STUDY ON OPTIMALITY CONDITIONS IN STOCHASTIC LINEAR PROGRAMMING

by

Lei Zhao

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
WITH A MAJOR IN SYSTEMS ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

2 0 0 5

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Lei Zhao
entitled Study on Optimality Conditions in Stochastic Linear Programming
and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Date: 22 November 2005
Julia L. Higle

Date: 22 November 2005
Suvrajeet Sen

Date: 22 November 2005
J. Cole Smith

Date: 22 November 2005
Daniel Zeng

Date: 22 November 2005
Kurt D. Fenstermacher

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation 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: 22 November 2005
Dissertation Director: Julia L. Higle

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Lei Zhao

ACKNOWLEDGEMENTS

I definitely thank my advisor, Professor Julia L. Higle, for her rigorous training in my research and her financial and spiritual support, without which I would not have been able to achieve this lifetime achievement.

Unfortunately, professor Suvrajeet Sen went to work in NSF in the past two years. But his earlier lectures provided insights and laid a foundation on my research. I thank him for kindly sharing his experience and insights on research during our discussions and collaboration after his return to school in the last semester.

Professor J. Cole Smith has been very helpful. I especially thank him for his valuable advice regarding starting off as a young researcher.

I thank two minor committee members, Professor Daniel Zeng and Professor Kurt D. Fenstermacher, for providing me advice on research areas besides my dissertation research.

Lewis Ntaimo, Chisonge Mofya, and I initiated our student study group on multistage stochastic programming. The group continued on other topics with new members Jingquan Li, Lefei Li, Yao Cheng, etc. I learned a lot from studying with these excellent students. I also enjoyed working with and thank the help from students in our research group: Anne Johnson, Jenn Barzeele, and Dale Henderson.

I especially want to thank Lefei Li for his kindly help in the last phase of my dissertation, when both my desktop and laptop computers failed to work at that crucial period of time.

Finally, I want to express my deep appreciation to my family and my dear friends for their support throughout these years. Among my friends, I especially thank Yiming Li, who encouraged and helped me to come to the University of Arizona, Jian Lan, who has been very supportive in the past nine years ever since I knew him. I thank my long time friends Haochun Qian, Dongming Zhang, and Jin Qu. I cannot imagine what the past 15 years would have been like without their constant friendship through the highs and lows.

DEDICATION

Dedicated to my grandma, my parents, and all my dear friends.

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In the rapidly changing world of today, people have to make decisions under some degree of uncertainty. At the same time, the development of computing technologies enables people to take uncertain factors into considerations while making their decisions. Stochastic programming techniques have been widely applied in financial engineering, supply chain management, logistics, transportation, etc. Such applications often involve a large, possibly infinite, set of scenarios. Hence the resulting programs tend to be large in scale.

The need to solve large scale programs calls for a combination of mathematical programming techniques and sample-based approximation. When using sample-based approximations, it is important to determine the extent to which the resulting solutions are dependent on the specific sample used. This dissertation research focuses on computational evaluation of the solutions from sample-based two-stage/multistage stochastic linear programming algorithms, with a focus on the effectiveness of optimality tests and the quality of a proposed solution.

In the first part of this dissertation, two alternative approaches of optimality tests of sample-based solutions, adaptive and non-adaptive sampling methods, are examined and computationally compared. The results of the computational experiment are in favor of the adaptive methods. In the second part of this dissertation, statistically motivated bound-based solution validation techniques in multistage linear stochastic programs are studied both theoretically and computationally. Different approaches of representations of the nonanticipativity constraints are studied. Bounds are established through manipulations of the nonanticipativity constraints.

CHAPTER 1

Introduction

In a wide variety of applications, such as supply chain management, logistics, transportation, revenue management, and financial planning, people often have to make decisions periodically with respect to an uncertain future. Under the pressures of increasingly intensive competition and narrow profit margins, it is no longer advisable for any decision makers to rely on "intuitive" decisions alone, especially as the scale of the decision problem increases and/or the impact of uncertain information is significant.

Deterministic models are not designed to address the requirements of decisions under uncertainty. When considered within the context of an uncertain environment, deterministic models yield suboptimal solutions. Stochastic programming methods, especially those that combine mathematical programming and applied statistical techniques, provide a powerful class of modeling and solution tools for dynamic decision problems under uncertainty.

When the problem scale is large, people often resort to sample-based solution techniques. In such cases, it is typically hard, or even impossible, to obtain solutions whose optimality can be deterministically verified and it is necessary to explore the quality of a proposed solution empirically. This dissertation explores the empirical evaluation of the effectiveness of optimality tests and the quality of a proposed solution.

Empirical investigations of the impact and effectiveness of optimality tests for two-stage stochastic linear programs with recourse (TS-SLP) are discussed in Chapter 3. Sample-based algorithms for TS-SLP can be classified as either adaptive or nonadaptive methods, depending on whether or not they incorporate sampling into their solution processes. As an adaptive solution algorithm for TS-SLP, *stochastic decomposition* (SD, Higle and Sen [1991, 1996b]) enables termination of the algo-

rithm based on the recognition of sufficient evidence of solution quality. However, repetitively solving "bootstrapped" master problems to obtain tight lower bounds on the master problems in the optimality tests can be cumbersome.

An improvement on the optimality test efficiency is proposed for the regularized version of the SD method (RSD, Higle and Sen [1994, 1999]), where a quadratic term is added to the master program. Instead of solving each bootstrapped master program, RSD calculates a lower bound on the objective value of the bootstrapped master program based on a feasible dual solution. Although asymptotic convergence of the SD method is not dependent on the form of its master program used, we are interested in exploring how the differences in the optimality tests of these two versions of the SD method impact the quality of the solution identified and the computational effort required to identify the solution.

We conclude our computational experiments with optimality tests for sample based algorithms for TS-SLP, by comparing the adaptive SD method with an non-adaptive optimality test framework, *sample average approximation* (SAA, Shapiro [2000]).

In multistage stochastic linear programs with recourse (MS-SLP), the characteristic *nonanticipativity constraints* ensure that decisions in each stage depend only on information available at the time that a commitment to the decision must be made, and not on information that will only become available in future stages. The objective function evaluation of any given first-stage solution in MS-SLP is not available for us to easily measure the quality of the proposed solution. Evaluating the objective function value requires the solution of MS-SLP problems with one less stage than the original problem. In chapters 4 and 5, we develop and test a statistically motivated bound-based approach to solution validation through which we can examine the gap between an upper bound on its objective function value and a lower bound on the optimal objective value.

In chapter 4, we explore lower bounds on the optimal objective value of the MS-SLP obtained through a partial relaxation of the nonanticipativity constraints in the *deterministic equivalent program* (DEP). In our computational experiment, we

examine the impact of various representations of the nonanticipativity constraints, schemes for selecting constraints to be relaxed, and relaxation levels on both the lower bound quality and the computational effort required to obtain the lower bound.

In Chapter 5, we obtain upper bounds on the objective function value of a given first-stage solution through manipulations of the solutions we obtained from the lower bound procedure, which satisfy a subset of the nonanticipativity constraints. Three procedures to obtain upper bounds are compared empirically and the quality of the solution obtained from the lower bound procedures is further examined.

Our empirical investigations indicate that, with appropriate representation and manipulation of the nonanticipativity constraints, a significant amount of nonanticipativity constraints can be eliminated from the DEPs without sacrificing the quality of the lower and upper bounds in MS-SLP. This insight can serve as guidelines in designing efficient solution validation procedures and sample-based solution algorithms in MS-SLP, especially by combining Lagrangian approaches. This is important when solving large scale real world problems.

CHAPTER 2

Literature Review

2.1 Multistage Stochastic Programming Models

In Multistage Stochastic Linear Programming (MS-SLP), information evolves randomly over "time" and decisions are made sequentially. Firstly we want to differentiate between the notions of *stages* in the decision problem and *time periods* in the model. *Stages* correspond to points in time when information becomes available. *Time periods* correspond to points in time at which actions must be taken. Consequently, a stage in the decision problem may correspond to multiple time periods in the model.

In a multistage optimization program, decisions are to be made in stages $t = 1, ..., T$. The decision vector, x, is a collection of sub-vectors,

$$x = (x_1, ..., x_T),$$

where $x_t \in R^{n_t}$ is the vector of decisions corresponding to stage t, $t = 1, ..., T$. Let $n = \sum_{t=1}^{T} n_t$ and consequently $x \in R^n$. For example, in a dynamic fixed-income portfolio selection problem, where business activities include borrowing and lending bonds with different maturities, x_t can be defined as maturities borrowed and/or lent in stage t.

Decisions in successive stages need to satisfy two groups of constraints. The first group of constraints are specific to a single stage t:

$$x_t \in X_t, \quad t = 1, ..., T,$$

where $X_t \subset R^{n_t}$, $t = 1, ..., T$. The second group describes the dynamics of the system and involves decisions from multiple stages. In the simplest linear model, they can be expressed as:

$$T_t x_{t-1} + W_t x_t = r_t, \quad t = 2, ..., T,$$

where $T_t \in R^{m_t \times n_{t-1}}$, $W_t \in R^{m_t \times n_t}$, and $r_t \in R^{m_t}$, $t = 2, ..., T$. T_t, $t = 2, ..., T$, are referred to as *technology matrices* and W_t, $t = 2, ..., T$, are referred to as *recourse matrices* in the literature. In this sense, decisions at each stage are affected by the decision history up to that stage, and affect decisions in future stages as well. In the fixed-income portfolio selection problem, there can be cash flow constraints and liability constraints in each stage t, some of which are affected by bonds borrowed or lent in previous stages.

The goal of a multistage linear program is to optimize a linear cost function:

$$c^\top x = c_1^\top x_1 + c_2^\top x_2 + ... + c_T^\top x_T,$$

where $c_t \in R^{n_t}$ for $t = 1, ..., T$. In the fixed-income portfolio selection problem, the objective can be to maximize the accumulated profit over a time horizon T.

In stochastic programming, some of the data elements in $\{X_t, c_t, T_t, W_t, r_t\}_{t=2}^T$ are random variables whose actual values become known over time. For example, in the fixed-income portfolio selection problem, interest rates and bond prices are usually dynamic and uncertain. Thus, information evolves over time and decisions in any given stage may vary as a function of the information that is available at that time. The first-stage data set, X_1, is fixed. Hence the first-stage decisions, x_1, the so-called *here-and-now* decisions, are deterministic. The data in other stages include random variables, and therefore the associated decisions, x_t, $t = 2, ..., T$, the so-called *wait-and-see* decisions, may depend on the manner in which the uncertain data is realized.

To model a multistage stochastic linear program, we use a multivariate random variable, $\tilde{\omega} = \{\tilde{\omega}_2, ..., \tilde{\omega}_T\}$, with distribution P on sample space Ω, to represent the complete random data process, and use $\omega = \{\omega_2, ..., \omega_T\}$ to denote a realization of such a process. We define the stage-wise notations as follows:

- $\tilde{\omega}_t$ represents random data in stage t with distribution P_t and sample space Ω_t;

- ω_t represents a realization of $\tilde{\omega}_t$, $\omega_t \in \Omega_t$;

- $\underline{\tilde{\omega}}_t = (\tilde{\omega}_2, ..., \tilde{\omega}_t)$, represents the history of the random data process up to, and including, t;
- $\underline{\omega}_t = (\omega_2, ..., \omega_t)$, represents a realization of $\underline{\tilde{\omega}}_t$;
- x_t represents a decision in stage t;
- $\underline{x}_t$ represents a decision history up to, and including, t, i.e., $\underline{x}_t = \{x_\tau\}_{\tau=1}^t$.

For each $t \in \{2, ..., T\}$, $\{X_t(\underline{\omega}_t), c_t(\underline{\omega}_t), T_t(\underline{\omega}_t), W_t(\underline{\omega}_t), r_t(\underline{\omega}_t)\}$ are stage t data components as functions of the realization, $\underline{\omega}_t$. Because the optimal values of the decision variables in stage t, $t = 2, ..., T$, depend on $\underline{\tilde{\omega}}_t$, they are induced random variables.

Each realization of $\omega \in \Omega$ corresponds to a sequence of data components

$$\omega = \{X_t(\underline{\omega}_t), c_t(\underline{\omega}_t), W_t(\underline{\omega}_t), r_t(\underline{\omega}_t), T_t(\underline{\omega}_t)\}_{t=1}^T,$$

and is referred to as a *scenario*. To represent the complete information process, we include the deterministic first stage data, although it does not vary with the scenarios.

In some cases, a MS-SLP is modeled via constraints on decision variables that would otherwise vary freely with the scenario. In such cases, we denote $x_t(\omega)$ as the decisions made at stage t corresponding to scenario ω. The program corresponding to a scenario is simply a multistage linear program, and we call it a *scenario problem*.

At each stage t, decisions can only be based on the information available so far, $\underline{\omega}_t$, as well as the decisions made in previous stages, $\underline{x}_{t-1}$. This structure is referred to as the *nonanticipativity condition*, and can be thought of as reflecting implementability requirements. As we will discuss later, there are different ways to express nonanticipativity constraints, explicitly or implicitly, which in turn will affect the design and computational performance of solution algorithms in MS-SLP.

The goal of MS-SLP is to optimize some statistical measure of the stochastic objective function values, subject to a variety of constraints with embedded uncertain elements that evolve over time, as well as the nonanticipativity constraints. For

example, in the fixed-income portfolio selection problem, we may wish to maximize the expected accumulated profit over a time horizon of five years, with bond borrowing and lending decisions made annually and subject to annual cash-flow and liability constraints in each year, within which interest rates and bond prices are uncertain. When the objective function and the constraints in each stage are linear, the resulting program is a Stochastic Linear Program (SLP) with five stages.

We assume that the time horizon, T, is finite. We also assume that probabilistic descriptions or distributions of the random variable, $\tilde{\omega}$, are discrete and finite, and are given. A multi-stage stochastic linear program may be modeled as

$$\begin{aligned} \text{minimize} \quad & E[c(\tilde{\omega})^\top x(\tilde{\omega})] && && \text{(MS-SLP)} \\ \text{subject to} \quad & x(\tilde{\omega}) \in X(\tilde{\omega}), && w.p.1 && \text{(2.1a)} \\ & x(\tilde{\omega}) \in \mathcal{N}, && w.p.1 && \text{(2.1b)} \end{aligned}$$

where,

$$X(\omega) = \{\{x_t(\omega)\}_{t=1}^T | x_t(\omega) \in X_t(\omega), \sum_{j<t} T_j(\underline{\omega}_t)\underline{x}_j(\underline{\omega}_t) + W_t(\underline{\omega}_t)\underline{x}_t(\underline{\omega}_t) = r_t(\underline{\omega}_t), \forall t\},$$

and $\mathcal{N}$ is the set of nonanticipative (implementable) solutions.

This formulation is referred to as the *Deterministic Equivalent Program* (DEP). It expresses one of the main features of MS-SLP, that is, the decisions in each stage are constrained explicitly by the decisions made in prior stages, and at the same time, are influenced by the decisions in later stages indirectly by incurring costs (expected objective function values).

The nonanticipativity constraints must ensure that for all $\omega, \omega' \in \Omega$, and any $t = 1, ..., T$,

$$x_t(\omega) = x_t(\omega'), \quad \text{if} \quad \underline{\omega}_t = \underline{\omega}'_t. \tag{2.2}$$

That is, decisions corresponding to scenarios sharing the same history up to stage t should be equal. A solution that satisfies nonanticipativity constraints is called a *nonanticipative* or an *implementable* solution.

When Ω is a finite set, so that $\Omega = \{\omega^s | s = 1, ..., |S|\}$, it is common to use a tree structure, known as a *scenario tree*, to depict the information structure of MS-SLP (see, for example, Figure 2.1). In the scenario tree, the root node represents the deterministic first stage data. A node in stage t represents a realization of the stage-wise random data $\tilde{\omega}_t$ in stage t. Except for the root node, each node has one *ancestor* node; and except for the leaf nodes, each node has one or more *descendant* nodes. A path in the scenario tree, from the root node to a leaf node, corresponds to a complete realization of the random data $\tilde{\omega}$. That is, each path is a *scenario* in MS-SLP.

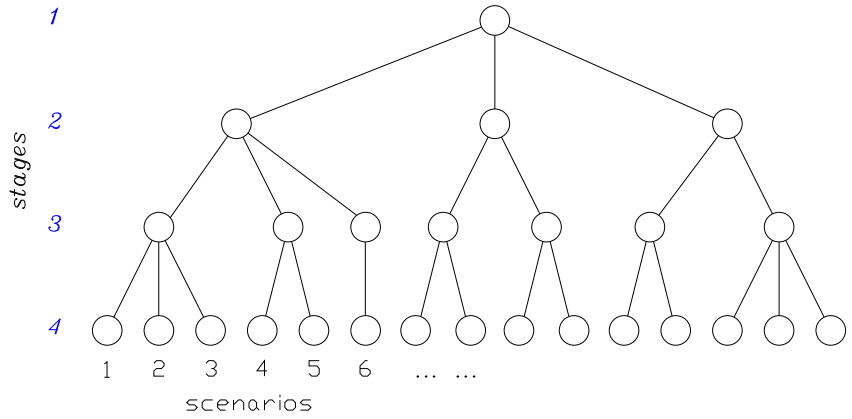


Figure 2.1: A Typical Scenario Tree for MS-SLP

Depending on the structure of the scenario tree, the size of MS-SLP can become very large. There are direct methods for solving MS-SLP such as those described in Birge [1997]. However, these standard optimization methods encounter difficulties as the problem size increases.

When solving large scale MS-SLP instances, people often apply decomposition methods, some of which have been summarized in Birge [1997] and Ruszczyński [1999]. Decomposition-based methods for MS-SLP, according to different perspectives of the MS-SLP models and decomposition schemes, can be classified into two categories: *scenario decomposition* algorithms and *stage decomposition* algorithms.

2.2 Scenario Decomposition

When there are a finite number of scenarios, $\Omega = \{\omega^s\}_{s=1}^{|S|}$, where S is a finite index set ($|S| < \infty$). Let $p^s = P\{\tilde{\omega} = \omega^s\}$, $c^s = c(\omega^s)$, and $X^s = X(\omega^s)$. (2.1) can be rewritten as

$$\text{minimize} \quad \sum_{s \in S} p^s c^{s\top} x^s \tag{MS-SLP}$$

$$\text{subject to} \quad x^s \in X^s, \qquad s \in S, \tag{2.3a}$$

$$\{x^s\}_{s \in S} \in \mathcal{N}. \tag{2.3b}$$

Here,

$$x^s = (x_1^s, ..., x_T^s),$$

where x_t^s denotes the decision to be made in stage t if the scenario happens to be s (i.e., $\tilde{\omega} = \omega^s$).

For each $s \in S$, we have a scenario problem

$$\text{minimize} \quad c^{s\top} x^s \tag{P^s}$$

$$\text{subject to} \quad x^s \in X^s.$$

In MS-SLP, the scenario problems are linked by the nonanticipativity constraints (2.3b). Without the nonanticipativity constraints, the scenario problems are all independent of each other and the program can be decomposed into independent scenario subproblems that can be solved individually. This leads to *scenario decomposition* algorithms, which seek different ways to represent and to relax the nonanticipativity constraints, and then impose them iteratively, until an optimal or near-optimal implementable solution is found.

2.2.1 Progressive Hedging Algorithm

In the *progressive hedging algorithm* (PHA) (Rockafellar and Wets [1991]), scenarios are bundled at each stage (according to the nodes in a scenario tree) to reflect the

availability of information (the nonanticipativity condition). By studying the scenario problems and their optimal solutions, one may be able to discover similarities and trends, make iterative adjustments to the decisions/solutions to adapt to the nonanticipativity structure, and eventually come up with a "well hedged" solution to the underlying problem, which seeks to optimize the expected objective function value across all scenarios, based on some probabilistic description of the scenario structure.

To model the nonanticipativity constraints, Rockafellar and Wets [1991] use the notation *scenario bundles*, denoted as $\mathcal{B}$. In each stage, t, the scenario set, S, is partitioned into finitely many disjoint subsets, which are called *scenario bundles*. The scenarios in any scenario bundle are regarded as observationally indistinguishable in stage t (i.e., they share the same information history, $\underline{\omega}_t$). Denoting the collection of all scenario bundles at stage t by $\mathcal{A}_t$, the nonanticipativity condition require that, for $t = 1, ..., T-1$, x_t^s must be constant relative to $s \in \mathcal{B}$ for each $\mathcal{B} \in \mathcal{A}_t$. That is, if $\mathcal{B} \in \mathcal{A}_t$, and if $s, s' \in \mathcal{B}$, then $\underline{x}_t^s = \underline{x}_t^{s'}$. A solution satisfying such constraints is an implementable solution.

A solution that is feasible to a scenario problem (P^s) is called an *admissible solution*, and the collection of admissible solutions is

$$\mathcal{L} = \{x^s | x^s \in X^s, \forall s \in S\}. \tag{2.4}$$

A solution that is obtained by solving a possibly perturbed version of a scenario problem (P^s) where the objective function differs from the original one, is called a *contingent solution*.

An admissible solution can be easily obtained by solving the individual scenario problem (P^s). Note that perturbing the objective function does not impact the admissability of a solution, and thus a contingent solution is always an admissible solution. The question becomes how to make use of the insights gained by solving the decomposed (perturbed) scenario problems to obtain a good *feasible solution* that is both admissible and implementable through an iterative adjustment of the perturbed scenario problems to adapt to the structure imposed by the nonanticipa-

tivity conditions.

Let $x_t(\mathcal{B})$ denote a "weighted average" of x_t^s for all scenarios in the scenario bundle $\mathcal{B}$,

$$\begin{aligned} x_t(\mathcal{B}) &= \sum_{s\in\mathcal{B}} p^s x_t^s / \sum_{s\in\mathcal{B}} p^s \\ &= E[x_t^s | s \in \mathcal{B}]. \end{aligned} \tag{2.5}$$

The nonanticipativity conditions can be expressed as:

$$\hat{x}_t^s = x_t(\mathcal{B}), \quad \forall s \in \mathcal{B}, \quad \forall \mathcal{B} \in \mathcal{A}_t, \quad t = 1, ..., T. \tag{2.6}$$

Clearly, (2.6) is equivalent to (2.2), and $\hat{x}$ is implementable, i.e. $\hat{x} \in \mathcal{N}$.

The transformation

$$J : x \to \hat{x} \text{ defined by } (2.5) - (2.6) \tag{2.7}$$

is linear and satisfies $J^2 = J$. It can be used to obtain a projection from $\mathcal{L}$ onto $\mathcal{N}$, which depends only on the information structure (scenario tree) as well as the probabilities, $\{p^s\}_{s\in S}$, and is called the *aggregation operator* or the *conditional expectation operator* relative to the information structure and probabilistic description.

Define another operator

$$K = I - J \quad (Kx = x - \hat{x}).$$

Then the nonanticipativity constraints are imposed by the linear constraint set $Kx = 0$ or $(x = \hat{x})$, and

$$\mathcal{N} = \{x | Kx = 0\}. \tag{2.8}$$

The complementary subspace to $\mathcal{N}$ is

$$\begin{aligned} \mathcal{U} = \mathcal{N}^\perp &= \{\pi | J\pi = 0\} \\ &= \{\pi | E[\pi_t^s | \mathcal{B}] = 0, \forall \mathcal{B} \in \mathcal{A}_t, t = 1, ..., T\}, \end{aligned} \tag{2.9}$$

and K is the orthogonal projection on $\mathcal{U}$.

Now (MS-SLP) can be equivalently expressed as

$$\begin{aligned} \text{minimize} \quad & c^\top x = E[c^{s\top} x^s] && \text{(MS-SLP}^{PHA}\text{)} \\ \text{subject to} \quad & x \in \mathcal{L}, && \text{(2.10a)} \\ & Kx = 0. && \text{(2.10b)} \end{aligned}$$

Taking the Lagrangian relaxation of the nonanticipativity constraints (2.10b), we obtain

$$\min_{x \in \mathcal{L}} \quad c^\top x + (\pi')^\top (Kx), \tag{2.11}$$

Let $\pi = K\pi'$. Since K is an orthogonal projection, we have $K^\top = K$. Therefore the objective function above can be rewritten as

$$L(x, \pi) = c^\top x + \pi^\top x, \quad \forall x \in \mathcal{L}, \forall \pi \in \mathcal{U}. \tag{2.12}$$

An *augmented* Lagrangian of (MS-SLP^{PHA}) with respect to (2.10b) is

$$\begin{aligned} L_a(x, \pi) &= c^\top x + \pi^\top x + \frac{\sigma}{2} ||Kx||^2 \\ &= c^\top x + \pi^\top x + \frac{\sigma}{2} ||x - \hat{x}||^2, \quad \forall x \in \mathcal{L}, \forall \pi \in \mathcal{U}, \sigma > 0. \end{aligned} \tag{2.13}$$

The MS-SLP with perturbed or modified scenario problems then becomes

$$\min_{x \in \mathcal{L}} L_a(x, \pi). \tag{2.14}$$

Through a good choice of of $\pi \in \mathcal{U}$ and $\sigma > 0$, one can expect that (2.14) can be used as a close representation of (MS-SLP^{PHA}), in the sense that its nearly optimal solutions will be good approximations to an optimal solution of (MS-SLP^{PHA}). Thus we have an *augmented Lagrangian method* as described in Algorithm 2.2.1.

The difficulty associated with algorithm 2.2.1 lies in the fact that $L_a(x, \pi)$ is not decomposable because of the term $||Kx||^2$. This is overcome by the use of the *alternating direction method* (Fortin and Glowinski [1983]), which enables decomposition of (2.14) into approximate scenario problems and solves them approximately in each iteration, as in Algorithm 2.2.2. The details are described in Rockafellar and Wets [1991].

Step 1 For fixed multipliers π^k, solve the problem $\min_{x \in \mathcal{L}} L_a(x, \pi^k)$. Let x^k be the optimal solution.

Step 2 If $Kx^k = 0$, stop (optimal solution found); otherwise, set

$$\pi^{k+1} = \pi^k + \sigma K x^k,$$

Increase k by 1 and go to step 1.

Algorithm 2.2.1: Augmented Lagrangian Method

The convergence of the progressive hedging algorithm as described in Algorithm 2.2.2 is based on the theory of the proximal point algorithm in nonlinear programming, as developed in Rockafellar [1976]. The basis for this approach is not dual ascent but the contraction of the pair of primal-dual solutions about a saddle point.

A strength of this algorithm is that it involves at every iteration an *admissible* solution x^k and an *implementable* solution $\hat{x}^k$. The distance expression

$$||x^k - \hat{x}^k|| = ||Kx^k|| \tag{2.15}$$

can be readily computed and taken as a measure of how far a candidate solution is from satisfying all the nonanticipativity constraints. One can always stop at any point with an implementable solution at hand, so that a graceful termination of the method is possible.

On the other hand, the nonanticipativity constraints defined by (2.5) and (2.6) result in a dense matrix of dimension $R^{n|S| \times n|S|}$, $n = \sum_{t=1}^{T} n_t$: each variable enters as many equations as the number of scenarios in its bundle in each stage. Numerical experience shows that convergence of this method can be slow in some cases (Halgason and Wallace [1991], Mulvey and Vladimirou [1991], Vladimirou [1990]).

2.2.2 Augmented Lagrangian Decomposition Algorithm

The manner in which the nonanticipativity constraints are defined can have a significant impact on the computational requirements of a solution procedure. A compact

step 0 [Initialization] $k = 0$. A set of admissible but not necessarily implementable solution $x^0 \in \mathcal{L}$ and dual multipliers $\pi^0 \in \mathcal{U}$ are given. (One can take $x^{0,s}$ to be the optimal solutions to the scenario problems (P^s), $s \in S$. And let $\pi^0 = 0$.)

Step 1 Calculate $\hat{x}^k = Jx^k$, which is implementable but not necessarily admissible. If $||x^k - \hat{x}^k|| \leq \epsilon$, where ϵ is a pre-defined tolerance, stop; otherwise, go to step 2.

Step 2 For each $s \in S$, solve perturbed scenario problem

$$\min_{x \in X^s} c^{s\top} x + \pi^{k,s\top} x + \frac{\sigma}{2} ||x - \hat{x}^{k,s}||^2$$

to get $x^{k+1,s}$, which is admissible but not necessarily implementable.

Step 3 Update $\pi^{k+1} \in \mathcal{U}$:

$$\pi^{k+1} = \pi^k + \sigma K x^{k+1}.$$

$k \leftarrow k + 1$, go to step 1.

Algorithm 2.2.2: Progressive Hedging Algorithm

and convenient description of the nonanticipativity constraints can help to ease the computational requirements.

The representation of the nonanticipativity constraints proposed in the *augmented Lagrangian decomposition algorithm* (ALD) (Rosa and Ruszczyński [1996]) and the *diagonal quadratic approximation method* (Mulvey and Ruszczyński [1992, 1995]) relies on the following definitions:

The last common stage of scenarios ω and ω' is defined by

$$t^{max}(\omega, \omega') = \max\{t : \underline{\omega}_t = \underline{\omega}'_t\}. \tag{2.16}$$

All the scenarios in Ω are ordered by assigning to them indices $s = 1, ..., |S|$ such that for every s, scenario $s + 1$, among all the remaining scenarios $s' > s$, has the

largest last common stage with s, i.e.:

$$t^{max}(\omega^s, \omega^{s+1}) = \max\{t^{max}(\omega^s, \omega^{s'}) : s' > s\}. \tag{2.17}$$

With such an arrangement, the *sibling* of scenario s in stage t is defined as

$$\nu(s,t) = \begin{cases} s+1, & if \quad t^{max}(\omega^s, \omega^{s+1}) \geq t, \\ \min\{s' : t^{max}(\omega^s, \omega^{s'}) \geq t\}, & otherwise. \end{cases} \tag{2.18}$$

In every stage t, the mapping $\nu : \Omega \times T \to \Omega$ defines a permutation of $\{1, ..., |S|\}$. The *inverse sibling* of scenario s is denoted by $\nu^{-1}(s,t)$. For example, the siblings of scenarios $s = 1, ..., 6$ in each stage t, $t = 1, ..., 4$, in Figure 2.1 are listed in Table 2.1.

	s = 1	2	3	4	5	6
t = 1	2	3	4	5	6	7
2	2	3	4	5	6	1
3	2	3	1	5	4	6
4	1	2	3	4	5	6

Table 2.1: Example: Siblings of Scenario 1 - 6 in Figure 2.1

With the above definitions, the nonanticipativity constraints (2.3b) can be expressed as

$$x_t^s = x_t^{\nu(s,t)}, \quad s = 1, ..., |S|, \quad t = 1, ..., T-1. \tag{2.19}$$

Then (MS-SLP) can be equivalently expressed as

$$\begin{aligned} \text{minimize} \quad & \sum_{s=1}^{|S|} p^s \sum_{t=1}^{T} c_t^\top x_t^s \\ \text{subject to} \quad & \text{(2.3a) and (2.19).} \end{aligned} \tag{2.20}$$

An augmented Lagrangian function for (2.20) has the form

$$\Lambda(x, \pi) = \sum_{s=1}^{|S|} p^s \sum_{t=1}^{T} c_t^\top x_t^s + \sum_{s=1}^{|S|} \sum_{t=1}^{T-1} \pi_t^{s\top}(x_t^s - x_t^{\nu(s,t)}) + \frac{\sigma}{2} \sum_{s=1}^{|S|} \sum_{t=1}^{T-1} ||x_t^s - x_t^{\nu(s,t)}||^2, \tag{2.21}$$

which is the Lagrangian relaxation with respect to the nonanticipativity constraints (2.19) augmented by a quadratic term.

Similar to Algorithm 2.2.1, the augmented Lagrangian decomposition algorithm can be described as follows:

Step 1 For fixed multipliers π^k, solve the problem minimize $\Lambda(x, \pi^k)$ subject to (2.3a). Let $x^k = (x^{k,1}, ..., x^{k,|S|})$ be the optimal solution.

Step 2 If $||x_t^{k,s} - x_t^{k,\nu(s,t)}|| \leq \epsilon$ for all s and t, stop (optimal solution found); otherwise, set for $s = 1, ..., |S|$ and $t = 1, ..., T-1$,

$$\pi_t^{s,k+1} = \pi_t^{s,k} + \sigma(x_t^{s,k} - x_t^{\nu(s,t),k}),$$

Increase k by 1 and go to step 1.

Algorithm 2.2.3: Augmented Lagrangian Decomposition Algorithm

Advantages of augmented Lagrangian dual methods over general Lagrangian dual methods include stability of the multiplier iterates and the possibility of warm start from an arbitrary x^0. However, as in the progressive hedging algorithm, the non-decomposable augmented term presents a serious disadvantage.

To overcome this difficulty, for each $s \in S$, let

$$\begin{aligned}\Lambda^s(x, \pi) =& p^s \sum_{t=1}^{T} c_t^\top x_t^s + \sum_{t=1}^{T-1} (\pi_t^s - \pi_t^{\nu^{-1}(s,t)})^\top x_t^s \\ &+ \frac{\sigma}{2} \sum_{t=1}^{T-1} \{||x_t^s - \tilde{x}_t^{\nu(s,t)}||^2 + ||x_t^s - \tilde{x}_t^{\nu^{-1}(s,t)}||^2\}.\end{aligned} \tag{2.22}$$

Assuming that other variables are temporarily fixed at their value $\tilde{x}_{s'}$ for $s' \in S$ and $s' \neq s$, we then minimize the augmented Lagrangian with respect to the variables associated with scenario s, as follows:

$$\min_{x^s \in X^s} \Lambda^s(x, \pi). \tag{2.23}$$

Step 1 of the algorithm 2.2.3 can then be implemented using the Jacobi method (algorithm 2.2.4) as described in details in Rosa and Ruszczyński [1996].

Step 1.0 Set $\pi = \pi^k$, $m = 1$, $\tilde{x}^{k,m} = x^{k-1}$, and $\alpha \in (0, 1/2)$.

Step 1.1 For $s = 1, ..., |S|$, solve (2.23) with $\tilde{x} = \tilde{x}^{k,m}$ to getting points $x^{k,m,s}$.

Step 1.2 If $||x_t^{k,m,s} - \tilde{x}_t^{k,m,s}|| \leq \epsilon, \forall (s,t) \ni s \neq \nu(s,t)$, then stop; otherwise, set for $s = 1, ..., |S|$, $t = 1, ..., T$

$$\tilde{x}_t^{k,m+1,s} = \tilde{x}_t^{k,m,s} + \alpha(x_t^{k,m,s} - \tilde{x}_t^{k,m,s}), \tag{2.24}$$

increase m by 1 and go to step 1.

Algorithm 2.2.4: Jacobi method

Mulvey and Ruszczyński [1995] discusses a parallel implementation of the augmented Lagrangian decomposition method.

By comparing the progressive hedging algorithm and augmented Lagrangian Method, the representations of nonanticipativity constraints not only affect the density of the nonanticipativity constraint matrices, but also have a strong impact on the approaches for seeking an implementable solution.

2.2.3 Stochastic Scenario Decomposition

In the *Stochastic Scenario Decomposition* (SSD) method, Higle et al. [2004] represent the nonanticipativity constraints in terms of state vectors, each of which is defined for each node in the scenario tree.

Let Γ denote the set of nodes in the scenario tree. Each node $\gamma \in \Gamma$ corresponds to a stage and a bundle of scenarios, denoted by $t(\gamma)$ and $\mathcal{B}_\gamma$, respectively. For each $\gamma \in \Gamma$, let z_γ denote the corresponding state vector. The nonanticipativity constraints (2.3b) can be expressed as

$$x_{t(\gamma)}^s = z_\gamma, \quad \forall s \in \mathcal{B}_\gamma, \quad \forall \gamma \in \Gamma. \tag{2.25}$$

Let $m(s,t)$ denote the node in the scenario tree associated with scenario s in stage t. Observe that $m(s,t) = \gamma$, for all $s \in \mathcal{B}_\gamma$ and $t = t(\gamma)$. (2.25) can also be

represented as

$$x_t^s = z_{m(s,t)}, \quad \forall s \in S, \quad t = 1, ..., T. \tag{2.26}$$

For every scenario $s \in S$, we introduce scaled dual multipliers $\{\xi^s\}_{s\in S}$ associated with the nonanticipativity constraints in (2.25). The Lagrangian dual problem of (MS-SLP) is:

$$\begin{aligned} \text{minimize} \quad & \sum_{s\in S} p^s\{c^\top x^s - \xi^{s\top} x^s\} + \sum_{\gamma\in\Gamma}(\sum_{s\in\mathcal{B}_\gamma} p^s \xi^s_{t(\gamma)})^\top z_\gamma, \\ \text{subject to} \quad & \text{(2.3a)}. \end{aligned} \tag{2.27}$$

Observe that $\{z_\gamma\}_{\gamma\in\Gamma}$ are unconstrained, so that the inner summation in the last term in (2.27) must equal zero in order to avoid an unbounded solution. Define

$$\Xi = \{\{\xi^s\}_{s\in S} | \sum_{s\in\mathcal{B}_\gamma} p^s \xi^s_{t(\gamma)} = 0, \quad \forall \gamma \in \Gamma\}. \tag{2.28}$$

The dual of (MS-SLP) can be formulated as

$$\max_{\xi\in\Xi} \min_{x\in X} \{\sum_{s\in S} p^s (c^\top x^s - \xi^{s\top} x^s)\} \tag{2.29}$$

Let

$$\nu(\xi) = \min_{x\in X} \sum_{s\in S} p^s (c^\top x^s - \xi^{s\top} x^s). \tag{2.30}$$

It follows that (2.29) can be restated as

$$\max_{\xi\in\Xi} \nu(\xi). \tag{2.31}$$

SSD is a sample-based cutting plane approximation in which successive approximations of the piecewise linear concave function ν are developed.

As the iterations progress, the number of cuts grows, and the number of variables grows in the master (dual) problem as well. The potential problem created by the growth in problem size is alleviated by a column aggregation scheme, in which columns are aggregated without regard for either the scenario or stage to which the variables correspond. Computational experience results in 20-25% reductions on the master problem size due to the aggregation.

In all the scenario decomposition based algorithms described above, MS-SLPs are decomposed into separable scenario problems first by taking the Lagrangian dual relaxation with regard to the nonanticipativity constraints. Information obtained from solving the scenario problems are then used to seek for an implementable solution. The number of the nonanticipativity constraints, consequently, the number of Lagrangian dual variables may have a strong impact on the computational performance of these algorithms.

2.3 Stage Decomposition

In the previous section, we consider scenario decomposition based MS-SLP solution algorithms and different representations of the nonanticipativity constraints. An alternative formulation of MS-SLP, in which the nonanticipativity constraints are expressed implicitly, is given by

$$\begin{aligned} \text{minimize} \quad & c_1^\top x_1 + E[h_2(x_1, \tilde{\underline{\omega}}_2)] \\ \text{subject to} \quad & A_1 x_1 = b_1, \\ & x_1 \geq 0, \end{aligned} \tag{MS-SLP^{Stg}}$$

where, $\forall t = 2, ..., T-1$,

$$\begin{aligned} h_t(x_{t-1}, \underline{\omega}_t) = \text{minimize} \quad & c_t(\underline{\omega}_t)^\top x_t + E[h_{t+1}(x_t, \underline{\omega}_{t+1}) | \tilde{\underline{\omega}}_t = \underline{\omega}_t] \\ \text{subject to} \quad & W_t(\underline{\omega}_t) x_t = r_t(\underline{\omega}_t) - T_t(\underline{\omega}_t) x_{t-1}, \\ & x_t \geq 0; \end{aligned}$$

and,

$$\begin{aligned} h_T(x_{T-1}, \underline{\omega}_T) = \text{minimize} \quad & c_T(\underline{\omega}_T)^\top x_T \\ \text{subject to} \quad & W_T(\underline{\omega}_T) x_T = r_T(\underline{\omega}_T) - T_T(\underline{\omega}_T) x_{T-1}, \\ & x_T \geq 0. \end{aligned}$$

This formulation suggests decomposition of MS-SLP by stage. MS-SLP algorithms based on temporal decomposition are extensions of the concepts for solving two-stage

stochastic linear programs (TS-SLP). Thus, we begin our discussion with methods for TS-SLP.

2.3.1 Two-stage Stochastic Linear Program

When there are only two stages, model (MS-SLP^{Stg}) becomes

$$\begin{aligned} \text{minimize} \quad & c_1^\top x_1 + E[h_2(x_1, \tilde{\omega}_2)] \\ \text{subject to} \quad & A_1 x_1 = b_1, \\ & x_1 \geq 0, \end{aligned} \tag{TS-SLP}$$

where,

$$\begin{aligned} h_2(x_1, \omega_2) = \text{minimize} \quad & c_2(\omega_2)^\top x_2 \\ \text{subject to} \quad & W_2(\omega_2) x_2 = r_2(\omega_2) - T_2(\omega_2) x_1, \\ & x_2 \geq 0. \end{aligned}$$

Note that if we denote $H(x) = E[h_2(x, \tilde{\omega}_2)]$, then (TS-SLP) can be equivalently written as

$$\begin{aligned} \text{minimize} \quad & c_1^\top x_1 + \eta \\ \text{subject to} \quad & A_1 x_1 = b_1, \\ & x_1 \geq 0, \\ & H(x_1) - \eta \leq 0. \end{aligned} \tag{2.32}$$

$H(x)$ is referred to as the recourse function of TS-SLP.

Benders' Decomposition

The major difficulties associated with solving (2.32) are:

1. There is no closed form representation of $H(x)$. As a result, subgradient information is not readily available, which hinders the application of any subgradient-based algorithms;

2. The number of realizations of $\tilde{\omega}$ is usually large, which makes it hard, or even impossible, to evaluate $H(x)$. As a result, the size of the deterministic equivalent problem (DEP) of TS-SLP can become unmanageable.

Benders' decomposition (Benders [1962], a version of *Kelly's cutting plane method* (Kelly [1960]), which is referred to as *the L-shaped method* (Van Slyke and Wets [1969]) when applied to TS-SLP) uses a piecewise linear function to create an outer approximation of $H(x)$. It identifies the optimal solution by iteratively adding constraints (cuts) that update the approximation.

Benders' decomposition iterates between two problems: an approximate master problem of the form

$$\begin{aligned} \text{minimize} \quad & c_1^\top x_1 + \eta && \text{(TS-SLP}^{mast}\text{)} \\ \text{subject to} \quad & A_1 x_1 = b_1, \\ & D_i x_1 \geq d_i, \quad i = 1, ..., p, && (2.33) \\ & E_i x_1 + \eta \geq e_i, \quad i = 1, ..., q, && (2.34) \\ & x_1 \geq 0; \end{aligned}$$

and the set of scenario subproblems of the form

$$\begin{aligned} h_2(x_1, \omega_2) = \text{minimize} \quad & c_2(\omega_2)^\top x_2 && \text{(TS-SLP}^{sub}\text{)} \\ \text{subject to} \quad & W_2(\omega_2) x_2 = r_2(\omega_2) - T_2(\omega_2) x_1, && (2.35a) \\ & x_2 \geq 0. && (2.35b) \end{aligned}$$

In each iteration, (TS-SLP^{mast}) is solved to obtain a candidate solution $\hat{x}_1$ and an optimistic estimator (lower bound) on the optimal function value of $c_1^\top x_1 + H(x_1)$, $c_1^\top \hat{x}_1 + \hat{\eta}$. For each $\omega_2 \in \Omega_2$, the subproblem corresponding to $h_2(\hat{x}_1, \omega_2)$ is solved. If a subproblem is infeasible, an unbounded direction of its dual problem is obtained and a *feasibility cut* of the form (2.33) is generated and added to (TS-SLP^{mast}). Note that in cases where the TS-SLP problem has *relatively complete recourse* (i.e., (TS-SLP^{sub}) is feasible for all $(x_1, \omega_2) \in X_1 \times \Omega_2$) the potential need for

feasibility cuts is eliminated. If all subproblems are feasible, a pessimistic estimator (upper bound) of the optimal function value of $c_1^\top x_1 + H(x_1)$ is obtained as $c_1^\top \hat{x}_1 + H(\hat{x}_1) = c_1^\top \hat{x}_1 + E[h_2(\hat{x}_1, \tilde{\omega}_2)]$. If $\hat{\eta} = H(\hat{x}_1)$ (optimality criterion), the algorithm terminates with $\hat{x}_1$ as the optimal solution and $f(\hat{x}_1) = c_1^\top \hat{x}_1 + H(\hat{x}_1)$ as the optimal objective function value. Otherwise, an *optimality cut* of the form (2.34) is added to (TS-SLP^{mast}).

Benders' decomposition algorithm can be interpreted as a dual version of the Dantzig-Wolfe Decomposition algorithm (Danzig and Wolfe [1961]), as demostrated by Van Slyke and Wets [1969]. It converges to the optimal solution in a finite number of iterations, but there are computational difficulties associated with it, such as the need to solve a large (possibly unmanageable) number of second-stage scenario subproblems in each iteration.

A Multicut Algorithm

Following the idea in Ruszczyński [1986] (described in the subsequent section), Birge and Louveaux [1988] propose a *multicut algorithm* to reduce the number of iterations before convergence to optimality. The basic idea is to replace the piecewise linear outer approximation of $H(x_1)$ used in the L-shaped method by outer linearizations of $h_2(x_1, \omega_2)$ for all $\omega_2 \in \Omega_2$. The logic behind this approach is that using outer approximations of all $h_2(x_1, \omega_2)$ sends more information than a single cut on $H(x_1)$ in one iteration and that, therefore, fewer iterations are needed. Limited computational results with this method are provided in the literature.

When the number of possible scenarios is large, the size of the master problem can grow very fast. Although some slack cuts may be dropped and re-enter the master problem when needed later, the manner in which the size of the master problem grows appears to be a major concern.

Regularized Decomposition

Besides a proposed multicut approach, in the *regularized decomposition method* by Ruszczyński [1986], the master program is augmented with a quadratic regularizing term (a penalty term to prevent the solution from deviating from the current *incumbent solution* (the best solution so far) too far without observing significant improvement on the objective function value). This helps to 1) stabilize the approximate master problem; 2) keep the number of cuts limited to an a priori bound; and 3) enable warm-start of the algorithm from a given "close" solution, i.e., because of the quadratic term, the solution will not deviate from the given "close" solution too much unless a significant improvement on the objective value is observed.

Finite convergence of the method to the optimal solution is proven without any non-degeneracy assumptions. No more than $n_1 + 2|S|$ cuts need to be stored, where n_1 is the size of first-stage decision vector and $|S|$ is the number of second-stage scenarios.

An *active set* based quadratic programming algorithm is proposed to solve the regularized master problem. It is essentially a feasible direction method for solving the dual of the regularized master problem. If the primal master problem is feasible, the algorithm finds the optimal solution after a finite number of iterations, since the number of possible active sets is finite.

Computational experiments, designed to obtain some "qualitative" conclusions on the behavior of the method, have been conducted. The results provide initial evidence of the numerical efficiency and stability of the method. In addition, an experiment is conducted to compare the multicut approach (approximating each subproblem objective function separately) and the single-cut approach (constructing aggregate cutting planes for the expected subproblem objective function). The results reported in the paper are in favor of the multicut approach.

Improvements to the implementation of the regularized decomposition algorithm have been reported by Ruszczyński [1993] and Ruszczyński and Świętanowski [1997]. The improvements include: 1) solving the master problem based on dynamic selec-

tion of *critical scenarios* to control the size of master problem; 2) using a penalty-based primal simplex method that allows quick hot starts and re-optimization to take advantage of the similar structure among subproblems in their solution process; and 3) using advanced starting points generated by a crash procedure.

Stochastic Decomposition

The dual of (TS-SLP^{sub}) may be written as

$$\begin{aligned} h_2(x_1,\omega_2) = \text{maximize} \quad & \pi^\top(r_2(\omega_2) - T_2(\omega_2)x_1) \\ \text{subject to} \quad & \pi^\top W_2(\omega_2) \le c_2(\omega_2), \end{aligned} \qquad \text{(TS-SLP}^{sub-dual}\text{)}$$

where π denotes the dual multipliers associated with the constraints (2.35a) in (TS-SLP^{sub}).

When the TS-SLP problem has fix recourse (i.e., $W_2(\omega_2) = W_2$ and $c_2(\omega_2) = c_2$), (TS-SLP$^{sub-dual}$) becomes

$$\begin{aligned} h(x_1,\omega_2) = \text{maximize} \quad & \pi^\top(r_2(\omega_2) - T_2(\omega_2)x_1) \\ \text{subject to} \quad & \pi^\top W_2 \le c_2. \end{aligned} \qquad (2.36)$$

That is, the dual feasible set $\Pi = \{\pi | \pi^\top W_2 \le c_2\}$ is common for all $\omega_2 \in \Omega_2$. Then for any $\pi \in \Pi$ and for any $x_1 \in X_1$, according to the theorem of weak duality, we have

$$\pi^\top(r_2(\tilde{\omega}_2) - T_2(\tilde{\omega}_2)x_1) \le h_2(x_1,\tilde{\omega}_2), \quad w.p.1. \qquad (2.37)$$

Hence, the dual solutions to (2.36) can be used to obtain lower bounds on the recourse function $H(x_1) = E[h_2(x_1,\tilde{\omega}_2)]$.

Higle and Sen [1991] exploit this property with the *stochastic decomposition* (SD) algorithm. Let V_k be a subset of the dual vertices that have been collected by the algorithm in the first k iterations, and let

$$\pi_m^k \in \arg\max\{\pi^\top(r_2(\omega_2^m) - T(\omega_2^m)x_1^k) | \pi \in V_k\}, \qquad (2.38)$$

where $\omega_2^m, m = 1, ..., k$ are the observations of $\tilde{\omega}_2$ collected by the algorithm so far. Then the k^{th} cutting plane can be constructed as

$$\alpha_k^k + (\beta_k^k)^\top x_1 = \frac{1}{k}\sum_{m=1}^{k}(\pi_m^k)^\top(r_2(\omega_2^m) - T_2(\omega_2^m)x_1^k)) \leq \frac{1}{k}\sum_{m=1}^{k} h_2(x_1, \omega_2^m) \qquad (2.39)$$

In Benders' decomposition, the evaluation of $H(x_1)$ requires solving (TS-SLP^{sub}) for all $\omega_2 \in \Omega_2$, which imposes a heavy computational burden when the sample space is large. The stochastic decomposition algorithm proposes an approach to alleviate this burden: in each iteration, SD generates one observation (a realization of $\tilde{\omega}_2$) and solves at most two LPs with the newly-generated observation (with the current candidate solution x_1^k and the current incumbent solution $\bar{x}_1^{k-1}$). The dual vertices obtained, $\pi(x_1^k, \omega_2^k)$ and $\pi(\bar{x}_1^{k-1}, \omega_2^k)$, are used to update V_k, $V_k = V_{k-1} \cup \{\pi(x_1^k, \omega_2^k), \pi(\bar{x}_1^{k-1}, \omega_2^k)\}$. The cut corresponding to the current set of observations, $\{\omega_2^m\}_{m=1}^k$, are generated using (2.38) and (2.39). The coefficients of previously generated cuts are updated dynamically to reflect the changing sample size.

Higle and Sen [1994] adopt the idea in *regularized decomposition* (Ruszczyński [1986] and regularize the master problem with a quadratic term to stabilize the solution process and to keep the size of the master problem under control (at most n_1+3 cuts are needed to ensure convergence to optimality, where n_1 is the dimension of x_1).

For all sampling-based algorithms, certain optimality tests are necessary to investigate to what extent the apparent quality of a proposed solution is sample-dependent. Higle and Sen [1996b] introduce an "in-sample" optimality test based on the *bootstrap method* (Efron [1979]). The test requires the solution of numerous bootstrapped LPs, which poses an obvious drawback if the test must be undertaken repeatedly.

Based on the duality analysis of the regularized version of SD, Higle and Sen [1999] develop a valid but potentially loose lower bound on $\hat{H}_m^k(\bar{x}_1^k)$, where $\hat{H}_m^k(\bar{x}_1^k)$ is the approximation of $H(x_1^k)$ in the m^{th} bootstrapped master problem in iteration k. In this way, the optimality test does not rely on solving numerous bootstrapped LPs, but instead calculates the lower bounds on the objective function values of

the bootstrapped master problems. We will discuss the optimality tests of the SD algorithm in detail in section 3.2 of this dissertation.

Conditional Stochastic Decomposition (CSD Higle et al. [1994]) is a multicut variation of the SD algorithm. While SD solves at most two subproblems per iteration, CSD solves subproblems for all observations of $\tilde{\omega}_2$ that are available so far. To justify the additional effort of solving more subproblems, CSD is only preferable to SD when the cost of generating new observations is high.

Cutting-Plane and Partial-Sampling Algorithm (CUPPS)

In iteration k, unlike Benders' decomposition, which solves subproblems for all $|S|$ realizations of $\tilde{\omega}_2$, and stochastic decomposition, which solves at most two subproblems to generate and accumulate new dual vertices and then constructs the new cut based on the k observations so far, the CUPPS algorithm (Chen and Powell [1999]) solves one subproblem for the new realization of $\tilde{\omega}_2$, then constructs the new cut based on all $\omega_2 \in \Omega_2$ as in

$$\alpha^k + (\beta^k)^\top x_1 = \sum_{s=1}^{|S_2|} p^s \pi^{s\top}(r_2(\omega_2^s) - T_2(\omega_2^s)x_1^k)) \leq \sum_{s=1}^{|S_2|} p^s h_2(x_1, \omega_2^s), \tag{2.40}$$

with π^s obtained as in (2.38).

To briefly summarize these three cutting plane approximations for TS-SLP: SD cuts provide statistically valid lower bounds on the objective function value, although the cuts so generated may not be valid in a finite number of iterations. CUPPS generates valid cuts in each iteration, but the cuts may not necessarily be tight. Benders' decomposition generates tight and valid cuts in each iteration. The computational effort associated with these three algorithms increases as the cuts become more accurate.

2.3.2 Multistage Stochastic Linear Program

Nested Decomposition Method

The *nested decomposition method* (Birge [1985]) employs an outer linearization decomposition strategy that extends the two-stage L-shaped method of Van Slyke and Wets [1969]. The two-stage L-shaped method (a.k.a. Benders' decomposition) approximates the convex recourse function in (TS-SLP) by successively appending supporting hyperplanes. In MS-SLP, the supporting hyperplanes are found by optimizing a sequence of nested subproblems of the same form, one for each node in the scenario tree. Each subproblem, as in (MS-SLP^{Stg}) has a linear objective term, $c_t^\top x_t$, for the cost of the current stage and a convex objective term, $H_{t+1}(x_t) = E[h_{t+1}(x_t, \underline{\omega}_{t+1})|\underline{\tilde{\omega}}_t = \underline{\omega}_t]$, which represents the future cost, with a set of constraints including the possible effect of past decisions.

Similar to Benders' decomposition, $H_{t+1}(x_t)$ can be replaced by an artificial variable, η_t and a set of constraints, as in (2.41).

$$
\begin{aligned}
\text{minimize} \quad & c_t^\top x_t + \eta_t && && && (2.41a)\\
\text{subject to} \quad & W_t x_t && = r_t(\omega_t) - T_t(\omega_t) x_{t-1} && && (2.41b)\\
& D_t^i x_t && \geq d_t^i, && i = 1, ..., p, && (2.41c)\\
& E_t^i x_t + \eta_t && \geq e_t^i, && i = 1, ..., q, && (2.41d)\\
& x_t && \geq 0. && && (2.41e)
\end{aligned}
$$

Feasibility cuts (2.41c) are used to induce feasible solutions at stage $t+1$. Optimality cuts (2.41d) reflect the information we gained from solving the problems of descendant (future) nodes that we use to establish the optimality condition of the problem of the current node.

Program (2.41) is initially solved by forming a relaxed problem excluding (2.41c) and (2.41d) and by successively adding constraints (2.41c) and (2.41d) until a solution, (x_t^*, η_t^*), of the relaxed problem is found that satisfies the optimality condition,

i.e., $H_{t+1}(x_t^*) \leq \eta_t^*$. In nested decomposition, program (2.41) must be solved for *every* node in the scenario tree at each iteration.

A *forward pass* is performed until a feasibility cut is sent back to the first stage subproblem or a feasible solution is found for each stage T subproblem. A *backward pass* involves solving the subproblems (2.41) recursively until the optimality condition is satisfied at each node in the scenario tree.

When implementing the nested decomposition, three sequencing protocols can be used to decide the order in which subproblems are solved. After a stage t subproblem is solved, one may choose to go forward to descendant nodes, or to go backward to the parent node, according to the following protocols:

1. Fast-Forward-Fast-Backward (FFFB): change directions as little as possible, i.e., the algorithm continues to move in the same direction until a move in that direction is blocked;

2. Forwards First (FF): the algorithm only moves to stage $t-1$ when *all* current solutions for stage t through T are optimal;

3. Backward First(BF): the algorithm always move back to stage $t-1$ until no new cuts to that stage are generated.

MSLiP (Gassmann [1990]) implements the nested decomposition algorithm and adapts many computational tricks that are developed for deterministic staircase problems. The computational results are in favor of the FFFB protocol. A parallel implementation of the nested decomposition algorithm is reported in Birge et al. [1996].

Sample Average Approximation

With the exception of SSD, SD, CSD, and CUPPS, all of the methods discussed thus far are deterministic in that they are based on the entire sample space and their solutions are deterministically verifiable. As the sample space increases, the application of such these deterministic algorithms becomes impractical. A combination

of mathematical programming techniques and statistically motivated approximation techniques is an intuitive and common approach for solving large scale stochastic programs.

For example, let $\omega^1, \omega^2, ..., \omega^N$ be a random sample of $\tilde{\omega}$. An estimate for $E[h(x, \tilde{\omega})]$ can be obtained from the sample mean,

$$\hat{H}_N(x) = \frac{1}{N} \sum_{i=1}^{N} h(x, \omega^i).$$

This leads to an approximating problem

$$\begin{aligned} \text{minimize} \quad & c^\top x + \hat{H}_N(x) \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{2.42}$$

Over the years, this approximation has been referred to as *"stochastic counterpart method"* (Shapiro [1991]), *"sample mean optimization"* (Higle and Sen [1996a]), *"sample-path optimization"* (Plambeck et al. [1996]), and *"sample average approximation"* (Shapiro [2000], Kleywegt et al. [2001]). Throughout this discussion, we refer to it as sample average approximation (SAA).

Let $\hat{x}_N$ denote the optimal solution to (2.42). Define

- $\hat{f}_N(\hat{x}_N) = c^\top \hat{x}_N + \hat{H}_N(\hat{x}_N)$, the objective function value estimate at $\hat{x}_N$ based on the current sample of size N;
- $\hat{f}_{N'}(\hat{x}_N) = c^\top \hat{x}_N + \hat{H}_{N'}(\hat{x}_N)$, the objective function value estimate at $\hat{x}_N$ based on an independent sample of size N'.
- $f(\hat{x}_N) = c^\top \hat{x}_N + H(\hat{x}_N)$, the true objective function value evaluated at $\hat{x}_N$.

Mak et al. [1999] establish that $E[\hat{f}_N(\hat{x}_N)] \leq f(x^*) \leq E[f(\hat{x}_N)]$, where $f(x^*)$ is the optimal objective function value. A *batch-means sampling procedure* is proposed to develop confidence intervals on the optimality gap to any candidate solution, $\hat{x}$. A common random numbers (CRN) sampling procedure is also proposed to ensure nonnegative gap estimates and for reduction on the variance of the gap estimates.

In Kleywegt et al. [2001], the selection of sample size N to obtain a candidate solution, $\hat{x}$, is discussed in detail. As a result, a lower bound on N to find an ϵ-optimal solution with probability at least $1-\alpha$ is derived. Two issues remain: 1) for many problems N is not easy to compute; 2) the lower bound on N may be too conservative to provide practical computational guidance. Therefore, in practice, we face the tradeoff between computational effort and accuracy of the objective function estimate.

When faced with solving the SAA problem when the sample size, N, is large, Kleywegt et al. [2001] propose solving M replications of SAA problems with smaller sample sizes to obtain a collection of candidate solutions, $\{\hat{x}^m\}_{m=1}^M$. Estimated objective function values $\{\hat{f}_{N'}^m(\hat{x}^m)\}_{m=1}^M$ derived from an independent sample of size N' can be used to estimate the optimality gap and the variance of the optimality gap estimator. If the optimal gap and the variance of the optimality gap estimator are sufficiently small, a screening and selection procedure is used to choose the best solution among all candidate solutions $\{\hat{x}^m\}_{m=1}^M$; Otherwise, increase N and/or N', and repeat.

Linderoth et al. [2002] implement the SAA algorithm on a computational grid. An *asynchronous bundle-trust-region method* (ATR, extensively discussed in Linderoth and Wright [2002]) based on multicut L-shaped method is implemented to enable parallel computing and to improve efficiency.

The computational efficiency of the SAA framework will be discussed in more details in Chapter 3 of this dissertation.

CHAPTER 3

Computational Experiment with Termination Criteria on Two-stage SLP Methods

3.1 Introduction

As described in Chapter 2, a two-stage stochastic linear program with recourse may be stated as

$$\begin{aligned} \text{Minimize} \quad & f(x) = c^\top x + E[h(x, \tilde{\omega})] && \text{(TS-SLP)} \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0, \end{aligned}$$

where $\tilde{\omega}$ is a multivariate random variable with distribution P and sample space Ω, which models uncertainty in the second-stage data. For each $\omega \in \Omega$,

$$\begin{aligned} h(x, \omega) = \text{Minimize} \quad & g(\omega)^\top y && \text{(Sub-P)} \\ \text{subject to} \quad & W(\omega)y = r(\omega) - T(\omega)x, && \text{(3.1a)} \\ & y \geq 0. && \text{(3.1b)} \end{aligned}$$

For simplicity, in this chapter, we use x to denote the first-stage solution and y to denote the second-stage solution.

The dual representation of (Sub-P) is

$$\begin{aligned} h(x, \omega) = \quad \text{Maximize} \quad & \pi^\top [r(\omega) - T(\omega)x] && (3.2) \\ \text{subject to} \quad & \pi^\top W(\omega) \leq g(\omega), \end{aligned}$$

where π is the vector of dual variables of (3.1a). In the discussion of this chapter, we assume that $W(\omega) \equiv W$ and $g(\omega) \equiv g$.

Calculating the objective function value $f(x)$ associated with any particular x, requires an evaluation of $h(x, \omega)$ for all $\omega \in \Omega$. When Ω is a large set, this

can become burdensome, if not impossible. Stochastic Decomposition (SD), which avoids the exact calculation of objective function values, is an efficient sample based algorithm for solving (TS-SLP). It is first proposed in Higle and Sen [1991] and has been studied and improved throughout the years (e.g. Higle and Sen [1994, 1996b, 1999]). In this chapter, we report some recent computational experimentation with regard to the optimality tests employed in this method.

3.2 Termination of the algorithms

In the k^{th} iteration, the SD master program has the form:

$$\begin{aligned} \text{Minimize} \quad & f(x) = c^\top x + \eta && (\text{LM}^k) \\ \text{subject to} \quad & Ax = b, && (3.3a) \\ & \eta \geq \alpha_t^k + \beta_t^{k\top} x, \quad t \in J_k, && (3.3b) \\ & x \geq 0, \end{aligned}$$

where $J_k \subset \{1, ..., k\}$ is the index set of cuts retained in the k^{th} iteration. The k^{th} cutting plane in (3.3b) is constructed as in (2.39).

In its regularized form, the master program is

$$\begin{aligned} \text{Minimize} \quad & f(x) = c^\top x + \eta + \frac{\sigma}{2}\|x - \bar{x}^k\|^2 && (\text{RM}^k) \\ \text{subject to} \quad & Ax = b, \\ & \eta \geq \alpha_t^k + \beta_t^{k\top} x, \quad t \in J_k, \\ & x \geq 0, \end{aligned}$$

where $\bar{x}^k$ is the incumbent solution in the k^{th} iteration.

The SD method incorporates sampling within Benders' decomposition. As a result, it is necessary to use stopping rules based on criteria that differ from its deterministic counterpart. For example, it is necessary to investigate to what extent the apparent quality of a solution is sample-dependent.

3.2.1 Simple Stopping Rule

Traditionally, algorithms that solve convex programs via the construction of an outer linearization of the objective function (e.g. a cutting plane approximation) terminate when the difference between the objective value at a given iterate (upper bound) and a known lower bound on the optimal objective value is sufficiently small. As a cutting plane algorithm, upper and lower bounds are available in SD. However, these bounds are statistical estimates of upper and lower bounds, and are therefore sample dependent.

In SD, the upper bound estimate in iteration k is obtained by evaluating

$$u_k = f_k(\bar{x}^k) = c\bar{x}^k + \nu_k(\bar{x}^k), \tag{3.5}$$

and the lower bound in iteration k is obtained as

$$\begin{aligned} l_k = f_k(x^{k+1}) = \min \quad & f_k(x) = cx + \nu_k(x) \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{3.6}$$

where

$$\nu_k(x) = \max\{\alpha_t^k + \beta_t^{k\top} x | t \in J_k\}. \tag{3.7}$$

We may consider terminating the SD algorithm in iteration k if the difference between these values is sufficiently small. That is, if

$$f_k(x^{k+1}) > (1 - \epsilon_{smpl}) f_k(\bar{x}^k), \tag{3.8}$$

the algorithm is terminated, where $\epsilon_{smpl} \in (0, 1)$ is a predefined error tolerance.

As simple as it is, this test fails to account for the variability of the solution due to sampling, and it should not be considered an appropriate termination criterion. However, it can serve as a "pre-test" before a full optimality test is undertaken. That is, when the difference between these bound values is not small enough, the computational expense of a full scale optimality test is unlikely to be worthwhile.

3.2.2 Full Stopping Rules

Linear SD (Higle and Sen [1991])

The variability of the objective function approximation, f_k, and therefore the estimated error bound, $u_k - l_k$, are directly attributable to the variability of the cut coefficients, as in (3.7), which in turn derive from the sampled observations. We shall study the variability of the error bounds by replicating the cut coefficients from which these bounds are derived.

The "bootstrap" method of Efron [1979] provides a means for estimating the variability of an estimator without requiring the use of additional samples. Although the details of the bootstrap based optimality test for SD can be found in Higle and Sen [1996b, 1999], we summarize the basic process here:

Suppose we wish to test a primal incumbent solution, $\bar{x}^k$, in iteration k. In SD, we have k observations of $\tilde{\omega}$, $\{\omega^t\}_{t=1}^k$. To sample from the empirical distribution $\{\omega^t\}_{t=1}^k$, we simply sample from the index set $\{1, ..., k\}$ with replacement. We label this sample as $\{t(i)\}_{i=1}^k$ and the bootstrapped sample consists of the observations $\{\omega^{t(i)}\}_{i=1}^k = (r(\omega^{t(i)}), T(\omega^{t(i)}))_{i=1}^k$.

For a cut indexed by $j \leq k$, let π_t^j denote the subproblem dual vertex associated with the observation $\omega^t, t \leq j$. The cut coefficients of the re-sampled cut are

$$\hat{\alpha}_t^k = \frac{1}{k} \sum_{\{i|t(i)\leq t, i\leq k\}} (\pi_{t(i)}^t)^\top r(\omega^{t(i)}) \tag{3.9a}$$

and

$$\hat{\beta}_t^k = -\frac{1}{k} \sum_{\{i|t(i)\leq t, i\leq k\}} (T(\omega^{t(i)}))^\top (\pi_{t(i)}^t) \tag{3.9b}$$

With the replicated cut coefficients as in (3.9), denoted by $\{(\hat{\alpha}_t^k, \hat{\beta}_t^k)\}_{t\in J_k}$, and

$$\hat{\nu}_k(x) = \max\{\hat{\alpha}_t^k + \hat{\beta}_t^{k\top} x | t \in J_k\}. \tag{3.10}$$

we can calculate the upper bound, $\hat{u}^{lsd}$, by evaluating (3.5), and solve the linear program (3.6), for lower bound, $\hat{l}^{lsd}$, using the reformulated master problem. If the

so-obtained error bound satisfies $(\hat{u}^{lsd} - \hat{l}^{lsd})/\hat{u}^{lsd} \leq \epsilon_{full}$ (for a small enough error tolerance ϵ_{full}), the solution appears to be acceptable within this replication.

This process can be repeated to obtain M independent bootstrapped estimates of the error bound.

For $i = 1, ..., M$

- Obtain $\hat{l}_i^{lsd} = \min\{c^\top x + \hat{\nu}_{k,i}(x) | Ax = b, x \leq 0.\}$
- $\hat{u}_i^{lsd} = c^\top \bar{x}^k + \hat{\nu}_{k,i}(\bar{x}^k)$.
- $\hat{e}_i^{lsd} = \hat{u}_i^{lsd} - \hat{l}_i^{lsd}$.

If for a sufficiently large number of these bootstrapped estimates, a significantly large portion (for example, 95%) of them are acceptable, we then conclude that the current solution passes the full stopping rule and terminate the SD algorithm with the current incumbent solution $\bar{x}^k$ as the optimal solution.

Regularized SD (Higle and Sen [1999])

For the simplicity of notations, we rewrite the feasible region $\{x | Ax = b, x \geq 0\}$ as $\{x | A'x \geq b.\}$ and denote it as X. We can rewrite (RM^k) in terms of the direction of change d from the incumbent $\bar{x}^k$ (i.e., $x = \bar{x}^k + d$) as

$$\text{Minimize} \quad f(d) = c^\top d + \eta + \frac{\sigma}{2}||d||^2 \qquad (\text{PRM}^k)$$

$$\text{subject to} \quad \bar{x}^k + d \in X, \qquad (3.11a)$$

$$-{\beta_t^k}^\top d + \eta \geq v_t^k, \quad t \in J_k, \qquad (3.11b)$$

where $v_t^k = \alpha_t^k + {\beta_t^k}^\top \bar{x}^k$. Note that the constant term $c^\top \bar{x}^k$ is dropped from the objective function in (PRM^k).

Let π and θ denote the dual vectors of (3.11a) and (3.11b). Let V_k denote the vector of scalars $\{v_t^k\}_{t \in J_k}$, and let B_k denote the matrix whose rows are given by the cut coefficients $\{\beta_t^{k\top}\}_{t \in J_k}$. Furthermore, let b_k denote the vector $A'\bar{x}^k - b$, the

necessary condition for (PRM^k) are given by

$$A'd + b_k \geq 0, \tag{3.12a}$$

$$-e\eta + V_k + B_k^\top d \leq 0, \tag{3.12b}$$

$$\pi^\top (A'd + b_k) = 0, \tag{3.12c}$$

$$\theta^\top (-e\eta + B_k^\top d + V_k) = 0, \tag{3.12d}$$

$$c + B_k^\top \theta + A'^\top \pi + \sigma d = 0, \tag{3.12e}$$

$$e^\top \theta = 1, \tag{3.12f}$$

$$\theta \geq 0, \quad \pi \leq 0 \tag{3.12g}$$

Note that when (3.12e) is satisfied, we have

$$c + B_k^\top \theta + A'^\top \pi = -\sigma d. \tag{3.13}$$

In Higle and Sen [1994], the dual to (PRM^k) is suggested as

$$\begin{aligned} \text{Maximize} \quad & V_k^\top \theta + b_k^\top \pi - \frac{1}{2\sigma}||c + B_k^\top \theta + A'^\top \pi||^2 \\ & e^\top \theta = 1, \theta \geq 0, \pi \leq 0. \end{aligned} \tag{DRM^k}$$

The necessary condition for (DRM^k) are given by

$$-\frac{1}{\sigma} A'(c + B_k^\top \theta + A'^\top \pi) + b_k \geq 0, \tag{3.14a}$$

$$-e\eta + V_k - \frac{1}{\sigma} B_k^\top (c + B_k^\top \theta + A'^\top \pi) \leq 0, \tag{3.14b}$$

$$\pi^\top (-\frac{1}{\sigma} A'(c + B_k^\top \theta + A'^\top \pi) + b_k) = 0, \tag{3.14c}$$

$$\theta^\top (-e\eta + V_k - \frac{1}{\sigma} B_k^\top (c + B_k^\top \theta + A'^\top \pi)) = 0, \tag{3.14d}$$

$$e^\top \theta = 1, \tag{3.14e}$$

$$\theta \geq 0, \quad \pi \leq 0. \tag{3.14f}$$

With (3.13), the necessary conditions for (PRM^k) and (DRM^k) are equivalent.

Note also from (3.12c) and (3.12e), we have

$$\begin{aligned}\pi^\top b_k &= -\pi^\top A' d \\ &= (c + B_k^\top \theta + \sigma d)^\top d \\ &= c^\top d + \sigma ||d||^2 + \theta^\top B_k d. \end{aligned} \tag{3.15}$$

Together with (3.12d) and (3.12f), the objective function of (DRM^k) is

$$\begin{aligned} & V_k^\top \theta + b_k^\top \pi - \frac{1}{2\sigma}||c + B_k^\top \theta + A'^\top \pi||^2 \\ &= \theta^\top V_k + c^\top d + \sigma ||d||^2 + \theta^\top B_k d - \frac{\sigma}{2}||d||^2 \\ &= c^\top d + \frac{\sigma}{2}||d||^2 + \theta^\top (V_k + B_k d) \\ &= c^\top d + \frac{\sigma}{2}||d||^2 + \theta^\top (e\eta) \\ &= c^\top d + \frac{\sigma}{2}||d||^2 + \eta. \end{aligned} \tag{3.16}$$

The optimal objective values of (PRM^k) and (DRM^k) are equal.

With regularized SD, we have an opportunity for estimating a lower bound on the objective function value without solving the linear program suggested by (3.6) and (3.7). Suppose that in iteration k, we are to test the primal incumbent, $\bar{x}^k$, and the corresponding dual solutions, (π^k, θ^k), for optimality. The same bootstrap procedure as in linear SD is used to re-sample the cuts. With the resulting replicated cut coefficients, $\{(\hat{\alpha}_t^k, \hat{\beta}_t^k)\}_{t \in J_k}$, we are able to formulate the replicated primal and dual problems, denoted as $\hat{P}^k$ and $\hat{D}^k$, with the same form as (PRM^k) and (DRM^k).

The upper bound on the optimal value of $\hat{P}^k$ is, similar to (3.7),

$$\hat{u}^{rsd} = \max_{t \in J_k} \{\hat{\alpha}_t^k + \hat{\beta}_t^{k\top} \bar{x}^k\}. \tag{3.17}$$

Let $\hat{V}_k$ denote the vector of scalars $\{\hat{v}_t^k = \hat{\alpha}_t^k + \hat{\beta}_t^{k\top} \bar{x}^k\}_{t \in J_k}$ and $\hat{B}_k$ denote the matrix whose rows are given by the cut coefficients $\{-\hat{\beta}_t^k\}_{t \in J_k}$. The dual objective evaluated at (π^k, θ^k) is

$$\hat{l}^{rsd} = \hat{V}_k^\top \theta^K + b_k^\top \pi^k - \frac{1}{2\sigma}||c + \hat{B}_k^\top \theta^k + A'^\top \pi^k||^2. \tag{3.18}$$

Since (π^k, θ^k) is feasible for $\hat{D}^k$, it follows that $\hat{l}^{rsd}$ is a lower bound on its optimal value, and therefore a lower bound on the optimal value of $\hat{P}^k$ (Higle and Sen [1999]). The error bounds ($\hat{e}^{rsd} = \hat{u}^{rsd} - \hat{l}^{rsd}$) described in this subsection can be implemented without solving any linear/quadratic programs: the calculations only require function evaluations as in (3.17, 3.18).

Note that the primary differences between the error bound estimates of the two SD methods (i.e. linear SD and regularized SD) are two fold – the effort required to calculate them and the potential quality of the bounds. Both $\hat{l}^{lsd}$ and $\hat{l}^{rsd}$ offer insight into the manner in which the lower bound estimate might vary if it were estimated via a different set of observations – that is the intent behind all bootstrap methods. The former, $\hat{l}^{lsd}$, looks directly at the impact of the variability of the cutting plane coefficients on the master program objective value. That is, because the bootstrapped master program is solved to optimality, it is a tight estimate of the lower bound (relative to the bootstrapped representation of the cutting plane coefficients). However, the need to repetitively solve the master program can become computationally burdensome. On the other hand, $\hat{l}^{rsd}$ looks at the impact of the variability of the cutting plane coefficients on the optimal dual solution to the regularized master program. The dual solution, (π^k, θ^k) remains dual feasible – but is not necessarily optimal. It follows that the lower bound that is calculated via the regularized master problem is potentially looser than that provided via the linear master, although the lower bound calculation in (3.18) is clearly easier than solving the bootstrapped linear master problem.

In order to explore the relative computational requirements of the two SD methods and their impact on the solution quality, computational experimentation is undertaken, as described in the following section.

3.3 Computational Investigation: SD Optimality Tests

Our initial computational experiment is intended to investigate the relative advantages of the two forms of the SD method and their associated optimality tests. These

tests are statistical in nature, and as such their accuracy warrants investigation.

We begin by exploring the quality of the termination criteria used by the SD methods. In what follows, we refer to SD with the linear master program as LSD and SD with the regularized master program as RSD. Asymptotic results are not dependent on the form of the master program used – both methods identify an optimal solution with probability one (see Higle and Sen [1991] and Higle and Sen [1994]). However, one expects that the form of the master program used will impact the sequence of points visited. Moreover, the differences in the optimality tests used in conjunction with the two master programs will impact the quality of the solution that is ultimately identified as well as the computational effort required to identify it.

3.3.1 Data

A collection of problem instances, most of which are well referenced in the stochastic programming literature and publicly available, are used. These problems are characterized as two-stage stochastic linear programs with general and complete recourse. The specific problems that we include in our computational experiment are listed in Table 3.1.

	First Stage		Second Stage			
Problem	rows	columns	rows	columns	No. of rvs	No. of outcomes
PGP2	2	4	7	12	3	576
SSN	1	89	175	706	86	$> 5^{86}$
20Term	3	63	124	764	40	2^{40}
Fleet20_3	3	60	320	1920	200	$> 3^{200}$

Table 3.1: Test Problems

PGP2 is a power generation planning problem described in Higle and Sen [1994] and Higle and Sen [1996b]. SSN (Sen et al. [1994]) is a telecommunication network planning problem. 20Term, described in Mak et al. [1999], is a motor freight scheduling problem. Fleet20_3 is a two-stage transportation problem, which comes to us via Huseyin Topaloglu (SORIE, Cornell University). PGP2 is a small and

easy-to-solve problem, while 20TERM, SSN, and Fleet20_3 are more challenging problems.

Besides the test problems listed above, we also explored many other problems. CEP1 (described in Higle and Sen [1994]) is a small capacity expansion planning problem with known deterministic optimal solution. It is more appropriate as a testing tool for implementations than as a test problem in a computational experiment. STORM (Mulvey and Ruszczyński [1995]) is an air freight scheduling problem. Although it has 118 random variables and 5^{118} possible scenarios, the gap between its upper bound and lower bound when evaluated using the *mean value problem* (MVP) is very small, indicating that the stochastic nature of the problem has a negligible impact on the optimal objective value. Both LandS and GBD are test problems we downloaded from the web site of the Stochastic Programming Community. LandS is a modified version of the electrical investment planning problem in Louveaux and Smeers [1988]. GBD is derived from the aircraft allocation problem described by Dantzig [1963]. We find that LandS is essentially PGP2, but with incomplete recourse, and that GBD is a simple recourse problem. These problems are not included in our final list of test problems.

3.3.2 Run-Time Parameter Settings

As always, performance measures can be greatly influenced by parameters embedded within the implementation of the algorithm under test. There are several such parameters within the SD program that impact conditions under which execution terminates.

The objective function approximations developed by SD are dynamic – they change from one iteration to the next as the set of observations vary. For this reason, even though a solution appears to be good during a given iteration, as the objective function approximation is updated the apparent quality of the solution can degrade. This is the case whether the linear or regularized master program is used. For this reason, optimality is not tested unless the apparent quality of the current solution does not appear to be appreciably affected by the updating of the objective

function estimate. That is, the simple stopping rule is tested as the preliminary test before the full test.

Pre-optimality Test: In each iteration k, no full optimality test is undertaken unless (3.8) is satisfied. In our experiment, the tolerance for the pre-optimality test is tied to that for the full optimality test. That is, $\epsilon_{smpl} = \epsilon_{full}$.

When the pre-optimality test is passed in iteration k, the full optimality test is undertaken on $\bar{x}^k$.

Full Optimality Test: A total of M bootstrapped estimates of the error associated with $\bar{x}^k$ are calculated. The program terminates if:

- At least $1 - \alpha$ of the M bootstrapped error estimates are within ϵ_{full} of the bootstrapped upper bound estimate (i.e., $(\hat{u} - \hat{l})/\hat{u} \leq \epsilon_{full}$).

In our experiment, we fix

- $M = 50$
- $\alpha = 0.05$

and test multiple values of ϵ_{full},

- $\epsilon_{full} \in \{0.005, 0.001, 0.0001\}$.

In the implementation of the SD methods, two additional parameters are used to control termination of the program: *Min_iter* and *Max_iter*. *Min_iter* is the number of iterations that must execute before any termination criteria are considered. *Max_iter* is the maximum number of iterations allowed. If the termination criterion is not satisfied by *Max_iter*, the program is terminated. The settings of these two parameters vary with the problem dimensions, as in Table 3.2.

A primary computational difference between LSD and RSD is a result of the master program. LSD uses a linear master program, identical to the master program used in Benders' Decomposition, except for the difference in the way that the cutting plane approximation of the recourse function is defined via the "argmax" procedure (see Higle and Sen [1996b]). The RSD objective is identical to the LSD objective,

Problem	Min_iter	Max_iter
PGP2	100	1000
SSN	1000	5000
20Term	1000	5000
Fleet20_3	1000	5000

Table 3.2: SD Parameter Settings

except for the addition of the term $\frac{\sigma}{2}(x - \bar{x})^\top(x - \bar{x})$. Other than this quadratic term in the objective the LSD and RSD master programs are structurally identical. Of course, the trajectory of iterates identified by LSD and RSD will differ, which can impact the overall effort required to identify a solution.

In order to provide a basis for comparisons of the impact of master program, we compare the following performance measures:

- *Objective Function Value* – the value of the objective function associated with the solution identified upon termination. For PGP2, this value is calculated precisely. For 20Tterm, SSN, and Fleet20_3, this value is estimated based on independently generated observations and is accurate to within 1% with 95% confidence. For example, the statistical evaluation typically requires less than 1,000 observations for 20TERM, and more than 700,000 observations for SSN.

- *Iterations required* – the total number of iterations undertaken prior to termination.

- *Time required* – the total time elapsed prior to termination of the algorithm.

We note that the solution methods in question use randomly generated data in their search for a solution. For this reason, we used 30 sets of runs (i.e., 30 different sequences of randomly generated observations). A single initial seed for our random number generator is used for each set, which ensures that LSD and RSD receive the same sequence of observations of $\tilde{\omega}$. Additionally, objective function value estimation is undertaken with a single stream of observations, generated independently from the 30 streams used in the LSD and RSD solution process. In this way, regardless

of the master problem form (e.g., linear or regularized) and the run-time parameter settings (e.g., Min_Iter, Max_Iter, ϵ), the two SD methods are exposed to identical sequences of observations in a controlled setting. As a result, differences in the output are due solely to the run-time settings that are under our control.

3.3.3 Computational Environment

All programs are written in C, using the CPLEX 7.0 callable library to execute master- and sub-problem optimization. The programs are run on a Sunw Sun-Fire-280R with 2xUltraSPARCIII processors at 900 MHz, 4GB of RAM, and a Solaris 9 operating system. The machine is a multi-user machine. Although efforts are made to ensure a dedicated computing environment, we cannot verify if, in fact, we are successful in this regard.

3.3.4 Result Analysis

Solution Quality

We begin by examining the quality of the solutions obtained by the two SD methods with virous run-time parameter settings. In Table 3.3 below, the objective values associated with the various solutions obtained are listed. The tabulated values represent averages over 30 independent replications at each of the indicated settings of ϵ_{full}, followed by the standard deviation associated with these 30 values in parentheses. In some cases, the method does not terminate prior to Max_iter. In such cases, the objective values are averaged over the subset of runs which terminate as a result of having passed the optimality test and the number of such runs appear in boldface within the parentheses. For example, when solving 20Term with $\epsilon_{full} = 0.001$, the average estimated objective value for the 30 replications involving RSD is 254,633 and the standard deviation is 84. All of these runs terminate prior to reaching Max_iter. A "–" is used to indicate that none of the 30 replications resulted in termination based on the optimality test (i.e., termination occurs when Max_Iter is reached).

Problem	ϵ_{full}	Objective Value	
		LSD	RSD
PGP2	0.005	448.2 (0.53, **13**)	450.2 (9.6)
	0.001	448.2 (0.54,**7**)	449.8 (9.6)
	0.0001	–	449.8 (9.8, **29**)
20Term	0.005	–	254,633 (84)
	0.001	–	254,633 (84)
	0.0001	–	254,633 (84)
Fleet20_3	0.005	–	141,731 (26)
	0.001	–	141,731 (26)
	0.0001	–	141,731 (26)
SSN	0.005	–	12.5 (0.93)
	0.001	–	12.5 (0.93)
	0.0001	–	10.3 (0.20)

Table 3.3: Objective Function Values

There are several items worth noting in Table 3.3. First, we know that the optimal objective value for PGP2 is 447.32. It is interesting to observe that in general, the solutions identified are nearly optimal, but there is evidence of error. The standard deviations of the solutions' objective value estimates are lowest with LSD (Stochastic Decomposition with a linear master program), but it exhibits a much stronger tendency toward reaching the maximum iteration count without satisfying the termination criterion. On the other hand, RSD (Stochastic Decomposition with a regularized master program) exhibits a much larger standard deviation. Closer inspection of the specific objective values obtained indicates that this is due to a single run which terminates based on the optimality test, but identifies a rather poor solution (with an objective value of 500.6). When that particular run is omitted, the performance on this measure is similar to the other values reported. Of course, one accepts the possibility of error when using statistical estimates and randomly generated data. Note that within our termination criteria, we set $\alpha = 0.05$ – indicating an acceptance of error beyond the specified level 5% of the time. In that sense, it is not surprising to see such error in one of the 30 runs undertaken.

The problem PGP2 is the smallest instance that we included in this experiment.

It is small enough to be solved exactly using the L-Shaped method, which probably makes it the least interesting problem to discuss. The other three problems are sufficiently large as to preclude precise solution. It is interesting to note that in all of these problems, LSD is consistently unable to terminate prior to reaching the maximum iteration count, while RSD exhibits the exact opposite behavior – it terminates via the optimality tests in every single case. The solution values are remarkably stable – especially for the problems 20term and Fleet20_3, for which ϵ_{full} does not impact the solution. The problem SSN is anecdotally held to be among the most challenging TS-SLPs available. We note that RSD terminates in every case, and the solution quality improves as the optimality error tolerance (i.e., ϵ_{full}) is reduced.

Iterations

It is interesting to observe the number of iterations required by the two SD methods prior to termination, as summarized in Table 3.4 below. The entries correspond to the average number of iterations over the 30 replications, followed by the standard deviation parenthetically. Similar to Table 3.3, we have also recorded the number of times that the methods terminated prior to reaching the maximum iteration count in boldface after the standard deviation. Thus, for example for $\epsilon_{full} = 0.001$, solving PGP2 with LSD requires 956 iterations on average, with a standard deviation of 133 and only 7 of the 30 replications terminate prior to the maximum iteration count. For the larger problems, the full optimality test is never undertaken prior to reaching the maximum iteration count with LSD, a phenomenon that is observed with $\epsilon_{full} = 0.005$. For this reason, we do not undertake these runs with the tighter tolerances.

Again we note distinct differences between the two SD methods, especially on the larger problems. While the linear master exhibits a clear tendency toward failing to satisfy the termination criteria within the allotted number of iterations, the regularized master exhibits the exact opposite behavior and exhibits a clear tendency toward satisfying the termination criteria very early on (e.g., as soon as

Problem	ϵ	Iterations	
		LSD	RSD
PGP2	0.005	920 (151 **13**)	185 (39)
	0.001	956 (133, **7**)	226 (39)
	0.0001	1000 (0, **0**)	316 (159, **29**)
20term	0.005	5000 (0, **0**)	1000 (0)
	0.001	5000 (0, **0**)	1000 (0)
	0.0001	5000 (0, **0**)	1000 (0)
Fleet20_3	0.005	5000 (0, **0**)	1000 (0)
	0.001	5000 (0, **0**)	1000 (0)
	0.0001	5000 (0, **0**)	1000 (0)
SSN	0.005	5000 (0, **0**)	1000 (0)
	0.001	5000 (0, **0**)	1005 (14)
	0.0001	5000 (0, **0**)	2409 (482)

Table 3.4: Iterations Required

it is permitted to undertake the full optimality test!). Notable exceptions to this tendency appear with PGP2, in which there is clearly some variability in the number of iterations required to pass the termination criteria (which occur in all cases), and SSN with the tight error tolerance of $\epsilon_{full} = 0.0001$. These exceptions illustrate the adaptive nature of the SD method – the tests permit a recognition of insufficient evidence of solution quality to warrant termination. When that occurs, iterations continue until such evidence is obtained. Given the high degree of consistency in both the objective values and the iterations required for the regularized SD to solve 20term and Fleet20_3, it is likely that the Min_Iter settings for these two instances are higher than necessary.

Solution Time

We note that "iterations" and "optimizations" alone are not sufficient for comparative purposes, as the "computational overhead" and the nature of the optimizations performed vary considerably. SD iterates with "master problems" and "subproblems." LSD works with a linear master program and RSD works with a regularized master program – a quadratic program, although the subproblems are identical in

both cases. In order to reflect the impact that the form of the master problem might have, we also look at the time required to execute the algorithms through termination.

Problem	ϵ	Time	
		LSD	RSD
PGP2	0.005	12.33 (16.48)	0.96 (0.35)
	0.001	9.12 (9.14)	1.26 (0.48)
	0.0001	5.74 (6.49)	2.67 (3.10)
20term	0.005	25,644 (1235)	259.42 (31.94)
	0.001	25,644 (1235)	259.27 (31.90)
	0.0001	25,644 (1235)	259.30 (31.85)
Fleet20_3	0.005	40,286.3 (943)	291.67 (2.54)
	0.001	40,286.3 (943)	291.67 (2.54)
	0.0001	40,286.3 (943)	293.57 (2.45)
SSN	0.005	20,424 (8,127)	391.81 (17.05)
	0.001	20,424 (8,127)	341.70 (88.53)
	0.0001	20,424 (8,127)	7491.81 (3728.67)

Table 3.5: Computational Times

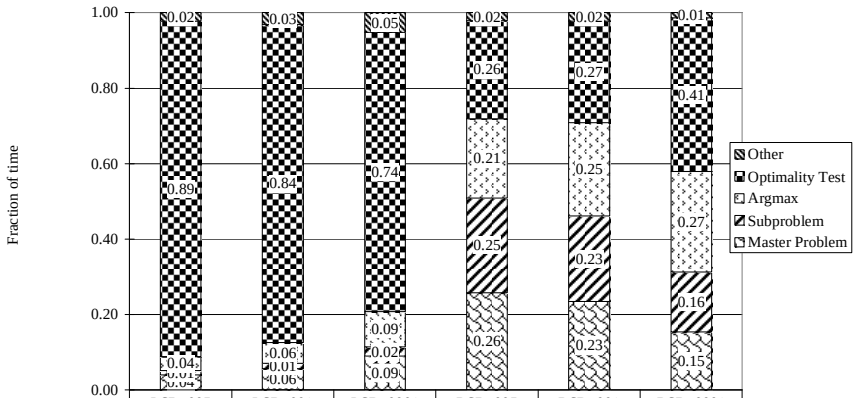
In Table 3.5, we summarize the time (in seconds) required to obtain these solutions. Again, we report the averages over 30 independent replications with standard deviations reported parenthetically. One immediately notices that RSD requires considerably less time than LSD on all problems. The average times reported are smaller, as are the standard deviations. We have already noted that LSD often runs all the way to the maximum number of iterations allowed, which has an obvious impact on the time required. On the other hand, RSD terminates well before this limit is reached. Additionally, RSD has a more streamlined method for estimating error bounds in the optimality tests. The added effort required to solve the quadratic master program (beyond that required for the linear master program) is more than compensated for by the reduced number of iterations required and the simplified termination criteria available.

It is interesting to compare the manner in which time/effort is allocated among the various computational activities by the two SD methods. The primary compu-

tational requirements are:

- master program solution
- subproblem solution
- argmax-related computations (see Higle and Sen [1996b])
- optimality tests (i.e., termination criteria)
- other "overhead" activities

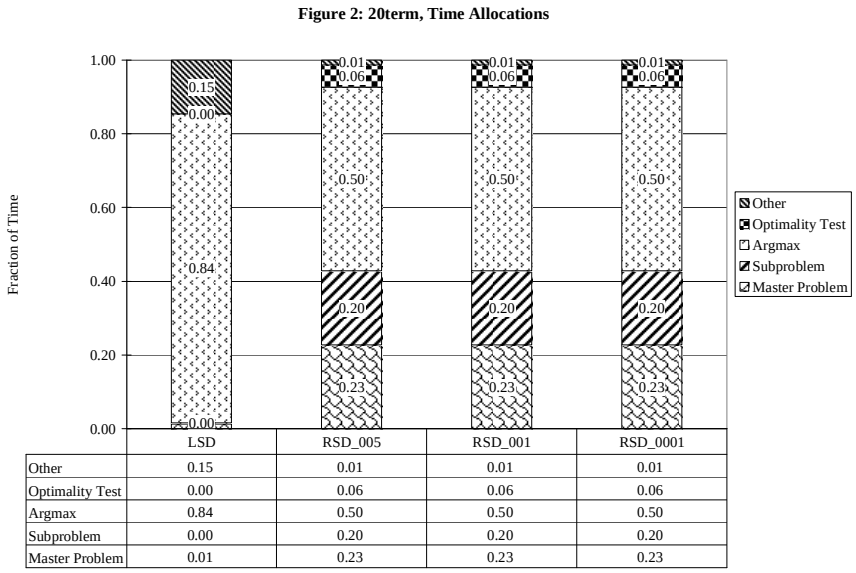
Figure 1: PGP2 Time Allocations



	LSD_005	LSD_001	LSD_0001	RSD_005	RSD_001	RSD_0001
Other	0.02	0.03	0.05	0.02	0.02	0.01
Optimality Test	0.89	0.84	0.74	0.26	0.27	0.41
Argmax	0.04	0.06	0.09	0.21	0.25	0.27
Subproblem	0.01	0.01	0.02	0.25	0.23	0.16
Master Problem	0.04	0.06	0.09	0.26	0.23	0.15

Figure 3.1: Time Allocation – PGP2

Figures 3.1 – 3.4 illustrate the allocation of time among these various activities for each problem instance. Within these graphs, the type of master program used, linear or regularized, is indicated by the terms "LSD" and "RSD", respectively. Since the value of ϵ_{full} varied, it is also reflected. Thus, for example, LSD_001 indicates solution with a linear master program using $\epsilon_{full} = 0.001$, while RSD_0001 indicates a regularized master program with $\epsilon_{full} = 0.0001$. Table 3.5 clearly indicates that LSD requires considerably more time than RSD for all problems – Figures 3.1 – 3.4 indicate how the time is distributed among the various activities. Note for example

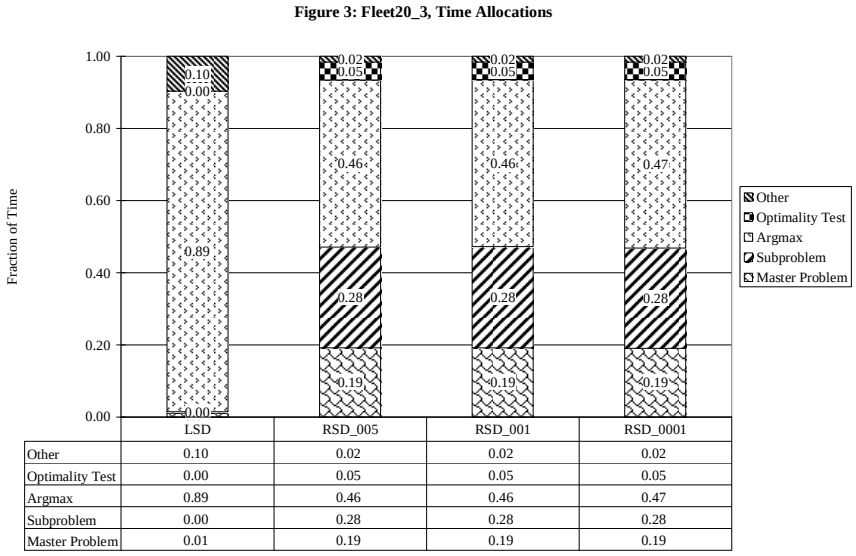


	LSD	RSD_005	RSD_001	RSD_0001
Other	0.15	0.01	0.01	0.01
Optimality Test	0.00	0.06	0.06	0.06
Argmax	0.84	0.50	0.50	0.50
Subproblem	0.00	0.20	0.20	0.20
Master Problem	0.01	0.23	0.23	0.23

Figure 3.2: Time Allocation – 20Term

that Figure 3.1 indicates that when solving PGP2 with LSD the vast majority of the time is allocated to the full optimality test (74% - 89%, depending on the value of ϵ_{full} used). Indeed, the potential for extreme computational effort toward this undertaking is discussed in Section 3.2. That the optimality time is reduced somewhat as ϵ_{full} is reduced results from the fact that given our run-time settings (i.e., the tolerance of the pre-optimality test, ϵ_{smpl}, is tied to ϵ_{full}), a tighter error tolerance will also serve to prevent full optimality tests from occurring in some iterations. Because so much time is spent on the full optimality test, it is difficult to appreciate the magnitude of the effort required for the remaining tasks. However, we note the computations required to perform the "argmax" procedures are comparable to the time allocated to the master problem solutions. This remains approximately true when PGP2 is solved with the regularized master where substantially less time is spent on optimality tests. Given that RSD is considerably faster than LSD in the first place, it appears that a substantial reduction in effort required to perform the full optimality test has been accomplished, as suggested in Section 3.2.

In all of the other instances solved, LSD fails to undertake an optimality test due to the preliminary tests that are in place. In 20term, SSN, and Fleet20_3, we



	LSD	RSD_005	RSD_001	RSD_0001
Other	0.10	0.02	0.02	0.02
Optimality Test	0.00	0.05	0.05	0.05
Argmax	0.89	0.46	0.46	0.47
Subproblem	0.00	0.28	0.28	0.28
Master Problem	0.01	0.19	0.19	0.19

Figure 3.3: Time Allocation – Fleet20_3

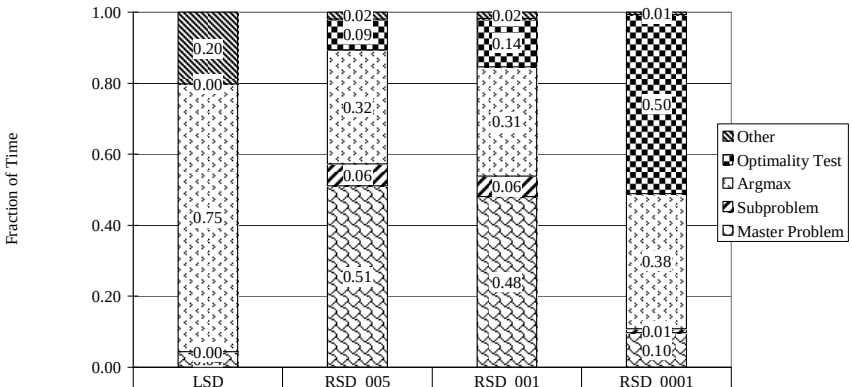
note that LSD allocates most of its effort to the argmax computations (75-89%). In contrast, RSD spends a smaller fraction of time on the argmax computations (31-50%). Given the variety of problem size statistics reported in Table 3.1, the effort required to solve the master and subproblems varies among the problems, as one expects.

3.4 Computational Investigation: RSD vs. SAA

In Section 3.3, we explore the computational impact of the error bound-based termination criteria of the two SD methods as described in Section 3.2. In this section, we explore the computational impact of the adaptive sample-based technique, SD, and the nonadaptive sample-based technique, SAA. Recall from Section 2.3.2 that unlike SD which potentially considers a large sample size within a single run (if necessary), SAA proposes to solve a fixed number of instances (M), each with a fixed sample size (n). Given that our goal is comparative, we select M and n so that SD and SAA use approximately the same observations per run.

To do this suppose that SD terminates in iteration K, indicating that a total of K observations were used to identify the solution. For each of $M \in \{1, 2, 5, 10\}$,

Figure 4: SSN, Time Allocations



	LSD	RSD_005	RSD_001	RSD_0001
Other	0.20	0.02	0.02	0.01
Optimality Test	0.00	0.09	0.14	0.50
Argmax	0.75	0.32	0.31	0.38
Subproblem	0.00	0.06	0.06	0.01
Master Problem	0.04	0.51	0.48	0.10

Figure 3.4: Time Allocation – SSN

we set $n_M = \lceil K/M \rceil$ (rounding up if necessary) to ensure that all M representations of SAA use the same number of observations within each replication of our experiment. Thus, for example, if SD terminates with $K = 2557$, then for $M = 5$, $n_5 = \lceil 2557/5 \rceil = 512$ while for $M = 10$, $n_{10} = 256$. In this fashion, SD and SAA use essentially the same set of observations of $\tilde{\omega}$ as they search for a solution to (TS-SLP). We note that SAA_1 is a misnomer. Without the replication formally required of SAA (see, e.g., Shapiro [2000]), SAA_1 is nothing more than the original problem instance in which the probability distribution is replaced by the empirical distribution associated with the observations generated in the course of the SD solution.

We use the term "SAA instance" to refer to the smaller stochastic program with n_M observations for some value of $M \in \{1, 2, 5, 10\}$. Thus, within any given "run", we solve 1 SAA instance, with a sample size of $n_1 = K$, 2 SAA instances each with a sample size of n_2, etc. Formally, SAA is a meta-method, because the SAA instances must still be solved. In our computations, all SAA instances are solved using Benders' Decomposition (aka, the L-Shaped method), and are terminated when the difference between the upper bound and lower bound is within 0.0005, or

when a maximum iteration count is reached. For PGP2, this maximum iteration count is set at 1000, and for the remaining problems, it is set at 4000. Recall that because SAA operates with a nonadaptive sample, the iteration count only controls the number of cuts used in the piecewise linear approximation of the objective function. All of the results involving SD in this section are obtained with RSD, using $\epsilon_{full} = 0.0001$. The minimum and maximum iteration counts for RSD are fixed as in Table 3.4 except for PGP2 for which Min_iter = 500 and Max_iter = 2000 are used.

Recall that SAA includes an intermediate step in which each of the M solutions obtained from a given SAA instance are evaluated so that the apparent best among them can be selected. Because the sample spaces associated with the problems under consideration are too large to permit precise objective function evaluations, these evaluations are undertaken with randomly generated data as in section 3.3. Whenever $M > 1$, each of the SAA instance solution evaluations are performed with an independent set of observations, and are accurate to within 1%, with 95% confidence. For each value of M, the SAA instance solution with the minimum objective value estimate (based on this phase of independent posterior evaluation) is identified as the SAA solution.

As in Section 3.3, we begin with a review of the objective values associated with the RSD and SAA solutions. In Table 3.6, we present the average objective values associated with the solutions obtained over multiple independent replications (10 for PGP2 and 5 for 20term, Fleet20_3, and SSN), with standard deviations reported parenthetically. As in Table 3.3, these objective values are exact for PGP2 and accurate to within 1%, with 95% confidence for 20term, Fleet20_3, and SSN. Because RSD is an adaptive technique, "K" (the number of observations used by SD, and subsequently SAA) varied among the independent replications. For each problem, we report the average value of K, followed by the standard deviation parenthetically.

There are several observations to make regarding Table 3.6. We will examine the output on a problem by problem basis, in combination with Figures 3.5 – 3.8

Problem	M	Objective Value	
		RSD	SAA
PGP2	1	448.7 (1.31)	448.0 (0.61)
K=604 (230)	2		447.7 (0.53)
	5		447.4 (0.05)
	10		447.4 (0.15)
		RSD	SAA
20Term	1	254,581 (79)	254,744 (57)
K=1000 (0)	2		254,677 (77)
	5		254,624 (17)
	10		254,512 (55)
		RSD	SAA
Fleet20_3	1	141,749 (18)	141,685 (27)
K=1000 (0)	2		141,687 (32)
	5		141,662 (21)
	10		141,654 (6.5)
		RSD	SAA
SSN	1	10.26 (0.14)	10.02 (0.06)
K=2293 (462)	2		10.08 (0.04)
	5		10.22 (0.15)
	10		10.57 (0.28)

Table 3.6: RSD vs. SAA: Objective Function Values

which illustrate the computational times required by the various methods.

PGP2: The average objective values obtained are all within 0.3% of each other. We note that RSD exhibits the highest average value as well as the largest standard deviation. Examination of the individual output indicates that as before, the high standard deviation associated with these values is due to a single run in which a high objective value (452.2) is obtained. On a run by run basis, the source of the maximum observed objective value is evenly divided between RSD and SAA_1, while the source of the minimum observed objective value is most often associated with SAA_5. Figure 3.5 illustrates the time required to execute the solution of PGP2 with the various methods, for each of the 10 independent replications. In general, solutions are obtained within 4 seconds, regardless of the method used. However, two replications require considerably more with RSD (11.2 and 14.7 seconds). The

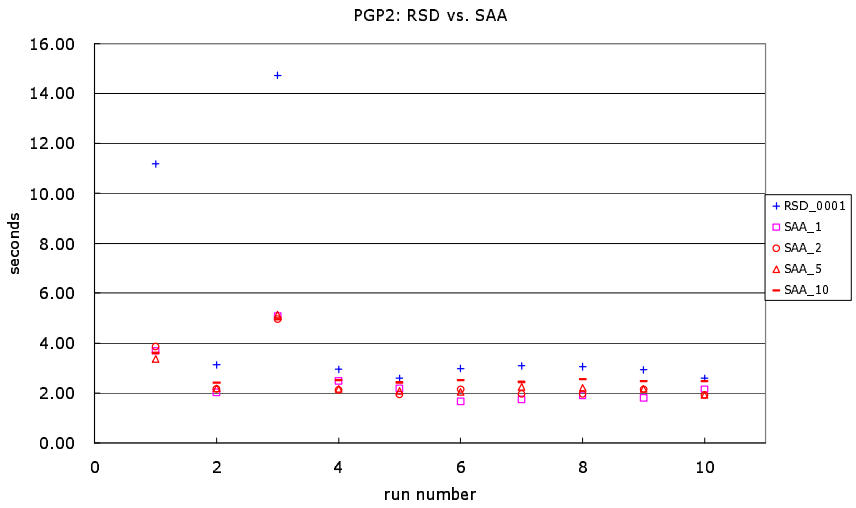


Figure 3.5: Solution Times – PGP2

non-adaptive method, SAA, appears to be well suited for this small problem.

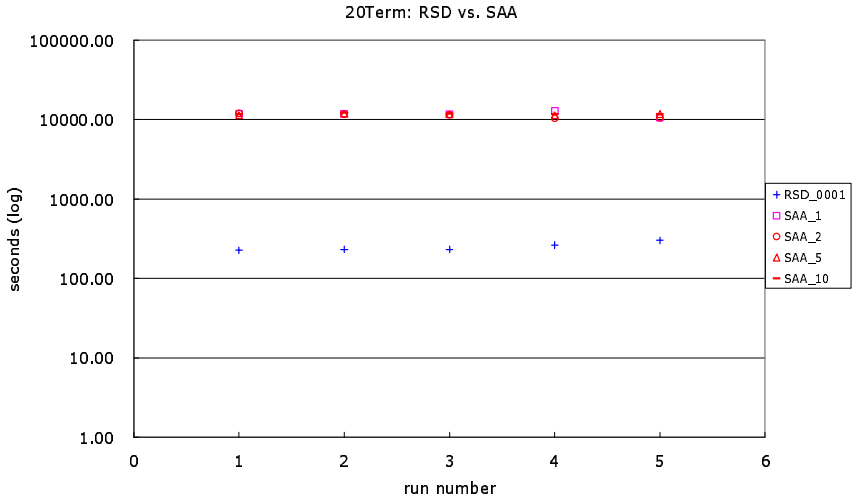


Figure 3.6: Solution Times – 20Term

20Term: It is fair to say that as far as objective values are concerned, all methods performed equally well. In addition to the averages being close and standard deviations being small, as indicated on Table 3.6, the difference between the minimum and maximum observed values is only 0.1%. Given that the estimated objective values are only accurate to within 1% with 95% confidence, there are no discernible differences among the objective values obtained. Figure 3.6 illustrates the solution times

required by the various methods to obtain their solutions. We note that because the differences are so extreme, it is necessary to view these times on a logarithmic scale. With regard to computational times, RSD exhibits a clear advantage. The solution times required by SAA are fairly stable across the four implementations tested (SAA_1, SAA_2, SAA_5, and SAA_10), and are nearly 1.5 orders of magnitude larger than the solution times required by RSD.

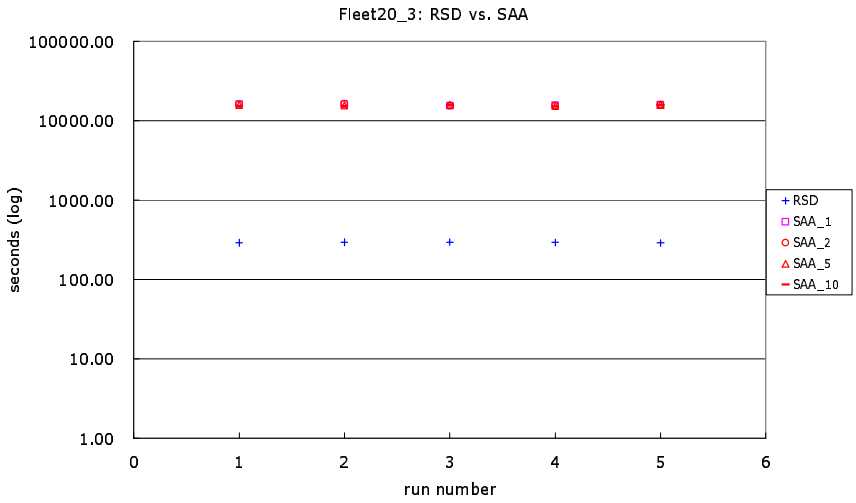


Figure 3.7: Solution Times – Fleet20_3

Fleet20_3: As with 20Term, the objective values obtained in all cases are within 0.1% of each other, indicating that there is no discernible differences among them. Figure 3.7 illustrates the solution times required to obtain solutions. Again, the differences in these times are so extreme that it is necessary to view them on a logarithmic scale, and again we see that the SAA solution times are nearly 1.5 orders of magnitude larger than for RSD.

SSN: For this particular instance, SAA_1 exhibits the best objective value. That is, simply replacing the original distribution with the empirical distribution obtained from the observations generated during the course of the RSD runs produced the best objective values on average (and the least variable values as well). In all cases, the worst objective values obtained are associated with SAA_10. Figure 3.8 illustrates the solution times required. Again, a logarithmic scale is necessary to view them, and SAA requires approximately 1.5 orders of magnitude more time

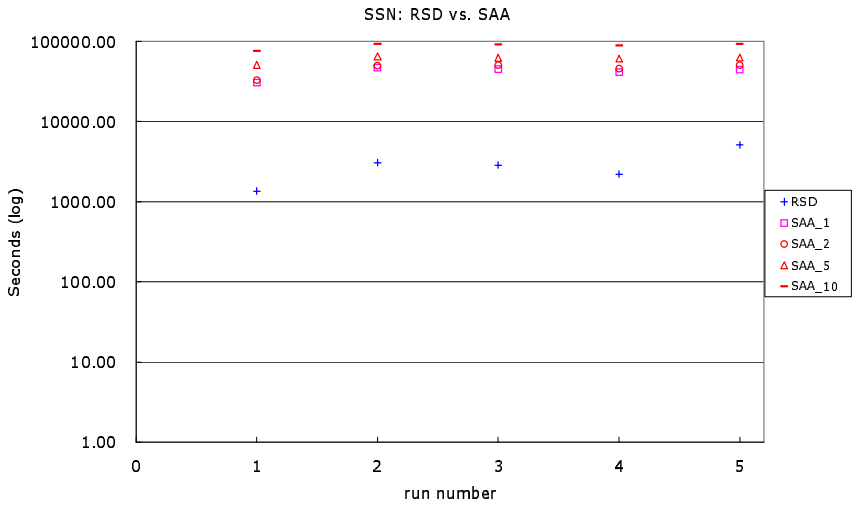


Figure 3.8: Solution Times – SSN

to solve SSN than RSD. Unlike the other large problems (20term and Fleet20_3, which exhibited fairly stable solution times), SSN exhibits greater variation in times attesting to its reputation as a difficult problem. We note that even on this logarithmic scale, there is a clear discrepancy among the SAA solution times, with SAA_1 (Benders' decomposition applied to the empirical distribution without the required SAA replications) considerably faster than the remaining SAA implementations. It would appear that in combination with the improved objective value estimates that it provides, among the various SAA runs tested, SAA_1 appears to be best suited for this particular instance. Of course, it is not difficult to note that even this requires nearly 13 times longer to obtain a solution than RSD, whose objective value is on average only slightly larger than the statistical error.

It is clear that for all but the smallest problem, the SAA solution times are substantially higher than the RSD solution times with approximately 1.5 orders of magnitude difference in all cases. Variation in the RSD solution times are attributable to the adaptive nature of its termination conditions, as we have discussed in Section 3.3. The source of variation in the SAA solution times, especially as observed in SSN is less clear. Consequently, we examine the time allocated to each of the major computational requirements associated with SAA. Recall that SAA

involves two major computational steps. Given M, the number of SAA replications undertaken, it is necessary to

- solve M SAA instances of the problem (i.e., where the actual distribution is replaced by an empirical distribution) to obtain the set of M proposed solutions, and
- evaluate the objective function value associated with each of the M proposed solutions.

Following this, the proposed solution with the smallest objective function value estimate is selected as the SAA solution. In Table 3.7, we tabulate these times for each problem. "Solve" refers to the time required to solve the set of M SAA instances and "Evaluate" refers to the time required to evaluate the M proposed solutions. The numbers reported are averages over the 5 replications (ten replications for the smaller problem, PGP2) followed by standard deviations parenthetically. Note that in all cases, the evaluation time associated with $M = 1$ is identically 0 – in the absence of the SAA replications, there is no need to evaluate the proposed solution.

Examining the SAA time allocations provided in Table 3.7, the explanation behind some of our earlier observations becomes apparent. First, note that in general there is only limited variation in the solution times as M varies from 1 to 10. That is, for these problem instances, it appears that the actual solution time is not critically dependent on the manner in which the observations are divided. While the number of observations naturally influences the solution times required, the manner in which they are distributed among the SAA instances does not in general. The exception to this is SSN, where we observe a fairly steady increase in the solution time required as M increases. Second, we note that two of the test problems, 20Term and Fleet20_3 require relatively minimal time to evaluate the objective function values. Clearly, as M increases the time required to evaluate the proposed solutions increases as well. However, in these two problems, the time required to evaluate these solutions is negligible compared to the solution times required. On the other hand, evaluating the objective function values associated with the solutions to

Problem	M	Time	
		Solve	Evaluate
PGP2	1	2.40 (1.08)	0 (0)
	2	2.38 (1.03)	0.14 (0.01)
	5	2.21 (1.00)	0.34 (0.01)
	10	2.16 (0.83)	0.68 (0.02)
20Term	1	11,757 (816)	0 (0)
	2	11,189 (717)	4.42 (0.18)
	5	11,547 (213)	11.15 (0.38)
	10	11,851 (224)	22.77 (0.94)
Fleet20_3	1	15,934 (340)	0 (0)
	2	15,833 (408)	1.48 (0.05)
	5	15,658 (228)	3.78 (0.09)
	10	15,378 (272)	7.32 (0.16)
SSN	1	36,893 (5,962)	0 (0)
	2	37,627 (6,976)	7,989 (660)
	5	39,100 (4,842)	21,146 (785)
	10	46,340 (4,901)	41,792 (2,381)

Table 3.7: SAA Time Allocations

SSN is a more arduous undertaking. As M increases, the evaluation time becomes comparable to the solution time.

As a final comment, our benchmarking comparisons between SD and SAA are based on the L-Shaped method, so that there is a common algorithmic root shared by the solution methodologies. We note that the L-Shaped method may not provide the most efficient solution procedure for the solution of the SAA instances. In order to understand the extent to which this might have been a factor in the extreme disparity in the solution times that we observed, we undertake a cursory investigation of alternative solution methods. In general, as one expects, we observe that for smaller sample sizes, CPLEX can solve the SAA instances faster than the L-Shaped method. This does not tend to be the case with the larger sample sizes, where the L-Shaped method can be considerably faster than CPLEX. That is, in contrast to our implementation of the L-Shaped method, where the manner in which observations are distributed among the SAA instances does not, in general, have an appreciable impact on the overall solution times, CPLEX is considerably faster with

the smaller sample sizes than with the larger ones. We note that our observations in this regard apply only to the solution times, and not to the intermediate step in which all of the SAA instance solutions' objective values are estimated. Additionally, we note that our observations regarding the relative performances of the linear and regularized versions of SD suggest that improvement in the solution of the SAA instances might also be obtained using a regularized L-Shaped method, as in Ruszczyński [1986]. Although we do not code a regularized L-Shaped method, we may look to the literature for results on comparisons based on a well-tuned implementation. Ruszczyński [1993], and Ruszczyński and Świętanowski [1997] provide numerical results with SSN using linear and regularized versions of the L-Shaped method, suggesting that the regularized version is approximately 3-4 times faster than the non-regularized version. This observation is based on a sample size of 200 observations, which approximately corresponds to the size of the SAA_10 instances for this problem (and of course, this excludes the intermediate objective function value evaluation required by SAA). Thus, while in some cases the SAA instances may be solved faster using methods other than the L-Shaped method, it appears that the regularized SD solution times are still faster, especially when the objective function evaluation requirements are taken into account.

3.5 Conclusions

Our computational results indicate fairly clearly that the regularized version of SD (RSD), with its more streamlined optimality tests, is more efficient than SD with a linear master program (LSD). The matrix computations associated with RSD's optimality tests are less demanding than the replicated master program solution associated with LSD's optimality tests. Combined with apparent differences in the solution trajectories (as indicated by the substantial differences in the number of iterations required), the additional computational effort required to solve the quadratic master program in RSD appears to be easily offset. With a small problem, such as PGP2, the differences in solution times are fairly minor, but with larger

problems the differences are dramatic. It would appear that RSD prefers a tighter tolerance in its termination requirements, as the solution values appear to be less variable when this tolerance is tightened.

It appears that SAA may be well suited to the solution of the smaller problem, such as PGP2. It exhibits an edge over RSD in the computational times, and in the objective values as well. In this particular case, M=1 does not perform as well as larger values of M. For all other instances tested, SAA requires substantially longer to obtain solutions than does RSD. Indeed, the differences in solution times are dramatic enough to require display with logarithmic scale, although this requirement would probably be lessened by an alternate choice of solution method for the SAA instances.

In general, the differences in the objective values obtained are well below the statistical error associated with their estimated values, although this is not the case uniformly. In the more challenging problem (SSN), there is a preference for simply using M=1, which bypasses SAA and uses an empirical distribution with the largest sample size available. As a general rule, the need to evaluate objective function values for all M proposed solutions is an obvious computational bottleneck, although problems with relatively easy objective function evaluations pose less of a burden in this regard.

Although in the second phase of experiment, our intent is to investigate differences in the computational behavior of a method that exploits adaptive sample sizes (SD) with one that uses non-adaptive samples (SAA), we note that our experimental design offers SAA an opportunity to take advantage of information regarding sample sizes learned by SD in its adaptive setting. That is, the samples to which SAA is exposed are determined a priori by SD. Guidance on sample sizes can be found in Shapiro et al. [2002], where the sample sizes tend to be much larger than those that we have used, which will naturally serve to increase the computational times required. Additionally, the effort required to determine these sample sizes can also be somewhat extensive (requiring, for example, the expected value and variance of directional derivatives at optimal solutions). Of course, the impact of SD's adap-

tive sample sizes is evident in the variable number of iterations and times required as it adjusts to the different sample paths to which it is exposed. The stability in the objective values produced by the RSD solution indicates some success in this adaptation. To be sure, there are cases in which there is variability in the objective values produced (most notably with PGP2), although this variability appears to be consistent with the statistical nature of the termination conditions imposed.

CHAPTER 4

Solution Validation in MS-SLP: Lower Bounds

Multistage stochastic linear programming, which allows more stages in the decision problems than two-stage stochastic linear programming does, provides more modeling flexibility and has been successfully applied to model and solve decision problems under uncertainty.

However, real world applications often result in large scale instances that are hard, if not impossible, to solve precisely. People often combine heuristic methods with approximation and sampling techniques to obtain quick, but not necessarily optimal, solutions to such instances. In many cases, solutions obtained in this manner are not deterministically verifiable and are subject to validation. As a result, techniques to evaluate solution quality become important.

4.1 MS-SLP and NNT Constraints

As described in Chapter 2, assuming a finite time horizon, $T < \infty$, and assuming the number of scenarios is finite, $|S| < \infty$, a multistage stochastic linear program (MS-SLP) may be formulated as

$$\begin{aligned} \text{minimize} \quad & \sum_{s \in S} p^s c^{s\top} x^s && && \text{(MS-SLP)} \\ \text{subject to} \quad & x^s \in X^s, && \forall s \in S, && \text{(2.3a)} \\ & \{x^s\}_{s \in S} \in \mathcal{N}. && && \text{(2.3b)} \end{aligned}$$

Here,

$$x^s = (x_1^s, ..., x_T^s),$$

where x_t^s denotes the decision vector in stage t of scenario s (i.e., if $\tilde{\omega} = \omega^s$). $\mathcal{N}$ is the set of nonanticipative solutions. Nonanticipativity (NNT) of a solution ensures

that the solution is implementable, so that decisions in stage t depend only on information available until stage t, but not on those that can only be obtained in stages $t+1, t+2, ...$

The existence of the NNT constraints is a bottleneck in solving an MS-SLP instance. This is true for two reasons: first, the NNT constraints are the "coupling constraints" that link the otherwise separable scenario problems to form a much larger LP (the DEP); second, the NNT constraints typically constitute a significant fraction of constraints in the DEP of an MS-SLP instance. For example, consider the five-stage financial portfolio decision problem instance, SGPF5Y5 (We describe this test instance in Section 4.4.1). In SGPF5Y5, each of the 625 scenario problems has 455 decision variables and 314 constraints. The DEP of SGPF5Y5 has more than 280,000 decision variables and 410,000 constraints. There are more than 220,000 NNT constraints, which are more than half of the total constraints in the DEP. (Note: the exact size of the DEP depends on the choice of NNT representation.)

Eliminating all the NNT constraints from the DEP results in a problem that is easier to solve, because the remaining constraints, (2.3a), are separable by scenario. The resulting solution, $\{\hat{x}^s\}_{s\in S}$, is known as the *wait-and-see* solution, which provides a lower bound on the optimal objective value of the MS-SLP instance. However, a full relaxation of these constraints generally leads to a loose bound on the optimal objective value. Our goal is to explore ways to obtain tighter lower bounds through partial elimination of the NNT constraints from the DEP.

4.2 Representations of Nonanticipativity Constraints

There are a number of ways to represent the NNT constraints, and the manner in which they are represented can have a significant impact on the computational requirements of a solution procedure. Consequently, the precise representation of these constraints is typically based on the solution method employed.

Returning to the scenario tree depicted in Figure 2.1, let us consider the node in stage 2 through which scenarios $1-6$ pass. In Figure 4.1, this node is labeled

"a". Taking the sub-tree rooted at "a", $\mathcal{T}_a$, as an example, we will briefly review some representations of the NNT constraints in the literature. The discussion of the impact of their manipulations on the information structure follows.

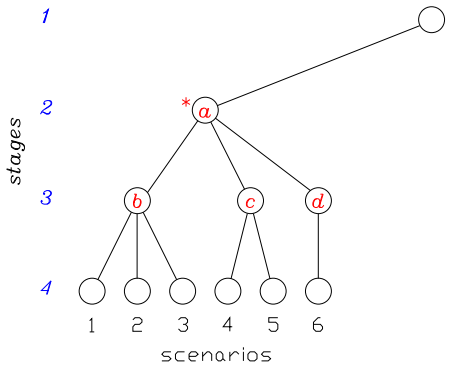


Figure 4.1: A Scenario Subtree (see Figure 2.1)

4.2.1 Sibling-type

Sibling-type NNT constraints are suggested in the *Augmented Lagrangian Decomposition Algorithm* (Rosa and Ruszczyński [1996]) and the *Diagonal Quadratic Approximation Method* (Mulvey and Ruszczyński [1992, 1995]). In this representation, the scenarios in each stage are ordered according to the "last common stage", which is defined, between scenarios ω and ω', by

$$t^{max}(\omega, \omega') = \max\{t : \underline{\omega}_t = \underline{\omega}'_t\}. \tag{2.17}$$

The *sibling* of scenario s in stage t is then defined as

$$\nu(s,t) = \begin{cases} s+1, & if \quad t^{max}(\omega^s, \omega^{s+1}) \geq t, \\ \min\{s' : t^{max}(\omega^s, \omega^{s'}) \geq t\}, & otherwise. \end{cases} \tag{2.18}$$

The *inverse sibling* of scenario s in stage t is denoted as $\nu^{-1}(s,t)$. That is, $\nu^{-1}(s,t) = s'$, if $s = \nu(s',t)$.

With the above definitions at hand, the NNT constraints (2.2) can be expressed as

$$x_t^s = x_t^{\nu(s,t)}, \quad s = 1, ..., |S|, \quad t = 1, ..., T-1, \tag{4.2}$$

or

$$x_t^s = x_t^{\nu^{-1}(s,t)}, \quad s = 1, ..., |S|, \quad t = 1, ..., T-1. \tag{4.3}$$

In the example of subtree $\mathcal{T}_a$ in Figure 4.1, the siblings of scenarios 1 - 6 in stage 2 are 2, 3, 4, 5, 6, 1; and are 2, 3, 1, 5, 4, 6 in stage 3. The inverse siblings of scenarios 1 - 6 in stage 2 are 6, 1, 2, 3, 4, 5; and are 3, 1, 2, 5, 4, 6 in stage 3. That is, the inverse sibling of scenario 4 is scenario 3 in stage 2 and is scenario 5 in stage 3.

(4.2) and (4.3) are equivalent mathematically. In our implementations, we choose to use (4.3). Then the NNT constraints of subtree $\mathcal{T}_a$ in stage 2 are

$$x_2^1 = x_2^6, \tag{4.4a}$$
$$x_2^2 = x_2^1, \tag{4.4b}$$
$$x_2^3 = x_2^2, \tag{4.4c}$$
$$x_2^4 = x_2^3, \tag{4.4d}$$
$$x_2^5 = x_2^4, \tag{4.4e}$$
$$x_2^6 = x_2^5, \tag{4.4f}$$

and in stage 3 are

$$x_3^1 = x_3^3, \tag{4.5a}$$
$$x_3^2 = x_3^1, \tag{4.5b}$$
$$x_3^3 = x_3^2, \tag{4.5c}$$
$$x_3^4 = x_3^5, \tag{4.5d}$$
$$x_3^5 = x_3^4, \tag{4.5e}$$
$$x_3^6 = x_3^6. \tag{4.5f}$$

The sibling-type constraints are sparse with exactly two nonzero entries in each constraint. The sibling indices of scenarios $s = 1, ..., |S|$ in each stage is a permutation of $1, ..., |S|$. The representation generates a loop of paired scenarios within each scenario bundle. There is one redundant constraint in each scenario bundle.

In our implementations, we remove the redundancy by eliminating the NNT constraint associated with the scenario with the lowest index in each scenario bundle.

For example, we eliminate the NNT constraints associated with scenario 1: (4.4a) in scenario bundle node 'a' and (4.5a) in scenario bundle node 'b'. Similarly, NNT constraint (4.5d) in bundle node 'c' and (4.5f) in bundle node 'd' are eliminated as well. The resulting NNT constraints are, in stage 2,

$$x_2^2 = x_2^1, \tag{4.6a}$$

$$x_2^3 = x_2^2, \tag{4.6b}$$

$$x_2^4 = x_2^3, \tag{4.6c}$$

$$x_2^5 = x_2^4, \tag{4.6d}$$

$$x_2^6 = x_2^5, \tag{4.6e}$$

and in stage 3,

$$x_3^2 = x_3^1, \tag{4.7a}$$

$$x_3^3 = x_3^2, \tag{4.7b}$$

$$x_3^5 = x_3^4. \tag{4.7c}$$

Note that, after removing redundancy, among the retained NNT constraints, we have

$$s > \nu^{-1}(s,t), \quad \forall s \in S, t = 1, ..., T-1. \tag{4.8}$$

This property is important in our upper bound procedures in Chapter 5.

4.2.2 Node-type

Node-type NNT constraints are used in the *Stochastic Scenario Decomposition* algorithm (Higle et al. [2004]). In the node-type representation, the NNT condition is enforced by the introduction of an artificial state vector at each node in the scenario tree.

Let Γ denote the set of nodes in the scenario tree. Each node $\gamma \in \Gamma$ corresponds to a stage and a bundle of scenarios, denoted by $t(\gamma)$ and $\mathcal{B}_\gamma$, respectively. For each $\gamma \in \Gamma$, let z_γ denote its "information state vector". The NNT constraints (2.2) can

be expressed as

$$x^s_{t(\gamma)} = z_\gamma, \quad \forall s \in \mathcal{B}_\gamma, \quad \forall \gamma \in \Gamma. \tag{2.25}$$

For example, in subtree $\mathcal{T}_a$ in Figure 4.1, the NNT constraints in stage 2 are

$$x^1_2 = z_a, \tag{4.9a}$$
$$x^2_2 = z_a, \tag{4.9b}$$
$$x^3_2 = z_a, \tag{4.9c}$$
$$x^4_2 = z_a, \tag{4.9d}$$
$$x^5_2 = z_a, \tag{4.9e}$$
$$x^6_2 = z_a. \tag{4.9f}$$

and the NNT constraints in stage 3 are

$$x^1_3 = z_b, \tag{4.10a}$$
$$x^2_3 = z_b, \tag{4.10b}$$
$$x^3_3 = z_b, \tag{4.10c}$$
$$x^4_3 = z_c, \tag{4.10d}$$
$$x^5_3 = z_c, \tag{4.10e}$$
$$x^6_3 = z_d. \tag{4.10f}$$

In each stage, scenarios are connected by the node through which they pass. Node-type constraints are sparse with exactly two nonzero entries in each constraint. Unlike the sibling-type representation, there is no redundancy in the resulting node-type NNT constraint matrix. Of course the sparsity and lack of redundancy here require the introduction of an additional set of variables for each node.

4.2.3 Branch-type

The idea of branch-type NNT constraints is from the SMPS "SCENARIOS" format (Edwards et al. [1985], Birge et al. [1987], Gassmann and Schweitzer [2001]).

SMPS format is an input format based on the MPS format for deterministic linear programs. The design of SMPS format aims to make it easy to convert existing deterministic linear programs into stochastic programs by adding information about the dynamic and stochastic structure.

In the SMPS "SCENARIOS" format, scenarios "branch" starting from the root node in a fashion similar to tree branches. Given the MS-SLP scenario tree, in each scenario bundle, the scenario of the lowest index is selected as the *branch scenario* and other scenarios branch from it. In the first stage scenario bundle, the scenario with the lowest index among $s = 1, ..., |S|$, scenario 1, is the branch scenario, and it branches from the root node by definition.

Let s_b^s denote the branch scenario of scenario s. The stage in which a scenario s diverges from its branch scenario is called the *branch stage* of scenario s, denoted by τ^s. In MS-SLP, it is possible that a scenario s has multiple branch scenarios from different scenario bundles that scenario s passes through. In this case, the scenario with which scenario s has the largest branch stage will be the branch scenario of scenario s.

For example, in subtree $\mathcal{T}_a$ in Figure 4.1, scenario 1 is the branch scenario for scenarios 2 and 3 with branch stage 4. Scenario 1 is also the branch scenario for scenarios 4 and 6 with branch stage 3. Scenario 5 branches from scenario 1 in stage 3 and it branches from scenario 4 in branch stage 4. Therefore, scenario 4 is the branch scenario of scenario 5.

In this manner, except for one scenario (scenario 1) that branches from the root node, each scenario s in the scenario tree $\mathcal{T}$ has one branch scenario s_b^s and a branch stage τ^s. The NNT constraints (2.2) can be equivalently expressed with a branch-type representation as

$$x_t^s = x_t^{s_b^s}, \quad \forall t = 1, ..., \tau^s - 1, \forall s \in S \setminus \{1\}. \tag{4.11}$$

In subtree $\mathcal{T}_a$ in Figure 4.1, the NNT constraints in stage 2 are

$$x_2^2 = x_2^1, \tag{4.12a}$$
$$x_2^3 = x_2^1, \tag{4.12b}$$
$$x_2^4 = x_2^1, \tag{4.12c}$$
$$x_2^5 = x_2^4, \tag{4.12d}$$
$$x_2^6 = x_2^1. \tag{4.12e}$$

The NNT constraints in stage 3 are

$$x_3^2 = x_3^1, \tag{4.13a}$$
$$x_3^3 = x_3^1, \tag{4.13b}$$
$$x_3^5 = x_3^4. \tag{4.13c}$$

In the branch-type representation, the scenarios in each bundle are linked through the branch scenario(s). The resulting NNT constraint matrix is sparse, with two nonzero entries in each constraint, and includes no redundancy. Note also that we may assume that the scenarios are numbered so that

$$s > s_b^s, \quad \forall s \in S. \tag{4.14}$$

Again, we will need this property in the upper bound procedures in Chapter 5.

4.2.4 Nonanticipativity Constraint Representation Types: Summary

Ease of these three representations form sparse NNT constraint matrices in the resulting DEPs. The sibling-type represents the NNT condition with a redundant constraint in each scenario bundle. The node-type representation does not have the redundant nonanticipativity constraints, but increases column space. The branch-type is the most compact representation with the least number of NNT constraints, and includes no redundancy.

If we compare (4.6) and (4.7) with (4.12) and (4.13), we observe that, after removing redundancy, the sibling-type representation is similar to the branch-type representation in the sparsity and compactness of the constraint matrix. The difference between these two representations lies in the way that scenarios in a scenario bundle are tied to each other. In the sibling-type representation, scenarios are tied to each other in a chain and the NNT constraints are almost evenly distributed among scenarios within each scenario bundle. In each scenario bundle, except for the scenarios with the lowest and the largest indices in the bundle, each scenario links to two other scenarios: its sibling and inverse sibling scenarios. In the branch-type representation, scenarios are linked via the branch scenario(s). Each scenario links with its branch scenario with one NNT constraint vector in each bundle, while the branch scenario(s) are heavily tied to all other scenarios in this bundle. Also, a scenario may have different sibling or inverse sibling scenarios in different stages, while it has only one branch scenario in the scenario tree.

The consequences of the relaxation of these three types of NNT constraints on the information structure as well as their impact on the computational requirements of solving the relaxed DEP are different, as we will discuss later in this dissertation.

4.3 Lower Bounds by Relaxation

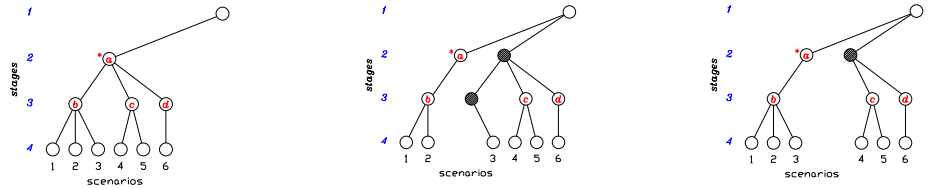
As with the relaxation of any constraints in minimization linear programs, lower bounds on the optimal objective values can be obtained by relaxing NNT constraints. Based on the representation of the NNT constraints used, the relaxation of a set of NNT constraints has different implications on the information structure of MS-SLP problems. Moreover, different ways to choose the set of NNT constraints to relax also have different implications on the information/decision structure. In the following subsections, we examine different schemes to choose the set of NNT constraints to relax, and discuss the implications of NNT representation types within each relaxation approach.

4.3.1 Relaxation by Forward Scenarios

We define a *forward scenario* (x^s_{t+}) as the continuation of x^s from stage t to T, $x^s_{t+} = \{x^s_\tau\}^T_{\tau=t}$. When we relax the NNT constraints of a forward scenario x^s_{t+}, we eliminate all NNT constraints associated with the forward scenario $\{x^s_\tau\}$, $\tau = t, ..., T$ in the DEP.

Sibling-type

After removing the redundant NNT constraints (for example, (4.4a), (4.5a), (4.5d), and (4.5f), as in subtree $\mathcal{T}_a$), in the sibling-type representation, the scenarios within each bundle, originally linked in a loop, are linearly linked as a chain. Eliminating the NNT constraints between the forward scenario decision vectors of a scenario and its inverse sibling scenario from a given stage will partition the scenario bundle into subsets. In effect, the scenario subtree will be split into two subtrees. Figure 4.2b illustrates the resulting structures after eliminating the NNT constraints associated with the forward scenario x^3_{2+}, that is, (4.6b) and (4.7b). Similarly, Figure 4.2c illustrates the effect of eliminating the NNT constraints associated with the forward scenario x^4_{2+}, (4.6c).



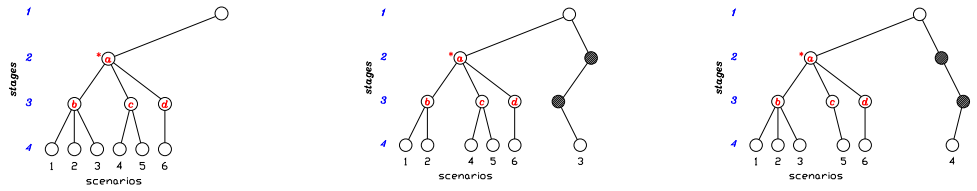
(a) Before elimination (b) After eliminating (4.6b) and (4.7b) (c) After eliminating (4.6c)

Figure 4.2: Relaxation by Forward Scenarios: Sibling-type

Node-type

In the node-type representation, if we eliminate the NNT constraints between the forward scenario from a given stage t of a scenario s and that of the bundle nodes

that scenario s passes through, scenario s will be separated from other scenarios in its bundles (in stages t through T). Taking scenario 3 in Figure 4.1 as an example, if we eliminate its NNT constraints from stage 2, that is, (4.9c) and (4.10c), scenario 3 will be separated from the bundle nodes 'a' in stage 2 and 'b' in stage 3. This is illustrated in Figure 4.3b. Similarly, Figure 4.3c depicts the separation of scenario 4 from bundles nodes 'a' and 'c', after eliminating the NNT constraints associated with x^4_{2+}, (4.9d) and (4.10d).



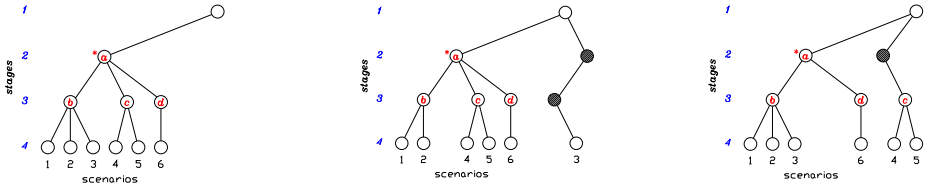
(a) Before elimination (b) After eliminating (4.9c) and (4.10c) (c) After eliminating (4.9d) and (4.10d)

Figure 4.3: Relaxation by Forward Scenarios: Node-type

Branch-type

In the branch-type NNT representation, scenarios in a scenario bundle are linked through their branch scenarios. If we eliminate the NNT constraints associated with the forward scenario, x^s_{t+}, scenario s is separated from its branch scenario s^s_b from stage t. Consequently, we separate scenario s from its scenario bundles from stage t. This is illustrated in Figure 4.4b, where the NNT constraints ((4.12b) and (4.13b)) associated with the forward scenario x^3_{2+} are eliminated.

On the other hand, as shown in Figure 4.4c, when a scenario s is separated from its branch scenario from a given stage t, all the scenario(s) that branch from scenario s remain(s) tied to it. Consequently, a subtree rooted at a node in stage t may be separated from the original subtree.

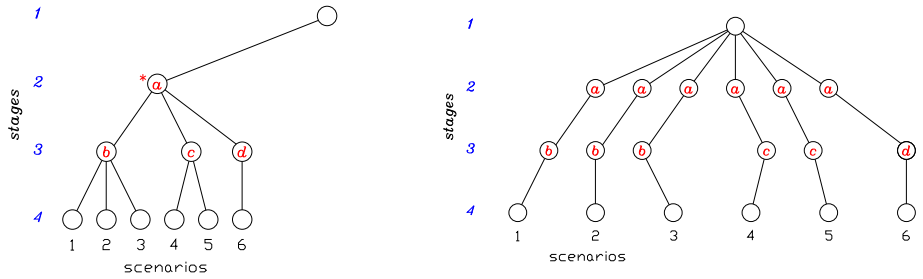


(a) Before elimination (b) After eliminating (4.12b) and (4.13b) (c) After eliminating (4.12c) and (4.13b)

Figure 4.4: Relaxation by Legs: Branch-type

Relaxation of Non-First-Stage Nonanticipativity Constraints

As a special case of relaxation by forward scenarios, we enforce NNT constraints of the first stage decisions for all scenarios, and relax the remaining NNT constraints, as depicted in Figure 4.5. If the problem has complete recourse, then this relaxation yields an implementable first-stage solution as well as a lower bound on the optimal objective value.



(a) Before relaxation (b) After relaxation

Figure 4.5: Non-First-Stage Type Relaxation

4.3.2 Relaxation by Scenario-Stage Pairs

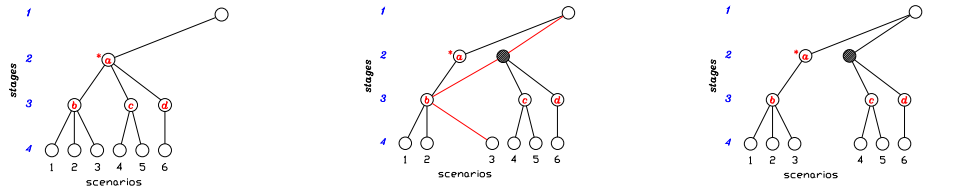
We define a *scenario-stage pair* (x_t^s) as the vector of decision variables of scenario s in stage t. By relaxing the NNT constraints associated with a scenario-stage pair x_t^s, we eliminate the NNT constraints from the DEP that are enforced on the decision vector only in stage t of scenario s.

Sibling-type

Again, after eliminating the redundant NNT constraints with the sibling-type representation, the scenarios in each scenario bundle are linearly linked in a chain. Eliminating the NNT constraints associated with a scenario-stage pair x_t^s will partition the scenarios within the bundle into two subsets in stage t. However, unlike relaxation by forward scenarios, scenario s may remain tied to its inverse sibling scenarios by the NNT constraints in future stages.

In Figure 4.6b, after eliminating the NNT constraints associated with the scenario-stage pair x_2^3, (4.6b), scenarios in bundle "a" are partitioned into two scenario subsets in stage 2, namely 1-2 and 3-4-5-6. However, the NNT constraints associated with scenario 3 in stage 3, (4.7b), keep scenarios 2 and 3 tied to each other from stage 3, as in bundle "b".

On the other hand, there are no NNT constraints between scenario 3 and 4 after stage 2. Therefore, after eliminating the NNT constraints associated with scenario-stage pair x_2^4, (4.6c), scenario bundle "a" will be split into two subtrees from stage 2, as illustrated in Figure 4.6c.



(a) Before elimination (b) After eliminating (4.6b) (c) After eliminating (4.6c)

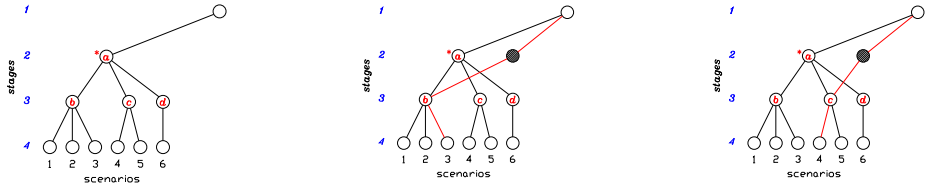
Figure 4.6: Relaxation by Scenario-Stage Pairs: Sibling-type

Node-type

In the node-type representation, eliminating the NNT constraints between a scenario-stage pair x_t^s and the state vector of the corresponding bundle node $m(s,t)$ will separate scenario s from the bundle in stage t. However, this scenario may still be constrained by NNT constraints in future stages so that it may not be completely

separated from its bundles since stage t. This is illustrated in Figure 4.7b. After we eliminate the NNT constraints between scenario-stage pair x_2^3 and the state vector of bundle node "a", i.e., (4.9c), we separate scenario 3 from bundle "a" in stage $t = 2$. But constraint (4.10c) in stage 3 prevents scenario 3 from separating from bundle node "b".

Figure 4.7c illustrates a similar example of the elimination of the NNT constraints associated with scenario-stage pair x_2^4, (4.9d).



(a) Before elimination (b) After eliminating (4.9c) (c) After eliminating (4.9d)

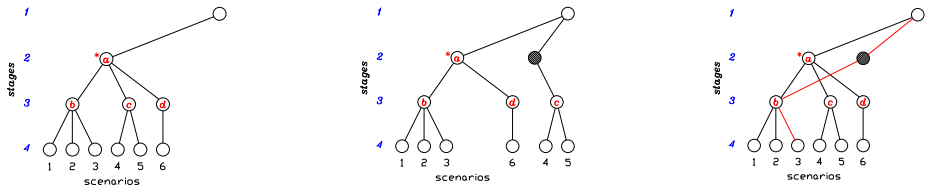
Figure 4.7: Relaxation by Scenario-Stage Pairs: Node-type

Branch-type

Similar to relaxation by forward scenarios, in the branch-type representation, eliminating the NNT constraints associated with a scenario-stage pair x_t^s will break the tie between scenario s and its branch scenario s_b^s in stage t. If t is the last stage before scenario s branches from scenario s_b^s, i.e. $t = \tau^s - 1$, eliminating the NNT constraints associated with the scenario-stage-pair x_t^s splits the subtree of scenario s and scenarios branching from it out of the scenario bundle. This is illustrated in Figure 4.8b, where the NNT constraints associated with scenario-stage pair x_2^4 are eliminated. In this example, scenario 4 branches from scenario 1 in stage 3. Eliminating the NNT constraints associated with scenario 4 in stage 2 separates the subtree of scenario 4 and scenario 5 from the original scenario bundle in stage 2.

However, if $t < \tau^s - 1$, that is, if scenario s remains tied to its branch scenario s_b^s in stages after t, eliminating the NNT constraints associated with scenario-stage pair x_t^s will only separate scenario s from its scenario bundle in stage t, but will not make it completely separable after stage t. This is illustuated in Figure 4.8c,

where the NNT constraints associated with scenario-stage pair x_2^3 are eliminated. In this example, scenario 3 branches from scenario 1 in stage 4. Eliminating the NNT constraints of scenario 3 in stage 2 will only make it separable in stage 2. Scenario 3 is still tied to its branch scenario (scenario 1) in stage 3.



(a) Before elimination (b) After eliminating (4.12c) (c) After eliminating (4.12b)

Figure 4.8: Relaxation by Scenario-Stage Pairs: Branch-type

4.3.3 Relaxation by Individual Nonanticipativity Constraints

All of the NNT relaxation schemes discussed so far have considered the structure associated with scenario-stage indices. We may also randomly choose a subset of NNT constraints for elimination, without regard to this structure. However, the implications of this type of relaxation is elusive to us.

4.4 Computational Experiment

In order to test the effectiveness and the quality of the lower bounds obtained via the partial relaxation of the NNT constraints from the DEPs as discussed in the previous section, we design and implement a series of computational experiments on a class of test instances, namely, SGPF.

4.4.1 Test Instances

SGPF – A Financial Portfolio Problem

SGPF test instances are a class of financial portfolio problem instances available at the Stochastic Programming Community Home Page (http://stoprog.org). The SGPF model deals with financial portfolio decisions - borrowing and lending bonds

with different maturity dates to maximize the expected profit over a finite time horizon. The portfolio decisions and the resulting profit are influenced by uncertain interest rates and changes in bond prices. The randomness appears in the objective and right-hand-side coefficients.

The salient features of the test instances are described in Table 4.1. SGPFmYn represents an n-stage instance of the SGPF model with a maximum maturity of m years. We include two subclasses of the test instances, namely, SGPF3Yn and SGPF5Yn, $n = 3, 4, 5$.

The numbers of stages of the test instances are listed in the column labeled "Stgs", and the numbers of scenarios in the test instances are listed in the column labeled "Scens". For each test instance, the number of columns/decision variables and the number of rows/constraints of each scenario problem are listed in the columns labeled "Cols" and "Rows", respectively, under label "Scenario".

The optimal objective values for these test instances are known and are listed in the column labeled "DEP" under "Optimal Objective Value". The wait-and-see lower bound values and the relative gaps between the wait-and-see lower bound values and the optimal objective values are also listed in the columns labeled "Wait-n-see" and "Gap", respectively.

			Scenario		Optimal Objective Value		
Instance	Stgs	Scens	Cols	Rows	DEP	Wait-n-see	Gap
SGPF3Y3	3	25	189	116	-2,967.91	-3,088.91	-4.1%
SGPF3Y4	4	125	240	155	-3,994.18	-4,136.39	-3.6%
SGPF3Y5	5	625	291	194	-5,172.07	-5,393.50	-4.3%
SGPF5Y3	3	25	297	188	-3,027.60	-3,362.86	-11.1%
SGPF5Y4	4	125	376	251	-4,031.30	-4,468.85	-10.9%
SGPF5Y5	5	625	455	314	-5,201.20	-5,765.22	-10.8%

Table 4.1: SGPF Test Instances

4.4.2 Experiment Design

In our computational experiment, we solve the DEPs with partial elimination of the NNT constraints (relaxed DEPs), varying the NNT representations, NNT relaxation

schemes, and relaxation levels (the fraction of NNT constraints that are eliminated) to observe the impact of these factors on the computational requirements and the quality of the lower bounds.

Nonanticipativity Constraint Relaxation Schemes

For each test instance, we test four NNT relaxation schemes (non-first-stage relaxation, relaxation by forward scenarios, relaxation by scenario-stage pairs, and relaxation by individual NNT constraints) to obtain lower bounds on the optimal objective values.

Nonanticipativity Constraint Representations

We compare three NNT representations (branch, node, sibling) in our experiment.

Nonanticipativity Constraint Relaxation Levels

In the non-first-stage relaxation, we keep the NNT constraints associated with the first stage decisions in the relaxed DEP while eliminating the NNT constraints in all other stages. Once an instance is given, the subset of NNT constraints to be eliminated from the relaxed DEP is fixed.

In all other cases, we select a subset of NNT constraints to be eliminated from the DEP to form the relaxed DEP. We want to explore how the fractions of NNT constraints to be eliminated will affect the lower bound quality. We define the relaxation level $\mathcal{P}_{rlx}$ as the fraction of NNT constraints to be eliminated from the DEP. Four relaxation levels are tested in our experiment: $\mathcal{P}_{rlx}$ = 25%, 50%, 75%, and 90%.

Factor	Num	Description
NNT representation	3	branch, node, and sibling
Deterministic relaxation scheme[1]	1	non-first-stage
Statistical relaxation scheme	3	forward scenarios, scenario-stage pairs, and individual NNT constraints
Relaxation level ($\mathcal{P}_{rlx}$)	4	25%, 50%, 75%, and 90%

[1] The relaxation level does not apply to non-first-stage relaxation.

Table 4.2: Experiment Design on Lower Bounds

Given a relaxation level, $\mathcal{P}_{rlx}$, we need a way to identify a subset of NNT constraints to be eliminated from the DEP. This subset is randomly selected as follows:

Relaxations by individual NNT constraints are accomplished by defining a Bernoulli random variable for each individual NNT constraint. Let $\tilde{\xi}_i$ represent the random variable associated with the i^{th} NNT constraint. Then $\{\tilde{\xi}_i\}$ are iid, where $p\{\tilde{\xi}_i = 1\} = \mathcal{P}_{rlx}$. To select the NNT constraints for relaxation, we generate observations of $\{\tilde{\xi}_i\}$. The NNT constraints for which $\xi_i = 1$ are eliminated from the DEP, while the rest are retained.

To relax the NNT constraints by scenario-stage pairs, we repeat this process, except that the Bernoulli random variables are assigned to the scenario-stage pairs rather than to individual NNT constraints. Finally, to relax the NNT constraints by forward scenarios, we select scenario-stage pairs in this fashion, but eliminate the entire set of scenario-stage pairs of each scenario associated with its forward scenario.

Because the manner in which we select the subset of NNT constraints that are relaxed is statistical in nature, the percentage of NNT constraints that are actually eliminated from the DEP is only an approximation of $\mathcal{P}_{rlx}$ and is dependent on the specific random stream employed. Replications are needed as in any statistical experiment design.

Replications

The actual subsets of NNT constraints to be eliminated from the DEPs are selected randomly. As a result, the specific subset of NNT constraints eliminated may have an impact on the lower bound quality and computational requirement. To account for this, we run 10 independent replications for each combination of NNT representation, NNT constraint relaxation scheme (except for non-first-stage relaxation), and relaxation level.

4.4.3 Computational Environment

All the implementations and data manipulations are programmed in C++. ILOG CPLEX 8.1 C Callable Library is encapsulated in a C++ class, which serves as the standard optimization library. We use dual simplex method to solve all the relaxed DEPs.

The experiment is run on two Sunw Sun-Fire-280R Sun servers with identical configurations: 2xUltraSPARCIII+ processors at 900 MHz, 4GB RAM, and Solaris 9 operating system. Best effort has been put to avoid interference with other users on these machines to ensure accurate recording of CPU times.

4.5 Experiment Result Analysis

4.5.1 Deterministic Results

We begin with the solution of the full DEPs for each of the three nonanticipativity constraint representations. The results of this preliminary investigation can be found in Table 4.3.

For each nonanticipativity (NNT) constraint representation, we list the total number of columns, rows, and nonzeros in the DEP in columns labeled “Cols”, “Rows”, and “Nzs” under “Total”. We also list the number of nonanticipativity columns, rows, and nonzeros under in the corresponding columns under “NNT”. The NNT column refers to the number of additional variables introduced for the

Instance	NNT-type	NNT			Total			Optimal	CPU
		Cols	Rows	Nzs	Cols	Rows	Nzs	Value	(sec)
SGPF3Y3	branch	0	3,108	6,216	4,725	6,008	16,066	-2,967.91	0.28
	node	342	3,450	6,900	5,067	6,350	16,750	-2,967.91	0.41
	sibling	0	3,108	6,216	4,725	6,008	16,066	-2,967.91	0.33
SGPF3Y4	branch	0	22,008	44,016	30,000	41,383	109,766	-3,994.18	10.39
	node	1,617	23,625	47,250	31,617	43,000	113,000	-3,994.18	17.40
	sibling	0	22,008	44,016	30,000	41,383	109,766	-3,994.18	7.21
SGPF3Y5	branch	0	142,008	284,016	181,875	263,258	695,266	-5,172.07	810.62
	node	7,992	150,000	300,000	189,867	271,250	711,250	-5,169.34	1,072.15
	sibling	0	142,008	284,016	181,875	263,258	695,266	-5,172.07	314.57
SGPF5Y3	branch	0	4,916	9,832	7,425	9,616	25,682	-3,027.60	0.52
	node	534	5,450	10,900	7,959	10,150	26,750	-3,027.60	0.78
	sibling	0	4,916	9,832	7,425	9,616	25,682	-3,027.60	0.51
SGPF5Y4	branch	0	34,616	69,232	47,000	65,991	174,982	-4,031.30	23.22
	node	2,509	37,125	74,250	49,509	68,500	180,000	-4,030.94	45.47
	sibling	0	34,616	69,232	47,000	65,991	174,982	-4,031.30	22.15
SGPF5Y5	branch	0	222,616	445,232	284,375	418,866	1,106,482	-5,201.20	2,316.87
	node	12,384	235,000	470,000	296,759	431,250	1,131,250	-5,201.20	3,047.71
	sibling	0	222,616	445,232	284,375	418,866	1,106,482	-5,201.20	722.18

Table 4.3: SGPF: Results on DEPs

representation of the nonanticipativity constraints, which appears only in the node-type NNT representation and corresponds to the number of state variables required. The optimal objective values and solution times (in seconds) are reported in columns "Optimal Value" and "CPU".

Combining Table 4.1 and Table 4.3, we observe that the DEP problem sizes increase dramatically as the number of stages and the number of scenarios increase. The NNT constraints constitute more than half of the total number of constraints for these test instances and the NNT nonzeros constitute more than 1/3 of the total number of nonzeros in the DEPs.

Because of the internal optimization mechanism in the CPLEX solver, in instances SGPF3y5 and SGPF5y4, the optimal objective values obtained from DEPs with the node-type representation are slightly higher than the optimal objective values from DEPs with the branch- and sibling-type representations. The opti-

mal objective values agree with regard to the NNT representations in all other test instances.

As the number of stages and scenarios increases, the time required to obtain the optimal solution for the DEP increases dramatically. In small test instances such as SGPF3y3 and SGPF5y3, the solution times are so short that the interference of computer system on the recorded CPU solution times may be significant. However, we note that in every case, the DEP with the node-type representation consistently takes the most time to solve, while the DEP with the sibling-type representation requires the least times to solve. These results suggest the use of the sibling-type representation if we want to solve the DEP directly.

4.5.2 Relaxation Schemes

Non-first-stage Relaxation

Let us first exam the special case where we enforce NNT constraints for all the scenarios in the first-stage and eliminate those in all other stages. In Figure 4.9, the lower bound gaps, $\mathcal{G}_{lb}$, are relative to the optimal objective values of the corresponding test instances, that is,

$$\mathcal{G}_{lb} = \frac{f_{lb} - f_{opt}}{|f_{opt}|}, \tag{4.15}$$

where f_{lb} are the lower bounds obtained from solving the relaxed DEPs and f_{opt} are the optimal objective value of the DEPs. To facilitate benchmark comparisons, the lower bound gaps of the wait-and-see solutions are plotted in the same figure.

Recall that in the naming convention SGPFmYn, m corresponds to the maximum maturity of m years and n corresponds to the number of stages. For each value of m ($m = 3, 5$), as n ($n = 3, 4, 5$) increases, the number of stages and scenarios increase, and the lower bound gaps for this relaxation increase accordingly. For example, in SGPF3Y3, the lower bound value is the same as the optimal objective value. In SGPF3y5, when the number of stages increases to five, the lower bound gap increase to 3%, close to the lower bound gap of the wait-and-see solutions (4%).

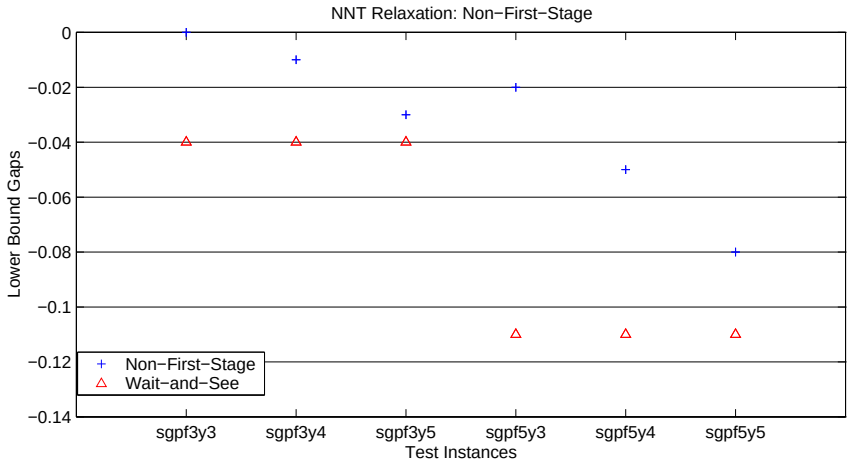


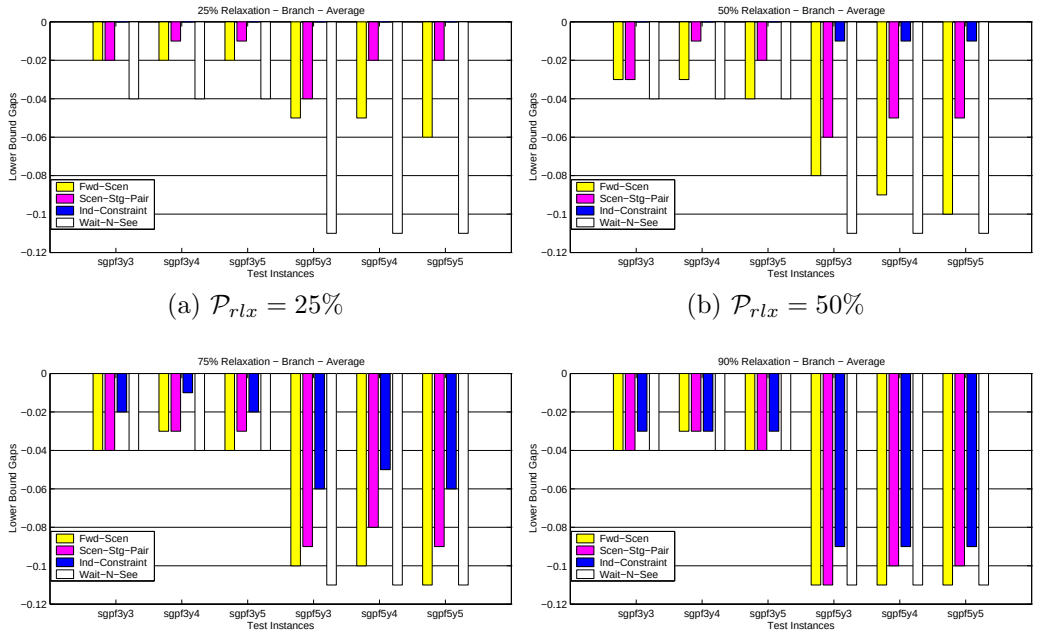
Figure 4.9: Lower Bound Gaps: Non-first-stage Type Relaxation

We observe a similar trend among the test instances SGPF5Yn, $n = 3, 4, 5$. The computational results are not unexpected: as the number of stages increases, the enforcement of NNT constraints only in the first stage reflects less information regarding the original nonanticipativity structure. Therefore, the lower bound gap increases accordingly.

Comparing Relaxation Schemes

In order to compare the three relaxation schemes (relaxation by forward scenarios, relaxation by scenario-stage pairs, and relaxation by individual NNT constraints), we fix the NNT representation as the branch-type representation (We will discuss the comparisons among the three NNT representations in the next section). In Figure 4.10, for each instance, we plot the average lower bound gap obtained from the ten replications of each relaxation scheme. We also include the lower bound gaps of the wait-and-see solutions as benchmarks. The lower bound gaps of each relaxation level ($\mathcal{P}_{rlx} = 25\%, 50\%, 75\%, 90\%$) are plotted in a sub-figure.

We observe that the lower bounds obtained from relaxation by forward scenarios are loose in general. When $\mathcal{P}_{rlx} = 25\%$, the lower bound gaps obtained from relaxation by forward scenario are about half of the lower bound gaps of the wait-and-see solutions. When $\mathcal{P}_{rlx} >= 50\%$, the lower bound gaps of this relaxation scheme are



(a) $\mathcal{P}_{rlx} = 25\%$ (b) $\mathcal{P}_{rlx} = 50\%$

(c) $\mathcal{P}_{rlx} = 75\%$ (d) $\mathcal{P}_{rlx} = 90\%$

Figure 4.10: Lower Bound Gaps – Comparing Relaxation Schemes

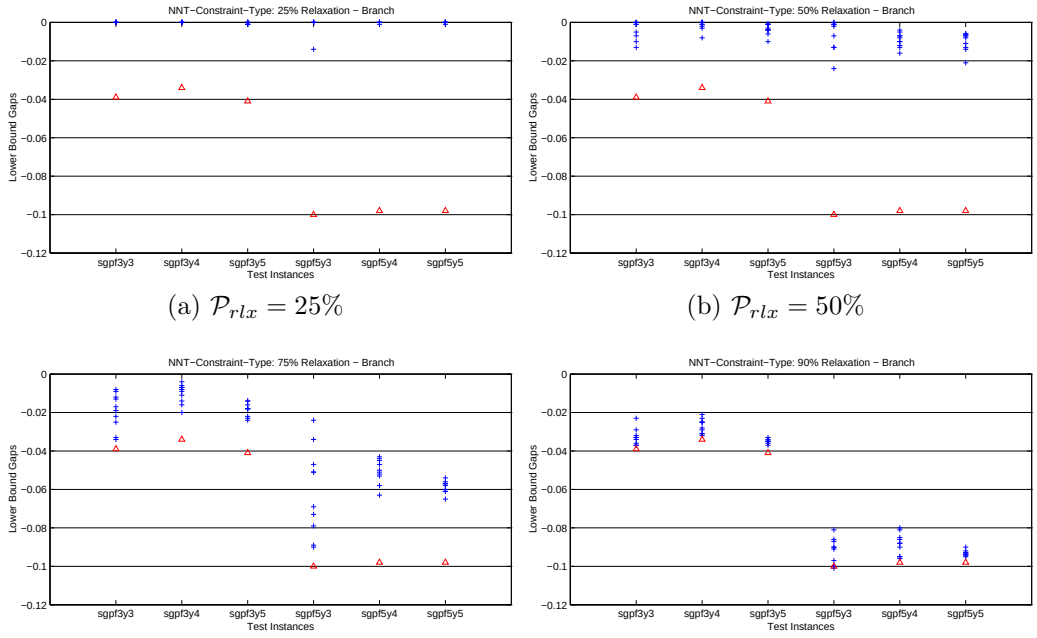
close to those of the wait-and-see solutions. The results suggest that relaxation by forward scenario can hardly provide better lower bounds on the optimal objective values than the wait-and-see solutions.

Comparing with relaxation by forward scenarios, relaxation by scenario-stage pairs results in smaller lower bound gaps, but the lower bounds obtained are still loose even when $\mathcal{P}_{rlx} = 25\%$, where only a small portion of the NNT constraints are eliminated from the DEPs.

On the other hand, when $\mathcal{P}_{rlx} = 25\%$, the lower bound gaps obtained from relaxation by individual NNT constraints are close to or equal to zero. With about half of the NNT constraints eliminated from the DEPs ($\mathcal{P}_{rlx} = 50\%$), the lower bound gaps are still very tight in instances SGPF3Yn and are small in instances SGPF5Yn, $n = 3, 4, 5$. The lower bounds become loose when $\mathcal{P}_{rlx} \geq 75\%$.

To further study the variance of the lower bounds obtained from relaxation by individual NNT constraints, in Figure 4.11, we plot the lower bound gaps of the ten replications with this relaxation scheme. Again, we fix the NNT representation as

the branch-type.



(a) $\mathcal{P}_{rlx} = 25\%$ (b) $\mathcal{P}_{rlx} = 50\%$

(c) $\mathcal{P}_{rlx} = 75\%$ (d) $\mathcal{P}_{rlx} = 90\%$

Figure 4.11: Lower Bound Gaps – Relaxation by Individual NNT Constraints

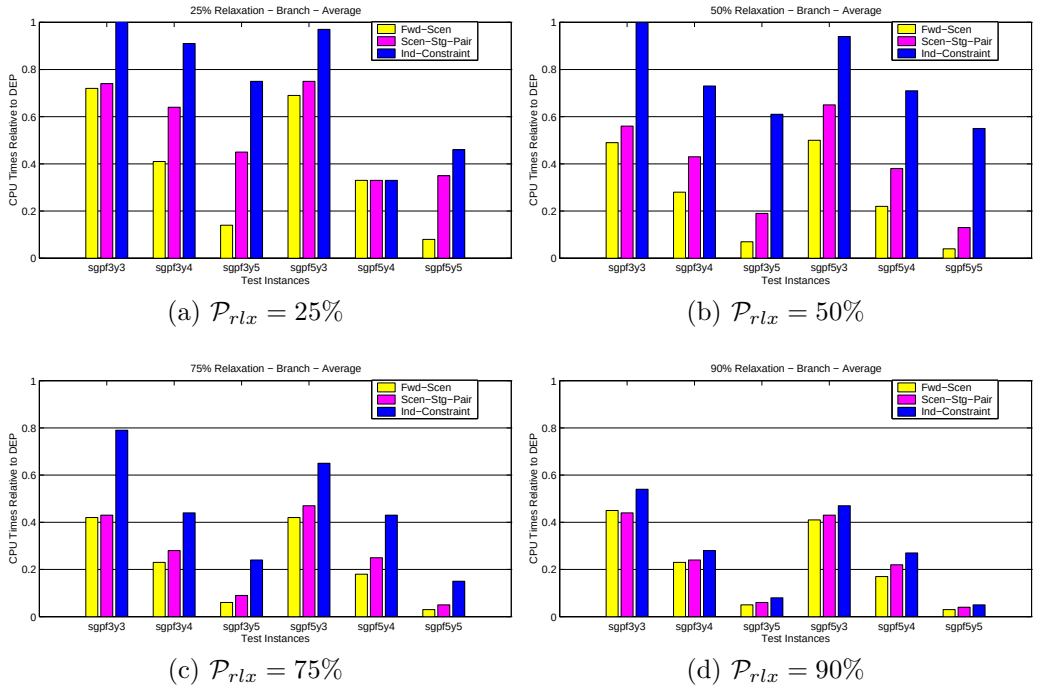
When $\mathcal{P}_{rlx} = 25\%$, all but two replications in SGPF5Y3 reach optimality without the complete set of NNT constraints. When $\mathcal{P}_{rlx} = 50\%$, we are still able to obtain small lower bound gaps even in the worst case replications (1% for SGPF3Yn and 2% for SGPF5Yn), which are about 1/4 of the lower bound gaps of the wait-and-see solutions. The lower bounds become loose when $\mathcal{P}_{rlx} \geq 75\%$

We observe similar relative comparison results among the three relaxation schemes when we fix the NNT representation type to the node-type or sibling-type.

Next, we examine the times required to solve the various relaxed DEPs. In Figures 4.12 and 4.13, we plot normalized solution times. That is, working once again with the branch-type representation, for each test instance, we divide the times to solve the relaxed DEPs by the time to solve the DEP of the instance with the branch-type representation. This offers an opportunity to observe the relative time required to obtain the lower bounds.

In Figure 4.12, we plot the average (of ten replications) normalized lower bound

times of the three relaxation schemes with each relaxation level presented as a subplot.



(a) $\mathcal{P}_{rlx} = 25\%$ (b) $\mathcal{P}_{rlx} = 50\%$

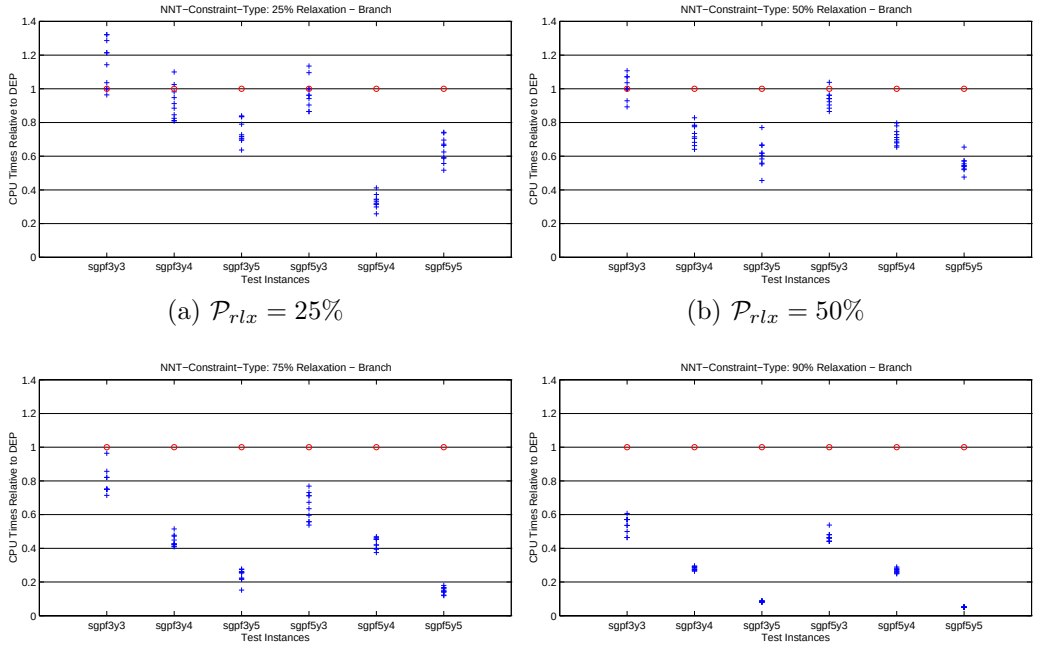
(c) $\mathcal{P}_{rlx} = 75\%$ (d) $\mathcal{P}_{rlx} = 90\%$

Figure 4.12: Solution Times – Comparing Relaxation Schemes

We note that test instances SGPF3y3 and SGPF5y3 are small, and the times required to solve their DEPs and relaxed DEPs are small enough that they may be impacted by the computer system variability. Therefore, we may not be able to draw meaningful conclusions based these solution times.

Among the three relaxation schemes, relaxation by individual constraints requires the longest times to obtain lower bounds, while relaxation by forward scenarios requires the least times. We have observed that for the SGPF test instances, we can obtain reasonably close lower bounds with relaxation by individual constraints and with relaxation level up to 50%. Except for the two small test instances (SGPF3y3 and SGPF5y3), at 50% relaxation level, the relaxed DEPs with individual NNT constraint type take about 60% to 70% of the DEP solution times on average.

To study the variability of the times to obtain lower bounds with relaxation by individual NNT constraints, in Figure 4.13, we plot the normalized times of this relaxation scheme for the 10 replications with branch-type NNT representation.



(a) $\mathcal{P}_{rlx} = 25\%$ (b) $\mathcal{P}_{rlx} = 50\%$

(c) $\mathcal{P}_{rlx} = 75\%$ (d) $\mathcal{P}_{rlx} = 90\%$

Figure 4.13: Solution Times – Relaxation by Individual NNT Constraints

Summary

Based on the observation of the our computational results, relaxed DEPs with either relaxation by forward scenarios or by scenario-stage pairs are usually not able to provide close lower bounds on the optimal objective values of MS-SLP instances. Such structured elimination of NNT constraint vectors usually results in loose lower bounds. The logical restructuring of the NNT structure contained in the scenario tree leads to a severe loss of nonanticipativity restrictions which in turn leads to loose lower bounds.

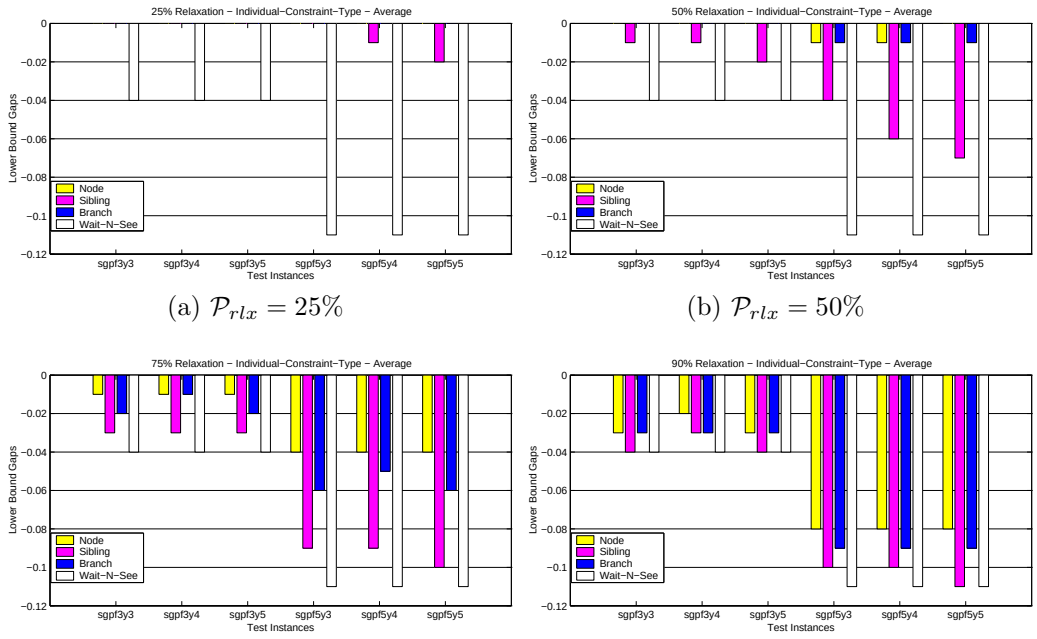
On the other hand, in relaxation by individual NNT constraints, the NNT constraints are eliminated randomly from the DEPs without regard to the stage and scenario indices. The nonanticipativity condition may be maintained to some extent

indirectly by the scenario constraints (2.3a). As a result, even when a large percentage (50%) of NNT constraints are eliminated from the DEP, we are still able to obtain reasonably tight lower bounds, with a saving of 30 - 40 % on solution times compared with solving DEPs directly to optimality.

4.5.3 Comparing Nonanticipativity Constraint Representations

In this section, we focus on the individual NNT constraint type relaxation scheme, and examine the impact of the NNT representations on the quality of the lower bounds and on the times required to obtain these bounds.

In Figure 4.14, we plot the average (of the 10 replications) lower bound gaps for the branch-, node-, and sibling-type representations. Again, we plot the data for each relaxation level ($\mathcal{P}_{rlx} = 25\%, 50\%, 75\%, 90\%$) in a sub-figure.



(a) $\mathcal{P}_{rlx} = 25\%$ (b) $\mathcal{P}_{rlx} = 50\%$

(c) $\mathcal{P}_{rlx} = 75\%$ (d) $\mathcal{P}_{rlx} = 90\%$

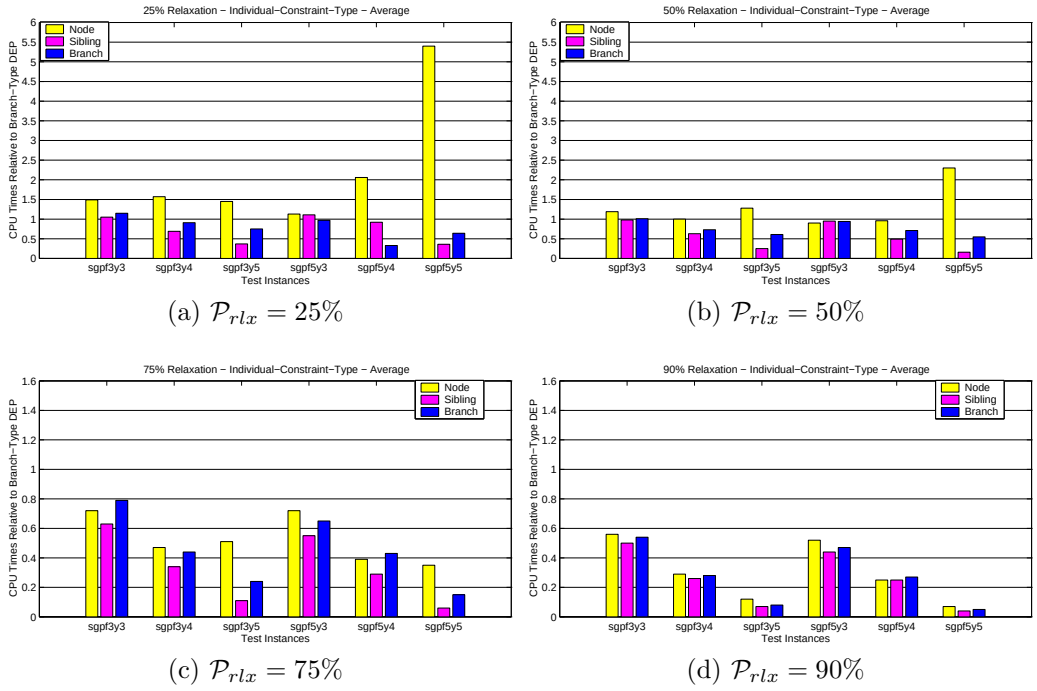
Figure 4.14: Lower Bound Gaps – Comparing NNT Representations

We observe that the node- and branch-type nonanticipativity constraints consistently provide better bounds than the sibling-type nonanticipativity constraints.

We conjecture that this may be due to the linear linking of scenarios in each bundle employed by the sibling-type representation, which tends to result in separation of scenario bundles in subsets in the relaxed DEP.

We note also that the node- and branch-type representations provide lower bounds with similar quality. Even when $\mathcal{P}_{rlx}$ is as high as 50%, the lower bounds obtained with these NNT representations provide bounds that are extremely tight.

In Figure 4.15, we plot the CPU times required to solve the relaxed DEPs. As before, we fix the relaxation scheme as relaxation by individual nonanticipativity constraints. The solution times of relaxed DEPs are normalized to the solution times of the DEPs with the branch-type NNT constraints of the corresponding test instance. We plot the data of each relaxation level in a sub-figure. Note that the scale of the normalized time in Figures 4.15a and 4.15b differs from that in Figures 4.15c and 4.15d.



(a) $\mathcal{P}_{rlx} = 25\%$ (b) $\mathcal{P}_{rlx} = 50\%$

(c) $\mathcal{P}_{rlx} = 75\%$ (d) $\mathcal{P}_{rlx} = 90\%$

Figure 4.15: Solution Times – Comparing NNT Representations

We observe that the times required to solve the relaxed DEPs with node-type

NNT constraints are almost consistently the largest among these three NNT constraint representations. With relaxation level 25% and 50%, the relaxed DEPs with node-type NNT constraints even take much longer to solve than the DEPs with branch type NNT constraints (normalized times > 1).

Similar to what we have observed in the full DEPs, the relaxed DEP with the sibling-type representation requires the least times to solve.

Summary

Our computational results indicate that relaxation of the DEPs with the sibling-type NNT constraints does not yield good lower bounds. Although we can obtain reasonably good lower bounds from solving the relaxed DEPs with both the node-type and branch-type NNT constraints with relaxation level up to 50%, the relaxed DEPs with the node-type NNT constraints take significantly longer to solve. Sometimes it takes longer to solve a relaxed DEP with the node-type NNT constraints than to solve the DEP with the branch-type NNT constraints directly to optimality.

4.5.4 Relaxation Levels

Based on the data plotted in Figure 4.10 to Figure 4.15, where lower bound gaps or normalized solution times of each relaxation level is plotted in a sub-figure, we observe that we are usually not able to obtain reasonably tight lower bounds with relaxation level 75% and 90%, regardless of the relaxation scheme and the NNT representation. Eliminating a large percentage of NNT constraints results in severe loss of the original nonanticipativity structure, which in turn results in loose lower bounds.

By relaxing a small portion (25%) of NNT constraints, we are usually able to obtain tight lower bounds due to a well maintained nonanticipativity structure, but we usually do not save much on the solution times. Eliminating a moderate percentage (up to 50%) of NNT constraints enables us to maintain the nonanticipativity structure to some extent, so that we still obtain reasonably tight lower bounds, while saving on solution time compared to solving the DEPs directly to optimality.

4.6 Conclusions

In this chapter, we experiment on the impact of three NNT representations, four relaxation schemes, and four relaxation levels on the lower bound quality and the times required to obtain these bounds. The computational results suggest that:

- Relaxation by forward scenarios and by scenario-stage pairs do not provide good lower bounds in general, due to the logical restructuring of the nonanticipativity structure of the MS-SLP. On the other hand, relaxation by individual NNT constraints is able to provide reasonably good lower bounds with a moderate elimination of NNT constraints.

- Among the three NNT representations, the sibling-type representation takes the least time to obtain the lower bounds, but the quality of the lower bounds are usually loose. Both the node- and branch- type representations can provide tight lower bounds, while the time required to obtain the lower bounds by the branch-type representation is much shorter than the node-type representation.

- Even with the right choice of relaxation scheme and NNT representation, we are usually not able to obtain good lower bounds if eliminating too many NNT constraints. A moderate relaxation level ($\mathcal{P}_{rlx} = 50\%$) is preferred.

As suggested by our computational results, we can obtain reasonably good lower bounds by solving the relaxed DEPs with branch-type NNT constraints, with relaxation by individual NNT constraints, and with relaxation level up to 50%. However, these results may be subject to the limitation of the test instances in our experiment. To thoroughly understand the impact of these factors on the lower bound quality and solution times, more investigation are needed, especially on test instances with different data sets and from diverse models with different characteristics.

CHAPTER 5

Solution Validation in MS-SLP: Upper Bounds

Assume we have obtained a "good" lower bound on the optimal objective value of a MS-SLP instance, denoted as f_{lb}. Given a feasible first-stage solution, $\hat{x}_1$, the gap between the lower bound and the objective function value at $\hat{x}_1$, $f(\hat{x}_1)$, or

$$\mathcal{G}(\hat{x}_1) = \frac{f(\hat{x}_1) - f_{lb}}{|f_{lb}|}, \tag{5.1}$$

is an indication of the "quality" of the solution. Unfortunately, in order to calculate the objective function value $f(\hat{x}_1)$, it is necessary to solve to optimality the problems associated with all the subtrees rooted at the second-stage nodes. These problems are themselves MS-SLP problems with one less stage than the original problem. In this chapter, we explore upper bounding approximations on $f(\hat{x}_1)$ in MS-SLP. We begin by introducing the following notation:

- $\mathcal{T}$ denotes the scenario tree of the original MS-SLP model.
- γ denotes a node in $\mathcal{T}$.
- $\mathcal{T}_\gamma$ denotes the subtree of $\mathcal{T}$ rooted at node γ.
- $t(\gamma)$ denotes the stage of node γ, as in Chapter 2.
- $\mathcal{B}_\gamma$ denotes the scenario bundle of γ, as in Chapter 2. Except for the leaf nodes in $\mathcal{T}$, we assume that $|\mathcal{B}_\gamma| > 1$, for all $\gamma \in \mathcal{T}$.
- Γ_t denotes the set of nodes in stage t of $\mathcal{T}$. $\Gamma_t = \{\gamma \in \mathcal{T} | t(\gamma) = t\}$.
- γ_t denotes a node in stage t of $\mathcal{T}$. That is, $\gamma_t \in \Gamma_t$. In the first stage of $\mathcal{T}$, $|\Gamma_1| = 1$, and γ_1 is the root node of $\mathcal{T}$.
- Γ^s denotes the set of nodes that scenario s passes through.

- Θ_γ denotes the set of γ's children nodes.
- θ_γ denotes a child node of γ, $\theta_\gamma \in \Theta_\gamma$.
- p^s denotes the probability of scenario s. $\sum_{s\in S} p^s = 1$.
- p_n^γ denotes the probability of node γ. $p_n^\gamma = \sum_{s\in\mathcal{B}_\gamma} p^s$. Note that for all $t \in \{1, ..., T\}$, $\bigcup_{\gamma_t\in\Gamma_t} \mathcal{B}_{\gamma_t} = S$, and $\mathcal{B}_{\gamma_t} \bigcap \mathcal{B}_{\gamma_t'} = \emptyset$ if $\gamma_t \neq \gamma_t'$. Therefore, $\sum_{\gamma\in\Gamma_t} p_n^\gamma = 1, \forall t \in \{1, ..., T\}$. At the root node of $\mathcal{T}$, $p_n^{\gamma_1} = 1$.
- $p_{cs}(s|\gamma)$ denotes the conditional probability of scenario s given node γ. That is

$$p_{cs}(s|\gamma) = \begin{cases} \frac{p^s}{p_n^\gamma} = \frac{p^s}{\sum_{s'\in\mathcal{B}_\gamma} p^{s'}}, & \forall s \in \mathcal{B}_\gamma; \\ 0, & \text{otherwise.} \end{cases}$$

- $p_{cn}(\theta|\gamma)$ denotes the conditional probability of node θ given node γ. That is

$$p_{cn}(\theta|\gamma) = \begin{cases} \frac{p_n^\theta}{p_n^\gamma} = \frac{p_n^\theta}{\sum_{\theta'_\gamma\in\Theta_\gamma} p_n^{\theta'_\gamma}}, & \forall \theta_\gamma \in \Theta_\gamma; \\ 0, & \text{otherwise.} \end{cases}$$

- $\underline{\omega}^\gamma$ denotes the realization of $\underline{\tilde{\omega}}_t$ that leads to node γ. Note that $\underline{\omega}_t^s = \underline{\omega}^\gamma$, $\forall s \in \mathcal{B}_\gamma$.
- x^γ denotes a decision at node γ. $x^\gamma \in R^{n_{t(\gamma)}}$.
- $\underline{x}^\gamma$ denotes a decision history up to, but not including, node γ, $\underline{x}^\gamma = \{x_\tau\}_{\tau=1}^{t(\gamma)-1}$.

Let $\hat{x}_1$ be a given first-stage feasible solution. Note that $\Theta_{\gamma_1} = \Gamma_2$. For each $\gamma_2 \in \Gamma_2$, we formulate the MS-SLP problem corresponding to the subtree $\mathcal{T}_{\gamma_2}$ as

$$\text{minimize} \quad \sum_{s\in\mathcal{B}_{\gamma_2}} p_{cs}(s|\gamma_2) \sum_{t=2}^{T} {c_t^s}^\top x_t^s \qquad \text{(MS-SLP}^{\gamma_2}\text{)}$$

$$\text{subject to} \quad x^s \in X_s, \quad \forall s \in \mathcal{B}_{\gamma_2}, \tag{5.2a}$$

$$\{x^s\}_{s\in\mathcal{B}_{\gamma_2}} \in \mathcal{N}, \tag{5.2b}$$

$$x_1^s = \hat{x}_1, \quad \forall s \in \mathcal{B}_{\gamma_2}. \tag{5.2c}$$

Let $x^{\gamma_2 *}$ be the optimal solution to (MS-SLP$^{\gamma_2}$) at node γ_2. Note that $x_2^s = x^{\gamma_2}$, for all $s \in \mathcal{B}_\gamma$. Let $f^{\gamma_2}(x^{\gamma_2 *}|\hat{x}_1)$ be the corresponding optimal objective value. We can evaluate the objective function $f(\hat{x}_1)$ as

$$\begin{aligned} f(\hat{x}_1) &= c_1^\top \hat{x}_1 + \sum_{\gamma_2 \in \Gamma_2} p_{cn}(\gamma_2|\gamma_1) f^{\gamma_2}(x^{\gamma_2 *}|\hat{x}_1) \\ &= c_1^\top \hat{x}_1 + \sum_{\gamma_2 \in \Gamma_2} p_n^{\gamma_2} f^{\gamma_2}(x^{\gamma_2 *}|\hat{x}_1). \end{aligned} \tag{5.3}$$

Note that since $p_n^{\gamma_1} = 1$, we have $p_{cn}(\gamma_2|\gamma_1) = p_n^{\gamma_2}$.

To generalize, with a given decision history $\underline{\hat{x}}^\gamma$, the MS-SLP problem corresponding to subtree $\mathcal{T}_\gamma$ can be formulated as

$$\text{minimize} \quad \sum_{s \in \mathcal{B}_\gamma} p_{cs}(s|\gamma) \sum_{\tau=t(\gamma)}^{T} {c_\tau^s}^\top x_\tau^s \tag{MS-SLP$^\gamma$}$$

$$\text{subject to} \quad x^s \in X_s, \quad \forall s \in \mathcal{B}_\gamma, \tag{5.4a}$$

$$\{x^s\}_{s \in \mathcal{B}_\gamma} \in \mathcal{N}, \tag{5.4b}$$

$$\underline{x}^s_{t(\gamma)-1} = \underline{\hat{x}}^\gamma, \quad \forall s \in \mathcal{B}_\gamma. \tag{5.4c}$$

In (MS-SLP$^\gamma$), (5.4c) fixes the decision history prior to node γ to agree with $\underline{\hat{x}}^\gamma$. Let $x^{\gamma *}$ be the optimal solution to (MS-SLP$^\gamma$) at node γ , and let $f^\gamma(x^{\gamma *}|\underline{\hat{x}}^\gamma)$ be the corresponding optimal objective value. We have

$$f^\gamma(x^{\gamma *}|\underline{\hat{x}}^\gamma) = {c_{t(\gamma)}^\gamma}^\top x^{\gamma *} + \sum_{\theta_\gamma \in \Theta_\gamma} p_{cn}(\theta_\gamma|\gamma) f^{\theta_\gamma}(x^{\theta_\gamma *}|\underline{\hat{x}}^\gamma, x^{\gamma *}), \tag{5.5}$$

where $x^{\theta_\gamma *}$ and $f^{\theta_\gamma}(x^{\theta_\gamma *}|\underline{\hat{x}}^\gamma, x^{\gamma *})$ are defined in a recursive fashion for each child node of γ.

When γ is a leaf node (i.e., $t(\gamma) = T$), $|\mathcal{B}_\gamma| = 1$, so that the nonanticipativity constraints (5.4b) in (MS-SLP$^\gamma$) become redundant and can be eliminated.

Note that in the discussion above, we assume that MS-SLP problem has the relatively complete recourse property. That is, for any subtree $\mathcal{T}_\gamma \subset \mathcal{T}$, the feasible set of the corresponding problem (MS-SLP$^\gamma$) is nonempty for any feasible decision history $\underline{\hat{x}}^\gamma = \{\hat{x}_\tau\}_{\tau=1}^{t(\gamma)-1}$.

Let $f_{ub}^{\gamma_2}(\gamma_2|\hat{x}_1)$ denote an upper bound on $f^{\gamma_2}(x_2^{\gamma_2 *}|\hat{x}_1)$. Then we may obtain an upper bound on $f(\hat{x}_1)$ as

$$f_{ub}(\hat{x}_1) = c_1^\top \hat{x}_1 + \sum_{\gamma_2 \in \Gamma_2} p_n^{\gamma_2} f_{ub}^{\gamma_2}(\gamma_2|\hat{x}_1) \geq f(\hat{x}_1). \tag{5.6}$$

We can evaluate the quality of $\hat{x}_1$ using the upper bound on $\mathcal{G}(\hat{x}_1)$,

$$\mathcal{G}_{ub}(\hat{x}_1) = \frac{f_{ub}(\hat{x}_1) - f_{lb}}{|f_{lb}|}. \tag{5.7}$$

5.1 Upper Bounds – Finding Implementable Solutions

One way to obtain an upper bound on the value of MS-SLP$^{\gamma_t}$ is to find a set of feasible (implementable) solutions, x^s, $s = 1, ..., |S|$, that satisfy all the nonanticipativity constraints. In Chapter 4, we describe and compare different approaches to obtain a lower bound on the optimal objective value by solving a relaxed DEP, in which a subset of the nonanticipativity constraints are eliminated from the DEP. As a result of this process, we find a lower bound on the optimal objective value, and a set of scenario solutions $\hat{x}^s$, $s = 1, ..., |S|$, as well. In this section, we exploit the fact that $\hat{x}^s$, $s = 1, ..., |S|$, satisfy a subset of the nonanticipativity constraints to obtain an implementable solution. In this way, we can establish an upper bound on the optimal objective value of the MS-SLP problem.

5.1.1 Upper Bounds via Solving Free Forward Scenarios

Recall from Chapter 2, for each $s \in S$, we have a scenario problem

$$\text{minimize} \quad c^{s\top} x^s \tag{P^s}$$

$$\text{subject to} \quad x^s \in X^s. \tag{5.8a}$$

If we fix part of the solution in (P^s) with a given decision history $\underline{\hat{x}}_t$, we then have a restricted scenario problem

$$\text{minimize} \quad c^{s\top} x^s \tag{RP^s}$$

$$\text{subject to} \quad x^s \in X^s, \tag{5.9a}$$

$$\{x_\tau^s\}_{\tau=1}^t = \underline{\hat{x}}_t. \tag{5.9b}$$

Assume we have a set of scenario solutions $\hat{x}^s$, $s = 1, ..., |S|$, which satisfy a subset of the nonanticipativity constraints. To find a set of feasible (and implementable) scenario solutions, an intuitive approach is to loop through the scenario problems one at a time, reinforce the nonanticipativity constraints up to a given stage, then solve the restricted scenario problem (RP^s) for the rest of stages. We call this the *free forward scenario* (FFS) procedure and describe it in Algorithm 5.1.1. In Algorithm 5.1.1, Υ denotes the subset of nodes in $\mathcal{T}$, $\Upsilon \subseteq \Gamma$, whose solutions are available (and have been fixed). $\overline{\Upsilon}$ denotes the complimentary subset of Υ (i.e., $\overline{\Upsilon} = \Gamma \setminus \Upsilon$).

step 1 $\hat{x}^1$ is given from solving the relaxed DEP.

step 1.1 $s = 1$. Let $\bar{x}^1 = \hat{x}^1$. Let $\bar{f}^1 = {c^1}^\top \bar{x}^1$. UB $= p^1 \bar{f}^1$.

step 1.2 Set $\bar{x}^\gamma = \hat{x}^1_{t(\gamma)}$, $\forall \gamma \in \Gamma^1$. $\Upsilon = \Gamma^1$, and $\overline{\Upsilon} = \Gamma \setminus \Gamma^1$.

step 2 For $s = 2, ..., |S|$, **do**

step 2.1 Enforce the solution in (P^s) to agree with the available decision history in $\Upsilon \cap \Gamma^s$ to form the restricted scenario problem (RP^s),

$$x^s_{t(\gamma)} = \bar{x}^\gamma, \quad \forall \gamma \in \Upsilon \cap \Gamma^s. \tag{5.10}$$

step 2.2 Solve the restricted scenario problem (RP^s) to obtain an optimal solution $\bar{x}^s$ with optimal objective value $\bar{f}^s$. UB $=$ UB $+ p^s \bar{f}^s$.

step 2.3 If $\Gamma^s \cap \overline{\Upsilon} \neq \emptyset$, set

$$\bar{x}^\gamma = \bar{x}^s_{t(\gamma)}, \quad \forall \gamma \in \Gamma^s \cap \overline{\Upsilon}, \tag{5.11}$$

and update $\Upsilon = \Upsilon \cup \Gamma^s$ and $\overline{\Upsilon} = \Gamma \setminus \Upsilon$.

Output: $\{\bar{x}^s\}_{s \in S}$ is a set of feasible (implementable) solution, and UB is an upper bound on $f(\hat{x}^1_1)$.

Algorithm 5.1.1: Upper Bound - Free Forward Scenario Approach

For any node $\gamma \in \mathcal{T}$, once we obtain a scenario solution $\bar{x}^s$ for the first $s \in \mathcal{B}_\gamma$, we fix the node solution as $\bar{x}^\gamma = \bar{x}_{t(\gamma)}$, as in step 1.2 and 2.3 of Algorithm 5.1.1. We then use this node solution to enforce the nonanticipativity constraints for all other scenarios in $\mathcal{B}_\gamma$, as in step 2.1 of Algorithm 5.1.1. Because the nonanticipativity constraints can be expressed in a variety of ways, the implementation in these steps depends on the nonanticipativity constraint representation type employed.

For the sibling-type representation,

$$x^s_{t(\gamma)} = \bar{x}^{\nu^{-1}(s,t(\gamma))}_{t(\gamma)}, \quad \forall \gamma \in \Upsilon \cap \Gamma^s, \tag{5.12}$$

and for the branch-type representation,

$$x^s_{t(\gamma)} = \bar{x}^{s^s_b}_{t(\gamma)}, \quad \forall \gamma \in \Upsilon \cap \Gamma^s. \tag{5.13}$$

In Chapter 4, we ensure that

$$s > \nu^{-1}(s,t), \quad \forall s \in S, t = 1, ..., T-1, \tag{4.8}$$

for sibling-type representation, and

$$s > s^s_b, \quad \forall s \in S, \tag{4.14}$$

for branch-type representation. Consequently, for these two nonanticipativity representations, for each scenario s, $s = 2, ...|S|$, the right-hand-side of (5.10) in step 2.1 are always available from the solutions of scenarios with lower indices. Note also that the assignment of the solutions for nodes $\gamma \in \Gamma^s \cap \overline{\Upsilon}$ in step 2.3, (5.11), does not affect the solutions of previously solved scenarios.

For the node-type representation, the implementation associated with $\bar{x}^\gamma$ is exactly the same as in Algorithm 5.1.1. It is obvious that (5.10) in step 2.1 is always available for any node $\gamma \in \Gamma^s \cap \Upsilon \subset \Upsilon$ (by definition, Υ is the set of nodes whose solutions are available), and the assignment of node solution for any node $\gamma \in \Gamma^s \cap \overline{\Upsilon}$ in step 2.3 does not affect the solutions of the nodes already in Υ.

In the example in Figure 5.1, we use scenarios 1 - 6 in Figure 4.1 to demonstrate how the free forward scenario approach works. In each sub-graph, the nodes in Υ are

shaded, while the rest are the nodes in $\overline{\Upsilon}$. Every time Υ is updated, $\overline{\Upsilon}$ is updated accordingly. The node solution, if available, is attached besides the corresponding node.

In the free forward scenario procedure, we ensure that scenarios in any scenario bundle $\gamma \in \Upsilon$ share the same solution. Therefore, the set of scenario solutions obtained, $\bar{x}^s$, $s = 1, ..., |S|$, satisfy all the nonanticipativity constraints and are therefore implementable. The upper bound calculated is a valid upper bound on $f(\hat{x}_1^1)$.

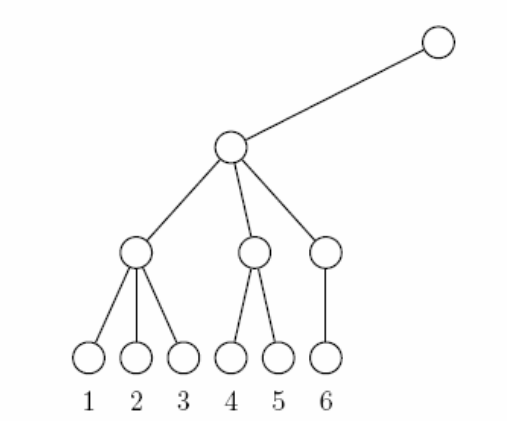
With a partial solution fixed by the enforcement of NNT constraints as in step 2.1, we expect that (RP^s) can be solved faster than the original scenario problem (P^s). Therefore, the implementation of Algorithm 5.1.1 is at least as fast as obtaining the wait-and-see solutions, where (P^s) is solved for each $s \in S$.

However, it is worth mentioning that, although we have obtained a set of scenario solutions $\{\hat{x}^s\}_{s\in S}$ from solving the relaxed DEP, only $\hat{x}^1$ is used in the free forward scenario approach. The upper bound so obtained is likely to be loose, so that the simplicity and time efficiency of the procedure are at the expense of the upper bound quality.

5.1.2 Upper Bounds via Solving Scenarios with Penalty Terms

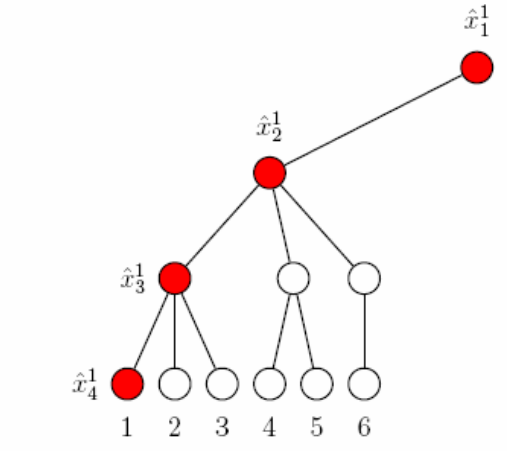
In order to take advantage of the fact that $\{\hat{x}^s\}_{s\in S}$ obtained from solving the relaxed DEP satisfies a subset of the nonanticipativity constraints, we use the *scenarios with penalty terms* procedure, as described in Algorithm 5.1.2.

Let L, E, and G denote the sets of less than inequality, equality, and greater than inequality constraints in (5.8a), respectively. Let A_L, A_E, and A_G be the constraint matrices and b_L, b_E, and b_G the right-hand-side vectors corresponding to L, E, and



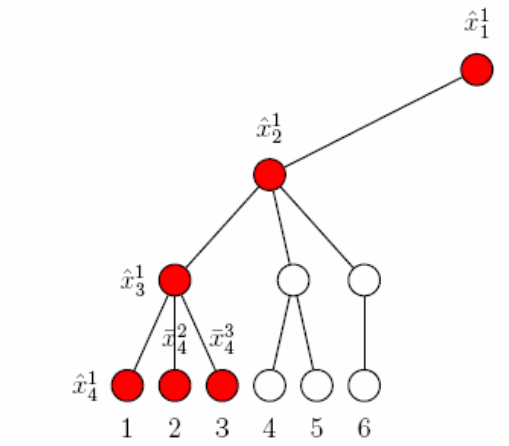
$\hat{x}^1$ is given. $\Upsilon = \emptyset$ and $\overline{\Upsilon} = \Gamma$. $UB = 0$.

(a) Subtree $\mathcal{T}_a$



First, as in step 1.1, let $\bar{x}^1 = \hat{x}^1$, and UB $= p^1 \bar{f}^1 = p^1 {c^1}^\top \bar{x}^1$. Then assign $\hat{x}^1$ to the solution of the nodes in Γ^1, and add these nodes to Υ.

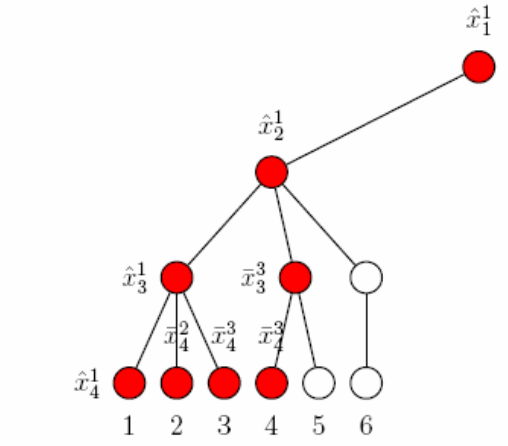
(b) Scenario 1



For scenarios $s = 2, 3$, first enforce the scenario solution x^s to agree with the available node solutions (up to stage 3) in (P^s) as in step 2.1. Then solve the restricted scenario problem (RP^s) and accumulate UB. Assign $\bar{x}_4^s$ to the stage 4 (leaf) node and add the node in Υ.

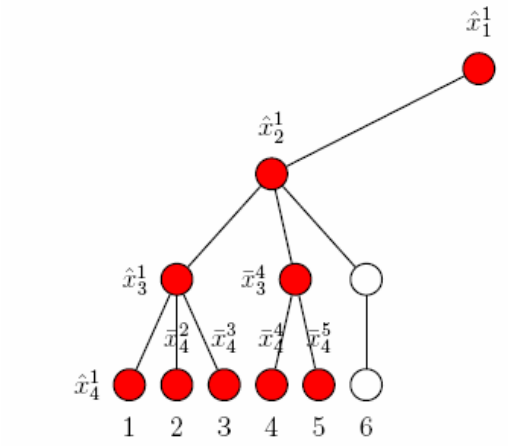
(c) Scenario 2 and 3

Figure 5.1: Free Forward Scenario (FFS) Approach



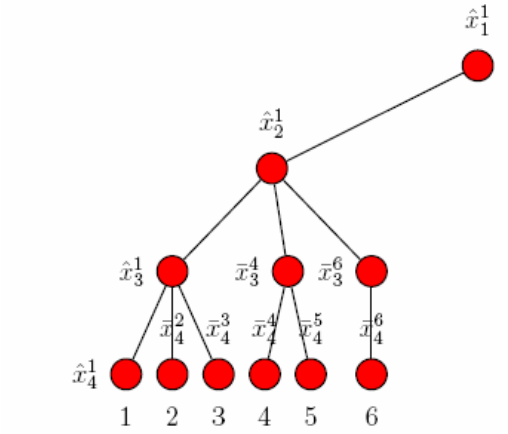
For scenario 4, first enforce the scenario solution to agree with the available node solutions up to stage 2 ($\hat{x}_1^1$ and $\hat{x}_2^1$). Solve (RP^4) and accumulate UB. Assign the solution $\bar{x}_3^4$ and $\bar{x}_4^4$ to its stage 3 and stage 4 nodes and add these two nodes in Υ.

(e) Scenario 4



For scenario 5, first enforce the scenario solution to agree with the available node solutions up to stage 3 ($\hat{x}_1^1$, $\hat{x}_2^1$, and $\bar{x}_3^4$). Then solve (RP^5) and Accumulate UB. Assign $\bar{x}_4^4$ to its stage 4 node and add it to Υ.

(f) Scenario 5



For scenario 6, first enforce the scenario solution to agree with the available node solutions up to stage 2 as for scenario 4. Then solve (RP^6) and accumulate UB. Assign its stage 3 and stage 4 solution to (RP^6) to the stage 3 and stage 4 nodes and add them to Υ.

(g) Scenario 6

Figure 5.1 Free Forward Scenario (FFS) Approach (continued)

G. We can write (P^s) equivalently as

$$\begin{aligned} \text{minimize} \quad & {c^s}^\top x^s && (5.14a)\\ \text{subject to} \quad & A_L x^s \leq b_L, && (5.14b)\\ & A_E x^s = b_E, && (5.14c)\\ & A_G x^s \geq b_G. && (5.14d) \end{aligned}$$

By adding slack variables (y_L^-, y_E^+, y_E^-, and y_G^+) and penalty terms (M_L, M_E, and M_G) in (5.14), we have

$$\begin{aligned} \text{minimize} \quad & {c^s}^\top x^s + M_L y_L^- + M_E y_E^+ + M_E y_E^- + M_G y_G^+ && (5.15a)\\ \text{subject to} \quad & A_L x^s - y_L^- \leq b_L, && (5.15b)\\ & A_E x^s + y_E^+ - y_E^- = b_E, && (5.15c)\\ & A_G x^s + y_G^+ \geq b_G. && (5.15d) \end{aligned}$$

When the penalty terms are sufficiently large, (5.15) is equivalent to (5.14).

In the relaxed DEP, scenarios are solved together and a subset of the nonanticipativity constraints are included. The solution obtained, $\{\hat{x}^s\}_{s\in S}$, are therefore likely to be better than those from solving scenario problems individually. The idea of (5.17) in step 2.1b is to take advantage of this fact and to keep as many of the solution values of $\{\hat{x}^s\}_{s\in S}$ as possible. However, the mixed solution enforcement in (5.16) and (5.17) may cause infeasibility in scenario constraints in (5.8a). The introduction of slack variables and penalty terms in (5.15) transfer the possible infeasibility and reflect it in the objective values. The quality of the so obtained upper bounds relies heavily on the quality of the solution $\{\hat{x}^s\}_{s\in S}$ and the appropriate choice of the penalty terms. It is possible that the resulting upper bound becomes huge.

5.1.3 Scenario-based Recursive Subtree Decomposition Approach

While the set of scenario solutions obtained from solving the relaxed DEP, $\{\hat{x}^s\}_{s\in S}$, optimize the scenario problems while satisfying a subset of the nonanticipativity

step 1 $\{\hat{x}^s\}_{s\in S}$ is given from solving the relaxed DEP.

step 1.1 $s = 1$. Let $\bar{x}^1 = \hat{x}^1$. Let $\bar{f}^1 = {c^1}^\top \bar{x}^1$. UB $= p^1 \bar{f}^1$.

step 1.2 Set $\bar{x}^\gamma = \hat{x}^1_{t(\gamma)}$, $\forall \gamma \in \Gamma^1$. $\Upsilon = \Gamma^1$, and $\overline{\Upsilon} = \Gamma \setminus \Gamma^1$.

step 2 For $s = 2, ..., S$, **do**

step 2.1a Enforce the scenario solution in (P^s) to agree with the available decision history in $\Upsilon \cap \Gamma^s$ to form the restricted scenario problem (RP^s) with penalty terms, (5.15)

$$x^s_{t(\gamma)} = \bar{x}^\gamma, \quad \forall \gamma \in \Upsilon \cap \Gamma^s. \tag{5.16}$$

step 2.1b (Further enforce the scenario solution with $\hat{x}^s$.) If $\overline{\Upsilon} \cap \Gamma^s \neq \emptyset$, let

$$x^s_{t(\gamma)} = \hat{x}^s_{t(\gamma)}, \quad \forall \gamma \in \overline{\Upsilon} \cap \Gamma^s. \tag{5.17}$$

step 2.2 Solve (5.15) to obtain optimal solution $\bar{x}^s$ and $\bar{f}^s$. UB $=$ UB $+ p^s \bar{f}^s$.

step 2.3 If $\Gamma^s \cap \overline{\Upsilon} \neq \emptyset$, let

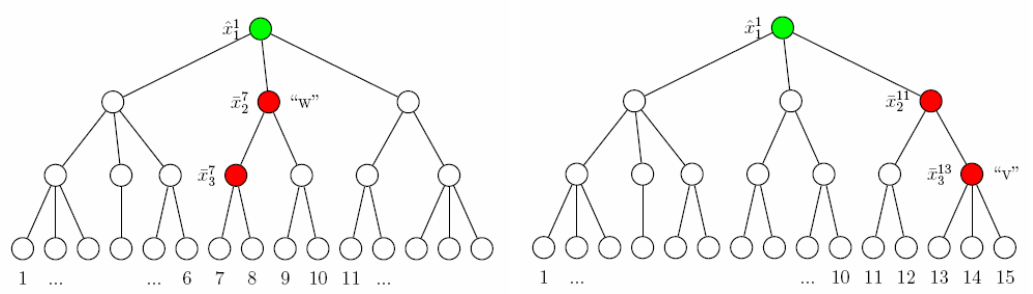
$$\bar{x}^\gamma = \bar{x}^s_{t(\gamma)}, \forall \gamma \in \Gamma^s \cap \overline{\Upsilon}, \tag{5.18}$$

and update $\Upsilon = \Upsilon \cup \Gamma^s$ and $\overline{\Upsilon} = \Gamma \setminus \Upsilon$.

Output: $\{\bar{x}^s\}_{s\in S}$ is a set of feasible (implementable) solution, and UB is an upper bound on $f(\hat{x}^1_1)$.

Algorithm 5.1.2: Upper Bound - Solving Scenarios with Penalty Terms

constraints, the upper bound calculated in the free forward scenario procedure relies only on $\hat{x}^1$. Solutions of other scenarios are developed based on the solutions of the scenarios with lower indices.



(a) Example 1 (b) Example 2

Figure 5.2: "Floating" subtrees in FFS

In Figure 5.2a, the solutions of scenarios 8 - 10 rely heavily on the scenario solution $\bar{x}^7$. In (RP^7), the first stage solution is fix to $\hat{x}_1^1$, while the solution of all other stages is only restrained by the scenario constraints. Consequently, $\bar{x}^7$ may deviate from $\hat{x}^7$ somehow arbitrarily. We refer the subtree rooted at the node "w" as a "floating" subtree. The "floating" subtrees may occur in any stage of the scenario tree. The subtree rooted at node "v" in Figure 5.2b is another example.

A floating subtree can result in a solution that deviates considerably from the original solution, which is presumed to be "good" (or at least, "nearly good"). To control the impact of "floating" subtrees, we introduce the *scenario-based recursive subtree decomposition* (SRSD) procdure, as described in Algorithm 5.1.3 and 5.1.4.

The recursive function, func_recursive_ub, takes two parameters: a node $\gamma \in \mathcal{T}$ and a decision history $\underline{\hat{x}}^\gamma$. It returns a real value as an upper bound on the optimal objective value of the subtree problem MS-SLP$^\gamma$ with give $\underline{\hat{x}}^\gamma$. We have several comments regarding func_recursive_ub:

- In step 1.1.2, if $|\mathcal{B}_\gamma| = 1$ (a leaf node), we simply solve the restricted scenario problem. In fact, when $\mathcal{T}_\gamma$ is small (i.e., there are only few scenarios in $\mathcal{B}_\gamma$), we may prefer to directly solving (MS-SLP$^\gamma$) to optimality. In this case, the

step 0 $\hat{x}_1$ is given. UB $= 0$.

step 1 For each $\gamma \in \Theta_{\gamma_1}$, or equivalently, $\gamma \in \Gamma_2$, **do**

step 1.1 $f_{ub}^{\gamma} =$ func_recursive_ub$(\gamma, \hat{x}_1)$.

step 1.2 UB $=$ UB $+ p^{\gamma} f_{ub}^{\gamma}$.

Algorithm 5.1.3: Upper Bounds - SRSD Approach

condition in step 1.1.2 can be generalized to $|\mathcal{B}_{\gamma}| \leq N$, where N is a pre-defined small integer.

- For each subtree $\mathcal{T}_{\gamma}$. We accept the free forward scenario upper bound in step 1.1.4 only if the gap between the lower bound and the free forward scenario upper bound, $\mathcal{G}^{\gamma}$, is small enough. Several factors may have impact on $\mathcal{G}^{\gamma}$:
  - The quality of the lower bound obtained in step 1.1.3.
  - The quality of the solution (of the first scenario in subtree $\mathcal{T}_{\gamma}$) obtained in step 1.1.3.
  - The existence and impact of "floating" subtrees on f_{ub}^{γ}.
- If the test in step 1.1.5 fails, before proceeding to the recursive function on the subtrees rooted at the children nodes of γ, we need to select a node solution $\bar{x}^{\gamma}$, as we will discuss next.

Selection of the Node Solution in $\mathcal{T}_{\gamma}$

In func_recursive_ub, before we proceed to step 1.1.7 to evaluate an upper bound for the problems associated with each subtree rooted at a child node of γ, we need to select the node solution $\bar{x}^{\gamma}$, which, together with $\underline{\hat{x}}^{\gamma}$, will become the decision history of each child node θ_{γ}, i.e., $\underline{\hat{x}}^{\theta_{\gamma}} = \{\underline{\hat{x}}^{\gamma}, \bar{x}^{\gamma}\}$.

The solution obtained from the solving the relaxed DEP in step 1.1.3 satisfy a subset of the nonanticipativity constraints. The selection of the node solution for

$\bar{x}^\gamma$ among the scenario solutions $\{\hat{x}^s\}_{s\in\mathcal{B}_\gamma}$ depends on the nonanticipativity representation type employed.

For the branch-type representation,

$$\bar{x}^\gamma = \hat{x}^{s'}_{t(\gamma)}, \tag{5.19}$$

where s' is the first scenario in $\mathcal{T}_\gamma$ (i.e., the scenario with the lowest index among scenarios in $\mathcal{B}_\gamma$). As a major branch scenario in $\mathcal{B}_\gamma$, its solution should be among the least affected by the partial elimination of NNT constraints in the lower bound procedure.

For the node-type representation,

$$\bar{x}^\gamma = \hat{z}_\gamma, \tag{5.20}$$

where z_γ is the state vector of node γ. Since all the scenarios in $\mathcal{B}_\gamma$ connect to each other through the nonanticipativity constraints enforced between their stage $t(\gamma)$ decision vector and this node state vector, z_γ is expected to be the least affected by the partial elimination of the nonanticipativity constraints. To avoid the possible unrestricted deviation of some state variable of z_γ caused by partial elimination of NNT constraints, we require that z_γ completely tie to the stage $t(\gamma)$ solution of the first scenario in $\mathcal{T}_\gamma$.

For the sibling-type representation, the nonanticipativity constraints are almost evenly allocated among scenarios in each scenario bundle. No scenario solution is more likely to be representative than others. Therefore, we take the expected stage $t(\gamma)$ solution of all scenarios in scenario bundle $\mathcal{B}_\gamma$ the node solution x^γ. That is,

$$\bar{x}^\gamma = \sum_{s\in\mathcal{B}_\gamma} p^s \hat{x}^s_{t(\gamma)}. \tag{5.21}$$

5.2 Computational Experiment

In our computational experiment, we test the performance of the three upper bound procedures described in the previous section. The computational environment and test instances are the same as described in Chapter 4.

We begin our experiment with a potential solution $\{\hat{x}^s\}_{s\in S}$, which is obtained using the lower bound procedure described in Chapter 4. More specifically, we solve the relaxed DEPs with

- each of the three nonanticipativity representation types (branch, node, and sibling),
- relaxation by individual nonanticipativity constraints, and
- relaxation level $\mathcal{P}_{rlx} = 50\%$.

In that way, we can examine the quality of these solutions. Again, we make 10 sets of replication runs to account for the impact of the random stream used.

Starting from $\{\hat{x}^s\}_{s\in S}$, which may not be implementable, we apply the three upper bound procedures as discuss in the previous section to obtain an implementable solution as well as an upper bound estimate on $f(\hat{x}_1)$.

5.3 Experiment Result Analysis

5.3.1 Free Forward Scenario Procedure

The free forward scenario upper bound procedure requires no more than 3 seconds in any case. As a result, we focus our analysis on the quality of the upper bounds obtained.

In Figure 5.3, we plot the relative gaps of

- $f_{ub}(\hat{x}_1)$, the upper bounds on $f(\hat{x}_1)$, obtained from the free forward scenario procedure;
- the function values $f(\hat{x}_1)$,
- the lower bounds, and
- the wait-and-see lower bounds.

step 1.1.1 Formulate the problem (MS-SLP$^\gamma$) with the decision history as $\underline{\hat{x}}^\gamma$.

step 1.1.2 If ($|\mathcal{B}_\gamma| = 1$), solve (MS-SLP$^\gamma$), and
return the optimal objective value $f^\gamma(x^{\gamma *}|\underline{\hat{x}}^\gamma)$.

step 1.1.3 Otherwise, using procedures described in Chapter 4, solve the relaxed DEP of (MS-SLP$^\gamma$), and obtain solution $\bar{x}$ and objective value f^γ_{lb}. f^γ_{lb} is a lower bound on the optimal objective of MS-SLP$^\gamma$. Note that in the relaxed DEP, the relaxation of nonanticipativity constraints only applies to subtree $\mathcal{T}_\gamma$.

step 1.1.4 Apply the free forward scenario approach in subtree $\mathcal{T}_\gamma$ and obtain an upper bound on the optimal objective value of MS-SLP$^\gamma$, denoted by f^γ_{ub}.

step 1.1.5 ϵ is a pre-defined tolerance. If ($\mathcal{G}^\gamma = \frac{f^\gamma_{ub} - f^\gamma_{lb}}{|f^\gamma_{lb}|} \leq \epsilon$)
return f^γ_{ub}.

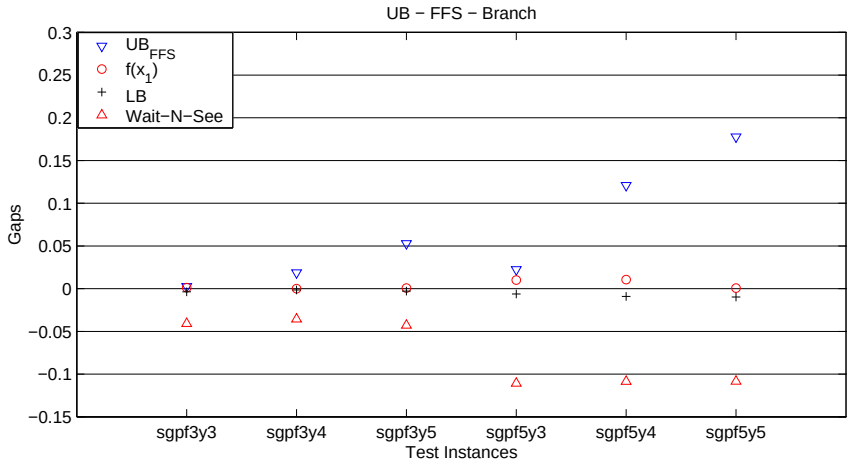
step 1.1.6 Otherwise, select a representative node solution $\bar{x}^\gamma$, based on $\bar{x}$ (as described in the text). Let $f^\gamma_{ub} = c^{\gamma\top}\bar{x}^\gamma$.

step 1.1.7 For each $\theta_\gamma \in \Theta_\gamma$, **do**

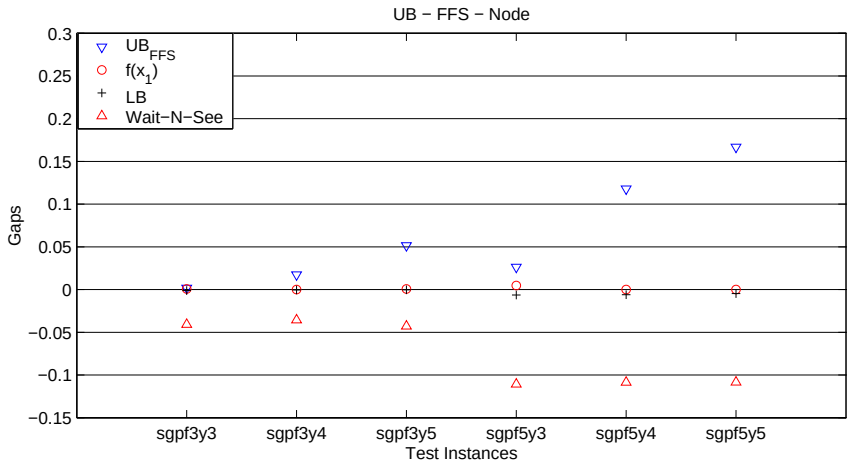
$$f^\gamma_{ub} = f^\gamma_{ub} + p_{cn}(\theta_\gamma|\gamma) \times \text{func\_recursive\_ub}(\theta_\gamma, \{\underline{\hat{x}}^\gamma, \bar{x}^\gamma\}).$$

Return f^γ_{ub}.

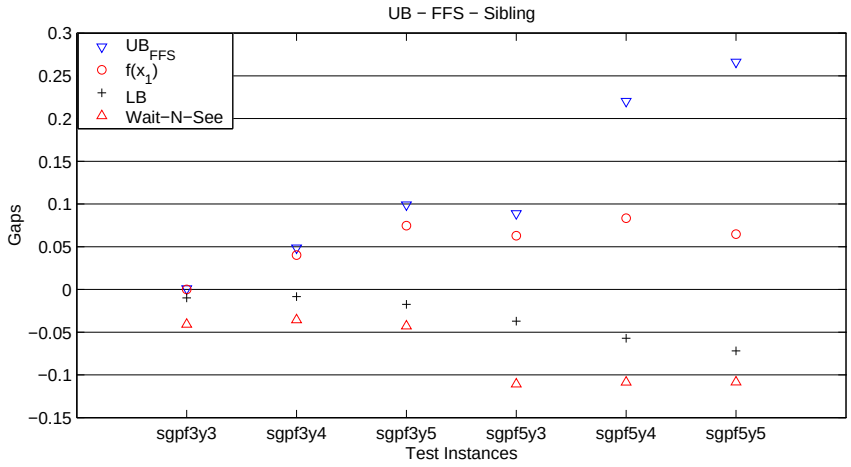
Algorithm 5.1.4: func_recursive_ub(γ, $\underline{x}^\gamma$)



(a) Branch



(b) Node



(c) Sibling

Figure 5.3: Bound Gaps – Free Forward Scenario

Note that in the free forward scenario approaches, we take $\hat{x}_1^1$ as the first stage solution of the MS-SLP problem. That is, $\hat{x}_1 = \hat{x}_1^1$.

The upper bound gaps are calculated as

$$\mathcal{G}_{ub}(\hat{x}_1) = \frac{f_{ub}(\hat{x}_1) - f_{opt}}{|f_{opt}|}, \tag{5.22}$$

where f_{opt} is the optimal objective value of the DEP. The gaps between $f(\hat{x}_1)$ and f_{opt} are calculated as

$$\mathcal{G}_f(\hat{x}_1) = \frac{f(\hat{x}_1) - f_{opt}}{|f_{opt}|}. \tag{5.23}$$

Lower bound gaps are calculated as in (4.15)

$$\mathcal{G}_{lb} = \frac{f_{lb} - f_{opt}}{|f_{opt}|}. \tag{4.15}$$

The gaps in Figure 5.3 are the average of the 10 replications, and the results for each nonanticipativity representation type appear as a sub-figure.

The values of the average bound gaps, $\mathcal{G}_{ub}(\hat{x}_1)$, are listed in Table 5.1 with the "normalized" standard deviations (standard deviation divided by the $|f_{opt}|$) listed parenthetically.

	branch	node	sibling
SGPF3Y3	0.25% (0.34%)	0.17% (0.26%)	0.09% (0.00%)
SGPF3Y4	1.87% (0.28%)	1.74% (0.00%)	4.87% (0.55%)
SGPF3Y5	5.28% (0.33%)	5.15% (0.00%)	1.24% (1.67%)
SGPF5Y3	2.24% (3.08%)	2.63% (2.77%)	8.88% (5.33%)
SGPF5Y4	12.08% (1.28%)	11.79% (0.05%)	22.02% (1.99%)
SGPF5Y5	17.77% (1.74%)	16.68% (0.00%)	26.60% (1.23%)

Table 5.1: Upper Bounds - Free Forward Scenario Approach

By examining the lower bounds gaps and the gaps of $f(\hat{x}_1)$, we note that solving the relaxed DEPs with the branch-type and node-type nonanticipativity constraints actually yields good solutions. In contrast, solutions of the relaxed DEPs with the sibling-type nonanticipativity constraints are relatively poorer.

In general, the upper bound estimates obtained from the free forward scenario procedure are too loose to provide an adequate judgement of solution quality. Therefore, we need more accurate estimates on the upper bounds of $f(\hat{x}_1)$.

5.3.2 Solving Scenarios with Penalty Terms

In all cases, these upper bounds require no more than 13 seconds to calculate. As with the free forward scenario procedure, we focus our analysis on the quality of the upper bound estimates.

	branch	node	sibling
SGPF3Y3	7,930.47 (12,830.56)	0.0014 (0.0029)	44,151.97 (40,693.75)
SGPF3Y4	3,646.02 (5,747.59)	3,324.23 (8,466.85)	55,185.42 (14,894.07)
SGPF3Y5	3,882.32 (2,942.32)	799.67 (1,234.19)	111,439.40 (33,005.84)
SGPF5Y3	14,518.38 (24,223.89)	5,703.91 (18,037.28)	101,359.70 (33,529.95)
SGPF5Y4	15,327.31 (13,002.64)	9,785.05 (9,026.59)	142,431.41 (25,654.27)
SGPF5Y5	9,905.24 (7,541.12)	6,289.36 (4,430.83)	160,571.46 (12,528.44)

Table 5.2: Upper Bounds - Solving Scenarios with Penalty Terms (1)

In Table 5.2, we list the upper bound gaps calculated as in (5.22). The gaps are the average of the 10 replications, with the "normalized" standard deviation (standard deviation divided by the $|f_{opt}|$) listed parenthetically.

Except for SGPF3Y3 with the node-type nonanticipativity representation, we observe huge average upper bound gaps with huge standard deviations in all cases. Note that the values of the bound gaps and standard deviations are so large that we do not present them in percentage.

On a run by run analysis, we find that not all the upper bounds are huge. In Table 5.3, the average gaps are calculated only including the runs with $\mathcal{G}_{ub}(\hat{x}_1) \leq 100\%$. We list the number of these runs in parentheses. For example, in SGPF5Y4, after solving the relaxed DEPs with the node-type nonanticipativity constraints, we obtain 4 upper bound estimates with $\mathcal{G}_{ub}(\hat{x}_1) \leq 100\%$ and the average gap of these 4 upper bound gaps is 0.82%. A "- (0)" entry indicates no such "ordinary" upper bound is found.

The results in Table 5.2 and 5.3 suggest that when we are able to obtain the "ordinary" upper bounds by solving scenario problems with penalty terms, they are usually very tight. However, there is a good chance that we encounter huge upper bounds. In general, the chance to encounter huge upper bounds increases as the

	branch	node	sibling
SGPF3Y3	0.00% (7)	0.14% (10)	0.08% (4)
SGPF3Y4	0.01% (6)	0.05% (7)	-
SGPF3Y5	0.00% (1)	0.10% (6)	-
SGPF5Y3	0.03% (6)	1.95% (9)	-
SGPF5Y4	0.00% (1)	0.82% (4)	-
SGPF5Y5	-	0.42% (1)	-

Table 5.3: Upper Bounds - Solving Scenarios with Penalty Terms (2)

number of stages and scenarios increases. Therefore, this upper bound procedure is not reliable.

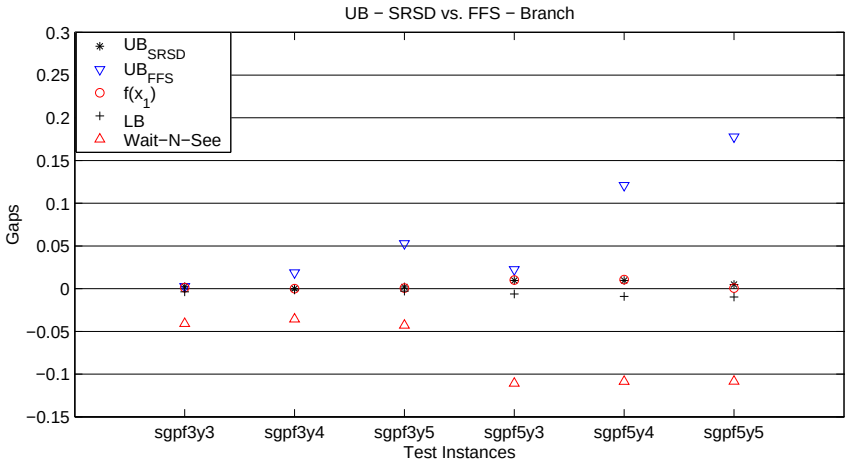
5.3.3 Scenario-based Recursive Subtree Decomposition Procedure

Note that besides the recursive manner and the subtree decomposition approach, the scenario-based recursive subtree decomposition procedure (SRSD) differs from the free forward scenario approach (FFS) in two major ways:

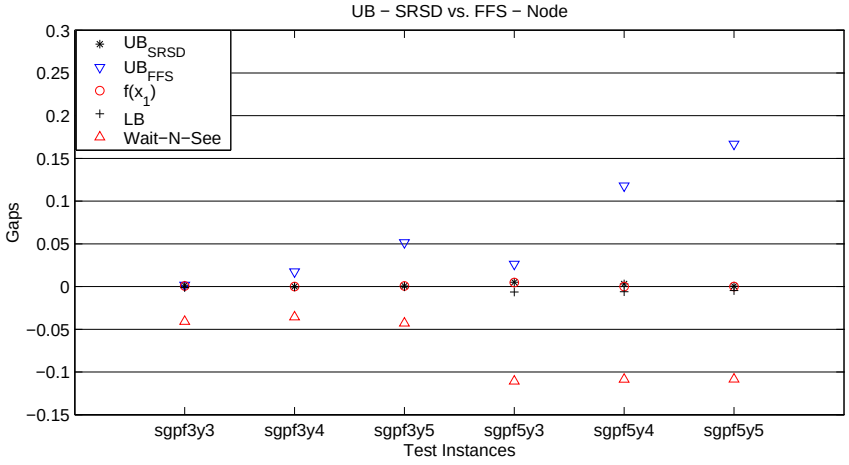
- We only need a candidate first stage solution $\hat{x}_1$ in SRSD, while we need a candidate scenario solution $\hat{x}^1$ in FFS.
- In FFS, $\hat{x}_1^1$ is always selected as the first stage solution, regardless of the nonanticipativity representation type used in the relaxed DEP. In SRSD, $\hat{x}_1^1$ is selected when we use the branch- or node-type nonanticipativity constraints. When the sibling-type nonanticipativity constraints are used, we take the expected values of the first stage solution of all scenarios as $\hat{x}_1$.

To examine the upper bound quality of SRSD and also compare SRSD with FFS, we include both upper bound gaps in Figure 5.4. Again, the bounds and function values in Figure 5.4 are the average over the 10 replications.

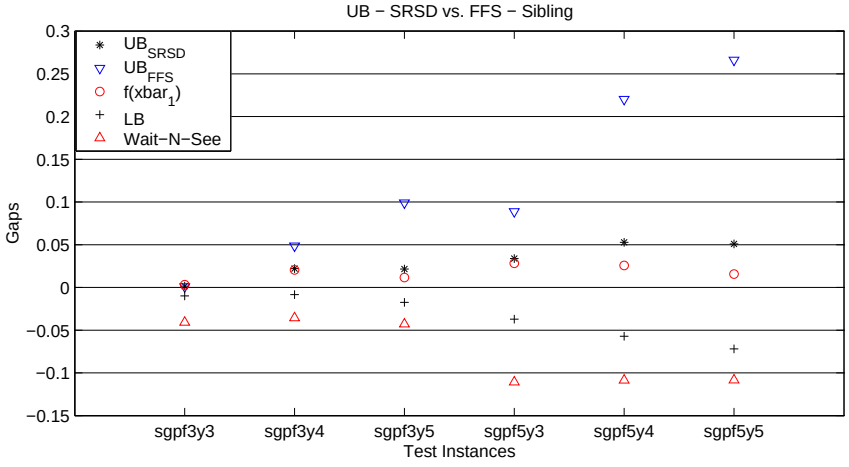
Note that for the sibling-type nonanticipativity constraints, the first stage solution is calculated as the expected first stage solution of all the scenarios, as in (5.21). We denote it as $\bar{x}_1$. Consequently, the result of the SRSD procedure is an upper bound estimate of $f(\bar{x}_1)$.



(a) Branch



(b) Node



(c) Sibling

Figure 5.4: Bound Gaps – SRSD vs. FFS

We observe a significant improvement in the upper bound quality of SRSD over FFS. In the branch-type and node-type nonanticipativity representations, the upper bound estimates from SRSD coincide with the function value $f(\hat{x}_1)$ in almost all test instances. Even in the sibling-type nonanticipativity representation, the upper bound estimates $f_{ub}(\hat{x}_1)$ from SRSD are much closer to $f(\hat{x}_1)$ than those in FFS.

We examine the time required to obtain the upper bounds with the SRSD approach. In Table 5.4, we list the upper bound time as well as the lower bound time for each test instance. The times are normalized relative to the time required to solve the corresponding DEP with the branch-type nonanticipativity constraints directly to optimality. Again, the times are the average of the 10 replications and the "normalized" standard deviation (standard deviation divided by the solution time of the DEP with the branch-type nonanticipativity representation) are listed parenthetically. The total of the lower and upper bound times are also included in Table 5.4.

	Branch			Node			Sibling		
	LB	UB	Total	LB	UB	Total	LB	UB	Total
SGPF3Y3	1.06 (0.07)	0.44 (0.19)	1.50	1.21 (0.06)	0.27 (0.13)	1.48	0.98 (0.17)	0.36 (0.22)	1.34
SGPF3Y4	0.71 (0.06)	0.27 (0.01)	0.97	1.02 (0.15)	0.25 (0.01)	1.27	0.62 (0.14)	0.23 (0.01)	0.85
SGPF3Y5	0.59 (0.08)	0.09 (0.00)	0.68	1.24 (0.22)	0.10 (0.00)	1.34	0.26 (0.02)	0.08 (0.00)	0.33
SGPF5Y3	0.98 (0.04)	0.46 (0.19)	1.44	0.94 (0.06)	0.45 (0.18)	1.40	0.94 (0.08)	0.56 (0.04)	1.50
SGPF5Y4	0.75 (0.06)	0.23 (0.01)	0.98	0.97 (0.14)	0.22 (0.01)	1.19	0.50 (0.05)	0.22 (0.01)	0.72
SGPF5Y5	0.55 (0.04)	0.06 (0.00)	0.61	2.32 (1.80)	0.07 (0.00)	2.40	0.16 (0.01)	0.06 (0.00)	0.22

Table 5.4: Normalized Times

From Table 5.4, we observe that the normalized times required to obtain upper bounds in SRSD reduces significantly as the number of stages and scenarios increase. It indicates that the upper bound estimates in SRSD become more efficient than directly solving the DEPs with the branch-type nonanticipativity constraints when the problem sizes increase.

Combining the times required to obtain both lower and upper bounds, we find that the node-type representation is not suitable for the bound-based approach discussed in this dissertation, because of the substantial time required to solve the

relaxed DEPs. Although the time required to obtain bounds with the sibling-type representation is small compared with the others, the resulting bounds are loose in general, as illustrated in Figure 5.4c. On the other hand, with the branch-type representation, both the lower and upper bounds are tight, and the computational requirements improve as the problem size increases. For example, in SGPF3Y5 and SGPF5Y5, the bound-based approach takes 60 - 70 % of the DEP solution time, but provides bounds with satisfactory quality, as illustrated in Figure 5.4a.

Last, we also observe that the time required to obtain upper bounds in SRSD are comparable among the three nonanticipativity representation types. Also, the time required to compute the lower bound is the bottleneck of the bound-based approach discussed in this dissertation. It is important to point out that the lower bound procedures are also used in the recursive subtrees in the SRSD procedure. Therefore, reducing the time required to obtain the lower bounds will improve the time efficiency of the SRSD procedure.

5.4 Conclusions

In this chapter, we discuss and compare three procedures to obtain an upper bound on the objective function value of a given first stage solution, $\hat{x}_1$. By combining the upper bound and a "good" lower bound on the optimal objective as discussed in Chapter 4, we may be able to evaluate the quality of the given solution.

Among the three upper bound procedures, the free forward scenario approach is the most time-efficient, but the bounds obtained are loose in most cases.

The upper bounds computed with penalty terms take advantage of the solution from solving the relaxed DEPs, but we may encounter situations where the resulting bounds are huge. This is more likely when the problem sizes become large. However, given that it requires very little time to compute, this upper bound procedure may serve as a pretest before implementing other more complicated upper bound procedures. In that way, we resort to more accurate but complicated upper bound procedures only if we fail to obtain a reasonable bound by solving scenarios with

penalty terms.

The scenario-based recursive subtree decomposition (SRSD) approach overcomes the "floating" subtree effect of FFS by recursively evaluating bound gaps for each subtree. SRSD significantly outperforms FFS on the quality of the upper bound obtained, although the increased time requirement suggests that it may only be justified for large instances.

The node-type representation is not suitable for the bound-based approach because of its substantial time requirement. The sibling-type nonanticipativity representation fails to provide good bounds, although it requires the least time among the three nonanticipativity representation types. The branch-type representation, combined with individual nonanticipativity constraint relaxation, tends to provide both good lower and upper bounds. When the problem size becomes large, it is more time-efficient than directly solving the DEP with branch-type nonanticipativity constraints.

CHAPTER 6

Conclusions

In the first part of this dissertation research, we investigate on the impact and effectiveness of optimality tests in sample-based algorithms for two-stage stochastic linear programs with recourse (TS-SLP). It is the first time that computational experiments are designed and implemented to compare the impact of the optimality tests employed in the two versions of the stochastic decomposition (SD) methods. Our computational results indicate that, with more streamlined optimality tests, the regularized version of SD (RSD) appears to be more efficient than its linear counterpart. This research provides the first empirical comparison of the sample average approximation (SAA) with another sample-based algorithm (RSD) in TS-SLP. The results of our empirical investigation suggest that, in solving large scale TS-SLP instances, RSD and SAA can achieve similar solution quality, although RSD demands substantially less computational effort than SAA.

During this research phase, we explore and design computational platforms for meaningful comparisons between sample-based algorithms for TS-SLP. The experience gained during the design and implementation of the computational experiments provides guidelines for research on the computational aspect of algorithm design in TS-SLP.

The second part of this dissertation research focuses on bound-based solution validation for multistage stochastic linear programs with recourse (MS-SLP). Valid lower bounds on the optimal objective value of MS-SLP can be obtained through eliminating a subset of the nonanticipativity constraints in the deterministic equivalent program (DEP). We examine three representations of the nonanticipativity constraints, four relaxation schemes, and four relaxation levels. To complement these lower bounds, we examine three methods for calculating upper bounds on the optimal objective value through manipulations of the partially implementable

solutions obtained from the lower bound procedures.

Based on the results of our empirical investigations, we observe that the nonanticipativity constraint representation has significant impact on both lower and upper bound quality. It also has a significant impact on the time required by the lower bound procedures, but has relatively insignificant impact on the time required by the upper bound procedures.

Based on our experimental results, with the branch-type nonanticipativity representation and individual nonanticipativity constraint relaxation scheme, we are able to obtain good lower bounds with up to 50% of the nonanticipativity constraints eliminated from the DEP. This observation provides some insights on improving directions in the design of Lagrangian relaxation based MS-SLP algorithms. The reduction in the number of nonanticipativity constraints corresponds to the reduction of dual multipliers in the Lagrangian relaxation of the MS-SLP problem, which in turn suggests reduction in the time required by these procedures. The reduction in the time required by the lower bound procedure, which is currently the bottleneck for our bound-based approach, can improve the efficiency of the approach.

In our proposed scenario-based recursive subtree decomposition (SRSD) approach, a scenario tree is decomposed into subtrees rooted at the children nodes of its root node. In this way, we are able to maintain the nonanticipativity structure within each subtree. The simplicity of the embedded free forward scenario procedure reduces the computational effort to obtain an upper bound, while the quality of the upper bound is improved by recursive evaluation of the bound gap in each subtree. SRSD provides upper bounds with satisfactory quality and becomes efficient when problem size is large.

The solution of the first scenario in the subtree and the solution of the root node of the subtree have a significant impact on the computational performance and upper bound quality of the SRSD procedure, which demands a proper representation of the nonanticipativity constraints. The rule of thumb is that we want to have a "representative" scenario whose solution is least affected by the partial elimination of the nonanticipativity constraints. The branch-type and node-type nonanticipativity

representations are appropriate in this regard, although the former yields superior computational times. There is still room to improve in this research direction.

When it is impossible to enumerate all the scenarios, we may resort to sample-based approaches. Still, for the MS-SLP corresponding to a collected sample, bound-based evaluation may be more preferable than exact objective function evaluation, especially in the early iterations of an adaptive sample-based algorithm. Also, when we add a new generated scenario into the existing sample, we may not need to enforce all of the associated nonanticipativity constraints.

Last, we build our analysis on the observations of our computational experiment, which may be subject to impact of the problem models and test instances. Empirical investigations with more problem models and test instances are needed.

REFERENCES

Benders, J. F. (1962). Partitioning Procedures for Solving Mixed Variables Programming Problems. *Numerische Mathematik*, **4**, pp. 238–252.

Birge, J. R. (1985). Decomposition and Partitioning Methods for Multistage Stochastic Linear Programs. *Operations Research*, **33**(5), pp. 989–1007.

Birge, J. R. (1997). Stochastic Programming Computation and Applications. *INFORMS Journal on Computing*, **9**(2), pp. 111–133.

Birge, J. R., M. Dempster, H. I. Gassmann, A. J. Gunn, King, E. A., and S. W. Wallace (1987). A Standard Input Format for Multiperiod Stochastic Linear Programs. *COAL Newsletter*, **17**, pp. 1–19.

Birge, J. R., C. J. Donohue, D. F. Holmes, and O. G. Svintsitski (1996). A Parallel Implementation of the Nested Decomposition Algorithm for Multistage Stochastic Linear Programs. *Mathematical Programming*, **75**, pp. 327–352.

Birge, J. R. and F. V. Louveaux (1988). A Multicut Algorithm for Two-stage Stochastic Linear Programs. *European Journal of Operational Research*, **34**, pp. 384–392.

Chen, Z. and W. B. Powell (1999). Convergent Cutting-Plane and Partial-Sampling Algorithm for Multistage Stochastic Linear Programs with Recourse. *Journal of Optimization Theory and Applications*, **102**(3), pp. 497–524.

Dantzig, G. B. (1963). *Linear Programming and Extensions*. Princeton University Press, New Jersey.

Danzig, G. B. and P. Wolfe (1961). The Decomposition Algorithm for Linear Programs. *Econometrica*, **29**(4), pp. 767–778.

Edwards, J., J. R. Birge, and L. Nazareth (1985). A Standard Input Format for Computer Codes Which Solve Stochastic Programs with Recourse and a Library of Utilities to Simplify Its Use. Working Paper WP-85-03, International Institute for Applied Systems Analysis, Laxenburg, Austria.

Efron, B. (1979). Another Look at the Jackknife. *Annals of Statistics*, **7**, pp. 1–26.

Fortin, M. and R. Glowinski (1983). On Decomposition Coordination Methods Using an Augmented Lagrangian. In Fortin, M. and R. Glowinski (eds.) *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems*, pp. 97–146. North Holland, Amsterdam.

Gassmann, H. I. (1990). MSLiP: A Computer Code for the Multistage Stochastic Linear Programming Problem. *Mathematical Programming*, **47**, pp. 407–423.

Gassmann, H. I. and E. Schweitzer (2001). A Comprehensive Input Format for Stochastic Linear Programs. *Annals of Operations Research*, **104**, pp. 89–125.

Halgason, T. and S. W. Wallace (1991). Approximate Scenario Solutions in the Progressive Hedging Algorithm. *Annals of Operations Research*, **31**, pp. 425–444.

Higle, J. L., W. W. Lowe, and R. Odio (1994). Conditional Stochastic Decomposition: An Algorithmic Interface for Optimization and Simulation. *Operations Research*, **42**(2), pp. 311–322.

Higle, J. L., B. Rayco, and S. Sen (2004). Stochastic Scenario Decomposition for Multi-Stage Stochastic Programs. *submitted to Annals of Operations Research*.

Higle, J. L. and S. Sen (1991). Stochastic Decomposition: An Algorithm for Two-stage Linear Programs with Recourse. *Mathematics of Operations Research*, **16**, pp. 650–669.

Higle, J. L. and S. Sen (1994). Finite Master Programs in Stochastic Decomposition. *Mathematical Programming*, **67**, pp. 143–168.

Higle, J. L. and S. Sen (1996a). Duality and Statistical Tests of Optimality for Two Stage Stochastic programs. *Mathematical Programming*, **75**, pp. 257–275.

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

Higle, J. L. and S. Sen (1999). Statistical Approximations for Stochastic Linear Programming Problems. *Annuals of Operations Research*, **85**, pp. 173–192.

Kelly, J. E. (1960). The Cutting Plane Method for Convex Programs. *Journal of SIAM*, **8**, pp. 703–712.

Kleywegt, A. J., A. Shapiro, and T. Homem-de mello (2001). The Sample Average Approximation Method for Stochastic Discrete Optimization. *SIAM Journal on Optimization*, **12**(2), pp. 479–502.

Linderoth, J., A. Shapiro, and S. Wright (2002). The Empirical Behavior of Sampling Methods for Stochastic Programming. Optimization Technical Report 02-01, University of Wisconsin-Madison.

Linderoth, J. and S. Wright (2002). Decomposition Algorithms for Stochastic Programming on a Computational Grid. Optimization Technical Report 02-07, University of Wisconsin-Madison.

Louveaux, F. V. and Y. Smeers (1988). Optimal Investments for Electricity Generation: A Stochastic Model and a Test Problem. In Y., E. and R. J.-B. Wets (eds.) *Numerical Techniques for Stochastic Optimization Problems*, pp. 445–452. Springer-Verla, Berlin.

Mak, W. K., D. P. Morton, and R. K. Wood (1999). Monte Carlo Bounding Techniques for Determining Solution quality in Stochastic Programs. *Operations Research Letters*, **24**, pp. 47–56.

Mulvey, J. M. and A. Ruszczyński (1992). A Diagonal Quadratic Approximation Method for Large-scale Linear Programs. *Operations Research Letters*, **12**, pp. 205–215.

Mulvey, J. M. and A. Ruszczyński (1995). A New Scenario Decomposition Method for Large-scale Stochastic Optimization. *Operations Research*, **43**(3), pp. 477–490.

Mulvey, J. M. and H. Vladimirou (1991). Solving Multistage Stochastic Networks: An Application of Scenario Aggregaion. *Networks*, **21**, pp. 619–643.

Plambeck, E. L., F. B.-R. R. S., and R. Suri (1996). Sample-path Optimization of Convex Stochastic Performance Functions. *Mathematical Programming*, **75**, pp. 137–176.

Rockafellar, R. T. (1976). Monotone Operators and the Proximal Point Algorithm. *SIAM Journal of Control Optimazation*, **14**, pp. 877–898.

Rockafellar, R. T. and R. J.-B. Wets (1991). Scenarios and Policy Aggregation in Optimization Under Uncertainty. *Mathematics of Operations Research*, **16**(1), pp. 119–147.

Rosa, C. H. and A. Ruszczyński (1996). On Augmented Lagrangian Decomposition Methods for Multistage Stochastic Programs. *Annals of Operations Research*, **64**, pp. 289–309.

Ruszczyński, A. (1986). A Regularized Decomposition Method for Minimizing A Sum of Polyhedral Functions. *Mathematical Programming*, **35**, pp. 309–333.

Ruszczyński, A. (1993). Regularized Decomposition of Stochastic Programs: Algorithmic Techniques and Numerical Results. Working Paper WP-93-21, International Institute for Applied Systems Analysis, Laxenburg, Austria.

Ruszczyński, A. (1999). Some Advances in Decomposition Methods for Stochastic Linear Programming. *Annals of Operations Research*, **85**, pp. 153–172.

Ruszczyński, A. and A. Świętanowski (1997). Accelerating the Regularized Decomposition Method for Two Stage Stochastic Linear Problems. *European Journal of Operational Research*, **101**, pp. 328–342.

Sen, S., R. D. Doverspike, and S. Cosares (1994). Network Planning with Random Demand. *Telecommunications Systems*, **3**, pp. 11–30.

Shapiro, A. (1991). Asymptotic Analysis of Stochastic Programs. *Annals of Operations Research*, **30**, pp. 169–186.

Shapiro, A. (2000). Stochastic Programming by Monte Carlo Methods. *Available at http://dochost.rz.hu-berlin.de/speps/ as of December 21, 2004*, (3), pp. 1–29.

Shapiro, A., T. Homem-De-Mello, and J. Kim (2002). Conditioning of Convex Piecewise Linear Stochastic Programs. *Mathematical Programming*, **94**(1), pp. 1–9.

Van Slyke, R. M. and R. J.-B. Wets (1969). L-shaped Linear Programs with Applications to Optimal Control and Stochastic Programming. *SIAM Journal on Applied Mathematics*, **17**, pp. 638–663.

Vladimirou, H. (1990). *Stochastic Networks: Solution Methods and Applications in Financial Planning*. Ph.D. thesis, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ.