \geq 0$ *be given. If* $f_0^k(u_0^k, x_0) - f_0^k(\hat{u}_0^{k-1}, x_0) \leq q[f_0^{k-1}(u_0^k, x_0) - f_0^{k-1}(\hat{u}_0^{k-1}, x_0)]$, $\hat{u}_0^k \leftarrow u_0^k, \sigma_k \leftarrow \mathrm{Max}\{\ \frac{\sigma_k}{2},\ \underline{\sigma}\}$ *otherwise, we continue with* $\hat{u}_0^k \leftarrow \hat{u}_0^{k-1}$; $\sigma_k \leftarrow \mathrm{Min}\{2\sigma^k, \bar{\sigma}\}$. *For nodes that do not belong to the* k^{th} *sample path, set* $h_n^k \leftarrow h_n^{k-1}$. *Increment the iteration counter* k*, and repeat from step 1.*

Figure (9) presents a caricature of one iteration of the algorithm. In the following, the solid lines joining the blank nodes represent the path generated in a forward pass of the algorithm, and the nodes marked with a cross represent previously visited children of nodes associated with the current sample path. The calculations in step 1.1 generate the scenario shown by the blank nodes, and the computations necessary for obtaining a candidate sequence for that scenario. The calculations in step 1.2 are intended to generate initial incumbents for a new scenario. As for other calculations, step 2 requires the solution of one LP for each non-root node on the sample path. Indeed, step 2 starts with the terminal node (on the sample path), and upon solving the NDA for that node (referred to as n), the algorithm proceeds to calculate the approximations for node $n-$, which will be designated as node n for step 3. Once the approximation is developed in step 3, it returns to step 2, with node n and one solves a new $NDA(n)$. In this manner, the algorithm moves backward in time until step 3 processes the root node (indexed by 0). At this point, a new approximation for the root node is at hand, and the algorithm proceeds to step 4.

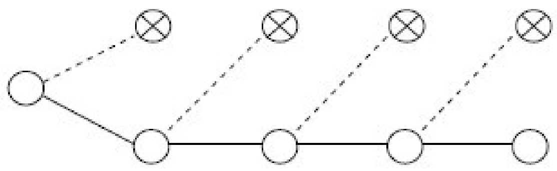
Figure 9: Caricature of visits to nodes within one iteration

It is important to recognize that in any iteration, approximations of the expected recourse functions are developed only for those nodes n that belong to the sample path generated in iteration k (i.e. $n \in \underline{\omega}_T^k$). Thus, unlike other approaches for optimization simulation in stochastic programming (e.g. [16] and [40]), the multistage SD algorithm creates approximations of the expected recourse function of non-terminal nodes of the sample path $\underline{\omega}_T^k$ generated in iteration k. Further details of each step are provided below.

Borrowing from the case of two-stage SD, we will make the following assumptions:

B1) The set of first stage decisions is compact.

B2) The relatively complete recourse assumption is satisfied at every stage; (i.e. (32) has a finite optimum for any setting of feasible x_n.

B3) Zero provides a lower bound on all conditional expectations. (This assumption can be easily relaxed as in the two-stage case.)

B4) Assume that for all t, D_t has full row rank. In addition, we reiterate the fixed recourse assumption ($D_n = D_{t(n)}$), as well as the requirement that the stochastic process has compact support.

B5) The scenario tree represents only nodes with conditional probability $p_n \geq 0$.

4.3.1 Simulate a sample path

We assume that the simulation will be consistent with the given stochastic process denoted $\underline{\tilde{\omega}}_T$. In developing the approximations however, we will use the empirical distribution observed for transitions from node n to a child node $n+$. At iteration k, this estimate ($p_{(n+|n)}$) will be denoted by p^k_{n+}. Our estimates of probabilities will reflect the number of times the simulation has visited a given node, given that the parent node was visited.

Steps 1.1 and 1.2 are often called forward pass because they are used to generate a sequence of states based on incumbent ($\hat{u}^k_n$) and candidate (u^k_n) decisions. Given a decision u_n, one is able to find the next "3d" states $\{x_{n+}\}_{n+\in\mathbb{C}_n}$ using the dynamics by traversing the sampled path $\underline{\omega}^k_T$ forward in time, by starting with $n = 0$ and then recursively calculating the "3d" states associated with the candidate sequence using the dynamics: $x^k_{n+} = a_{n+} + A_{n+}x^k_n + B_{n+}u^k_n$, followed by the decisions:

$$u^k_{n+} \in \arg\min\{f^{k-1}_{n+}(u_{n+}, x^k_{n+}) + \frac{\sigma_k}{2}||u_{n+} - \hat{u}^{k-1}_{n+}||^2 : u_{n+} \in U_{n+}(x^k_{n+})\}, \tag{30}$$

where $n+ \in \underline{\omega}^k$ and follows n.

We refer to a decision problem in (30) as a nodal decision simulation because these forward passes generate decisions along the sample path during the forward simulation. Because sample paths may change from iteration to iteration, incumbent state trajectories $\{\hat{x}_n^{k-1}\}$ along a sampled path must be updated to be consistent with the current incumbent decision $\hat{u}_0^{k-1}$. However, due to the relatively complete recourse assumption, the previously calculated incumbent decisions $(\{\hat{u}_n^{k-1}\})$ at a sampled node need not be changed. Thus we solves only one nodal decision simulation for any non-terminal node on the sample path for iteration k.

The MSD algorithm will update the approximate functions $\{f_n^k\}$ from iteration to iteration by using piecewise linear approximations developed during the backward pass (step 3). The parameter σ_k is also updated for computational efficiency; however it is important to maintain its value within a range that does not contain zero, and its upper limit should not be so large as to cause difficulties due to scaling ([30]). Without loss of generality, we can therefore assume a lower bound to be $\underline{\sigma} = 1$.

Step 1.2 plays a role in those iterations in which the sampling process discovers new nodes of the scenario tree. For such nodes, function approximations have not been created in any previous iteration, and as a result it is not possible to perform (30). What we recommend in this case is that a linear program be solved, starting from the first newly revealed node, with data corresponding to the rest of the sample path. The solution of this LP will provide the dual multiplier estimates to be used in the backward pass (step 2 - see next subsection).

4.3.2 Solve nodal dual approximations

These calculations are carried out backward in time, along the path that was generated in the forward pass. Accordingly, the definitions that follow are best carried out in a recursive manner, starting from a terminal node. For nodes along the sample path $\underline{\omega}^k$, the backward pass updates the functions f_n^k using two state trajectories $\{\hat{x}_n^{k-1}\}$, and $\{x_n^k\}$, $n \in \underline{\omega}^k$, and the corresponding decisions $\{\hat{u}_n^{k-1}\}$ and $\{u_n^k\}, n \in \underline{\omega}^k$, respectively. Whenever a particular calculation applies to both incumbent and candidate solutions, we will use the notation x_n to denote either trajectory. One of the main ideas behind the multistage SD setup is that it highlights the use of state variables ($\underline{s}_n = (x_n, \underline{\omega}_n)$) in the approximation process.

Moreover, for a terminal node n, we define $E[h_{n+}] = 0$ and let h_n^k provide a lower bound on h_n. In order to define approximations for non-terminal nodes, we set forth a recursive definition by assuming that for all $n+ \in \mathbb{C}_n$, we have approximations h_{n+}^{k-1}, as well as empirical probabilities $\{p_{n+}^k\}_{n+\in\mathbb{C}_n}$. Then, we define h_n^k as an empirical lower bounding approximation of the nodal recourse function at node n as follows:

$$h_n^k(\underline{s}_n) \leq c_n^T x_n + \text{Min}\{d_n^T u_n + \sum_{n+\in\mathbb{C}_n} p_{n+}^k h_{n+}^k(\underline{s}_{n+}) : D_{t(n)} u_n \leq b_n - C_n x_n\}. \qquad (31)$$

If the quantities h_{n+}^k on the right hand side of (31) are replaced by exact values h_{n+}, then the above statement becomes an equality, and in that case (31) is simply an application of the DP principle of optimality. Of course, if n is a terminal node, the above inequality will always hold as an equation because $h^k_{n+} = h_{n+} = 0$.

For finite k, our simulation based successive approximation scheme has two sources of errors: the probability distribution (using empirical distributions), and the future value functions h^k_{n+}. The latter will be approximated using subgradients of approximations. Now, if we could ensure that the collection of approximations satisfy $\{h^k_{n+}\}_{n+\in\mathbb{C}_n} \to h_{n+}$ in some sense (e.g. epi-convergence), then one could (using the law of large numbers) ensure that $h^k_n \to h_n$ with probability one. However, it is well known that the approximations generated by SD obeys epi-nesting relative to h_n, but not epi-convergence to h_n ([29], [57]). Thus the approximations h^k_n developed below (using subgradients) should be designed to ensure epi-nesting, which will be sufficient for convergence of state trajectories with probability one.

Suppose that for all $n+ \in \mathbb{C}_n$, we have already calculated a subgradient (with respect to x_{n+}) $\beta^k_{n+} \in [\partial h^k_{n+}(\underline{s}_{n+}) - c_{n+}]$ and let α^k_{n+} denote the constant term associated with the corresponding supporting hyperplane. Assuming relatively complete recourse, and substituting for the "3d" state variables, that is, $x_{n+} = a_{n+} + A_{n+}x_n + B_{n+}u_n$, and replacing the primal value function by its dual representation, we obtain an affine lower bounding approximation as follows:

$$h^k_n(\underline{s}_n) \le c_n^T x_n + \sum_{n+\in\mathbb{C}_n} p^k_{n+}\{\alpha^k_{n+} + (c_{n+} + \beta^k_{n+})^T[a_{n+} + A_{n+}x_n]\} +$$

$$\text{Max}\{\pi_n^T[b_n - C_n x_n] : \pi_n \le 0, D^T_{t(n)}\pi_n = d_n + \sum_{n+\in\mathbb{C}_n} p^k_{n+}B^T_{n+}(c_{n+} + \beta^k_{n+})\} \qquad (32)$$

The LPs in the second line of the above equation will be referred to as nodal dual approximations (NDA). In order to instantiate the NDA for node n, denoted $NDA(n)$, note that the right-hand side requires quantities $p^k_{n+}, B_{n+}, c_{n+}, \beta^k_{n+}$ for $n+ \in \mathbb{C}_n$. In order to do the recursion. we assume that p^k_{n+}, α^k_{n+} and β^k_{n+} are estimated and available for all $n+ \in \mathbb{C}_n$. Indeed, while we will maintain these quantities for all nonterminal nodes that have been visited at least once, they will only be updated periodically, depending on their "proximity" to the sampled path. The notion of "proximity" as well as the updating procedure will be described subsequently. Let π^k_n denote solutions obtained by setting $x_n = x^k_n$ (a candidate) for the LPs in (32). Hence using π^k_n, we obtain:

$$h^k_n(\underline{s}_n) \geq (\pi^k_n)^T b_n + (c_n - C^T_n \pi^k_n)^T x_n + \sum_{n+\in\mathbb{C}_n} p^k_{n+}\{\alpha^k_{n+} + (c_{n+} + \beta^k_{n+})^T[a_{n+} + A_{n+}x_n]\}.$$

Letting

$$\beta^k_n = -C^T_n \pi^k_n + \sum_{n+\in\mathbb{C}_n} p^k_{n+} A^T_{n+}(c_{n+} + \beta^k_{n+}) \tag{33}$$

and

$$\alpha^k_n = (\pi^k_n)^T b_n + \sum_{n+\in\mathbb{C}_n} p^k_{n+}\{\alpha^k_{n+} + (c_{n+} + \beta^k_{n+})^T a_{n+}\}, \tag{34}$$

we recursively obtain a subgradient $\beta^k_n \in [\partial h^k_n(\underline{s}^k_n) - c_n]$ (with respect to x_n) and the quantity α^k_n represents the "constant" term of the hyperplane. These quantities will be used for approximations at the parent node $n-$.

4.3.3 Collect information and update approximations

Using the pairs (33) and (34) (and their analogs for incumbent trajectories when necessary), we can now summarize the approximations developed at the candidate state trajectory for the multistage SD algorithm:

$h_n^k(\underline{s}_n) =$

$$\begin{cases} \text{Max}\{\alpha_n^k + (c_n + \beta_n^k)^T x_n, \hat{\alpha}_n^k + (c_n + \hat{\beta}_n^k)^T x_n, h_n^{k-1}(\underline{s}_n)\} \text{ if } t(n) = T; \\ \text{Max}\{\alpha_n^k + (c_n + \beta_n^k)^T x_n, \hat{\alpha}_n^k + (c_n + \hat{\beta}_n^k)^T x_n, \frac{\kappa_n^k - 1}{\kappa_n^k} h_n^{k-1}(\underline{s}_n)\} \text{ if } 1 \leq t(n) < T \end{cases}$$

where κ_n^k denotes the number of visits to node n when the k^{th} approximation is created. Note that if node n belongs to the sample path for iteration k, then $\kappa_n^{k-1} = \kappa_n^k - 1$. Hence if sample mean approximations were previously constructed using κ_n^{k-1} visits to node n, the multiplier $(\frac{\kappa_n^k - 1}{\kappa_n^k})$ reflects the increase in sample size by using the lower bound (assumed to be zero) as a new observation for all newly generated affine approximations for a non-terminal node n. Of course, in case of terminal nodes, there is no uncertainty in the future, and as a result the approximations require no further estimation (because sampling produces the same scenario).

In order to keep the notation manageable, we introduce an index set I_n^k that will index all affine functions defining h_n^k as follows: $h_n^k(\underline{s}_n) = \text{Max}\{\alpha_{in}^k + (c_n + \beta_{in}^k)^T x_n : i \in I_n^k\}$. Note that this is precisely the form of approximation used in L-shaped method/Benders' decomposition, and its variants. Finally, we put $h_n^k \leftarrow h_n^{k-1}$ for all those nodes that do not belong to the path sampled in iteration k. Thus the approximations associated with all other nodes remain unaltered, although their impact on the parent node changes via the estimated probability update p_n^k. In any event, setting $n \leftarrow n-$, we obtain an updated approximation of the following form,

$$f_n^k(u_n, x_n) = c_n^T x_n + d_n^T u_n + \sum_{n+\in\mathbb{C}_n} p_{n+}^k h_{n+}^k(\underline{s}_{n+}) : x_{n+} = a_{n+} + A_{n+}x_n + B_{n+}u_n (a.s.) \tag{35}$$

where the almost sure requirement is to be interpreted with respect to the empirical probability distribution observed at node n. Note that since $\underline{s}_{n+} = (x_{n+}, \underline{\omega}_{n+})$, the constraints in (35) provide terminal conditions whose cost-to-go is $\sum_{n+\in\mathbb{C}_n} p_{n+}^k h_{n+}^k(\underline{s}_{n+})$, the expected recourse (value) function. Clearly, this approximation scheme which is based on the arguments of SD, has the same form as ADP, and hence forms a bridge between SP and ADP. Moreover, using (35) further clarifies the role of the principle of optimality in both SP and ADP.

It is interesting to recognize that the functions (f_n^k) which are optimized in the nodal decision simulation (i.e. (35)) can be accommodated in several ways: a) as in the original L-shaped method using aggregated cuts from all observations ([69] and [28]), or b) via a multi-cut version as in ([8] and [60]). Depending on the degree of uncertainty and nonlinearity of the value function, one can use either of these extremes (or some combination of the two) in solving the approximations. Furthermore, we remind the reader that the forward simulation pass is performed by fixing x_n^k, and optimizing $f_n^{k-1}(u_n, x_n^k)$, whereas, the backward pass uses subgradients at the candidate and incumbent state trajectories (i.e. $\beta_n^k \in \partial[h_n^k(x_n^k) - c_n]$ and $\hat{\beta}_n^k \in \partial[h_n^k(\hat{x}_n^k) - c_n]$) to generate new approximations. Note that unlike nested Benders' decomposition and related stochastic dual dynamic programming, the backward pass in MSD uses specific future subgradients of NDA(n) (32), rather than optimizing over the entire collection of available sub-gradients. This has the advantage of maintaining a pure LP structure, rather than a piecewise LP structure as done in other previously mentioned multistage SP algorithms. An alternative, somewhat weaker version of these updates, appears in a related paper ([62]).

For situations in which the relatively complete recourse assumption is not justifiable, the algorithm can be modified to accommodate so-called feasibility cuts, which would have the form $[\delta_{n+}^k][b_{n+} - C_{n+}x_{n+}] \leq 0$, where δ_{n+}^k denote some dual extreme direction along which the objective in (32) recedes to $+\infty$. For the purposes of this dissertation however, we continue with the relatively complete recourse assumption as stated earlier.

4.3.4 Update incumbent solutions

As with the two-stage regularized SD algorithm (See chapter 2), the choice of the incumbent is based on objective value estimates at the root node; that is, we estimate whether there is a reduction in the value $f_0(u_0; x_0)$; that is, $\hat{u}_0^k \leftarrow u_0^k$ if

$$f_0^k(u_0^k, x_0) - f_0^k(\hat{u}_0^{k-1}, x_0) \leq q[f_0^{k-1}(u_0^k, x_0) - f_0^{k-1}(\hat{u}_0^{k-1}, x_0)] \tag{36}$$

If the above inequality is satisfied, then all incumbent solutions on the sample path are updated to assume the values of the candidate decisions, although nodes that are not on the sample path retain previous values of incumbent decisions.

4.4 Asymptotic convergence of multistage SD

While the algorithmic approach described above is entirely recursive, the analysis that follows will be more akin to the analysis of scenario-based SP algorithms, thus making SD an appropriate bridge between the SP and DP approaches. Because SP allows very general dependence structures, the multistage SD algorithm also inherits this generality. We emphasize that while the computational constructs for the algorithmic process have already been introduced in the previous section, some of the notations introduced below are simply intended for the purposes of analysis.

The collection of all scenarios generated will be denoted $\mathcal{S}_0$, and for nodes $n+ \in \mathbb{C}_0$, we define $\mathcal{S}_{0+} \subseteq \mathcal{S}_0$ as the subset of scenarios that visit node "0+". Recursively, let $\mathcal{S}_{n+}$ denote the subset of scenarios of $\mathcal{S}_n$ passing through node $n+$. This process is the same as that used to form scenario trees from filtrations in section 4.2, and consequently, the frequency estimates lead to the conditional probability estimates, asymptotically. For any $j \in (\cup_{n+\in\mathbb{C}_n}\mathcal{S}_{n+})$, let $\mathcal{X}^j_{n+}$ and $\mathcal{U}^j_{n+}$ denote sequences of state and decision vectors associated with scenario j, starting with node $n+$. Accordingly, assume that the vectors $(x_n, \mathcal{X}^j_{n+})$ and $(u_n,\mathcal{U}^j_{n+})$ satisfy feasibility, non-anticipativity, as well as dynamics as stated in (29), although it is the sampled subproblem starting at node n. Then, define a nodal sample mean approximation $(NSMA(n))$ as

$$\mathcal{H}_n(\underline{s}_n|\mathcal{S}_n) = c_n^T x_n + \text{Min}\{d_n^T u_n + \sum_{j\in(\cup_{n+\in\mathbb{C}_n}\mathcal{S}_{n+})} (\frac{\kappa_j}{\kappa_n})[\mathcal{C}_j(\mathcal{X}^j_{n+}) + \mathcal{D}_j(\mathcal{U}^j_{n+})]\}$$

$$= c_n^T x_n + \mathrm{Min}\{d_n^T u_n + \sum_{n+\in\mathbb{C}_n} (\frac{\sum_{j\in\mathcal{S}_{n+}} \kappa_j}{\kappa_n})[\mathcal{H}_{n+}(\underline{s}_{n+}|\mathcal{S}_{n+})]\}, \tag{37}$$

where $\kappa_n = |\mathcal{S}_n|$. To avoid further complicating the notation, we are not distinguishing between counters for scenarios (κ_j) and counters for visits to node (κ_n), and because the context (scenario/node) will be clear there should be no confusion. The functions $\mathcal{C}_j(\mathcal{X}_{n+}^j)$ and $\mathcal{D}_j(\mathcal{U}_{n+}^j)$ represent the cost for scenario j, starting with state $\underline{s}_n$. Since we are interested in asymptotic analysis, we assume that κ_n, the number of sampled paths is large enough for all $n \in \mathcal{N}$. Next assume that both non-anticipativity and dynamics are stated in polyhedral form (see e.g. [42] and [33]), and the frequency estimates ($\frac{\kappa_j}{\kappa_n}$) appear only in the objective function, but not in the constraints. The notation $\mathcal{H}_n(x_n|\mathcal{S}_n)$ is intended to convey the sense that the sample is given, and accordingly, its size κ_n as well as the counts κ_j are also given. In the following we will indicate the dependence of $NSMA(n)$ on the iteration counter (k) as well as the frequency estimates ($\frac{\kappa_j}{\kappa_n}$) by using the sample $\mathcal{S}_n^k$ in $\mathcal{H}_n(x_n|\mathcal{S}_n^k)$.

In the design of SD presented in the previous section, we do not solve any of the nodal sample mean approximations; instead we sequentially develop approximations of these nodal sample mean approximations, and they will be shown to provide asymptotically accurate estimates of the objective function. To see how the approximations in SD compare with (37), let us relate the latter to the nodal calculations of SD. We first observe that $p_{n+}^k = (\sum_{j\in\mathcal{S}_{n+}^k} \kappa_j^k)/\kappa_n^k$, and for large enough k such that $p_{n+}^k > 0$ for all $n+ \in \mathcal{C}_n$, let us examine how the approximations f_n^k in (35) compare with $NSMA(n)$ in (37). Both (35) and (37) require three properties: feasibility of u_n

(in the sense that $u_n \in U(x_n)$), as well as feasibility, non-anticipativity and dynamics of all future states and decisions. However, there is an important difference: (37) requires optimality with respect to the exact sample mean objective for nodes $n+$ and beyond, whereas (35) requires optimality with respect to future sample mean approximations h^k_{n+}. Through the nodal decision simulations in the forward pass, SD chooses u_n, and based on the choice of nodes dictated via sampling, the backward pass uses one child node in $\mathbb{C}_n$ whose piecewise linear approximation is updated. Other approximations remain unchanged during any iteration. Let

$$u_n^{k+1}(x_n) \in \text{argmin}\{d_n^T u_n + \sum_{n+\in\mathbb{C}_n} [p^k_{n+} h^k_{n+}(x_{n+})] + \frac{\sigma_k}{2}||u_n - \hat{u}^k_n||^2\}, \tag{38}$$

where $u_n \in U_n(x_n)$, and $x_{n+} = a_{n+} + A_{n+}x_n + B_{n+}u_n$.

Using $x^{k+1}_{n+}(x_n) = a_{n+} + A_{n+}x_n + B_{n+}u^{k+1}_n(x_n)$, define

$$F^k_n(\underline{s}_n) = c_n^T x_n + d_n^T u^{k+1}_n(x_n) + \sum_{n+\in\mathbb{C}_n} [p^k_{n+} h^k_{n+}(x^{k+1}_{n+}(x_n))], \tag{39}$$

$$\phi^k_n(\underline{s}_n) = F^k_n(\underline{s}_n) + \frac{\sigma_k}{2}||u^{k+1}_n(x_n) - \hat{u}^k_n||^2. \tag{40}$$

Note that as with the sample mean approximation $\mathcal{H}_n(x_n|\mathcal{S}^k_n)$, the approximations defined in (39) and (40) are also dependent on the sample $\mathcal{S}^k_n$. Hence, strictly speaking it would have been appropriate to indicate the sample dependence for these functions too. However, recognizing that all three functions depend on the same sample $\mathcal{S}^k_n$, we have chosen to simplify the notation for (39) and (40) by using the superscript k.

In order to study for the asymptotic behavior of the MSD algorithm, we investigate how the functions $\phi_n^k(\underline{s}_n), F_n^k(\underline{s}_n), \mathcal{H}_n(\underline{s}_n|\mathcal{S}_n^k)$ and $h_n(\underline{s}_n)$ compare at limiting states (if such exist). The proof is based on a dynamic version of regularization ([60]), subdifferential compatibility ([29], [57]), and epi-consistency ([38]). These concepts were combined previously for asymptotic results of two-stage SD in chapter 2. The following concept will be useful in the analysis.

Definition 1 *A sequence of functions $\{\phi^k\}$ and a function ϕ are said to be subdifferentially compatible with respect to a sequence of points $\{x^k\}$ if for any subsequence such that $\{x^k\}_{k\in K} \to \bar{x}$, one has limsup$_{k\in K}\partial\phi^k(x^k) \subseteq \partial\phi(\bar{x})$. A shorthand representation of this notion is $\{\phi^k\} \overset{\partial}{\to} \ \phi(wrt\{x^k\})$.*

Corollary 8 of [29] states that under subdifferential compatibility, the decisions generated via minimization of the approximation creates a sequence whose accumulation points belong to the set of optimal solutions to the original problem of optimizing ϕ. The terrain that we explore in proving asymptotic optimality can be summarized by the following relations,

$$\phi_n^k \overset{\partial}{\to} F_n^k \overset{\partial}{\to} \mathcal{H}_n(.|\mathcal{S}_n^k) \xrightarrow{wp1} h_n(\text{wrt}\{\hat{\underline{s}}_n^k\}) \tag{41}$$

which requires us to show that for the sequence of states $\{\hat{\underline{s}}_n^k\}$, the sequence of function values and subgradients on the left hand side of an arrow satisfy subdifferential compatibility for the sequence on the right hand side of the arrow. Of the three arrows above, the right-most one is classical (see e.g. [58] and [51]): it focuses attention on the so-called sample mean (average) approximation (for multistage problems). This is what one relies on when an algorithm samples the original scenario tree to create a more manageable sub-tree, as in previously studied sampling algorithms for MSLP. The middle arrow is what distinguishes SD from sample mean approximation, whereby piecewise linear approximations learn the sample mean approximation, asymptotically. Finally, the left-most arrow is used to ensure certain regularizing properties such as compactness and convergence of state and decision trajectories. We start with the regularizing properties (e.g. compactness).

Lemma 7 *Lemma for subgradient compactness. Suppose the multistage SD algorithm runs for infinitely many iterations. For any node n, let I_n^k denote the index set of affine functions in the definition of h_n^k. Under assumptions B1, B2 and B3, the collection of vectors $\{V_n^k = (\alpha_{in}^k, \beta_{in}^k), i \in I_n^k\}$ belongs to a compact set for all k and n.*

Proof: It is sufficient to show that the subgradients generated in each iteration, denoted β_{in}^k, belong to a compact set because all subsequent updates can be interpreted as convex combinations of β_{in}^k and 0. It is therefore convenient to drop the index i for the remainder of the proof. For any terminal node n, the complete recourse assumption ensures that an optimal solution of the nodal dual approximation is obtained at a basic feasible solution, and using $p_{n+}^k = 0$, we conclude that for all n with $t(n) = T, \beta_n^k = -C_n^T \pi_n^k$ belongs to a compact space (because π_n are vertices of a fixed dual polyhedron). Hence the result is true for all terminal nodes. Next assume that for some fixed $\tau \leq T$ this property is true for all nodes n such that $\tau = t(n) \leq T$. We show that this implies compactness of $\{\beta^k_{n-}\}$ also holds for all parent nodes $n-$, where $t(n-) = \tau - 1$. From (33), it is clear that for nodes $n-$ such that $t(n-) = \tau - 1$, the future reflected by $\sum_{n \in \mathbb{C}_{n-}} p^k_n A^T_n (c_n + \beta^k_n)$ is bounded due to the induction hypothesis. Hence β^k_{n-} is bounded if and only if π^k_{n-} is bounded. Since the complete recourse assumption allows us to restrict our attention to optimal nodal dual solutions that are basic feasible solutions of (32), the boundedness of $\{d_{n-} + \sum_{n \in \mathbb{C}_{n-}} p^k_n B^T_n (c_n + \beta^k_n)\}$ and the fixed matrix $D_{\tau-1}$ imply that π^k_{n-} is bounded and the result follows. ■

Lemma 8 *Lemma for solution stability and compactness. Suppose assumptions B1-B5 hold, and $\sigma \leq 1$. Let $\{\hat{u}^k_0\} \subseteq U_0$ denote any infinite sequence of first stage incumbent decisions. Then for each $n \in \mathcal{N}$, there exists a subsequence of iterations indexed by K_n such that the incumbent states $\{\hat{x}^k_n\}_{K_n}$ as well as incumbent decisions $\{\hat{u}^k_n\}_{K_n}$ have accumulation points.*

Proof: First consider any node n with $t(n) = 1$. Since $U_0(x_0)$ is compact set by assumption, there exists a subsequence indexed by K_0 such that for $k \in K_0$, $\{\hat{u}^k_0\}_{K_0} \rightarrow \bar{u}_0$. Then the linearity of the state dynamics implies that for any n such that $t(n) = 1$, $\{\hat{x}^k_n\}_{K_n} \rightarrow \bar{x}_n$ and $\bar{x}_n = a_n + A_n x_0 + B_n \bar{u}_0$, where K_n is a subsequence of K_0, starting with the earliest iteration in K_0 after the first visit to node n. Next we consider the incumbent solutions $\hat{u}^k_n$, where once again $k \in K_n$. Since incumbent solutions are a subsequence of candidate solutions, we study the stability of candidate solutions. Note that the variables in the nodal decision simulation (NDS) can be written in the form $u^k_n = \hat{u}^{k-1}_n + \Delta u^k_n$, where $\Delta u^k_n \in \operatorname{argmin}\{f^{k-1}_n(\hat{u}^{k-1}_n + \Delta u_n, \hat{x}^k_n) + \frac{\sigma_k}{2}||\Delta u_n||^2 :$

$\hat{u}_n^{k-1} + \Delta u_n \in U_n(\hat{x}_n^k)\}$. As before, this optimization problem is denoted NDS(n), and we denote its solution set by argmin(NDS(n)). Since $\sigma_k \geq \underline{\sigma} \geq 1$ (by construction), this is a positive definite piecewise linear quadratic program with linear constraints, which has unique primal and dual optimal solutions (see Chapter 4 in [31]). Let θ_n^k denote the vector of dual multipliers corresponding to the subgradient inequalities in NDS(n). Dualizing all subgradient inequalities by using the optimal dual multipliers θ_n^k, one obtains a quadratic programming subproblem whose solution set is identical to that of NDS(n). Since this quadratic programming subproblem is stable to perturbations of the right-hand side and the linear part of the objective (see [26] section 5), it follows that the solution set mapping (for NDS(n))$\psi_n : \hat{x}_n \rightarrow$ argmin(NDS(n)) is also continuous. Consequently it follows that $\{\hat{x}_n^k\}_{K_n} \rightarrow \bar{x}_n$ implies that $\{\hat{u}_n^k\}_{K_n} \rightarrow \bar{u}_n$ for all nodes n such that $t(n) = 1$. Using the above argument recursively, one concludes the validity of the result for all nodes $n \in \mathcal{N}$. ■

Lemma 9 *Corollary for uniform convergence. Suppose assumptions B1-B5 hold.*

a) The sequence of functions $\{h_n^k\}_k$ is uniformly equicontinuous, and uniformly convergent for all n.

b) The sequence of functions $\{F_n^k\}_k$ is uniformly equicontinuous, and uniformly convergent for all n.

c) The sequence of functions $\{\mathcal{H}_n(.|\mathcal{S}_n^k)\}_k$ converges uniformly with probability one to the expectation h_n.

Proof: a) From Lemma 8 one is able to restrict attention to a compact subset of decisions for each n. As observed in concluding line of the proof of Lemma 7, the boundedness of $\{d_{n-} + \sum_{n \in \mathbb{C}_{n-}} p_n^k B_n^T (c_n + \beta_n^k)\}$ together with the fixed recourse matrix $D_{t(n-)}$ implies that $\{h_n^k\}$ must have a uniform Lipschitz constant, leading to the conclusion stated in the corollary.

b) The complete recourse assumption ensures that the functions F_n^k have finite value, and recall from quadratic programming optimality conditions that the solution mapping u_n^{k+1} is piecewise linear. Hence the sequence of functions $F_n^k(\underline{s}_n) = c_n^T x_n + d_n^T u_n^{k+1}(x_n) + \sum_{n+ \in \mathbb{C}_n} [p_{n+}^k h_{n+}^k (x_{n+}^{k+1}(x_n))]$ is also bounded over a compact set of decisions, and moreover, a uniform Lipschitz constant also exists, thus leading to the same conclusion as in part a).

c) This is the classical result of sample mean approximations converging uniformly to the mean with probability one, so long as the approximations are constructed by i.i.d sampling with an ever increasing sample size, the existence of a uniform Lipschitz constant (Lemma 7) and compactness of the decision/control/parameter space (Lemma 7) can be established ([58] and [51]). ■

Theorem 8 *Theorem for convergence of decisions. Suppose assumptions B1-B5 hold, and $\underline{\sigma} \geq 1$. Then there exists $\bar{u}_0 \in U_0$, such that $\{\hat{u}_0\} \to \bar{u}_0$, and more generally, there exist $\bar{u}_n(\bar{x}_n) \in U_n(\bar{x}_n)$, where the dynamics in (29) are satisfied, and $\{\hat{u}_n^k(\hat{x}_n^k)\} \to \bar{u}_n(\bar{x}_n)$ for all $n \in \mathcal{N}$.*

Proof: If the incumbent at the root node changes only finitely many times, then the result is obviously true. So, consider the case in which the incumbent changes infinitely many times. First consider node 0. Since x_0 is given, it will be convenient to refer to the objective function at node 0, simply by $f_0^k(u_0)$, rather than the more cumbersome $f_0^k(u_0, x_0)$. The optimality conditions for the regularized approximation at the root node (see equation (2.6) on page 115 of [31]) and our choice $\underline{\sigma} \geq 1$ implies that:

$$f_0^k(u_0^{k+1}) - f_0^k(\hat{u}_0^k) \leq -||u_0^{k+1} - \hat{u}_0^k||^2 \leq 0. \tag{42}$$

Let K_0 denote iterations at which the incumbent changes, and for m successive indices in K_0, we have $\hat{u}_0^{k_{l+1}} = u_0^{k_l+1}$, and the left hand side of this inequality can be written as

$$\Delta_m = \frac{1}{m}[f_0^{k_m}(\hat{u}_0^{k_{m+1}}) - f_0^{k_0}(\hat{u}_0^{k_0})] + \frac{1}{m}\sum_{l=0}^{m-1}[f_0^{k_l}(\hat{u}_0^{k_{l+1}}) - f_0^{k_{l+1}}(\hat{u}_0^{k_{l+1}})]. \tag{43}$$

From lemma 9 the summation term in (43) converges to 0, whereas, the first term (in square brackets) has a finite upper bound (due to upper and lower bounds on the approximations f_0^k). Hence $\lim_{m\to\infty}\Delta_m \to 0$. It follows that $\lim_{m\to\infty}\frac{1}{m}\sum_l ||\hat{u}_0^{k_{l+1}} - \hat{u}_0^{k_l}||^2 \to 0$. Hence the entire sequence of incumbents generated at node 0 must converge, and we denote such a point by $\bar{u}_0 \in U_0$.

Let us now proceed to other nodes on the tree. If a node is visited finitely many times, the result is clearly true. Next consider nodes that are visited infinitely many times. Since the incumbent decisions converge for $t = 0$, the future incumbent states $\{\hat{x}_n^k\}$ also converge (as in Lemma 8). Hence there exists a sequence of incumbent states that satisfy the equivalent of (42); that is, $F_n^k(\underline{\hat{s}}_n^{k+1}) - F_n^k(\underline{\hat{s}}_n^k) \leq$

$-||u_n^{k+1}(x_n^{k+1}) - \hat{u}_n^k(\hat{x}_n^k)||^2 \leq 0$, where $k \in K_n$ (as in the proof of Lemma 8 and the previous paragraph). Once again, using the same arguments as in the previous paragraph, we conclude that $\{\hat{u}_n^k(\hat{x}_n^k)\} \to \bar{u}_n(\bar{x}_n)$ for all $n \in \mathcal{N}$. ■

Finally, we present the main asymptotic optimality result of MSD.

Theorem 9 *Theorem for asymptotic consistency and optimality. Suppose assumptions B1-B5 hold, and $\underline{\sigma} \geq 1$. For all n such that $1 \leq t(n) \leq T$, we have*

$$\phi_n^k \overset{\partial}{\to} F_n^k \overset{\partial}{\to} \mathcal{H}_n(.|\mathcal{S}_n^k)(wrt\{\hat{\underline{s}}_n^k\}) \tag{44}$$

(which requires us to show that for the sequence of states $\{\hat{\underline{s}}_n^k\}$, the sequence of function values and subgradients on the left hand side of an arrow satisfy subdifferential compatibility for the sequence on the right hand side of the arrow). Moreover, subdifferential compatibility for node 0 implies that $\bar{u}_0 \in U_0$ is optimal (wp1).

Proof: . First note that $\{\hat{u}_n^k(\hat{x}_n^k)\} \to \bar{u}_n(\bar{x}_n)$ implies that the sequence of state $\{\hat{\underline{s}}_n^k\}$ converges for all n. Now, let us investigate the subdifferential compatibility as stated. The definition of subgradients and the updates for h_n^k together with Theorem 8 imply subdifferential compatibility of ϕ_n^k and F_n^k for all n. In order to prove the validity of the next arrow, consider n such that $t(n) = T$ (terminal nodes). Since $\{\hat{\underline{s}}_n^k\}$ and the functions h_n^k and $\mathcal{H}_n(.|\mathcal{S}_n^k)$ are equal for such (terminal) n and all states $\hat{\underline{s}}_n^k$, they are also subdifferentially compatible. Next we consider any parent node n, such that $t(n) = T - 1$. Let $F_n = \lim_{k\to\infty} F_n^k$. Because the incumbent sequence is created from the candidate sequence (i.e. $\hat{u}_n^k = u_n^k$ for the chosen indices k) and $\{\underline{\hat{s}}_{n+}^k\} \to \underline{\bar{s}}_{n+}$, (35), (38), and (39) imply

$$\lim_{k\to\infty} h_n^k(\underline{\hat{s}}_n^k) = F_n(\underline{\bar{s}}_n) \tag{45}$$

Moreover, the induction hypothesis implies that $h_{n+}^k(.)$ and $\mathcal{H}_{n+}(.|\mathcal{S}_{n+}^k)$ are subdifferentially compatible wrt $\{\underline{\hat{s}}_{n+}^k\} \to \underline{\bar{s}}_{n+}$ for $n+ \in \mathbb{C}_n$. From the development immediately preceding (38), we recall that $p_{n+}^k = (\sum_{j\in\mathcal{S}_{n+}^k} \kappa_j^k)/\kappa_n^k$. Consequently, $\{\underline{\hat{s}}_n^k\} \to \underline{\bar{s}}_n$, and $\{\underline{\hat{s}}_{n+}^k\} \to \underline{\bar{s}}_{n+}$ for all $n+ \in \mathbb{C}_n$, together with (37), (38) and (39) imply that $F_n(\underline{\bar{s}}_n) = \lim_{k\to\infty}\mathcal{H}_n(\underline{s}_n^k|\mathcal{S}_n^k)$. Combining this asymptotic behavior with that in (45), and the lower bounding behavior (32) obeyed by h_n^k, we conclude that F_n^k and h_n^k are both subdifferentially compatible with $\mathcal{H}_n(\underline{s}_n^k|\mathcal{S}_n^k)$ with respect to $\{\underline{\hat{s}}_n^k\} \to \underline{\bar{s}}_n$, where n satisfies $t(n) = T - 1$. Using this reasoning backwards recursively leads us to the conclusion that F_{0+}^k and $\mathcal{H}_{0+}(\underline{\hat{s}}_{0+}^k|\mathcal{S}_{0+}^k)$ are also subdifferentially compatible. Finally, applying lemma 9(c) and Corollary 8 of [29] to approximations $\{f_0^k\}_k$ at node 0 we conclude that $\bar{u}_0 \in U_0$ is optimal (wp1). ■

4.5 Remarks on the asymptotic analysis

A review of the convergence analysis reveals that our approach can be easily applied to models for which the nodal problems are convex programs whose objective and constraints are separable by states and decisions, and the stochastic dynamics are linear. For the case of stochastic nonlinear dynamics however, one may not achieve global optimality.

With a few exceptions (e.g. [52]), approximations in ADP do not rely on convexity. The asymptotic analysis presented above highlights some of the main differences between SP and ADP: the former uses convex approximations which satisfy certain properties (e.g. subdifferential compatibility) to ensure asymptotic convergence. This is achieved through dual problems, as in the nodal dual approximations used in MSD.

4.6 A production and inventory application

In the applications of stochastic programming, a number of studies have been focused on including uncertainties in production and inventory systems ([71] and [63]). Moreover, different methods have been proposed to solve specific uncertainties in production and inventory systems. The dynamic lot-sizing method using stochastic dynamic programming was suggested by Burstein et al.([10]) by analyzing the multistage production and inventory systems with stochastic delivery time. On the other hand, Escudero and Kamesam ([21]) considered stochastic demands for multiple products and periods in a multistage production system. Discrete multistage stochastic programming model and scenario tree are applied to describe the stochastic demands.

Furthermore, [46] studied a production planning and control problem when the demand is stochastic. On the algorithmic side, they chose a robust optimization method to solve the stochastic nonlinear programming. Another class of production and inventory problem is to consider stochastic capacity and demands. [20] proposed a stochastic programming model to study the capacity planning model in automobile manufacture and [37] formulated a multistage stochastic programming model in a semi-conductor manufacturer.

In the presence of uncertainty, optimization problems in production planning system are very complicated. Therefore, new methods and algorithms to effectively handle such complex systems are required to be developed. In our application, we consider a multi-period production system with stochastic demands and capacities. In most cases of these type of systems, one aims at obtaining the optimal inventory policy to minimize the costs or maximize the profits, which leads to multistage stochastic programming models. In most studies of production planning systems, stochastic programming was frequently applied. Especially, when the system become very large and complicated with multi-period and multiple uncertainties, simulations and heuristic approaches are usually utilized. Moreover, due to the computational difficulties in multistage stochastic models, the attempts to incorporate the production capacity constraints in multilevel systems are moderate ([72]). However, we proposed a simulation optimization paradigm/algorithm to handle large scale multistage stochastic linear programs, which makes the computation possibilities as well as maintaining the asymptotic optimality properties.

4.7 Computational results

We present our preliminary computations within the context of a production and inventory instance available from website (http://myweb .dal.ca/gassmann/). First, we run a small instance, which has 5 stages and every node in the scenario tree has three children nodes except the nodes at last stage.

Therefore, there are totally 81 scenarios in this instance. Figure 10 reports the computational results of the first stage solution as iterations proceed. One can observe that after 32 iterations, the first stage solution converges to (8,0). This solution is verified to be optimal by solving the deterministic equivalent linear program.

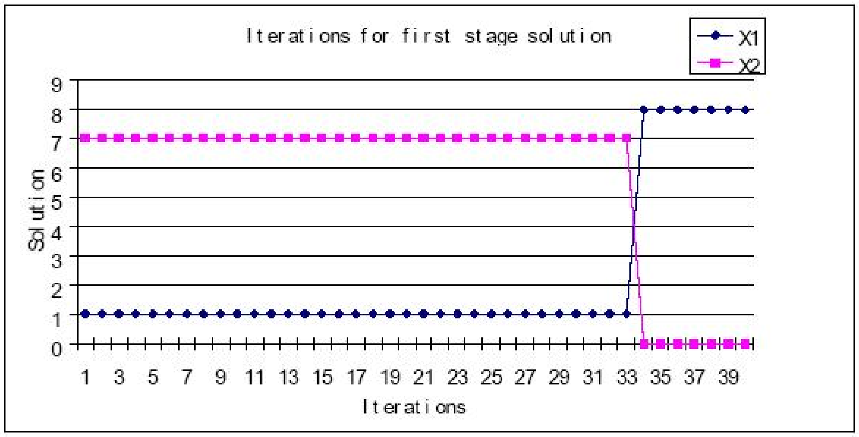

Figure 10: Sequence of first-stage solutions for a small production-inventory instance (81 scenarios)

We also replicated the solution of this instance by using 20 different seeds to run this algorithm. A summary of the outcomes of these runs is provided in Table 2.

Table 2: Results from replications of the multistage SD algorithm.

First-stage Solution	Sample Average	Std. Deviation	CI Half Width ($\alpha = 0.05$)
x_1	7.978	0.141	0.062
x_2	0.068	0.032	0.019

In order to study whether the method would scale to problems with more scenarios, we run a similar production-inventory instance, but for this one we allow 6 stages and every node has 32 children nodes. This results in an instance with 32^5 scenarios. Furthermore, we fix the number of iterations as 500. Figure 11 reports the computational results of this instance. It is not difficult to observe that the solutions stabilize after approximately 50 iterations. Since it is not possible to solve the deterministic equivalent problem in this instance, we use the scenario tree generated by multistage SD method and solve the corresponding deterministic linear program. Once again, we find that solutions by solving deterministic problem are identical to that obtained by running the multistage SD algorithm.

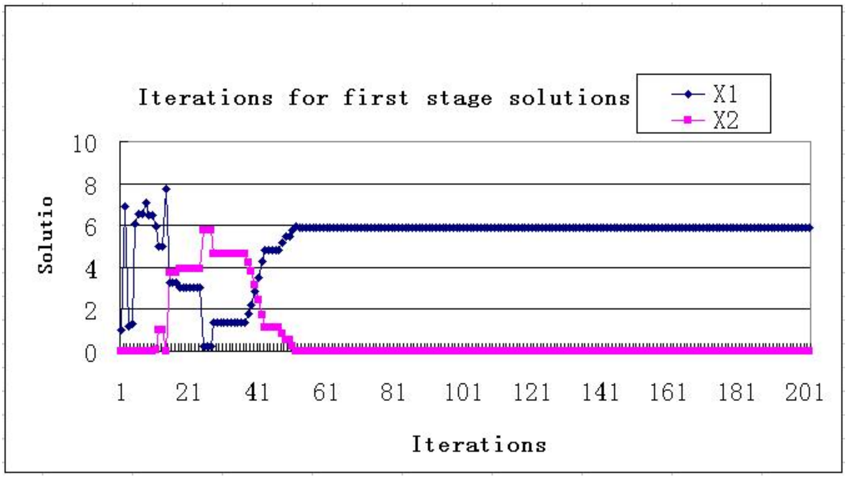

Figure 11: Sequence of first-stage solutions for a large production-inventory instance (32^5 scenarios)

CHAPTER 5: CONCLUSIONS

The enhancements that we have developed lie in both convergence theory and algorithmic considerations for two-stage SD. In the area of convergence theory, we prove the subsequence of the candidate solution has a unique limit. On the algorithmic side, we show that without loss of asymptotic properties, it is possible to reduce the number of outcomes used for the argmax process. This modification helps two-stage SD algorithm overcome the need to use every previously generated outcome in the process.

As for the multistage stochastic decomposition paradigm, our main goal is to present a unified framework for both two-stage as well as multistage stochastic decomposition algorithms. In a sense, MSD is similar to Stochastic Dual Dynamic Programming ([47], [35] and [16]), although the differences are significant: a) MSD uses sample means for recursive estimates of subgradients, where as SDDP uses the probability given via the entire scenario tree, b) MSD solves regularized (quadratic) approximations the forward simulation, which leads to unique nodal decisions, however, it is not clear how one can obtain a stable sequence of decisions by incorporating the regularized term in SDDP, c) generating a cut in MSD requires one LP per non-terminal node of a sample path generated during forward simulation, on the other hand, SDDP needs to update the entire sampled subtree in the forward pass, d) MSD allows subgradient generation to use special structure (e.g. network structure), but, in SDDP, forward and backward passes solve LPs with similar structures, both of which include cuts that may destroy specialized matrices (e.g. network flows), e) MSD overcomes the stagewise independence assumption of the stochastic process in obtaining the asymptotic convergence (with probability one), while the stagewise independence assumption seems critical for the convergence of SDDP ([40] and [66]), and f) unlike most SP algorithms which require that the probability distribution be specified a priori, the MSD approach allows scenarios to be generated via simulations, and so long as the outputs of the simulated process are discrete (e.g. finite state stochastic models), one can directly use the sample paths within the SD framework. In our opinion, these differences, especially (c-f) are particularly important for large scale applications. Items e) and f) are quite far reaching because they allow for the possibility of including more general simulators than ordinary Monte Carlo simulation. In contrast to Simulation Optimization which injects optimization into a simulation model, our new paradigm allows us to inject simulators into an optimization model, leading to a new approach for stochastic programming (Optimization Simulation in [62]).

As to the future research, to design the stopping rules for the MSD algorithm is one of our interests. We believe one of the stopping rules for MSD will be similar to the stopping rule framework in two-stage SD ([31]). Moreover, we can extend our MSD algorithm to handle random variables with continuous distributions in MSLP, which is one of the most challenging problems in stochastic programming. We have also explored common ground between SP and DP, illustrating how SP treats its state variables in a manner that tends to reduce the curse of dimensionality by distinguishing endogenous state variables, with exogenous state variables $\underline{\omega}$. As a result of this framework, we have opened the door of SP methods to dynamic systems optimization, including approximate DP, Differential DP, and Model Predictive Control. While these variants rely on differentiability, SD and more generally SP, handle non-differentiable objective functions.

REFERENCES

[1] G. Bayraksan and D. P. Morton. Assessing solution quality in stochastic programs. *Mathematical Programming*, 108:495–514, 2006.

[2] G. Bayraksan and D. P. Morton. A sequential sampling procedure for stochastic programming. *Operations Research*, 59(3):898–913, 2011.

[3] E.M.L Beale. On minimizing a convex function subject to linear inequalities. *J. of the Royal Statistical Society, Series B*, 17:173–184, 1955.

[4] J. F. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[5] J. O. Berger. *Statistical decision theory and Bayesian Analysis (2nd ed.)*. Springer-Verlag, New York, 1985.

[6] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming,*. Athena Scientific,, Belmont, MA., 1996.

[7] J. R. Birge. Decomposition and partitioning methods for multistage stochastic linear programs. *Operations Research*, 33:989–1007, 1985.

[8] J. R. Birge and F. Louveaux. *Introduction to stochastic programming*. Springer-Verlag, New York, 1997.

[9] J.R. Birge. Stochastic programming computation and applications. *INFORMS J. on Computing*, 9:111–133, 1997.

[10] M.C. Burstein, C.H. Nevision, and R.C. Carlson. Dynamic lot-sizing when demand timing is uncertain. *Operations Research*, 32:362–379, 1984.

[11] M. Casey and S. Sen. The scenario generation algorithm for multi-stage stochastic linear programming. *Mathematics of Operations Research*, 30:615–631, 2005.

[12] A. Charnes and W.W. Cooper. Chance-constrained programming. *Management Science*, 5:73–79, 1959.

[13] Z.L. Chen and W.B. Powell. Convergent cutting plane and partial-sampling algorithm for multistage stochastic linear programs with recourse. *Journal of Optimization Theory and Applications*, 102:497–524, 1999.

[14] G. B. Dantzig and G. Infanger. Large-scale stochastic linear programs: importance sampling and benders' decomposition, in: C. brezinski and u. kulisch (eds.). In *Computational and Applied Mathematics I–Algorithms and Theory*, pages 111–120. Proceedings of the 13th IMACS World Congress, Dublin, Ireland, 1991.

[15] George B. Dantzig. Linear programming under uncertainty. *Management Science*, 1(3/4):197–206, April-July 1955.

[16] C. J. Donohue and J. R. Birge. The abridged nested decompositionmethod formultistage stochastic linear programs with relatively complete recourse. *Algorithmic Operations Research*, 1:20–30, 2006.

[17] C.J. Donohue. *Stochastic network programming and the dynamic vehicle allocation problem*. Ph.D. Dissertation, University of Michigan, Ann Arbor, MI, 1996.

[18] N.C.P. Edirisinghe and W.T. Ziemba. Implementing bounds-based approximations in convex-concave two-stage stochastic programming. *Math. Programming*, 75(2, Ser. B):295–325, 1996. Approximation and computation in stochastic programming.

[19] B. Efron. Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, 7:1–26, 1979.

[20] G. Eppen and L. Schrage. *Centralized ordering policies in a multi-warehouse system with lead times and random demand. In: Schwarz, L. (Ed.), Multi-Level Production/Inventory Control Systems: Theory and Practice, vol. 16. Studies in Management Science*. North-Holland, Amsterdam, 1981.

[21] L.F. Escudero and P.V. Kamesam. *MRP modelling via scenarios. In: Ciriani, T.A., Leachman, R.C. (Eds.), Optimization in Industry*. Wiley, New York, 1993.

[22] K. Frauendorfer. *Stochastic two-stage programming*, volume 392 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, 1992. Habilitationsschrift, University of Zürich, Zürich, 1992.

[23] K. Frauendorfer. Multistage stochastic programming: error analysis for the convex case. *Z. Oper. Res.*, 39(1):93–122, 1994.

[24] H. Gassmann. Mslip: a computer code for the multistage stochastic linear programming problem. *Mathematical Programming*, 47:407–423, 1990.

[25] N. Gröwe-Kuska, J. Dupačová, and W. Römisch. Scenario reduction in stochastic programming: An approach using probability metrics. *Mathmatical Programming Ser. A*, 95:493–511, 2003.

[26] J Guddat. Stability in convex quadratic parametric programming,. *Statistics*, 7(2):223–224, 1976.

[27] Y. Herer, M. Tzur, and E. Yücesan. The multilocation transshipment problem. *IIE Transactions*, 38(3):185–200, 2006.

[28] J. L. Higle and S. Sen. Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Mathematics of Operations Research*, 16:650–669, 1991.

[29] J. L. Higle and S. Sen. On the convergence of algorithms with implications for stochastic and nondifferentiable optimization. *Mathematics of Operations Research*, 17:112–131, 1992.

[30] J. L. Higle and S. Sen. Finite master programs in regularized stochastic decomposition. *Mathematical Programming*, 67:143–168, 1994.

[31] J. L. Higle and S. Sen. *Stochastic Decomposition: A Statistical Method for Large Scale Stochastic Linear Programming*, volume 8 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publisher, 1996.

[32] J. L. Higle and S. Sen. Statistical approximations for stochastic linear programming problems. *Annuals of Operations Research*, 85:173–192, 1999.

[33] J. L. Higle and S. Sen. Duality for multistage convex stochastic programs,. *Annals of Operations Research*, 142:129–146, 2006.

[34] M. Hindsberger and A.B. Philpott. Resa: A method for solving multi-stage stochastic linear programs. *SPIX Stochastic Programming Symposium, Berlin,*, 2001.

[35] G. Infanger and D.P. Morton. Cut sharing for multistage stochastic linear programs with interstage dependency,. *Mathematical Programming*, 75:241–256, 1996.

[36] P. Kall and S.W. Wallace. *Stochastic Programming*. John Wiley and Sons, Chichester, England, 1994.

[37] S. Karabuk and S. D. Wu. Coordinating strategic capacity planning in the semiconductor industry. *Operations Research*, 51:839–49, 2003.

[38] A. J. King and R. J.-B. Wets. Epi-consistency of convex stochastic programs. *Stochastics and Stochastics Reports*, 34:83–92, 1991.

[39] J. T. Linderoth and S. J. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24:207–250, 2003.

[40] K. Linowsky and A. B. Philpott. On the convergence of sampling-based decomposition algorithms for multistage stochastic programs. *Journal of optimization theory and applications*, 125:349–366, 2005.

[41] W. K. Mak, D. P. Morton, and R. K. Wood. Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24:47–56, 1999.

[42] J. M. Mulvey and A. Ruszczyński. A new scenario decomposition method for large scale stochastic optimization. *Operations Research*, 43:477–490, 1995.

[43] D. H. Myers, D. R. Carino, and W. T. Ziemba. Concepts, technical issues and uses of the russell-yasuda-kasai financial planning model. *Operations Research*, 46(4):450–462, 1998.

[44] S. Nielsen and S. Zenios. A massively parallel algorithm for nonlinear stochastic network problems. *Operations Research*, 41(2):319–337, 1993.

[45] M. P. Nowak and W. Römisch. Stochastic lagrangian relaxation applied to power scheduling in a hydrothermal system under uncertainty. *Annals of Operations Research*, 100:251–272, 2000.

[46] D. Paraskevopoulos, E. Karakitsos, and B. Rustem. Robust capacity planning under uncertainty. *Management Science*, 37:787–800, 1991.

[47] M. V. F. Pereira and L. M. V. G. Pinto. Multistage stochastic optimization applied to energy planning. *Mathmatical Programming*, 52:359–375, 1991.

[48] R. J. Peters, K. Boskma, and H. A. E. Kuper. Stochastic programming in production planning: a case with nonsimple recourse. *Statistica Neerlandica*, 31:113–126, 1977.

[49] A.B. Philpott and Z. Guan. On the convergence of stochastic dual dynamic programming and related methods. *Operations Research Letters*, 36:450–455, 2008.

[50] E.L. Plambeck, B-R. Fu, S.M. Robinson, and R. Suri. Sample path optimization of convex stochastic performance functions. *Mathematical Programming*, 75:137–176, 1996.

[51] W. Powell. *Principles of Mathematical Analysis.* McGraw-Hill, New York, 1976.

[52] W. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* John Wiley and Sons, New York, 2007.

[53] W. Powell, A. Ruszczynski, and H. Topaloglu. Learning algorithms for separable approximations of discrete stochastic optimization problems. *Mathematics of Operations Research*, 29:814–836, 2004.

[54] A. Prekopa. *Stochastic Programming.* Kluwer Academic Publishers, Dordrecht, 1995.

[55] W. Römisch and R. Schultz. Distribution sensitivity in stochastic programming. *Mathematical Programming*, 50:197–226, 1991.

[56] R.T. Rockafellar and R. J-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 1991.

[57] R.T. Rockafellar and R.J-B. Wets. *Variational Analysis*. Springer-Verlag, Berlin, Germany., 1997.

[58] H. Rubin. Uniform convergence of random functions with applications to statistics,. *The Annals of Mathematical Statistics*, 27(1):200–203, 1956.

[59] R. Rubinstein and A. Shapiro. *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method*. Wiley, New York, 1993.

[60] A Ruszczynski. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35:309–333, 1986.

[61] S. Sen, R. D. Doverspike, and S. Cosares. Network planning with random demand. *Telecommunications Systems*, 3:11–30, 1994.

[62] S. Sen and Z. Zhou. Optimization simulation: The case of multi-stage decision models. *Proceedings of the Winter Simulation Conference, S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu (eds.)*, 2011.

[63] S.P. Sethi, H. Yan, and Q. Zang. Optimal and hierarchical controls in dynamic stochastic manufacturing systems: A survey. *Manufacturing and Service Operations Management*, 4:133–170, 2002.

[64] A. Shapiro. Asymptotic analysis of stochastic programs. *Annals of Operations Research*, 30:169–186, 1991.

[65] A. Shapiro. On complexity of multistage stochastic programs. *Operations Research Letters*, 34:1–8, 2006.

[66] A. Shapiro. Analysis of stochastic dual dynamic programming method. *European Journal of Operational Research*, 209:63–72, 2011.

[67] A. Shapiro and T. Homem-de Mello. A simulation-based approach to stochastic programming with recourse. *Mathematical Programming,*, 81:301–325, 1998.

[68] K. Singh. On the asymptotic accuracy of efron's bootstrap. *The Annals of Statistics*, 9(6):1187–1195, 1981.

[69] R. M. Van Slyke and R. J-B Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17:638–663, 1969.

[70] J. Wu and S. Sen. A stochastic programming model for currency option hedging. *Annals of Operations Research*, 100:227–250, 2000.

[71] C. Yano and H. Lee. Lot sizing with random yields: A review. *Operations Research*, 43:311–334, 1995.

[72] W.H.M. Zijm. Towards intelligent manufacturing planning and control systems. *OR Spektrum*, 22:313–334, 2000.