Modelling and solution methods for stochastic optimisation

Victor Zverovich

A thesis submitted for the degree of Doctor of Philosophy

School of Information Systems, Computing and Mathematics,
Brunel University

May 2011

Dedicated to my wife Natallia

Table of Contents

Abstract

In this thesis we consider two research problems, namely, (i) language constructs for modelling stochastic programming (SP) problems and (ii) solution methods for processing instances of different classes of SP problems. We first describe a new design of an SP modelling system which provides greater extensibility and reuse. We implement this enhanced system and develop solver connections. We also investigate in detail the following important classes of SP problems: single-stage SP with risk constraints, two-stage linear and stochastic integer programming problems. We report improvements to solution methods for single-stage problems with second-order stochastic dominance constraints and two-stage SP problems. In both cases we use the level method as a regularisation mechanism. We also develop novel heuristic methods for stochastic integer programming based on variable neighbourhood search. We describe an algorithmic framework for implementing decomposition methods such as the L-shaped method within our SP solver system. Based on this framework we implement a number of established solution algorithms as well as a new regularisation method for stochastic linear programming. We compare the performance of these methods and their scale-up properties on an extensive set of benchmark problems. We also implement several solution methods for stochastic integer programming and report a computational study comparing their performance. The three solution methods, (a) processing of a single-stage problem with second-order stochastic dominance constraints, (b) regularisation by the level method for two-stage SP and (c) method for solving integer SP problems, are novel approaches and each of these makes a contribution to knowledge.

Acknowledgements

I am grateful to OptiRisk Systems for the financial support of my research and to my supervisor, Prof. Gautam Mitra, for his input and guidance. Also I would like to thank my colleagues, Prof. Csaba Fabian, Dr. Francis Ellison, Dr. Cormac Lucas, Dr. Diana Roman and fellow PhD students, Christian Valente, Katharina Schwaiger, Michael Sun and Dominik Hollmann. With their help the time I spent at Brunel University was both enjoyable and productive.

I would like to thank my wife Natallia whose love and care gave me strength throughout these years. I am also grateful to my parents who despite being far away inspired me by their lives dedicated to teaching and scientific work and to my brothers, Alexey and Vadim. They both encouraged me and I knew that I could always count on them.

List of Abbreviations

ALM	Asset Liability Management
AML	Algebraic Modelling Language
AST	Abstract Syntax Tree
CC	Chance Constraint
CDF	Cumulative Distribution Function
COM	Component Object Model
CVaR	Conditional Value at Risk
DBB	Decomposition-based Branch and Bound
DEP	Deterministic Equivalent Problem
ICC	Integrated Chance Constraint
IDE	Integrated Development Environment
IPM	Interior Point Method
JVM	Java Virtual Machine
LP	Linear Programming
MIP	Mixed Integer Programming
MP	Mathematical Programming
OR	Operational Research
QP	Quadratic Programming
RD	Regularised Decomposition
RHS	Right Hand Side
RINS	Relaxation Induced Neighbourhood Search
RO	Robust Optimisation
SIP	Stochastic Integer Programming
SOCP	Second-Order Cone Programming
SP	Stochastic Programming
SSD	Second-order Stochastic Dominance

TR	Trust Region
TSP	Travelling Salesman Problem
VaR	Value at Risk
VNDS	Variable Neighbourhood Decomposition Search
VNS	Variable Neighbourhood Search
VNSB	Variable Neighbourhood Search Branching

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Since the seminal work of Dantzig (1955) which introduced linear programming under uncertainty, it has been widely recognised that real world problems often include some degree of uncertainty. Several modelling frameworks have been established that rely on different types of information available about the uncertain parameters. One such framework is stochastic programming (SP) with recourse which assumes that the probability distribution of random parameters is known. Another approach which only assumes that the random parameters are known within certain bounds is robust optimisation.

The history of the development of SP can be traced in the following way. In 1980s linear programming (LP) as a decision model became established as methods for solving large linear programs evolved (Karmarkar, 1984). The theory of SP was well developed by the end of 1970s (Prékopa, 1973; Wets, 1974). However, only after the success of the application of LP was interest focused on these applications where uncertainty in model parameters could not be ignored. We set out below a range of papers and case studies which are examples of optimum decision making under uncertainty and illustrative applications of SP.

Transportation and logistics

Transportation problems are among the first applications of stochastic programming starting with the problem of optimal allocation of aircraft to airline routes under uncertain demand (Ferguson and Dantzig, 1956). Ermol'ev et al. (1976) describe a stochastic network model for planning of empty container shipment. SP has also a long history of applications in railroad car distribution problems.

These problems are inherently stochastic due to uncertain supply, demand and travel times over the railroad network. Jordan and Turnquist (1983) describe a model for distribution of empty freight cars. A comprehensive review of freight vehicle transportation problems is given by Dejax and Crainic (1987). Powell (1986) address a more general multistage model that is applicable in the context of truckload motor carriers.

Supply chain

In supply chain management models some parameters such as customer demands, prices, and resource capacities are uncertain which makes it a sensible application area for SP. Many stochastic models have been proposed both at strategic and tactical levels. At the strategic level much research has focused on facility location problems. Eppen et al. (1989) describe a model for finding optimal strategic investment decisions on types and locations of facilities. The model is motivated by the application in the automobile industry. An extensive review of strategic facility location models is given by Owen and Daskin (1998). A two-stage stochastic integer programming (SIP) problem for optimal design of production system topology under uncertainty is studied by Alonso-Ayuso et al. (2003a). Santoso et al. (2005) propose both an SP model and a solution method for an optimal design of real-scale supply chain networks.

Energy

Escudero et al. (1999) propose a modelling and solution framework for optimisation of oil supply, transformation and distribution under uncertainty. The stochasticity in this model is due to uncertain product demand, spot supply cost and spot selling price. A recent gas portfolio planning model by König et al. (2007) is formulated as a two-stage SP problem with recourse. Wallace and Fleten (2003) give a review of SP models in energy sector.

Finance

Bradley and Crane (1972) formulate a portfolio model as a multistage decision problem which incorporates uncertainty in future interest rates and cash flows. Kallberg et al. (1982) and Kusy and Ziemba (1986) develop basic concepts of asset-liability management (ALM) models under uncertainty. Their work has been followed by a number of sophisticated practical applications, such as the

Russel-Yasua Kasai model by Cariño et al. (1994) which is formulated as a multistage SP problem for optimising investment strategy. Consigli and Dempster (1998) describe the CALM model, a multistage stochastic programming model for asset-liability management "designed to deal with uncertainty in both assets (in either the portfolio or the market) and liabilities (in the form of scenario dependent payments or borrowing costs)".

1.2 Taxonomy of stochastic programming problems

In this section we introduce preliminary definitions and outline the following classes of stochastic programming models:

- single-stage SP,
- two-stage SP with recourse,
- multistage SP with recourse,
- chance constraints,
- integrated chance constraints,
- robust optimisation.

Here we use the term *stochastic programming* in a broad sense to denote optimisation under uncertainty in general.

In the SP setting some of the problem components, that is, model parameters such as constraint matrix elements, objective function coefficients, variable or constraint bounds may take random values. To distinguish these from decision variables we use the term *random parameters*.

Consider a probability space $(\Omega, \mathcal{F}, P)$, where

- Ω is the set of all possible outcomes,
- $\mathcal{F}$ is the σ-algebra on Ω,
- $P : \mathcal{F} \to [0, 1]$ is the probability measure.

The σ-algebra $\mathcal{F}$ is a collection of events, where each event $F \in \mathcal{F}$ is a subset of Ω. As a σ-algebra $\mathcal{F}$ satisfies the following three properties:

1. $\emptyset \in \mathcal{F}$,

2. $\mathcal{F}$ is closed under the set complement operation:
   if $A \in \mathcal{F}$, then $\Omega \setminus A \in \mathcal{F}$,

3. $\mathcal{F}$ is closed under the union of a countable number of sets:
   if $A_i \in \mathcal{F}, i = 1, 2, \ldots$, then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$.

Define by $\mathcal{A}$ the set of naturally measurable sets in $\mathbb{R}^k$. A k-dimensional random vector $\boldsymbol{\xi}$ defined on the probability space $(\Omega, \mathcal{F}, P)$ is a function $\boldsymbol{\xi} : \Omega \to \mathbb{R}^k$ such that $\boldsymbol{\xi}^{-1}(A) = \{\omega | \boldsymbol{\xi}(\omega) \in A\} \in \mathcal{F}$, $\forall A \in \mathcal{A}$. A random vector induces a probability measure $P_{\boldsymbol{\xi}}$ on $\mathcal{A}$. The support of the probability space for the random vector $\boldsymbol{\xi}$ is defined as the smallest closed set $\Xi \subseteq \Omega$ such that $P_{\xi}(\Xi) = 1$.

General SP problem

A general stochastic programming problem is stated as

$$\begin{array}{ll} \text{"minimize"} & g_0(\boldsymbol{x}, \boldsymbol{\xi}) \\ \text{subject to} & g_i(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0, \quad i = 1, \ldots, m, \\ & \boldsymbol{x} \in X \subset \mathbb{R}^n, \end{array} \tag{1.1}$$

where $\boldsymbol{\xi}$ is a vector of random parameters defined on the probability space $(\Omega, \mathcal{F}, P)$. It is assumed that the probability measure P does not depend on $\boldsymbol{x}$ and that at the time the decision is made the probability distribution, but not the actual realisations, is known. The functions $g_i(\boldsymbol{x}, \cdot) : \Xi \to \mathbb{R}$ are random variables for every $\boldsymbol{x}$ and $i = 0, \ldots, m$.

However, the meaning of optimisation with unspecified realisation of $\boldsymbol{\xi}$ is not clear. Hence the problem (1.1) is not well-defined and we need to further specify it in the following alternative formulations.

Single-stage SP problem

A single stage SP problem involves one set of decisions which are made before the uncertainty is disclosed. This type of problem is often used in finance, in particular in portfolio management, where preferences between random returns of portfolios are established in a single-stage environment. Mean-risk and stochastic dominance models are classical examples of single-stage models. Chance constraints and integrated chance constraints are also often considered in the context of single-stage problems. In Chapter 3 we consider this class of models.

Two-stage SP problem with recourse

A two-stage stochastic programming problem with recourse consists of two sets of decisions. The sequence of events in this problems is as follows:

1. the first-stage or "here-and-now" actions are made,
2. uncertainty is revealed,
3. the second-stage or recourse actions are made.

The two-stage linear stochastic programming problem with recourse is defined as

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} + \mathrm{E}_{\boldsymbol{\xi}}[Q(\boldsymbol{x},\omega)] \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \in \mathbb{R}^{n_1}_+, \end{array} \tag{1.2}$$

where the n_1-dimensional vector $\boldsymbol{x}$ represents the first-stage decisions, A is a fixed $m_1 \times n_1$ matrix, $\boldsymbol{b} \in \mathbb{R}^{m_1}$ and $\boldsymbol{c} \in \mathbb{R}^{n_1}$ are fixed vectors and $Q(\boldsymbol{x},\omega)$ is the value function of the recourse problem

$$\begin{array}{ll} \text{minimize} & \boldsymbol{q}(\omega)^T\boldsymbol{y} \\ \text{subject to} & T(\omega)\boldsymbol{x} + W(\omega)\boldsymbol{y} = \boldsymbol{h}(\omega), \\ & \boldsymbol{y} \in \mathbb{R}^{n_2}_+. \end{array} \tag{1.3}$$

In (1.3) the n_2-dimensional vector $\boldsymbol{y}$ represents the second-stage recourse decisions and $\omega \in \Omega$ represents a random outcome. For a given realisation ω, $T(\omega)$ is a fixed $m_2 \times n_1$ matrix, $W(\omega)$ is a fixed $m_2 \times n_2$ matrix, $\boldsymbol{h}(\omega) \in \mathbb{R}^{m_2}$ and $\boldsymbol{q}(\omega) \in \mathbb{R}^{n_2}$ are fixed vectors.

The problem (1.2)-(1.3) can be generalised to a non-linear case. Consider the general problem formulation (1.1). The i-th constraint is violated for realisation $\hat{\boldsymbol{\xi}}$ if and only if $g_i(\boldsymbol{x},\hat{\boldsymbol{\xi}})^+ > 0$, where the plus sign in a superscript denotes a positive part, i.e. $g_i(\boldsymbol{x},\hat{\boldsymbol{\xi}})^+ = \max(g_i(\boldsymbol{x},\hat{\boldsymbol{\xi}}),0)$. For each violated constraint a recourse action defined by decision $y_i(\boldsymbol{\xi})$ can be taken such that $g_i(\boldsymbol{x},\hat{\boldsymbol{\xi}}) - y_i(\hat{\boldsymbol{\xi}}) \leq 0$.

Generalising the recourse functions to be nonlinear and introducing the costs for the recourse actions leads to the following formulation:

$$\begin{array}{ll} \text{minimize} & \mathrm{E}_{\boldsymbol{\xi}}[g_0(\boldsymbol{x},\boldsymbol{\xi}) + Q(\boldsymbol{x},\boldsymbol{\xi})] \\ \text{subject to} & \boldsymbol{x} \in X \subset \mathbb{R}^{n_1}, \end{array} \tag{1.4}$$

where $Q(\boldsymbol{x},\boldsymbol{\xi})$ is a value function of the recourse problem

$$\begin{aligned} &\text{minimize} && q(\boldsymbol{y}) \\ &\text{subject to} && H_i(\boldsymbol{y}) \geq g_i(\boldsymbol{x}, \boldsymbol{\xi})^+, \quad i = 1, \ldots, m, \\ & && \boldsymbol{y} \in Y \subset \mathbb{R}^{n_2}. \end{aligned} \tag{1.5}$$

In (1.5) the function $H_i(\boldsymbol{y})$ represents a recourse action for the i-th constraint and $q(\boldsymbol{y})$ is the recourse cost function.

Multistage SP problem with recourse

In a multistage stochastic programming problem there is a set of decisions associated with each stage and between the stages realisation of random events take place. It is assumed that at each stage t the decisions at previous stages $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_{t-1}}$ and the realisations of the random vectors $\boldsymbol{\xi_2}, \boldsymbol{\xi_3}, \ldots \boldsymbol{\xi_t}$ are known.

A multistage linear SP with recourse is stated as

$$\begin{aligned} &\text{minimize } \boldsymbol{c_1^T x_1} + \mathrm{E}_{\boldsymbol{\xi_2}}[\min \boldsymbol{c_2}(\omega)^T \boldsymbol{x_2} + \cdots + \mathrm{E}_{\boldsymbol{\xi_K}}[\min \boldsymbol{c_K}(\omega)^T \boldsymbol{x_K}] \ldots] \\ &\text{subject to} \\ &\begin{aligned} W_1 \boldsymbol{x_1} & & & & & = \boldsymbol{h_1}, \\ T_1(\omega)\boldsymbol{x_1} + W_2(\omega)\boldsymbol{x_2} & & & & & = \boldsymbol{h_2}(\omega), \\ T_2(\omega)\boldsymbol{x_2} + & W_3(\omega)\boldsymbol{x_3} & & & & = \boldsymbol{h_2}(\omega), \\ & \ddots & & & & \vdots \\ & & T_{K-1}(\omega)\boldsymbol{x_{K-1}} + & W_T(\omega)\boldsymbol{x_K} & & = \boldsymbol{h_K}(\omega), \end{aligned} \\ &\boldsymbol{x_t} \in \mathbb{R}^{n_t}_+, \quad t = 1, \ldots, K, \end{aligned} \tag{1.6}$$

where W_1 is a fixed $m_1 \times n_1$ matrix, $\boldsymbol{h_1} \in \mathbb{R}^{m_1}$ and $\boldsymbol{c_1} \in \mathbb{R}^{n_1}$ are fixed vectors and for each realisation of $(\xi_2, \ldots \xi_t), t = 2, \ldots, K$, $W_t(\omega)$ is a fixed $m_t \times n_t$ matrix, $T_{t-1}(\omega)$ is a fixed $m_t \times n_{t-1}$ matrix, $\boldsymbol{h_t}(\omega) \in \mathbb{R}^{m_t}$ and $\boldsymbol{c_t}(\omega) \in \mathbb{R}^{n_t}$ are fixed vectors.

The random vector $\boldsymbol{\xi_t}(\omega)$ consists of random components of vectors $\boldsymbol{c_t}(\omega)$ and $\boldsymbol{h_t}(\omega)$ and matrices $T_{t-1}(\omega)$ and $W_t(\omega)$. In the case of discrete finite distribution of random parameters the structure of the model can be represented in the form of a scenario tree or an event tree.

Figure 1.1 illustrates the scenario tree of a 4-stage SP problem. Each arc represents a realisation of a random vector of parameters between the two stages. Each node represents a subproblem corresponding to a given stage and a sequence of realisations of random parameters determined by a path from the root node. A path from the root to a leaf node represents a particular scenario.

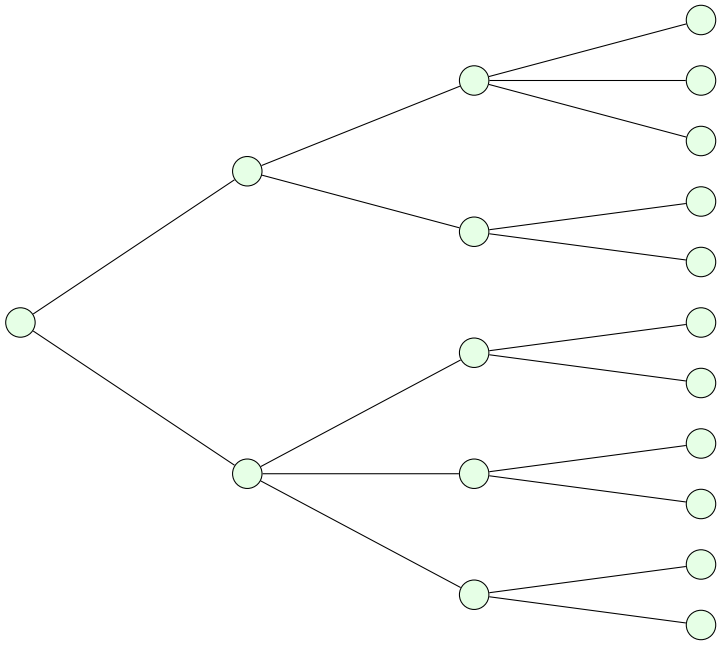


Figure 1.1: Example of a scenario tree

Chance constraints

In SP problems with recourse it is assumed that a constraint must be satisfied almost surely. Within the chance-constrained (CC) programming framework this is replaced with the requirement that a constraint must be satisfied with some probability. Consider the i-th constraint of a general SP problem:

$$g_i(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0. \tag{1.7}$$

It can be restated as a chance or probabilistic constraint as

$$P\{\boldsymbol{\xi} | g_i(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0\} \geq \alpha_i, \tag{1.8}$$

where $\alpha_i \in (0, 1)$ is a parameter which specifies the minimum probability of satisfying the constraint (1.7).

A joint chance constraint is stated as

$$P\{\boldsymbol{\xi} | g_i(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0, i \in I\} \geq \alpha, \tag{1.9}$$

where $\alpha \in (0, 1)$ is a parameter which specifies the minimum joint probability of satisfying the constraints with indices in $I \subseteq \{1, \ldots, m\}$.

Chance constraints are based on qualitative risk concept in a sense that they take into account only the fact of constraint violation but not the amount by which it is violated.

Integrated chance constraints

Integrated chance constraints (ICC) introduced by Klein Haneveld (1986) are related to probabilistic constraints. Both CC and ICC allow violation of the underlying constraints. In integrated chance constraints instead of dealing only with the probability of this realisation, the expected amount of violation (expected shortfall or surplus) is also restricted. Therefore ICCs are considered to be based on quantitative risk concept.

Consider the i-th constraint of a general SP problem:

$$g_i(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0. \tag{1.10}$$

It can be restated as an integrated chance constraint as

$$\mathrm{E}_{\boldsymbol{\xi}}[g_i(\boldsymbol{x}, \boldsymbol{\xi})^+] \leq \beta_i, \tag{1.11}$$

where $\beta_i \in \mathbb{R}_+$ is a parameter which specifies the maximum expected amount of violation of constraint (1.10).

A joint integrated chance constraint is stated as

$$\mathrm{E}_{\boldsymbol{\xi}}\left[\max_{i \in I} g_i(\boldsymbol{x}, \boldsymbol{\xi})^+\right] \leq \beta, \tag{1.12}$$

where $\beta \in \mathbb{R}_+$ is a parameter which specifies the maximum expected amount of violation of constraints with indices in $I \subseteq \{1, \ldots, m\}$.

Robust optimisation

A general form of a robust formulation problem is as follows:

$$\begin{array}{ll} \text{minimize} & g_0(\boldsymbol{x}) \\ \text{subject to} & g_i(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0, \quad i = 1, \ldots, m, \\ & \boldsymbol{x} \in X \subset \mathbb{R}^n, \\ & \boldsymbol{\xi} \in \Xi. \end{array} \tag{1.13}$$

Unlike models considered earlier in this section, robust optimisation does not assume the knowledge of the distribution of the uncertain parameters represented by the vector $\boldsymbol{\xi}$. It only assumes that $\boldsymbol{\xi}$ belongs to a known set Ξ.

There are several models that rely on different representation of uncertainty set Ξ. For instance in the classic model of Soyster (1973) the columns of a constraint matrix belong to convex sets. Alternative formulations of robust optimisation models are considered in Chapter 2.

1.3 Structure of the thesis

In this thesis we consider optimisation under uncertainty both from the perspective of modelling and that of solving such models. We recognise the importance of having a choice of modelling constructs because different problems may require different modelling approaches. It may depend on the availability of information about the random parameters as discussed above or it may be due to some reliability requirements as is the case of risk constraints. Therefore we do not focus on one particular modelling paradigm but propose a number of modelling language extensions that facilitate the development of optimisation models using different frameworks.

In addition to two-stage and multistage SP with recourse we have identified the following important modelling approaches:

- chance constraints,
- integrated chance constraints,
- robust optimisation.

In Chapter 2 we describe extensions for expressing these additional types of models and discuss the interface between a modelling system and an SP solver.

There are several reasons why it is important to have an algebraic modelling language (ALM) support for additional classes of models.

First it frees the modeller from the necessity of using deterministic equivalent formulations of corresponding constructs, which are often error-prone and verbose, cluttering the model with auxiliary variables and constraints. Instead direct representation of such constructs in an ALM allows the model to be kept clean with the modeller's intent clear. It is then the responsibility of a translator or a solver to reformulate a problem introducing extra variables and constraints.

Second it allows the translator to capture important information about the structure of the model and pass it to the solver facilitating the use of specialised algorithms which exploit this information. Advantages of such explicit formulation and capturing special problem structures has been discussed by Fourer and Gay (1995) for the case of network and piecewise-linear constructs and by Colombo et al. (2009) for problems with block structure such as two-stage SP problems. In Chapter 3 we illustrate both the modelling and computational benefits of this approach on a portfolio choice model formulating it directly with the extensions for representing integrated chance constraints and using a specialised cutting-plane algorithm of Klein Haneveld and van der Vlerk (2006) to solve it.

Although we recognise the practical importance of scenario generation we do not try to address it in this thesis and refer the reader to Mitra et al. (2007) and Di Domenica et al. (2009) for a discussion of this aspect of an SP modelling system.

In Chapter 3 we consider problems with risk constraints. By risk we mean the possibility and impact of undesirable outcomes. It is often assumed that a decision maker exhibits risk averse behaviour. We describe a portfolio choice model based on second-order stochastic dominance (SSD) criterion which is consistent with this assumption. The portfolio selection problem is an example of decision making under risk and it is of great importance in the area of quantitative fund management. In this problem which is formally set out in Chapter 3 an investor has to decide what proportion of the initial wealth to invest in each asset with uncertain return distributions. Our model is based on the one by Roman, Darby-Dowman, and Mitra (2006). We discuss the relationship between SSD, Conditional Value-at-Risk (CVaR) and integrated chance constraints and present alternative formulations of this model.

The computational results presented in Section 3.6 demonstrate that our model gives higher overall outcomes at the cost of being slightly more risky than the model of Roman et al. (2006) although optimal portfolios constructed by both models are SSD efficient, meaning that there are no other portfolios dominating them. We use a cutting-plane method to solve the model and investigate the advantages of using regularisation by the level method of Lemaréchal et al. (1995).

In Chapter 4 we focus on solution methods for two-stage stochastic programming problems with recourse. Significance of this class of problems is easily seen; for instance, in the review of the application in Section 1.1 most of the models reported are two-stage SPs. So the solution methods for two-stage SP remain an important research topic.

SP problems are known to be computationally challenging and the recent study of Dyer and Stougie (2006) shows that their complexity is primarily determined by the computation of a multidimensional integral. There are several approaches to solving two-stage SP problems. One approach relies on the solution of the deterministic equivalent problems (DEP) and it has received attention recently due to the advances in interior point methods (IPM) which are especially suitable for solving such large-scale problems with block structure.

Another approach which can be traced back to (Dantzig and Wolfe, 1960) and Dantzig and Madansky (1961) is decomposition. We describe a computational

framework for implementing decomposition-based methods. Based on this infrastructure we implement a classic L-shaped method of Van Slyke and Wets (1969) and several regularisation methods such as regularised decomposition and trust region method. We also propose a regularisation approach which is based on the level method.

We carry out an empirical study comparing the performance of the above methods to each other and to the DEP solution with an IPM and simplex solver. One of the aspects of this study is evaluation of scale-up properties of the algorithms. The tests are performed on a large number of benchmark problems from several established collections and to make the results comprehensible we use performance profiles (Dolan and Moré, 2002). This provides a clear visualisation allowing comparison of the algorithms across the whole set of test problems.

Stochastic integer programming problems are known to be computationally challenging; at the same time there is a practical need to solve this type of problems (some applications are given by Midler and Wollmer (1969), Subrahmanyam et al. (1994) and Sen et al. (1994)). Over the last two decades significant progress has been made both in exact and heuristic solution methods for two-stage and multistage stochastic integer programming (Laporte and Louveaux, 1993; Alonso-Ayuso et al., 2003b; Sen and Higle, 2005; Cristóbal et al., 2009; Escudero et al., 2011). The research team led by Escudero has reported successful application of SIP solution methods to large-scale two- and multistage problems (Escudero et al., 2007a,b; Escudero, 2009; Escudero et al., 2009a,b,c, 2010a,b). In this work we focus on a heuristic approach and consider a variant of variable neighbourhood search which has been successfully applied in a deterministic environment (Brimberg and Mladenović, 1996; Aouchiche et al., 2006; Dražić et al., 2008).

In Chapter 5 we investigate the applicability of variable neighbourhood decomposition search to stochastic integer problems. We implement a classic integer L-shaped method of Laporte and Louveaux (1993) and use it both in performance benchmarks comparing it with our heuristic method and as an underlying solution method to solve subproblems. We also describe out implementation of the integer L-shaped method which is interesting because it to a large extent reuses existing branch-and-cut infrastructure.

In Chapter 6 we summarise the findings reported in the thesis and present our conclusions.

Chapter 2

Stochastic programming modelling constructs and extensions

In this chapter we consider the language structure and syntax for presenting alternative stochastic programming (SP) models to an SP modelling system. In Section 2.1 we discuss existing formats for representing SP problems and in particular SMPS and SP extensions to established modelling languages. We compare several extensions of the AMPL modelling language and describe one of them, SAMPL, in more details. Design and implementation of a new SAMPL translator is discussed in Section 2.2. The three remaining sections give details for one particular set of extensions. Section 2.3 is devoted to describing chance-constrained modelling constructs. Section 2.4 describes the syntax of integrated chance constraints. In Section 2.5 we introduce language constructs for alternative robust optimisation models.

2.1 Introduction to the SAMPL modelling language

AMPL is an algebraic modelling language (AML) for mathematical programming (Fourer, Gay, and Kernighan, 2003) designed for representing linear and nonlinear optimisation problems in discrete or continuous variables. One of the notable features of AMPL is the similarity of its syntax to the mathematical notation for describing optimisation models. Stochastic AMPL or SAMPL (Valente, Mitra, Sadki, and Fourer, 2009) is an extension of the AMPL language that enables

formulation of stochastic programming problems. SAMPL supports two-stage and multistage scenario-based SP problems with recourse. SAMPL is used in Stochastic Programming Integrated Environment (SPInE) which is an integrated development environment for SP modelling (Valente, Mitra, and Poojari, 2005).

SAMPL is a relatively new language. The most important among earlier formats is arguably SMPS that was introduced by Birge, Dempster, Gassmann, Gunn, King, and Wallace (1987). In a way comparable with the role of the MPS format for linear programming, SMPS has become a de facto standard representation of stochastic programming problems. The following established collections of benchmark problems for stochastic linear and integer programming use this format:

- POSTS (Holmes, 1995)
- Test Set for Stochastic Linear Programming (Ariyawansa and Felt, 2004)
- Test problems by Linderoth, Shapiro, and Wright (2002)
- SIPLIB (Ahmed, 2004)

However SMPS has certain limitations most of which are inherited from MPS. For instance, in both formats the direction of optimisation (minimisation or maximisation) is not specified and the precision is limited due to fixed width of numeric fields. Also both MPS and SMPS are column-oriented which is different from usual equation-oriented algebraic formulation of MP problems.

In the last decades algebraic modelling languages have gained wide acceptance of the OR community. This is supported by the statistics of the NEOS Server for Optimisation (Czyzyk et al., 1998) given in Table 2.1. Figures show that algebraic modelling languages (AMPL and GAMS) together account for almost 90% of the NEOS submissions in January 2011. Interestingly MPS is used in less than 1% of cases despite being supported by most solvers. Among other low-level formats LP (IBM Corp., 2009a) leads by a wide margin.

At the same time stochastic programming has become an important decision tool as suggested by various developments in this field reflected on the active website `http://stoprog.org` of the SP community and the triennial international conference on stochastic programming. The edited volume by Wallace and Ziemba (2005) describes many applications of stochastic programming in diverse domains and outlines the SP modelling systems. Our analysis of the SP modelling and solver requirements reveals that modelling support, scenario generation and

Table 2.1: Solver inputs statistics for January 2011 from the NEOS Server for Optimization

Solver Input	Submissions	Percentage
AMPL	7199	59.4
GAMS	3602	29.7
LP	714	5.9
CPLEX	110	0.9
MATLAB_BINARY	101	0.8
SPARSE_SDPA	98	0.8
MPS	94	0.8
TSP	90	0.7
SMPS	27	0.2
Other	76	0.6

solution methods are three important aspects of a working SP system. In the current chapter we focus entirely on the first aspect and we refer the readers to Mitra et al. (2007) and Di Domenica et al. (2009) for scenario generation and Chapters 3, 4 and 5 of this thesis for solution methods.

The developments in the SP field, together with growing popularity of AMLs, have resulted in a number of extensions to the modelling systems providing facilities to express stochastic programming problems. Major vendors of optimisation software, namely, XPRESS, AIMMS, MAXIMAL, and GAMS have started offering such extensions to their optimisation suites (Dormer et al., 2005; Roelofs and Bisschop, 2010; Dirkse, 1998).

Several SP extensions to the AMPL modelling language have been proposed by different authors:

- Stochastic programming extensions to AMPL by Fourer (1996),
- SML (Colombo et al., 2009),
- StAMPL (Fourer and Lopes, 2009) - multistage stochastic programming problem with recourse.
- SAMPL (Valente et al., 2009),

The extensions proposed by Fourer introduce a concept of scenarios into AMPL allowing the association of different data within the same model. Together with a new statement for defining a stochastic framework for a model and

scenarios, this enables formulation of stochastic programming problems with recourse. At the time of writing, these extensions have not been implemented in the AMPL translator whereas Fourer has participated with the CARISMA team to define SAMPL (Valente, Mitra, Sadki, and Fourer, 2009).

SML is described by Colombo et al. as a structure-conveying algebraic modelling language for mathematical programming based on AMPL. Unlike the previous set of extensions SML is implemented, although not in the AMPL translator itself but as a sequence of pre- and post-processing passes for AMPL. This modelling language allows the block structure of a problem to be preserved and passed to the solver which can exploit this information. SML also provides modelling facilities to express SP problems with recourse based on the nodal description of the scenario tree.

StAMPL is an extension of the AMPL modelling language for multistage stochastic programming problems with recourse. It is based on the idea that every such problem contains a filtration process and provides a notation for representing this process.

SAMPL is an algebraic modelling language based on AMPL which allows representation of two- and multistage SP problems with recourse. The language supports scenario-based formulation and compact representation of several common tree structures as well as arbitrary scenario trees.

In this research we further develop the SAMPL modelling language providing support for additional classes of SP modelling constructs involving uncertainty and risk. Apart from stochastic programming with recourse which is already supported in SAMPL we have identified the following important approaches to handling uncertainty and risk in SP models:

- chance constraints which are closely related to Value at Risk (VaR),
- integrated chance constraints (ICCs) which are closely related to Conditional Value at Risk (CVaR),
- robust optimisation.

For a detailed description see CARISMA lecture notes on stochastic programming (CARISMA, 2010). The first two are important as they provide risk measures and can be used in scenario-based recourse problems thus complementing existing features of the SAMPL language. Computational aspects of solving problems with risk constraints are discussed in Chapter 3.

To introduce the extensions we first briefly describe the basic concepts and syntax of SAMPL. The latter inherits five types of entities from AMPL:

- sets,
- parameters,
- variables,
- objectives,
- constraints.

The entity names are self-descriptive, we should only clarify that variables denote decision variables and parameters are problem parameters. Each entity is either a single value or a collection indexed over a set called *indexing set*.

Listing 2.1 gives an example of an AMPL model which illustrates all five types of constructs mentioned above. This example is a deterministic version of the farmer's problem from Birge and Louveaux (1997).

An algebraic formulation of the farmer's problem is as follows:

Given	C	a set of crops (element 3 denotes sugar beets),
and	a	total area (acre),
	b_c	$c \in C$: yield of crop c (T / acre),
	d_c	$c \in C$: planting cost of crop c (\$ / acre),
	e_c	$c \in C$: selling price of crop c (\$ / T),
	f	selling price of sugar beets produced above quota (\$ / T),
	g_c	$c \in C$: purchase price of crop c (\$ / T),
	r_c	$c \in C$: minimum requirement of crop c (T),
	q	quota for sugar beets (T),
define	$x_c \geq 0$	$c \in C$: acres of land devoted to crop c,
	$w_c \geq 0$	$c \in C$: tons of crop c sold (at favourable price),
	$z \geq 0$	tons of sugar beets sold at the lower price,
	$y_c \geq 0$	$c \in C$: tons of crop c purchased,

Listing 2.1: Deterministic version of the farmer's problem formulated in AMPL

```
###  SETS  ###
set Crops;

###  PARAMETERS  ###
param TotalArea;              # acre
param Yield{Crops};           # T/acre
param PlantingCost{Crops};    # $/acre
param SellingPrice{Crops};    # $/T
param ExcessSellingPrice;     # $/T
param PurchasePrice{Crops};   # $/T
param MinRequirement{Crops}; # T
param BeetsQuota;             # T

###  VARIABLES  ###
# Area in acres devoted to crop c
var area{c in Crops} >= 0;

# Tons of crop c sold (at favourable price)
var sell{c in Crops} >= 0;

# Tons of sugar beets sold in excess of the quota
var sellExcess >= 0;

# Tons of crop c bought
var buy{c in Crops} >= 0;

###  OBJECTIVE  ###
maximize profit:
  sum{c in Crops} (SellingPrice[c] * sell[c] -
      PurchasePrice[c] * buy[c] - PlantingCost[c] * area[c]) +
  ExcessSellingPrice * sellExcess;

###  CONSTRAINTS  ###
subject to totalArea: sum{c in Crops} area[c] <= TotalArea;

subject to requirement{c in Crops}:
  Yield[c] * area[c] - sell[c] + buy[c] >= MinRequirement[c];

subject to quota: sell['beets'] <= BeetsQuota;

subject to sellBeets:
  sell['beets'] + sellExcess <= Yield['beets'] * area['beets'];
```

$$
\begin{aligned}
\text{maximize} \quad & \sum_{c \in C} (e_c w_c - g_c y_c - d_c x_c) + f z \\
\text{subject to} \quad & \sum_{c \in C} x_c \leq a, \\
& b_c x_c - w_c + y_c \geq r_c, \quad c \in C, \\
& w_3 \leq q, \\
& w_3 + z \leq b_3 x_3.
\end{aligned}
\tag{2.1}
$$

This example shows one of the main AMPL features, that is, the similarity of its syntax to mathematical notation. Indeed, it is straightforward to obtain the AMPL formulation in Listings 2.1 from the algebraic formulation (2.1).

In addition to the entities mentioned above SAMPL introduces extensions involving new entities which are

- scenario information:
  - scenario set,
  - tree structure,
  - scenario probabilities,
- random parameters,
- aggregation of variables into stages.

The SAMPL formulation of the farmer's problem restated as a two-stage SP model with recourse is shown in Listing 2.2. This model illustrates the additional constructs.

The SAMPL model is similar to the deterministic equivalent problem expressed in AMPL. However, the model in SAMPL has some important features that convey the underlying SP structure of the problem to the solver. First, the stage of each variable is specified with the help of `stage` suffixes. Second, the scenario set is clearly identified which allows subproblems for each specific scenario to be distinguished. Finally, the structure of the scenario tree is given explicitly.

Listing 2.2: Stochastic version of the farmer's problem formulated in SAMPL

```
###  SETS  ###
set Crops;

###  SCENARIO INFORMATION  ###
scenarioset Scenarios;          # Scenario set
probability P{s in Scenarios}; # P[s] is a probability
                                # of scenario s
tree Tree := twostage;          # Scenario tree structure

###  PARAMETERS  ###
param TotalArea;                # acre
param PlantingCost{Crops};      # $/acre
param SellingPrice{Crops};      # $/T
param ExcessSellingPrice;       # $/T
param PurchasePrice{Crops};     # $/T
param MinRequirement{Crops};    # T
param BeetsQuota;               # T

###  RANDOM PARAMETERS  ###
random param Yield{Crops, Scenarios}; # T/acre

###  VARIABLES  ###

# Area in acres devoted to crop c
var area{c in Crops} >= 0;

# Tons of crop c sold (at favourable price) under scenario s
var sell{c in Crops, s in Scenarios} >= 0, suffix stage 2;

# Tons of sugar beets sold in excess of the quota under
# scenario s
var sellExcess{s in Scenarios} >= 0, suffix stage 2;

# Tons of crop c bought under scenario s
var buy{c in Crops, s in Scenarios} >= 0, suffix stage 2;

###  OBJECTIVE  ###
maximize profit: sum{s in Scenarios} P[s] * (
    ExcessSellingPrice * sellExcess[s] +
    sum{c in Crops} (SellingPrice[c] * sell[c, s] -
                     PurchasePrice[c] * buy[c, s])) -
    sum{c in Crops} PlantingCost[c] * area[c];
```

Listing 2.3: Stochastic version of the farmer's problem formulated in SAMPL (continued)

```
###  CONSTRAINTS  ###

subject to totalArea:
    sum {c in Crops} area[c] <= TotalArea;

subject to requirement{c in Crops, s in Scenarios}:
    Yield[c, s] * area[c] - sell[c, s] + buy[c, s]
        >= MinRequirement[c];

subject to quota{s in Scenarios}:
    sell['beets', s] <= BeetsQuota;

subject to sellBeets{s in Scenarios}:
    sell['beets', s] + sellExcess[s]
        <= Yield['beets', s] * area['beets'];
```

A corresponding algebraic formulation of this two-stage SP problem in the deterministic equivalent form is given below.

Given the sets

C	a set of crops (element 3 denotes sugar beets),
S	a set of scenarios,

and the parameters

a	total area (acre),
b_{cs}	$c \in C, s \in S$: yield of crop c under scenario s (T / acre),
d_c	$c \in C$: planting cost of crop c (\$ / acre)
e_c	$c \in C$: selling price of crop c (\$ / T)
f	selling price of sugar beets produced above quota (\$ / T)
g_c	$c \in C$: purchase price of crop c (\$ / T)
r_c	$c \in C$: minimum requirement of crop c (T)
q	quota for sugar beets (T),
p_s	$s \in S$: probability of scenario s,

define the decision variables

$x_c \geq 0$	$c \in C$: acres of land devoted to crop c,
$w_{cs} \geq 0$	$c \in C, s \in S$: tons of crop c sold (at favourable price) under scenario s,

$z_s \geq 0$ $\quad s \in S$: tons of sugar beets sold at the lower price under scenario s,

$y_{cs} \geq 0$ $\quad c \in C, s \in S$: tons of crop c purchased under scenario s,

$$\begin{aligned} \text{maximize} \quad & \sum_{s \in S} p_s \left(\sum_{c \in C} (e_c w_{cs} - g_c y_{cs}) + f z_s \right) - \sum_{c \in C} d_c x_c \\ \text{subject to} \quad & \sum_{c \in C} x_c \leq a, \\ & b_{cs} x_c - w_{cs} + y_{cs} \geq r_c, \quad c \in C, s \in S, \\ & w_{3s} \leq q, \quad s \in S, \\ & w_{3s} + z_s \leq b_{3s} x_3, \quad s \in S. \end{aligned} \tag{2.2}$$

In the proposed extensions we mostly deal with the syntax of constraint declarations, therefore let us consider it in more detail. A simplified syntax of the AMPL/SAMPL constraint declaration is as follows:

[`subject to`*] name [indexing] [*: *constraint-expr]* ;

where the optional *indexing* expression defines a single- or multidimensional set over which the constraint is indexed.

The *constraint-expr* construct takes one of the following forms:

expr `=` *expr*

expr `<=` *expr*

expr `>=` *expr*

const-expr `<=` *expr* `<=` *const-expr*

const-expr `>=` *expr* `>=` *const-expr*

The following conventions are used in the syntax definition above and later in this chapter. Syntactic categories are printed in an *italic font*, while literal text such as a keyword is printed in a `monospaced font`. Constructs enclosed in slanted brackets *[ ]* are optional. The *expr* construct denotes an arithmetic expression while *const-expr* denotes a constant expression, the one that may not contain decision variables.

2.2 Design and implementation of the SAMPL translator

In this section we describe a new version of the SAMPL translator. A previous version (OptiRisk Systems, 2009) was implemented in a way similar to SML with pre- and post-processing passes that used the standard AMPL translator. The latter approach had certain limitations, in particular due to the fact that the AMPL translator was designed as an end-user tool and therefore its extensibility is limited. The new translator is designed with extensibility in mind.

The architecture of the SAMPL translator is shown in Figure 2.1. The translator consists of the following modules:

- lexical analyser that converts input into a sequence of tokens such as keywords or strings,
- parser that takes a sequence of tokens as an input, recognizes various grammar constructs and reports syntax errors,
- semantic analyser that checks for semantic errors and constructs abstract syntax trees (AST),
- AST module that provides classes to represent the SAMPL AST,
- bytecode emitter that converts AST into the Java bytecode,
- interpreter that passes ASTs corresponding to the top-level SAMPL declarations and statements to the bytecode emitter and then forwards the generated bytecode to the Java Virtual Machine (JVM) for execution,
- runtime library that provides support code required at runtime such as implementations of various AMPL functions,
- driver program connecting other modules together and providing a command-line interface to the translator,
- error handler that receives error messages from other modules, formats them and presents to the user.

The SAMPL translator is written in Java which makes it portable to a wide range of platforms including GNU/Linux, Mac and Windows. Also the translator produces Java bytecode which enables use of just-in-time compilation techniques

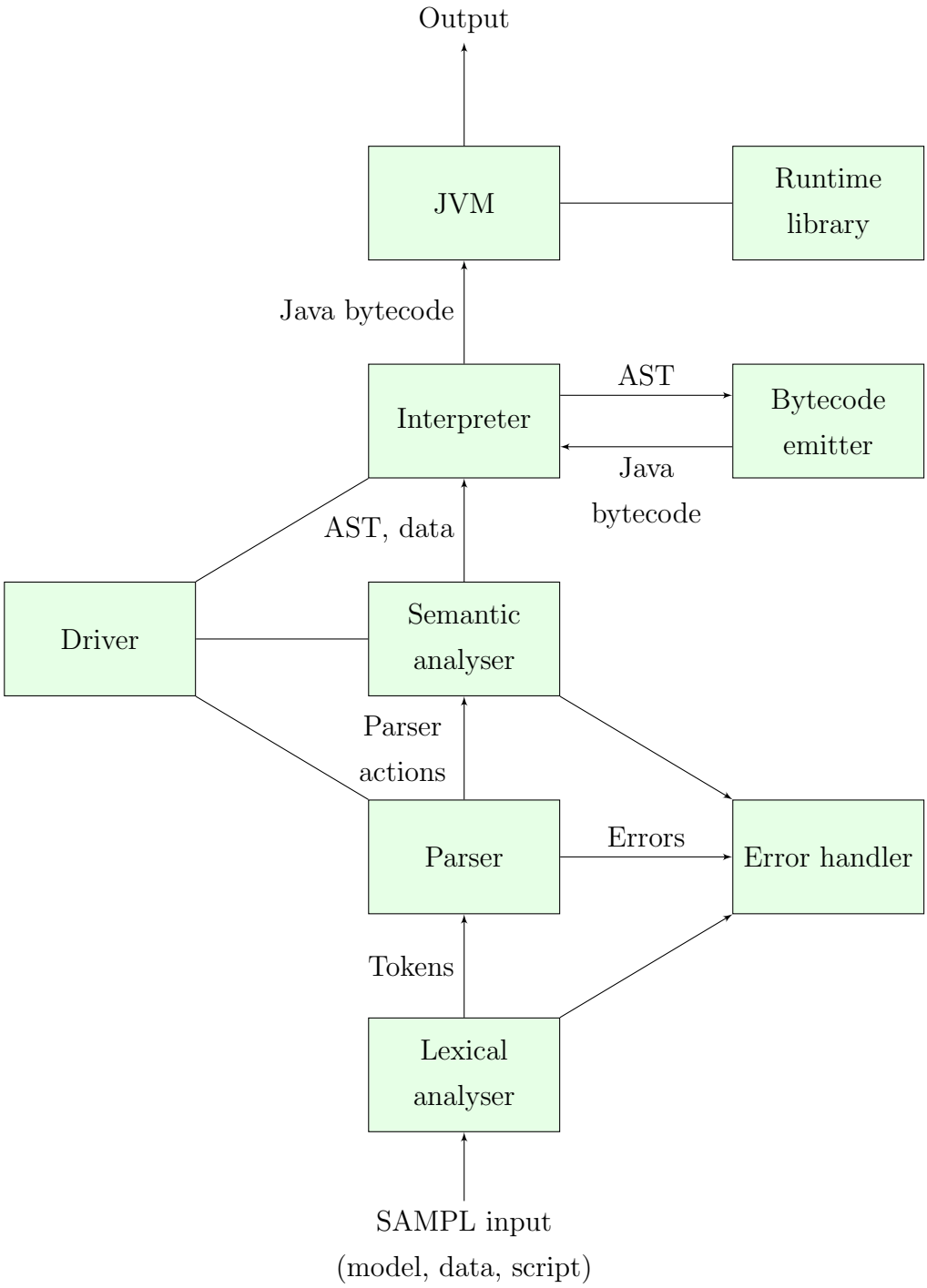


Figure 2.1: Architecture of the SAMPL translator

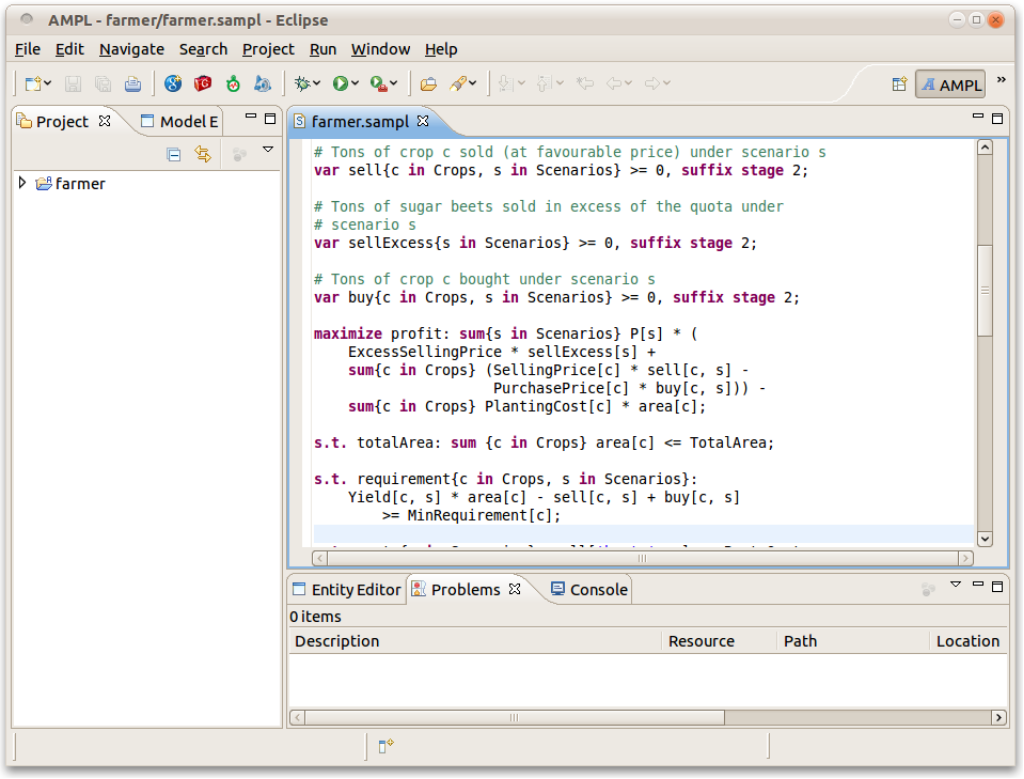


Figure 2.2: The Eclipse IDE showing an SAMPL file opened in the editor

available in modern Java implementations (Suganuma et al., 2000) for improving runtime performance.

Modular architecture of the SAMPL translator allows to reuse its components in other applications. In particular, the parser has been reused to implement syntax highlighting for the AMPL/SAMPL plug-in in Eclipse, an open-source integrated development environment (IDE) supporting a large number of programming languages (Des Rivières and Wiegand, 2004). A screenshot of the Eclipse IDE showing an opened SAMPL file with syntax highlighting is given in Figure 2.2. It is also possible to embed the complete SAMPL translator available as a software library in a client application.

During the development of the extensions our design goals have been to make proposed new language constructs

- close to established mathematical notation where one exists,
- consistent with other language features,
- compatible with existing models.

2.3 Chance constraints

Probabilistic or chance constraints are constraints that must hold with a given probability level. Chance-constrained programming has been extensively studied by Charnes and Cooper (1959), Prékopa (2003) and others. It has a wide range of applications from agricultural problems (Van de Panne and Popp, 1963) to portfolio selection (Agnew et al., 1969).

A chance constraint can be formulated as follows:

$$P\{A^i(\omega)x \geq \mathbf{h}^i(\omega)\} \geq \alpha^i,$$

where $0 < \alpha^i < 1$ and $i = 1, 2, \ldots, I$ is an index of the constraints that must hold jointly.

In this section we propose SAMPL extensions for expressing individual chance constraints. The case of joint chance constraints is postponed as a direction for future development.

We reuse the keyword `probability` in a new context of the chance constraint definitions. This keyword is introduced in SAMPL (OptiRisk Systems, 2009) to specify the parameter that provides scenario probabilities; this is a natural extension that does not break compatibility with existing models. Under the proposed extension the constraint expression takes one of the following forms:

basic-constraint-expr

`probability { ` *scenario-index* `: ` *basic-constraint-expr* ` } >= ` *const-expr*

const-expr ` <= probability { ` *scenario-index* `: ` *basic-constraint-expr* ` }`

where the *basic-constraint-expr* construct is one of the following:

expr ` = ` *expr*

expr ` <= ` *expr*

expr ` >= ` *expr*

const-expr ` <= ` *expr* ` <= ` *const-expr*

const-expr ` >= ` *expr* ` >= ` *const-expr*

and *scenario-index* is

dummy-member `in` *scenarioset-name*

Having the additional *basic-constraint-expr* construct that represents the original *constraint-expr* ensures that expressions containing `probability` cannot be nested. The *scenario-index* expression consists of a scenario set name and a dummy index whose scope covers *basic-constraint-expr*.

Consider the following chance-constrained problem:

Given the sets

F	a set of factories,
P	a set of products,
D	a set of dealers,
S	a set of scenarios,

and the parameters

T	number of time periods,
q_{ji}	$j \in P, i \in F$: cost of production of a unit of product j at factory i,
c_{ik}	$i \in F, k \in D$: cost of transportation of one unit of product from factory i to dealer k,
a_{ji}	$j \in P, i \in F$: production capacity of product j at factory i,
h_{ji}	$j \in P, i \in F$: cost of holding one unit of product j at factory i,
l_{ji}	$j \in P, i \in F$: initial inventory of product j at factory i,
n_{ji}	$j \in P, i \in F$: storage capacity of product j at factory i,
r_{jkt}	$j \in P, k \in D, t = 1, \ldots, T$: minimum acceptable probability that the demand for product j is satisfied at dealer k and time period t,
d_{jkts}	$j \in P, k \in D, t = 1, \ldots, T, s \in S$: demand for product j at dealer k and time period t under scenario s,
p_s	$s \in S$: probability of scenario s,

define the decision variables

$x_{jits} \geq 0$	$j \in P, i \in F, t = 1, \ldots, T, s \in S$: number of units of product j manufactured at factory i in period t under scenario s,
$y_{jits} \geq 0$	$j \in P, i \in F, t = 1, \ldots, T, s \in S$: number of units of product j stored in inventory at factory i in period t under scenario s,
$z_{jikts} \geq 0$	$j \in P, i \in F, k \in D, t = 1, \ldots, T, s \in S$: number of units of product j sent from factory i to dealer k in period t under scenario s,

$$
\begin{aligned}
\text{minimize} \quad & \sum_{s\in S} p_s \left(\sum_{j\in P}\sum_{i\in F}\sum_{t=1}^{T} q_{ji} x_{jits} + \sum_{j\in P}\sum_{i\in F}\sum_{k\in D}\sum_{t=1}^{T} c_{ik} z_{jikts} \right. \\
& \qquad \left. + \sum_{j\in P}\sum_{i\in F}\sum_{t=1}^{T} h_{ji} y_{jits} \right) \\
\text{subject to} \quad & P\left\{ s \in S : \sum_{i\in F} z_{jikts} \geq d_{jkts} \right\} \geq r_{jkt}, \\
& \qquad j \in P, k \in D, t = 1, \ldots, T, \\
& x_{ji1s} + l_{ji} = y_{ji1s} + \sum_{k\in D} z_{jik1s}, \quad j \in P, i \in F, s \in S, \\
& x_{jits} + y_{ji(t-1)s} = y_{jits} + \sum_{k\in D} z_{jikts}, \\
& \qquad j \in P, i \in F, t = 2, \ldots, T, s \in S, \\
& y_{jits} \leq n_{ji}, \quad j \in P, i \in F, t = 1, \ldots, T, s \in S, \\
& x_{jits} \leq a_{ji}, \quad j \in P, f \in F, t = 1, \ldots, T, s \in S, \\
& x_{ji1s} = x_{ji1s'}, \quad j \in P, f \in F, s \in S, s' \in S, \\
& y_{ji1s} = y_{ji1s'}, \quad j \in P, f \in F, s \in S, s' \in S, \\
& z_{jik1s} = z_{jik1s'}, \quad j \in P, f \in F, k \in D, s \in S, s' \in S.
\end{aligned}
\tag{2.3}
$$

Problem (2.3) is based on a two-stage SP formulation of a production planning model from Valente et al. (2009). The objective is defined as the expected cost and the shortage penalty is replaced by a chance constraint limiting the probability of not satisfying the demand. The SAMPL formulation of problem (2.3) is given in Listings 2.4 - 2.6.

The last three sets of constraints in (2.3) represent nonanticipativity restrictions. Note that these constraints are not used in the SAMPL formulation because nonanticipativity is implied by the structure of the scenario tree and partitioning of variables into stages.

Consider the chance constraint from problem (2.3):

$$
P\left\{ s \in S : \sum_{i\in F} z_{jikts} \geq d_{jkts} \right\} \geq r_{jkt}, \quad j \in P, k \in D, t = 1, \ldots, T.
$$

Listing 2.4: Production model with chance constraints

```
###  SETS  ###
set Prod; # products
set Fact; # factories
set Deal; # dealers

###  SCENARIO INFORMATION  ###
param S;                      # number of scenarios
scenarioset Scen = 1..S;      # scenario set
tree scen_tree := twostage;   # scenario tree
probability P{Scen} = 1 / S; # P[s] is the probability of
                              # scenario s
###  PARAMETERS  ###

param T; # number of production periods

# prod_cost[p, f] is the cost of production of a unit
# of product p at factory f
param prod_cost{Prod, Fact} >= 0;

# send_cost[f, d] is the cost of transportation of one
# unit of product from factory f to dealer d
param send_cost{Fact, Deal} >= 0;

# prod_cap[p, f] is the production capacity of product
# p at factory f
param prod_cap{Prod, Fact} >= 0;

# inv_cost[p, f] is the cost of holding one unit of
# product p at factory f
param inv_cost{Prod, Fact} >= 0;

# init_inv[p, f] is the initial inventory of product p
# at factory f
param init_inv{Prod, Fact} >= 0;

# inv_cap[p, f] is the storage capacity of product p
# at factory f
param inv_cap{Prod, Fact} >= 0;

# reliability[p, d, t] is the minimum acceptable probability that
# the demand for product p is satisfied at dealer d and time
# period t
param reliability{Prod, Deal, 1..T};
```

Listing 2.5: Production model with chance constraints (continued)

```
# random demand
random param demand{Prod, Deal, 1..T, Scen};

###  VARIABLES  ###

# make[p, f, t, s] is the number of units of product p
# manufactured at factory f in period t under scenario s
var make{Prod, Fact, t in 1..T, Scen} >= 0,
    suffix stage if t = 1 then 1 else 2;

# hold[p, f, t, s] is the number of units of product p
# stored in inventory at factory f in period t under
# scenario s
var hold{Prod, Fact, t in 1..T, Scen} >= 0,
    suffix stage if t = 1 then 1 else 2;

# send[p, f, d, t, s] is the number of units of product
# p sent from factory f to dealer d in period t under
# scenario s
var send{Prod, Fact, Deal, t in 1..T, Scen} >= 0,
    suffix stage if t = 1 then 1 else 2;

###  OBJECTIVE ###

# expectation of total cost which is the sum of
# production, transportation and inventory costs
minimize cost: sum{s in Scen} P[s] *
    (sum{p in Prod, f in Fact, t in 1..T}
        prod_cost[p, f] * make[p, f, t, s] +
    sum{p in Prod, f in Fact, d in Deal, t in 1..T}
        send_cost[f, d] * send[p, f, d, t, s] +
    sum{p in Prod, f in Fact, t in 1..T}
        inv_cost[p, f] * hold[p, f, t, s]);

###  CONSTRAINTS ###

# definition of the constraint satisfy_demand as a
# chance constraint
subject to satisfy_demand{p in Prod, d in Deal, t in 1..T}:
  probability{s in Scen:
    sum{f in Fact} send[p, f, d, t, s] >= demand[p, d, t, s]}
      >= reliability[p, d, t];
```

Listing 2.6: Production model with chance constraints (continued)

```
subject to inv_balance_init{p in Prod, f in Fact, s in Scen}:
  make[p, f, 1, s] + init_inv[p, f] =
    hold[p, f, 1, s] + sum{d in Deal} send[p, f, d, 1, s];

subject to inv_balance
  {p in Prod, f in Fact, t in 2..T, s in Scen}:
    make[p, f, t, s] + hold[p, f, t - 1, s] =
      hold[p, f, t, s] + sum{d in Deal} send[p, f, d, t, s];

subject to inv_capacity
  {p in Prod, f in Fact, t in 1..T, s in Scen}:
    hold[p, f, t, s] <= inv_cap[p, f];

subject to prod_capacity
  {p in Prod, f in Fact, t in 1..T, s in Scen}:
    make[p, f, t, s] <= prod_cap[p, f];
```

The formulation in SAMPL using the proposed extension is

```
subject to satisfy_demand{p in Prod, d in Deal, t in 1..T}:
  probability{s in Scen:
    sum{f in Fact} send[p, f, d, t, s] >= demand[p, d, t, s]}
      >= reliability[p, d, t];
```

One can see that SAMPL formulation of a chance constraint is nothing more than a transcription of the algebraic one that takes into account the conventions of the AMPL language. For consistency with the rest of SAMPL, the scenario set is specified explicitly. So the proposed syntax allows expressing chance constraints in a natural way following the design goals stated in Section 2.2.

We have also considered alternative representations of chance constraints in the SAMPL modelling language. The most notable alternative is probably the one suggested in SAMPL/SPInE manual (OptiRisk Systems, 2009). In this representation the above example is formulated as follows:

```
subject to satisfy_demand
  {p in Prod, d in Deal, t in 1..T, s in Scen}:
    sum{f in Fact} send[p, f, d, t, s] >= demand[p, d, t, s];
chance{p in Prod, d in Deal, t in 1..T, s in Scen}
  satisfy_demand[p, d, t, s] >= reliability[p, d, t];
```

This notation allows specifying only some of the constraint from a collection probabilistic. However, the same result can be achieved by other means, e.g. setting reliability to one for a combination of indices that correspond to the

constraints that should always hold.

The reasons why this representation was rejected are as follows. First, it breaks backward compatibility by introducing an additional keyword `chance`. This, however, can be overcome by making this keyword context-sensitive or replacing `chance` with `probability`. Second, it differs considerably from the algebraic formulation. Third and the most important reason is that the scope of the scenario set index should be different from the scopes of other indices, e.g. `reliability` cannot have the subscript `s`, which is not consistent with the rest of the language.

2.4 Integrated chance constraints

Integrated chance constraints (ICC) were introduced by Klein Haneveld (1986) and have found many applications in finance; for instance, see asset-liability management model of Van der Vlerk (2003). In general, integrated chance constraints can be used in cases when quantitative measure of risk is preferred to a qualitative one provided by chance constraints.

An individual ICC is defined as

$$\mathbb{E}_\omega[(A^i(\omega)x - h^i(\omega))_-] \leq \beta^i, \tag{2.4}$$

where $\beta^i \geq 0, i = 1, 2, \ldots, I$ and $(a)_- := max\{-a, 0\}$ is the negative part of $a \in \mathbb{R}$ or, equivalently,

$$\mathbb{E}_\omega[(h^i(\omega) - A^i(\omega)x)_+] \leq \beta^i, \tag{2.5}$$

where $(a)_+ := max\{a, 0\}$ is the positive part of $a \in \mathbb{R}$.

In AMPL $(a-b)_+$ can be naturally expressed as `a less b` using the operator `less` defined as

$$a \texttt{ less } b \equiv \texttt{max}\{a - b, 0\}$$

We propose extensions to represent integrated chance constraints as defined in equation (2.5) in the SAMPL modelling language. Under these extensions the constraint expression takes one of the following forms:

basic-constraint-expr

`expectation { `*scenario-index*` } ( `*expr* `less` *expr*` ) <=` *const-expr*

const-expr `>= expectation { `*scenario-index*` } ( `*expr* `less` *expr*` )`

The *basic-constraint-expr* construct represents the original *constraint-expr* which is the same as in the case of chance constraints and therefore is not repeated here. The *scenario-index* nonterminal consists of a scenario set name and a dummy index whose scope covers the expression in brackets.

We introduce the `expectation` keyword, but to preserve compatibility with existing models we allow it to be redefined as an entity name. We follow the AMPL convention that some keywords such as `product` can be redefined.

```
# expectation is redefined as a parameter
param expectation;
```

As an example consider the production planning model with integrated chance constraints. Since this model has a lot in common with its chance-constrained version introduced in the previous section, we do not repeat the complete definition of the problem but only give the algebraic and SAMPL formulation of ICC that replace the chance constraints in problem (2.3) and Listings 2.4 - 2.6 respectively.

The integrated chance constraint in the production planning model is formulated as follows:

$$E_s\left[\left(d_{jkts} - \sum_{i\in F} z_{jikts}\right)_+\right] \leq \beta_{jkt}, \quad j \in P, k \in D, t = 1, \ldots, T.$$

where β_{jkt} is the bound on the expected shortage in respect of demand for product j at dealer k and time period t. The integrated chance constraints above can be formulated in SAMPL using the proposed extension as follows:

```
subject to satisfy_demand{p in Prod, d in Deal, t in 1..T}:
  expectation{s in Scen}
    (demand[p, d, t, s] less sum{f in Fact} send[p, f, d, t, s])
      <= max_exp_shortage[p, d, t];
```

In Chapter 3 we discuss the relation between ICC, CVaR and second-order stochastic dominance (SSD). A portfolio choice model based on SSD criterion formulated as a SAMPL model with large number of integrated chance constraints is presented in Section 3.6. We solve this model using the deterministic equivalent approach and a cutting-plane method.

2.5 Robust optimisation

Robust optimization allows suboptimal solutions of the problems with nominal data to ensure that the solution remains feasible and close to optimal when the

data change. This modelling methodology originates from the work of Soyster (1973) and Falk (1976). It has a wide range of applications in various domains such as engineering (Ben-Tal and Nemirovski, 2002), finance (Costa and Paiva, 2002; El Ghaoui et al., 2003) and supply chain management (Bertsimas and Thiele, 2006).

Consider a linear programming problem

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & A\boldsymbol{x} \leq \boldsymbol{b}, \\ & \boldsymbol{x} \in \mathbb{R}^n_+, \end{aligned}$$

where $\boldsymbol{x}$ is a vector of decision variables, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$ are fixed vectors and A is an $m \times n$ matrix, with some coefficients a_{ij} being random.

We use the model of uncertainty described in Ben-Tal and Nemirovski (2000) where each random element of matrix A is modelled as a symmetric bounded random variable taking values from the range $[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}]$.

In order to be able to represent this kind of random parameters in SAMPL we introduce a new form of an attribute that can only be used in the declarations of random parameters. A simplified syntax of a parameter declaration with a proposed new attribute is given below.

parameter-decl:

`random param` *name [indexing] [attribute-list]* ;

attribute-list:

attribute-list attribute

attribute

attribute:

`dist` *name* (*expr-list*)

The `dist` attribute specifies the probability distribution of a random variable represented by a parameter. The `dist` keyword is followed by the name of a distribution and the list of expressions in parentheses represents its arguments. Currently we only support one type of distribution named `symmetric` which denotes some unspecified symmetric distribution and is used only for the purpose of representing the model of uncertainty introduced above. The `symmetric` distribution takes two arguments representing the lower and the upper bound of the interval $[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}]$ respectively.

This extension can be used in the future to specify univariate distributions of random parameters in a way which is comparable with the convention defined in the `INDEP` section of SMPS. For example, the following SMPS input

```
INDEP          UNIFORM
    COL1       ROW8        8.0            PERIOD2      9.0
```

could be expressed in SAMPL as

```
random param p dist uniform(8, 9);
```

We consider the following established robust formulations based on alternative representation of uncertainty sets:

- Soyster (1973),
- Ben-Tal and Nemirovski (2002),
- Bertsimas and Sim (2004).

Soyster (1973) considered the case of the columns of a constraint matrix belonging to convex sets. As shown by Bertsimas and Sim (2004) under the model of uncertainty described above the robust formulation of Soyster takes the following form:

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j + \sum_{j \in J_i} \hat{a}_{ij} y_j \leq b_i, \quad i = 1, \ldots, m, \\ & -y_j \leq x_j \leq y_j, \quad j = 1, \ldots, n, \\ & \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n_+, \end{aligned}$$

where J_i is a set of column indices of random elements a_{ij} in row i.

The formulation of Soyster leads to an LP problem which is advantageous from computational perspective. However its solutions can be too conservative in a sense that the objective value may be much worse than the one of the correspondent problem with nominal data.

Ben-Tal and Nemirovski (2002) proposed another robust formulation which

under the current model of uncertainty takes the following form:

$$\begin{aligned}
&\text{maximize} \quad \boldsymbol{c}^T\boldsymbol{x} \\
&\text{subject to} \quad \sum_j a_{ij}x_j + \sum_{j\in J_i} \hat{a}_{ij}y_{ij} + \Omega_i \sqrt{\sum_{j\in J_i} \hat{a}_{ij}^2 z_{ij}^2} \leq b_i, \quad i = 1,\ldots,m, \qquad (2.6)\\
&\qquad -y_{ij} \leq x_j - z_{ij} \leq y_{ij}, \quad i = 1,\ldots,m, \quad j \in J_i,\\
&\qquad y_{ij} \geq 0, \quad i = 1,\ldots,m, \quad j \in J_i,\\
&\qquad \boldsymbol{x} \in \mathbb{R}^n_+.
\end{aligned}$$

The inequality (2.6) defines the interior of an ellipsoid. The formulation of Ben-Tal and Nemirovski is thus based on ellipsoidal uncertainty sets and leads to a second-order cone programming (SOCP) problem. The level of conservatism is controlled through the weighting parameters Ω_i. The probability of violation of constraint i is at most $e^{-\Omega_i^2/2}$.

The robust formulation of Bertsimas and Sim (2004) is as follows:

$$\begin{aligned}
&\text{maximize} \quad \boldsymbol{c}^T\boldsymbol{x} \\
&\text{subject to} \quad \sum_{j=1}^{n} a_{ij}x_j + z_i\Gamma_i + \sum_{j\in J_i} p_{ij} \leq b_i, \quad i = 1,\ldots,m,\\
&\qquad z_i + p_{ij} \geq \hat{a}_{ij}y_j, \quad i = 1,\ldots,m, \quad j \in J_i,\\
&\qquad -y_j \leq x_j \leq y_j, \quad j = 1,\ldots,n,\\
&\qquad p_{ij} \geq 0, \quad i = 1,\ldots,m, \quad j \in J_i,\\
&\qquad \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n_+,\\
&\qquad \boldsymbol{z} \in \mathbb{R}^m_+.
\end{aligned}$$

The formulation of Bertsimas and Sim results in a linear programming problem based on convex polyhedral uncertainty sets similar to the model of Soyster. However, in this model it is possible to control the level of conservatism through the parameters Γ_i. The solution is feasible if no more than Γ_i of random coefficients change. Even if this is not the case the probability that the solution will be feasible is high.

The parameters Ω_i and Γ_i can be specified with the help of suffixes in the same way as it is done to assign stages to variables in SAMPL. For this purpose we use a predefined suffix `robustness`. The value of the parameter Γ_i should be in the range $[0, k_i]$, where k_i is the number of random coefficients in the i-th constraint.

Given the LP constraint $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$ with uncertain parameters $a_{ij} \in [0.95\bar{a}_{ij}, 1.05\bar{a}_{ij}]$ we can specify its robust counterpart with the Γ_i parameter set to $n/3$ in the following way:

```
# random parameter with symmetric uncertainty interval
random param a{i in 1..m, j in 1..n}
  dist symmetric(0.95 * abar[i, j], 1.05 * abar[i, j]);

# for a fixed i one third of a[i, j] coefficients can
# be changed without making the solution infeasible
subject to c{i in 1..m} suffix robustness n / 3:
  sum{j in 1..n} a[i, j] * x[j] <= b[i];
```

In order to select a specific formulation we introduce the option `RobustForm` which takes one of the following three values:

`Soyster`, `Bertsimas_Sim` or `BenTal_Nemirovski`.

As an example consider a simple portfolio management problem from Bert-

Listing 2.7: A portfolio management problem in SAMPL with robust optimisation extension

```
param NAssets = 150; # Number of assets
param MeanRet{i in 1..NAssets} =
  1.15 + i * (0.05 / 150);
param Delta{i in 1..NAssets} =
  (0.05 / 450) * sqrt(2 * i * NAssets * (NAssets + 1));

# Random returns
random param Return{i in 1..NAssets}
  dist symmetric(MeanRet[i] - Delta[i],
                 MeanRet[i] + Delta[i]);

# Fraction of the initial wealth invested in each asset
var invest{1..NAssets} >= 0, <= 1;

var w;
maximize wealth: w;

# Robust constraint
subject to robust suffix robustness 22:
  sum{i in 1..NAssets} Return[i] * invest[i] >= w;

subject to budget: sum{i in 1..NAssets} invest[i] = 1;

option RobustForm Bertsimas_Sim;
```

simas and Sim (2004). An investor wants to construct a portfolio of assets in order to maximize the return. There are 150 assets in total and the return of i-th asset belongs to the interval $[r_i - s_i, r_i + s_i]$, where $r_i = 1.15 + i(0.05/150)$ and $s_i = (0.05/450)\sqrt{2in(n+1)}$. The SAMPL formulation of the problem is given in Listing 2.7.

Syntax Summary

The combined syntax for the extensions introduced in previous sections is given below.

constraint-decl:

[`subject to`*] name [indexing] [*`:` *constraint-expr]* `;`

constraint-expr:

basic-constraint-expr

`probability {` *scenario-index* `:` *basic-constraint-expr* `} >=` *const-expr*

const-expr `<=` `probability {` *scenario-index* `:` *basic-constraint-expr* `}`

`expectation {` *scenario-index* `} (` *expr* `less` *expr* `) <=` *const-expr*

const-expr `>=` `expectation {` *scenario-index* `} (` *expr* `less` *expr* `)`

basic-constraint-expr:

expr `=` *expr*

expr `<=` *expr*

expr `>=` *expr*

const-expr `<=` *expr* `<=` *const-expr*

const-expr `>=` *expr* `>=` *const-expr*

scenario-index:

dummy-member `in` *scenarioset-name*

parameter-decl:

`random param` *name [indexing] [attribute-list]* `;`

attribute-list:

attribute-list attribute

attribute

attribute:

`dist` *name* `(` *expr-list* `)`

2.6 Architecture of an integrated modelling and solver system for stochastic programming

To make the modelling language extensions really useful it is not enough just to define their syntax and implement parsing and semantic analysis of the new constructs in the translator. It is equally important to have corresponding solver support for the additional types of problems that can be formulated using these constructs.

One possible way of providing solver support is to translate models into deterministic equivalent form where one exists. In Section 2.5 we described the alternative robust formulations that are automatically generated during translation. In the case of a finite discrete distribution of random parameters deterministic equivalents exist for chance constraints and integrated chance constraints as well.

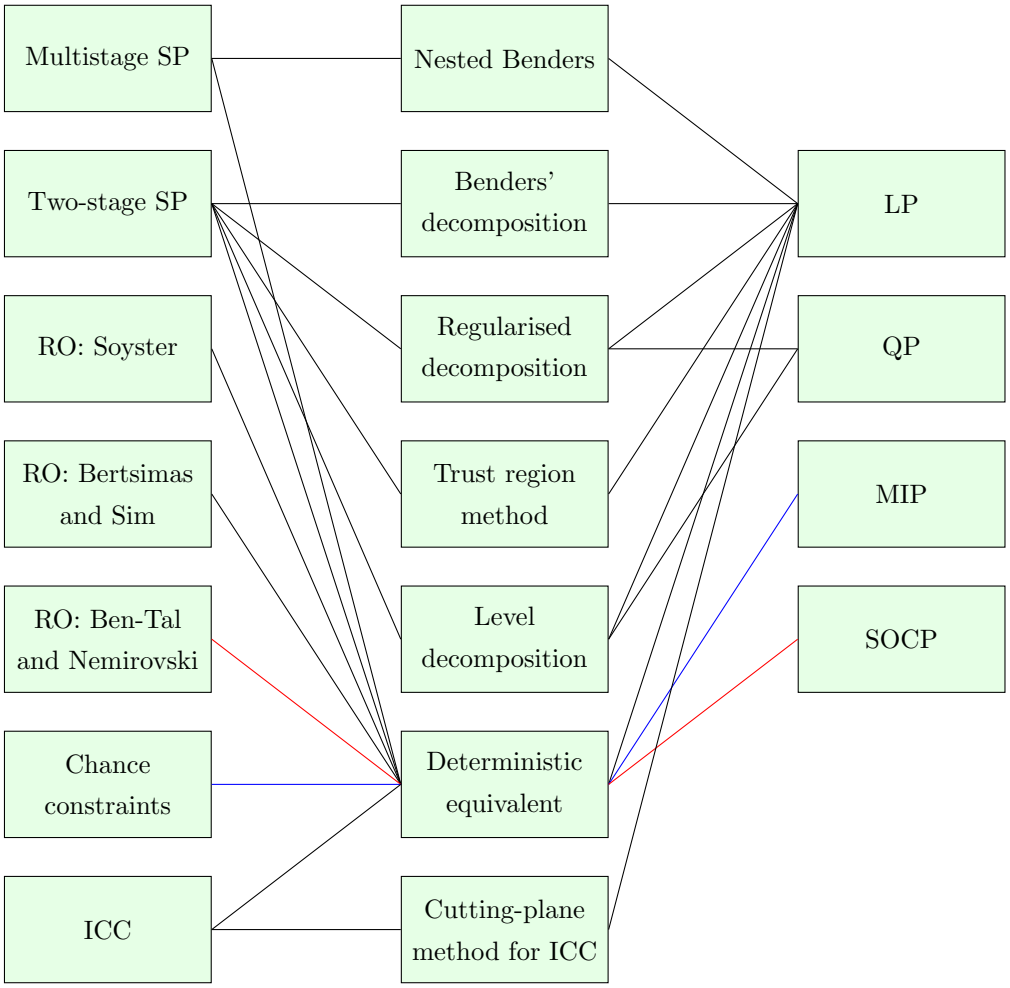
Another possibility is to use specialised algorithms designed to solve the additional types of problems that have been introduced. Having the translator capture structural information and pass it further to the solver enables use of such algorithms. To illustrate the feasibility of this approach we have implemented the cutting plane algorithm of Klein Haneveld and van der Vlerk (2006) for integrated chance constraints and applied it to a portfolio choice model formulated in SAMPL (see Section 3.6).

Solver connectivity

In our modelling and solver system we support multiple external linear programming (LP), mixed integer programming (MIP) and quadratic programming (QP) solvers that are used to optimise the DEP and subproblems in various decomposition algorithms. Each external solver implements different solution algorithms and even for the same algorithm the performance may vary greatly across solvers (see, for example, Mittelmann (1998)) due to implementation details. Therefore this ability to select a solver is important from the practical point of view.

Also it is desirable to have support for multiple deterministic solvers in order to be able to implement various classes of SP models and algorithms. For instance, the robust formulation of Ben-Tal and Nemirovski (2002) requires a SOCP solver and regularised decomposition requires LP and QP solvers. Figure 2.3 illustrates the dependencies between SP models, SP solution methods and deterministic models.

We have designed a generic solver interface which provides a uniform access



RO stands for robust optimisation

Figure 2.3: SP models and solution algorithms

to alternative solver functionality. This interface allows the problem to be incrementally constructed and manipulated in the target solver format through the following operations:

- add rows (constraints) to the problem,
- add columns (variables) to the problem,
- delete rows and columns from the problem,
- set row and column bounds,
- set linear and quadratic objective,
- change solver options,
- solve the problem,

- get problem elements (bounds, matrix coefficients, objective), primal and dual solution and basis.

We have successfully implemented the above solver interface for a number of solvers, namely, CPLEX (IBM Corp., 2009b), FortMP (Ellison et al., 2008) and Gurobi (Gurobi Optimization, 2010).

Chapter 3

Solving single-stage stochastic programming problems with risk constraints

A single-stage SP problem may be viewed as a special case of two-stage SP problem (1.2)-(1.3) where the second stage decision vector space is 0-dimensional ($n_2 = 0$). This problem comprises a first-stage decision, realisation of a random vector and evaluation of the outcomes over different scenarios. In this basic form the problem decomposes into a set of S weakly coupled deterministic problems, where S is the number of scenarios.

This type of problem becomes a more interesting object to study when combined with risk constraints. There are different models of representing risk in a stochastic programming problem such as chance constraints and integrated chance constraints, which were introduced in Chapter 2 in the context of modelling languages. Another important class of problems are those which require imposing stochastic dominance relations; this is considered later in this chapter.

Stochastic dominance is a fundamental concept in decision making under risk. Its importance has been recognized by Hadar and Russell (1969), Whitmore and Findlay (1978), Levy (1992) and many others. Of particular interest is the application of second-order stochastic dominance (SSD) relation since it captures risk-averse preferences (Fishburn, 1964) which is a common assumption about investment behaviour. This makes SSD a theoretically sound and rational choice criterion in portfolio selection models.

In Section 3.1 the notion of second-order stochastic dominance is introduced and its alternative definitions are given. Existing portfolio choice models with SSD criteria are described in Section 3.2. An enhanced version of one of these

models is presented in Section 3.3. Alternative formulations of the enhanced model in computationally tractable forms and connection between SSD constraints, integrated chance constraints and the conditional value-at-risk is discussed in Section 3.4. Solution methods are presented in Section 3.5 followed by a computational study in Section 3.6.

3.1 Second-order stochastic dominance

Let R and R' be random variables defined on the probability space $(\Omega, \mathcal{F}, P)$. By definition, R dominates R' with respect to SSD if and only if the following condition holds:

$$\mathrm{E}[U(R)] \geq \mathrm{E}[U(R')] \quad \begin{array}{l}\text{for any nondecreasing and concave} \\ \text{utility function } U.\end{array} \tag{3.1}$$

Since a concave utility function corresponds to risk-aversion, (3.1) shows that the SSD relation is consistent with preferences of a risk-averse decision maker.

There exist alternative definitions of stochastic dominance based on pointwise comparison of performance functions associated with distribution functions of random variables. In particular, Fishburn and Vickson (1978) proved that (3.1) is equivalent to the following:

$$F_R^{(2)}(t) \leq F_{R'}^{(2)}(t) \text{ for all } t \in \mathbb{R}, \tag{3.2}$$

where the performance function $F_R^{(2)}(t) = \int_{-\infty}^{t} F_R(u)du$ represents the area under the graph of the cumulative distribution function $F_R(t) = P(R \leq t)$ of a real-valued random variable R. The performance function can be expressed as the expected shortfall (see, for example, Ogryczak and Ruszczyński, 1999):

$$F_R^{(2)}(t) = \mathrm{E}[(t - R)_+] \tag{3.3}$$

Ogryczak and Ruszczyński (2002) showed equivalence between (3.2) and the following condition:

$$\mathrm{Tail}_\alpha(R) \geq \mathrm{Tail}_\alpha(R') \text{ for all } 0 < \alpha \leq 1, \tag{3.4}$$

where $\mathrm{Tail}_\alpha(R)$ denotes the unconditional expectation of the smallest $\alpha \cdot 100\%$ of the outcomes of R.

Figure 3.1 illustrates the concept of SSD. The left diagram shows the cumulative distribution functions of two random variables R and R' and the right

diagram shows the correspondent performance functions. The random variables have distributions of similar shaping; they have the same expected value but R has much lower variance. It is clear from the right diagram that R dominates R' with respect to SSD because the graph of the performance function $F_R^{(2)}$ is uniformly below or coincides with the graph of $F_{R'}^{(2)}$.

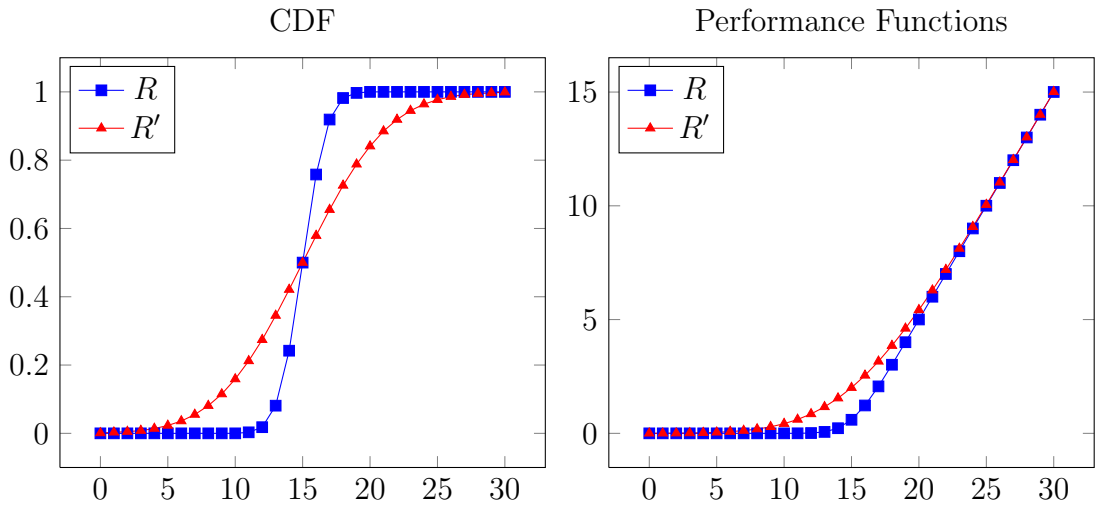


Figure 3.1: Illustration of second-order stochastic dominance

The notation $R \succeq_{SSD} R'$ is used to denote that R dominates R' with respect to SSD criteria. The corresponding strict relation is defined as follows:

$$R \succ_{SSD} R' \Leftrightarrow R \succeq_{SSD} R' \text{ and } R' \not\succeq_{SSD} R. \tag{3.5}$$

3.2 Portfolio selection models with SSD constraints

Consider the following portfolio problem. There are n assets and at the beginning of a time period an investor has to decide what proportion x_i of the initial wealth to invest in asset i. So a portfolio is represented by a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in X \subset \mathbb{R}^n$, where X is a bounded convex polytope representing the set of feasible portfolios; in particular it is defined as

$$X = \{\boldsymbol{x} \in \mathbb{R}^n_+ : \sum_{i=1}^{n} x_i = 1\},$$

if short positions are not allowed and there are no other modelling restrictions.

Let $\boldsymbol{R}$ denote the n-dimensional random vector of asset returns at the end of the time period. Then the real-valued random variable $R_{\boldsymbol{x}} = \boldsymbol{R}^T\boldsymbol{x}$ is the random return of portfolio $\boldsymbol{x}$.

Several portfolio models based on the concept of second-order stochastic dominance have been proposed in the literature. Those considered below assume existence of a reference random return $\widehat{R}$ for which the distribution is known. These are the models of Dentcheva and Ruszczyński (2003, 2006) and Roman et al. (2006). The reference return can be the return of a stock market index such as NASDAQ Composite, FTSE-100 or Hang Seng Index.

Portfolio $\boldsymbol{x} \in X$ is said to be SSD-efficient if there is no other portfolio $\boldsymbol{y} \in X$ such that $R_{\boldsymbol{y}} \succ_{SSD} R_{\boldsymbol{x}}$.

Dentcheva and Ruszczyński (2006) proposed the following model with an SSD constraint:

$$\begin{array}{ll} \text{maximize} & f(x) \\ \text{subject to} & x \in X, \\ & R_{\boldsymbol{x}} \succeq_{SSD} \widehat{R}, \end{array} \tag{3.6}$$

where f is a concave continuous function. They considered a special case of $f(x) = \mathrm{E}[R_{\boldsymbol{x}}]$ and described a solution method based on the regularized decomposition by Ruszczyński (1986) applied to the dual representation of the problem. With this method they were able to solve relatively large test problems consisting of 719 assets with 616 realisations of their joint return rates in a reasonable time.

Dentcheva and Ruszczyński also showed that if the reference portfolio return $\widehat{R}$ has a finite discrete distribution with realisations $\widehat{r}^{(1)}, \widehat{r}^{(2)}, \ldots, \widehat{r}^{(S)}$ then the second-order stochastic dominance constraint can be formulated as a finite set of integrated chance constraints (Klein Haneveld, 1986) based on the equation (3.3):

$$\mathrm{E}[(\widehat{r}^{(i)} - R_{\boldsymbol{x}})_+] \leq \mathrm{E}[(\widehat{r}^{(i)} - \widehat{R})_+], \quad i = 1, 2, \ldots, S. \tag{3.7}$$

This is an important formulation because in general representing second-order stochastic dominance relation requires continuum of inequalities as in (3.4) making SSD constrained optimisation difficult to apply in practice.

In the model (3.6) one seeks an optimal portfolio with return distribution that dominates the benchmark which is the reference return. The advantage of this model is that it only requires a benchmark portfolio, unlike traditional mean-risk optimisation (Markowitz, 1952) where one has to choose a particular risk measure and a trade-off between risk and return which can be sometimes difficult

to justify. Also the difficulty of selecting an appropriate utility function as in expected utility maximisation is avoided.

Roman, Darby-Dowman, and Mitra (2006) formulated a multiobjective linear programming model, the Pareto efficient solutions of which are SSD efficient portfolios. Subsequently Fábián et al. (2009) introduced a more efficient computational model. Based on the assumption of finite discrete distributions of returns with equiprobable outcomes, they proved that SSD constraint $R_{\boldsymbol{x}} \succeq_{SSD} \widehat{R}$ is equivalent to a finite system of inequalities $\mathrm{Tail}_{\frac{i}{S}}(R_{\boldsymbol{x}}) \geq \mathrm{Tail}_{\frac{i}{S}}(\widehat{R})$, $i = 1, 2, \ldots, S$, where S is the number of outcomes (scenarios). They converted the problem into a single-objective form by using the reference point method which resulted in the following formulation with the Tail functions:

$$\begin{array}{ll} \text{maximize} & \vartheta \\ \text{subject to} & \vartheta \in \mathbb{R}, \boldsymbol{x} \in X, \\ & \mathrm{Tail}_{\frac{i}{S}}(R_{\boldsymbol{x}}) \geq \mathrm{Tail}_{\frac{i}{S}}(\widehat{R}) + \vartheta, \quad i = 1, 2, \ldots, S. \end{array} \tag{3.8}$$

In the latter model one seeks a portfolio with a distribution which dominates the reference one or comes close to it uniformly. Uniformity here means that the smallest tail difference ϑ is maximized.

This model has the same advantages as (3.6) and in addition it provides an SSD-efficient portfolio. Roman et al. report favourable results with the model using the data from the Hang Seng Index.

3.3 An enhanced model

In Fábián, Mitra, Roman, and Zverovich (2010) we proposed an enhanced version of the model (3.8) which is expressed in the following SSD constrained form:

$$\begin{array}{ll} \text{maximize} & \vartheta \\ \text{subject to} & \vartheta \in \mathbb{R}, \boldsymbol{x} \in X, \\ & R_{\boldsymbol{x}} \succeq_{SSD} \widehat{R} + \vartheta. \end{array} \tag{3.9}$$

In this model we compute a portfolio that dominates a sum of the reference return and a riskless return ϑ. Depending on the reference distribution three outcomes of optimisation are possible:

1. If there exist portfolios that dominate the reference plus some value, the model returns one of such portfolios with maximum surplus ϑ.

2. If the reference distribution is efficient and attainable, meaning that there is a feasible portfolio with this return distribution, then the model returns such a portfolio.

3. If the reference distribution is unattainable then the model provides a portfolio that dominates the reference minus some value.

3.4 Formulation of a computational model

The number of constraints and variables in the original formulation of Roman et al. (2006) is of the order S^2, where S is the number of scenarios. Hence it is inefficient as a computational model.

Alternative formulations of the enhanced model are presented in this section. The main focus is on the representations that make it possible to apply efficient solution methods. The dominance relation is expressed using tail functions according to (3.4) in the first formulation and using integrated chance constraints in the second one.

Formulation using tails

Assume that the joint distribution of the random vector of asset returns $\boldsymbol{R}$ and the reference random return $\widehat{R}$ is a finite discrete distribution with equiprobable outcomes. Let S denote the number of outcomes, $\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}, \ldots, \boldsymbol{r}^{(S)}$ - the realisations of $\boldsymbol{R}$ and $\widehat{r}^{(1)}, \widehat{r}^{(2)}, \ldots, \widehat{r}^{(S)}$ - the realisations of $\widehat{R}$.

Taking into consideration the identity $\mathrm{Tail}_{\frac{i}{S}}(\widehat{R} + \vartheta) = \mathrm{Tail}_{\frac{i}{S}}(\widehat{R}) + \frac{i}{S}\vartheta,\ i = 1, 2, \ldots, S$, the model (3.9) is reformulated as

$$\begin{array}{ll} \text{maximize} & \vartheta \\ \text{subject to} & \vartheta \in \mathbb{R}, \boldsymbol{x} \in X, \\ & \mathrm{Tail}_{\frac{i}{S}}(R_{\boldsymbol{x}}) \geq \mathrm{Tail}_{\frac{i}{S}}(\widehat{R}) + \frac{i}{S}\vartheta, \quad i = 1, 2, \ldots, S. \end{array} \tag{3.10}$$

This model is referred to as a scaled model because it differs from (3.8) due to the second term $\frac{i}{S}\vartheta$ which we call a scaled tail.

There is a relation between the Tail function and the Conditional Value-at-Risk (Rockafellar and Uryasev, 2000). If we consider a loss distribution then Conditional Value-at-Risk (CVaR) at α level is the conditional expectation of the largest $\alpha \cdot 100\%$ of the outcomes. Then, taking into account that R represents

a random return and therefore $-R$ represents a loss, we obtain the following relation:

$$\mathrm{CVaR}_\alpha(R) = -\frac{1}{\alpha}\mathrm{Tail}_\alpha(R), \quad 0 < \alpha \leq 1. \tag{3.11}$$

As shown by Roman et al. $\mathrm{Tail}_{\frac{i}{S}}(R_{\boldsymbol{x}})$ is the optimal value of the following optimisation problem:

$$\begin{aligned} \text{maximize} \quad & \frac{i}{S}t_i - \frac{1}{S}\sum_{j=1}^{S}[t_i - \boldsymbol{r}^{(j)T}\boldsymbol{x}]_+ \\ \text{subject to} \quad & t_i \in \mathbb{R}. \end{aligned} \tag{3.12}$$

This follows from the Conditional Value-at-Risk optimisation formula of Rockafellar and Uryasev (2000, 2002) and relation (3.11). The problem 3.12 can be reformulated as a linear programming problem by introducing additional variables d_{ij} for representing $[t_i - \boldsymbol{r}^{(j)T}\boldsymbol{x}]_+$:

$$\begin{aligned} \text{maximize} \quad & \frac{i}{S}t_i - \frac{1}{S}\sum_{j=1}^{S} d_{ij} \\ \text{subject to} \quad & d_{ij} \geq t_i - \boldsymbol{r}^{(j)T}\boldsymbol{x}, \quad j = 1, 2, \ldots, S, \\ & d_{ij} \in \mathbb{R}_+, \quad j = 1, 2, \ldots, S, \\ & t_i \in \mathbb{R}. \end{aligned} \tag{3.13}$$

Künzi-Bay and Mayer (2006) reformulated the CVaR optimisation problem of Rockafellar and Uryasev as a two-stage stochastic programming problem with recourse. Based on this formulation they proposed a cutting-plane algorithm for CVaR optimisation that is a specialisation of the L-shaped method and is similar to the the cutting-plane method of Klein Haneveld and van der Vlerk (2006) for integrated chance constraints. Fábián, Mitra, and Roman (2009) adapted this approach to obtain the cutting-plane representation of the Tail function:

$$\begin{aligned} \mathrm{Tail}_{\frac{i}{S}}(R_{\boldsymbol{x}}) = \quad & \min \frac{1}{S}\sum_{j\in J_i} \boldsymbol{r}^{(j)T}\boldsymbol{x} \\ & \text{such that } J_i \subset \{1, 2, \ldots, S\}, \quad |J_i| = i. \end{aligned} \tag{3.14}$$

Using (3.14) the following cutting-plane representation of the problem (3.10) can be obtained:

$$\begin{aligned} \text{maximize} \quad & \vartheta \\ \text{subject to} \quad & \vartheta \in \mathbb{R}, \boldsymbol{x} \in X, \\ & \frac{1}{S}\sum_{j\in J_i} \boldsymbol{r}^{(j)T}\boldsymbol{x} \geq \widehat{\tau}_i + \frac{i}{S}\vartheta, \quad \text{for each } J_i \subset \{1, 2, \ldots, S\}, \\ & \qquad\qquad |J_i| = i, \ i = 1, 2, \ldots, S, \end{aligned} \tag{3.15}$$

where $\widehat{\tau}_i = \mathrm{Tail}_{\frac{i}{S}}(\widehat{R})$.

Formulation using integrated chance constraints

Using the representation (3.7) the second-order stochastic dominance constraint in the model (3.9) can be expressed as the following set of integrated chance constraints (ICC):

$$\sum_{j=1}^{S} p_j[\widehat{r}^{(i)} + \vartheta - \boldsymbol{r}^{(j)T}\boldsymbol{x}]_+ \leq \sum_{j=1}^{S} p_j[\widehat{r}^{(i)} - \widehat{r}^{(j)}]_+, \quad i = 1, 2, \ldots, S. \tag{3.16}$$

For the case of a discrete finite distribution Klein Haneveld and van der Vlerk (2006) developed a method for solving problems with integrated chance constraints. This method is based on a cutting-plane representation of ICC. The authors reported computational experiments demonstrating computational effectiveness of their cutting-plane approach.

Rudolf and Ruszczyński (2008) proposed an extension of the cutting-plane representation of Klein Haneveld and van der Vlerk. Based on it they developed primal and dual cutting-plane solution algorithms for problems with SSD constraints and demonstrated favourable performance characteristics of the primal method.

Let $\widehat{\nu}_i = \sum_{j=1}^{S} p_j[\widehat{r}^{(i)} - \widehat{r}^{(j)}]_+$ denote the right-hand side of (3.16) which does not depend on $\boldsymbol{x}$; then the cutting-plane representation of the i-th constraint takes the form:

$$\sum_{j \in J_i} p_j(\widehat{r}^{(i)} + \vartheta - \boldsymbol{r}^{(j)T}\boldsymbol{x}) \leq \widehat{\nu}_i \quad \text{for each } J_i \subset \{1, 2, \ldots, S\}. \tag{3.17}$$

The complete model in this case can be formulated as follows:

$$\begin{aligned} &\text{maximize} && \vartheta \\ &\text{subject to} && \vartheta \in \mathbb{R}, \boldsymbol{x} \in X, \\ &&& \sum_{j \in J_i} p_j(\widehat{r}^{(i)} + \vartheta - \boldsymbol{r}^{(j)T}\boldsymbol{x}) \leq \widehat{\nu}_i \quad \text{for each } J_i \subset \{1, 2, \ldots, S\}, \\ &&& \qquad\qquad\qquad\qquad\qquad\qquad i = 1, 2, \ldots S. \end{aligned} \tag{3.18}$$

The formulation using integrated chance constraints is more general than the one using tails because the former does not rely on the assumption of equiprobable outcomes.

3.5 Solution methods

Consider the formulation (3.15) of the portfolio choice model. It can be transformed into a problem of minimising a piecewise-linear convex function by changing the scope of optimisation:

$$\begin{array}{ll} \text{minimize} & \varphi(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in X, \end{array} \tag{3.19}$$

where

$$\begin{aligned} \varphi(\boldsymbol{x}) = \quad & \max \left(-\frac{1}{i} \sum_{j \in J_i} \boldsymbol{r}^{(j)T} \boldsymbol{x} + \frac{S}{i} \widehat{\tau}_i \right), \\ & \text{such that } J_i \subset \{1, 2, \ldots, S\}, |J_i| = i, \\ & i = 1, 2, \ldots, S. \end{aligned}$$

Cutting-plane method

The cutting-plane method constructs a piecewise-linear function which is an outer approximation of $\varphi(\boldsymbol{x})$. It evaluates the value of the objective function at the current iterate starting from $\boldsymbol{x}^0$ and constructs a supporting linear function (cut) at this point. If the stopping criterion is not reached it generates the next iterate by minimizing the current approximation function which is the upper cover of the constructed cuts.

The cut $l^k(x)$ at the iteration k is constructed in the following way:
Let $\boldsymbol{x}^k \in X$ denote the solution of the approximation function at iteration k and $\boldsymbol{r}^{(j_1^k)} \leq \boldsymbol{r}^{(j_2^k)} \leq \ldots \leq \boldsymbol{r}^{(j_S^k)}$ denote the ordered realisations of $R_{\boldsymbol{x}^k}$.

Select $i^k \in \underset{1 \leq i \leq S}{\operatorname{argmax}} \left(-\frac{1}{i} \sum_{j \in J_i^k} \boldsymbol{r}^{(j)T} \boldsymbol{x}^k + \frac{S}{i} \widehat{\tau}_i \right)$.

Then $l^k(\boldsymbol{x}) = -\frac{1}{i^k} \sum_{j \in J_{i^k}^k} \boldsymbol{r}^{(j)T} \boldsymbol{x} + \frac{S}{i^k} \widehat{\tau}_{i^k}$.

Regularisation by the level method

We applied regularisation by the level method of Lemaréchal, Nemirovskii, and Nesterov (1995) to the above cutting-plane method. The regularisation methods are discussed in more detail in Chapter 4. Its main idea is that the next iterate is obtained by projecting the current iterate on the level set of the linear approximation of the objective function. This implies that at each iteration an additional

quadratic programming problem has to be solved to obtain the projection. The regularised method often results in faster convergence both in terms of time and the number of iterations.

3.6 Computational study

In this computational study the enhanced model (3.9) is compared to the original model of Roman, Darby-Dowman, and Mitra both from the modelling and computational perspectives. Also the effect of regularisation by the level method on the number of iterations required to reach the given optimality tolerance is investigated and the results of experiments with an alternative formulation using ICC are reported.

Test problems

Scenarios for the test problems are generated using geometric Brownian motion, which is a standard method in finance for modelling asset prices (Ross, 2002). Parameters for scenario generation are derived from a data set of 132 historical monthly returns of 76 stocks (all the stocks that belonged to the FTSE 100 index during the period January 1993 - December 2003).

For the reference return $\widehat{R}$, the FTSE 100 index is used. Scenarios for the FTSE 100 monthly return are generated in the same way (using geometric Brownian motion and historical returns of the index during the period from January 1993 to December 2003).

We use test problems with the number of scenarios ranging from 1000 to 30000. Every scenario consists of 77 return values: one for each of the 76 available stocks, and one for the index.

Implementation issues

Both the scaled and original methods are implemented using the AMPL modelling language (Fourer et al., 2003) and the AMPL Component Object Model (COM) Library (Sadki, 2005), integrated with a C library. The problems are solved with FortMP linear and quadratic optimiser developed at Brunel University and NAG Ltd by Ellison et al. (2008). The architecture of the system is shown in Figure 3.2.

For efficiency reasons the cut generation is implemented in the C programming language. The constructed cuts are added to the AMPL data at each iteration and the AMPL translator is controlled through the COM interface.

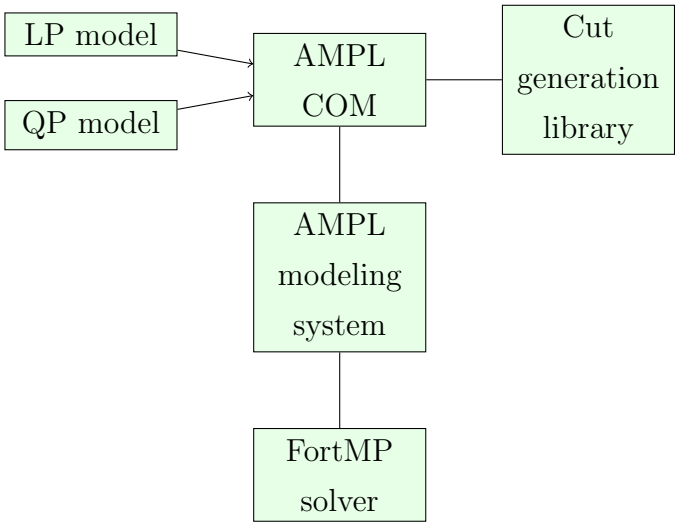


Figure 3.2: SSD solver architecture

Although the implementation of the methods is suboptimal and leaves many possibilities for speed-up, the performance of the methods is reasonably good. Even the largest problem instances with 30000 scenarios are solved within a minute on a computer with 1.73 GHz Intel Core Duo CPU and 2 GiB of RAM running Windows XP.

The methods are terminated when the absolute gap between the lower and upper bound on the objective function becomes less or equal to the value of the parameter ϵ which is set to 10^{-7} in all the experiments. The level method parameter is set to 0.5.

We have also implemented the cutting-plane method of Klein Haneveld and van der Vlerk and applied it to the representation of the enhanced model with integrated chance constraints of the form (3.16). Observing that the term $\boldsymbol{R}^T\boldsymbol{x}$ is repeated in S constraints we have introduced a second stage variable y to represent it which results in the following two-stage formulation:

$$\begin{array}{ll} \text{maximize} & \vartheta \\ \text{subject to} & \vartheta \in \mathbb{R}, \boldsymbol{x} \in X, \\ & \mathrm{E}[(\widehat{r}^{(i)} + \vartheta - y)_+] \leq \widehat{\nu}_i, \quad i = 1, 2, \ldots, S, \\ & y = \boldsymbol{R}^T\boldsymbol{x}. \end{array} \tag{3.20}$$

This allows us to reduce the number of nonzeros in the constraint matrix. Using the extensions for representing integrated chance constraints introduced in the previous chapter the problem (3.20) is expressed in SAMPL as shown in Listing 3.1.

Listing 3.1: Portfolio choice model with ICCs in SAMPL

```
# SETS
set Assets;
scenarioset Scenarios;

# PARAMETERS
probability P{Scenarios} = 1 / card{Scenarios};
random param Returns{Assets, Scenarios};
random param Reference{Scenarios}; # Reference return
param ICCRHS{Scenarios}; # Right hand side of ICC

# SCENARIO TREE
tree T := twostage;

# VARIABLES
var v;
var x{Assets} >= 0;
var y{Scenarios} suffix stage 2;

# OBJECTIVE
maximize obj: v;

# CONSTRAINTS
subject to balance:
    sum{a in Assets} x[a] = 1;
subject to link{s in Scenarios}:
    y[s] = sum{a in Assets} Returns[a, s] * x[a];
subject to icc{i in Scenarios}:
    expectation{s in Scenarios}
        (Reference[s] + v less y[s]) <= ICCRHS[i];
```

Table 3.1: Iteration counts

	Basic cutting-plane		Regularised	
Scenarios	Original	Scaled	Original	Scaled
5,000	60	74	23	39
7,000	84	79	27	45
10,000	73	97	28	45
15,000	91	74	24	39
20,000	120	97	27	45
30,000	92	97	27	48

Analysis of test results

Scale-up properties: Using a cutting-plane method allows us to solve the problems with tens of thousands of scenarios in a reasonable time (less than minute) which is not possible to achieve with the model of Roman et al. due to quadratic number of variables.

First we compare the performance of the basic cutting-plane method with its regularised counterpart on problems with increasing number of scenarios. The figures in Table 3.1 show that regularisation by the level method results in significant reduction in the number of iterations required to reach the given optimality tolerance ϵ. It can be also seen that in regularised method the number of iterations grows much slower with increase in the number of scenarios.

In the second set of experiments we compare the return distributions of the optimal portfolios obtained by solving the original model of Roman, Darby-Dowman, and Mitra and the scaled model (3.10). We observe that the main feature of the scaled model is that its return distribution is shifted to the right of the distribution obtained from the original model indicating overall higher outcomes.

The first benchmark problem contained 30000 scenarios from the historical data for the period from January 1993 to December 2003. The histograms for the return distributions are shown in Figure 3.3. Both return distributions obtained from the original and scaled model dominate the reference distribution (FTSE 100). But neither of them dominate the second one with respect to the second-order stochastic dominance.

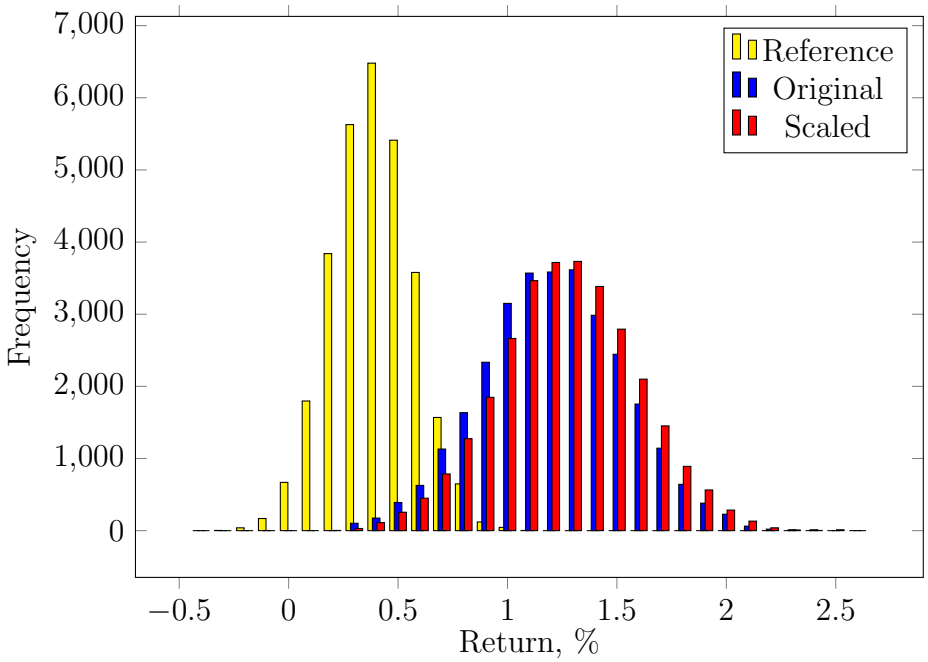


Figure 3.3: Dataset Jan 1993 - Dec 2003. Histograms for the return distributions of the optimal portfolios of SSD based models ("original" and "scaled") and for the FTSE100 Index ("reference").

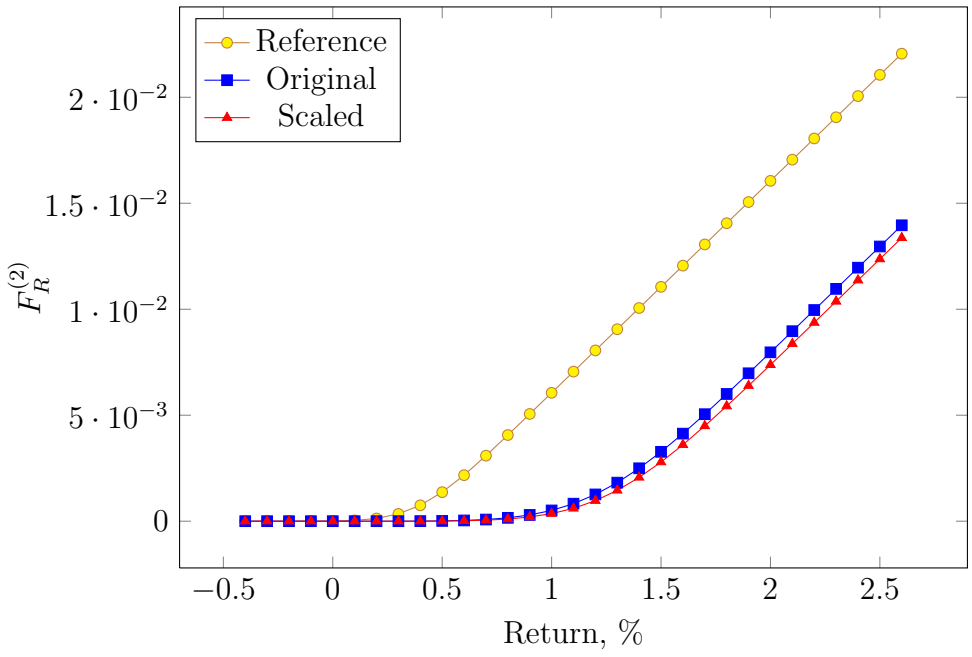


Figure 3.4: Dataset Jan 1993 - Dec 2003. Performance functions for the return distributions of the optimal portfolios of SSD based models ("original" and "scaled") and for the FTSE100 index ("reference"). Lower is better.

Table 3.2: Statistics of the return distributions

	Original	Scaled	Reference
Mean	0.0115	0.0121	0.0035
Median	0.0115	0.0121	0.0034
Std. Deviation	0.0032	0.0032	0.0018
Range	0.0215	0.0233	0.0136
Minimum	0.0023	0.0017	-0.0034
Maximum	0.0238	0.0250	0.0102

The plots of the performance functions $F_R^{(2)}$ also known as the Outcome-Risk diagrams (Ogryczak and Ruszczyński, 1999) for each return distribution are shown in Figure 3.4. The performance function of the return distribution for the scaled model is generally lower and it may seem that it dominates the one of the original model. However this is not the case and although it cannot be seen on the diagram there is a small bin belonging to the histogram for the scaled model situated at the left of the histogram for the original model in Figure 3.3. So compared to the scaled model the original one has slightly better worst case at the cost of lower overall outcomes.

Table 3.2 gives statistics for the return distributions. It shows that the distribution for the scaled model has higher mean value and approximately the same standard deviation as the one for the original model.

The tests are repeated using the problems with the following scenario sets:

1. 30000 scenarios from the historical data for the period from December 1992 to April 2000. Figure 3.5 depicts the histograms for the return distributions. The performance functions are plotted in Figure 3.6.

2. 30000 scenarios from the historical data for the period from May 2000 to September 2007. The histograms for the return distributions are shown in Figure 3.7 and the performance functions in Figure 3.8.

The results of these experiments are similar to the results of the first one. As can be seen from the diagrams the return distribution of the optimal portfolio for the scaled model was shifted to the right and the performance function is mostly lower except for the small part on the left.

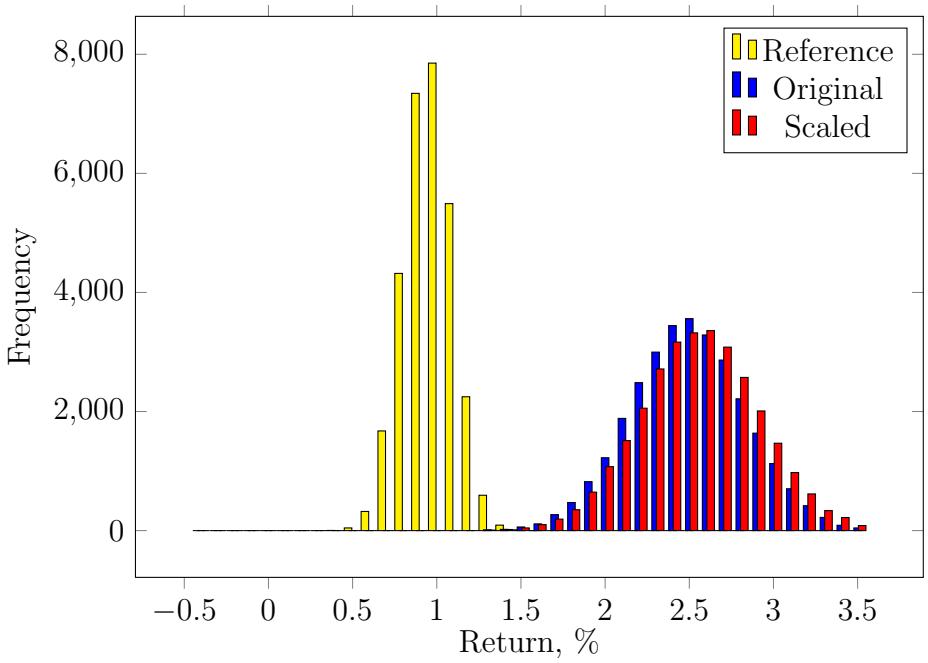


Figure 3.5: Dataset Dec 1992 - Apr 2000. Histograms for the return distributions of the optimal portfolios of "original" and "scaled" models and for the FTSE100 index ("reference").

Taking into account the observed properties of the scaled model we believe that the investor will prefer it to the original one.

Another computational study that analyses the effectiveness of SSD-based portfolio selection models is given by Roman, Mitra, and Zverovich (2011).

The test results for the cutting-plane method by Klein Haneveld and van der Vlerk on the formulation of the enhanced model using integrated chance constraints are shown in Table 3.3. The Roman numerals in the first column denote the type of the reference return distribution:

I index,

II unattainable,

III SSD efficient.

With the deterministic equivalent approach it was possible to solve only the smallest problem instance with 1000 scenarios within the time limit of 5 hours. Figure 3.9 shows the change in solution time with increase in the number of scenarios.

The results show that while the cutting-plane algorithm for the ICC formulation of the enhanced model is many times faster than solving corresponding

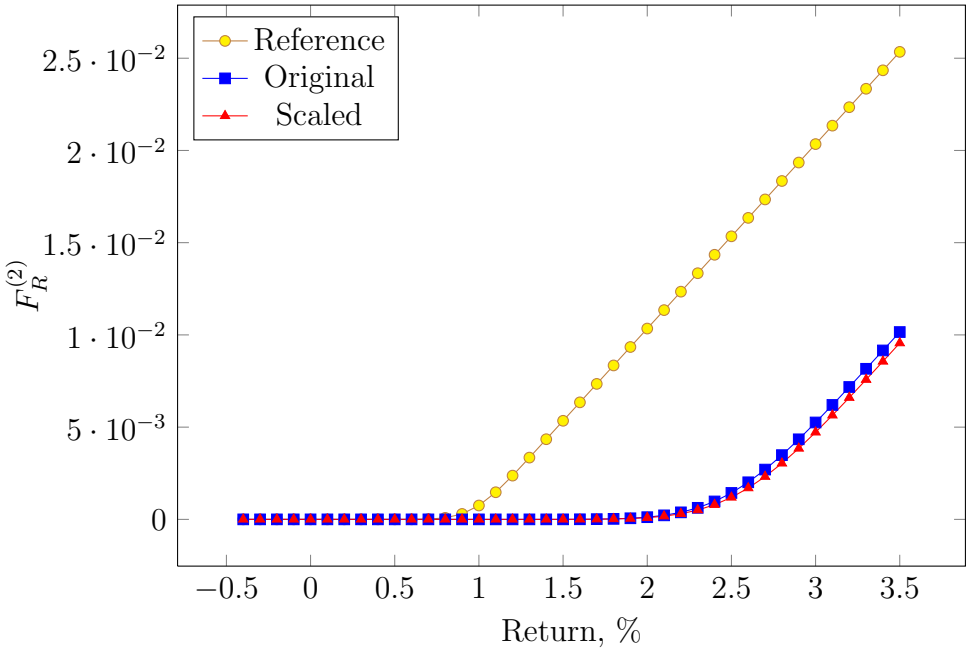


Figure 3.6: Dataset Dec 1992 - Apr 2000. Performance functions for the return distributions of the optimal portfolios of SSD based models ("original" and "scaled") and for the FTSE100 index ("reference"). Lower is better.

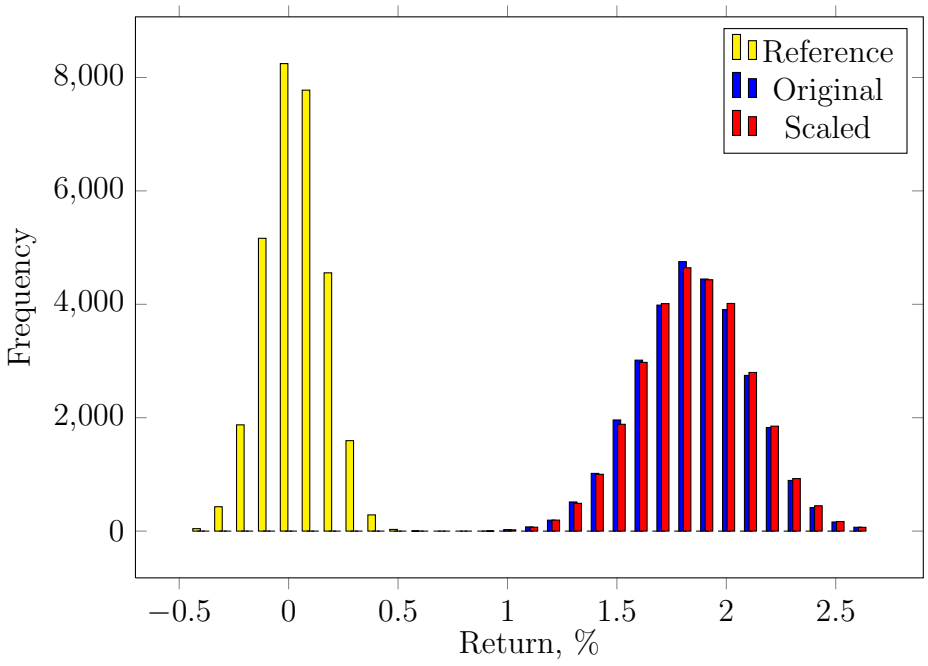


Figure 3.7: Dataset May 2000 - Sep 2007. Histograms for the return distributions of the optimal portfolios of "original" and "scaled" models and for the FTSE100 Index ("reference").

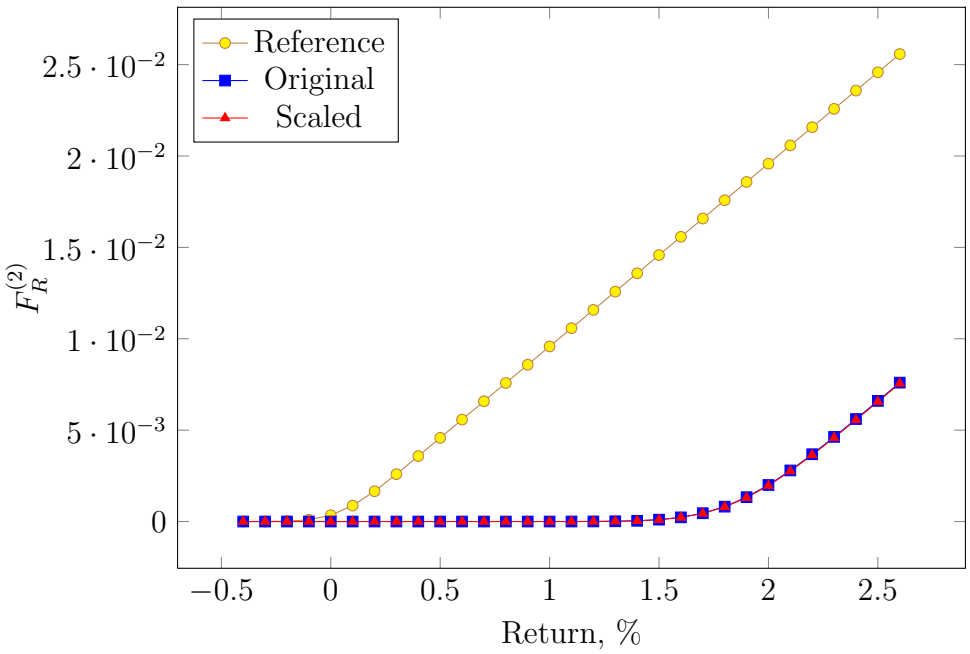


Figure 3.8: Dataset May 2000 - Sep 2007. Performance functions for the return distributions of the optimal portfolios of SSD based models ("original" and "scaled") and for the FTSE100 Index ("reference"). Lower is better.

deterministic equivalent problems it is still much less efficient then the cutting-plane method based on the formulation using the *Tail* functions. However the former is more general because it does not rely on the assumption of equiprobable outcomes.

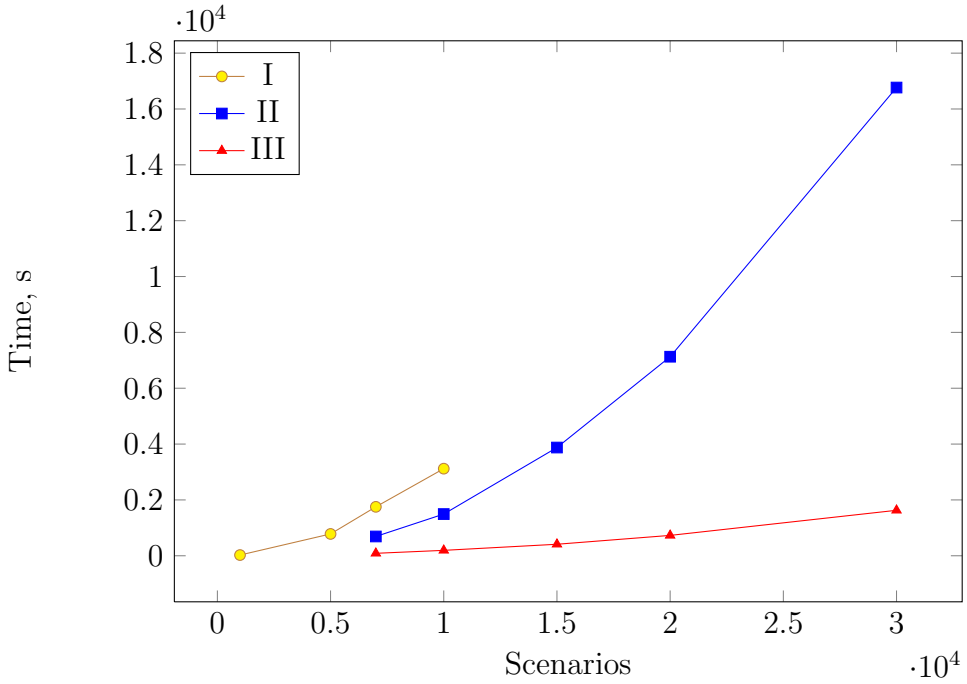


Figure 3.9: Performance of the ICC cutting-plane method

Table 3.3: Performance of the ICC cutting-plane method

	Scenarios	Iters	Cuts	Time, s	DEP Time, s
I	1,000	40	13283	25.59	11880.60
	5,000	62	95841	781.68	-
	7,000	71	127394	1752.60	-
	10,000	71	189596	3119.16	-
II	7,000	29	27596	691.23	-
	10,000	31	42544	1493.50	-
	15,000	35	59791	3875.88	-
	20,000	36	92057	7128.20	-
	30,000	38	118872	16767.70	-
III	7,000	3	12368	88.07	-
	10,000	3	19294	194.00	-
	15,000	3	20038	412.32	-
	20,000	3	39873	729.68	-
	30,000	3	56398	1628.44	-

Chapter 4

Solution methods for two-stage stochastic linear programming

This chapter is devoted to computational methods for solving two-stage stochastic linear programming problems with recourse. These problems which originate from the pioneering work of Dantzig (1955), Beale (1955) and Wets (1974) comprise arguably the most important class of stochastic programming (SP) problems. As discussed in Chapter 1 two-stage stochastic programming has a wide range of applications which underlines the importance of having efficient and robust solution methods for SP problems. Some multistage stochastic programming problems such as problems with block separable recourse can be reformulated as two-stage problems (Birge and Louveaux, 1997). Also two-stage solution methods can be used as a basis for building algorithms for multistage SP problems such as nested Benders' decomposition (Louveaux, 1980).

SP problems are known to be computationally challenging. This is mainly due to computation of a multidimensional integral which is required for evaluation of the expected recourse function.

We consider the two-stage SP problem set out in (1.2)-(1.3). The objective function in (1.2) is denoted by $f(\boldsymbol{x})$:

$$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + \mathrm{E}[Q(\boldsymbol{x}, \omega)]$$

We also assume that the vector of random coefficients has a finite discrete distribution with S realisations (scenarios) $\omega_1, \omega_2, \ldots, \omega_S$ and probability $P(\omega_i) = p_i, i = 1, 2, \ldots, S$.

The structure of this chapter is as follows. In Section 4.1 we describe implementation and improvements of several established algorithms for solving two-stage SP problems. This is not intended to be a comprehensive review of SP

solution algorithms but rather a detailed description of our implementations of selected methods. First we briefly discuss the deterministic equivalent approach. Then we consider the L-shaped method of Van Slyke and Wets (1969) and describe the application of the level method of Lemaréchal et al. (1995) to regularisation of the expected recourse function. We also consider two more decomposition-based algorithms, namely regularised decomposition of Ruszczyński (1986) and the trust region method of Linderoth and Wright (2003). In the numerical study presented in Section 4.2 we report the results of a series of computational experiments to compare the performance and scale-up properties of these methods for an extensive set of problems taken from sources considered to be established benchmarks for testing SP solution methods. Finally we present the results in the form of performance profiles that are particularly useful in our case since they provide a visual representation of a large set of test results.

4.1 Solution methods for two-stage SP

Solution of the deterministic equivalent problem

Under the assumption of finite discrete distribution of random parameters a two-stage linear SP problem can be formulated as a large-scale LP problem with a lower block angular structure:

$$
\begin{array}{rlllllll}
\text{minimize} & \boldsymbol{c^T x} & + & p_1 \boldsymbol{q_1}^T \boldsymbol{y_1} & + \ \ldots \ + & p_S \boldsymbol{q_S}^T \boldsymbol{y_S} & & \\
\text{subject to} & A\boldsymbol{x} & & & & & = & \boldsymbol{b}, \\
& T_1 \boldsymbol{x} & + & W_1 \boldsymbol{y_1} & & & = & \boldsymbol{h_1}, \\
& \vdots & & & \ddots & & & \vdots \\
& T_S \boldsymbol{x} & + & & & W_S \boldsymbol{y_S} & = & \boldsymbol{h_S}, \\
& \boldsymbol{x} \in \mathbb{R}^{n_1}_+, & & & & & & \\
& \boldsymbol{y_s} \in \mathbb{R}^{n_2}_+, & s = 1, ..., S. & & & & &
\end{array}
\tag{4.1}
$$

Problem (4.1) can be solved directly using the simplex method or an interior point method.

Benders' decomposition for stochastic programming problems

The structure of the deterministic equivalent problem enables application of decomposition methods. In this section we describe a selection of such methods; and

we refer the readers to Ruszczyński (2003) for a comprehensive overview of the state-of-the-art decomposition methods. As observed by Dantzig and Madansky (1961), Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960) can be applied directly to the dual of problem (4.1).

Van Slyke and Wets (1969) proposed another decomposition-based algorithm and called it the L-shaped method. This is a cutting-plane method which is an application of Benders' decomposition (Benders, 1962) to the solution of the deterministic equivalent problem (4.1). During the iteration k the L-shaped method solves the current problem of the following form:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T\boldsymbol{x} + \theta \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b}, \\ & D_k\boldsymbol{x} \geq \boldsymbol{d_k}, \\ & E_k\boldsymbol{x} + \theta \geq \boldsymbol{e_k}, \\ & \boldsymbol{x} \in \mathbb{R}^{n_1}_+,\ \theta \in \mathbb{R}. \end{aligned} \tag{4.2}$$

In (4.2) $D_k\boldsymbol{x} \geq \boldsymbol{d_k}$ are known as the feasibility cuts and $E_k\boldsymbol{x} + \theta \geq \boldsymbol{e_k}$ are known as the optimality cuts.

The L-shaped method iteratively builds approximations of the expected recourse function $\tilde{Q}(\boldsymbol{x}) = \mathrm{E}[Q(\boldsymbol{x}, \omega)]$ and the feasible region.

The optimality cuts are defined as follows (Birge and Louveaux, 1997):

$$\left(\sum_{s=1}^{S} p_s(\boldsymbol{\pi}_s^*)^T T_s\right) x + \theta \geq \sum_{s=1}^{S} p_s(\boldsymbol{\pi}_s^*)^T h_s,$$

where $\boldsymbol{\pi}_s^*$ is the vector of simplex multipliers associated with an optimal solution of the recourse problem for scenario s:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{q_s}^T\boldsymbol{y} \\ \text{subject to} \quad & W\boldsymbol{y} = \boldsymbol{h_s} - T_s\boldsymbol{x^*}, \\ & \boldsymbol{y} \in \mathbb{R}^{n_2}_+, \end{aligned}$$

where $\boldsymbol{x^*}$ is an optimal solution of the current problem.

The feasibility cuts are defined as follows:

$$\left((\boldsymbol{\sigma}_s^*)^T T_s\right) x \geq (\boldsymbol{\sigma}_s^*)^T h_s,$$

where $\boldsymbol{\sigma}_s^*$ is the vector of simplex multipliers associated with an optimal solution of the following problem for scenario s for which the recourse problem is infeasible:

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}^T(\boldsymbol{u} + \boldsymbol{v}) \\ \text{subject to} \quad & W\boldsymbol{y} + I(\boldsymbol{u} - \boldsymbol{v}) = \boldsymbol{h_s} - T_s\boldsymbol{x^*}, \\ & \boldsymbol{y} \in \mathbb{R}^{n_2}_+, \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{m_2}_+, \end{aligned}$$

Algorithm 1: Scenario clustering

$s \leftarrow 0$, $i \leftarrow 1$
while $s < S$ **do**
  $S_i \leftarrow \left\lceil i \max\left(\frac{S}{\lceil 1/r - 0.5 \rceil}, 1\right) - s - 0.5 \right\rceil$
  $s \leftarrow s + S_i$
  $i \leftarrow i + 1$
end while

where $\mathbf{1} = (1, 1, \ldots, 1)$ and I is the identity matrix of an appropriate size.

Birge and Louveaux (1988) proposed a multicut version of the algorithm. Unlike the original L-shaped method in the multicut version S optimality cuts are added at each iteration where all second-stage subproblems are feasible and instead of a single additional variable θ one such variable per scenario is used.

We consider a more general version of the multicut L-shaped method, in particular used by Linderoth and Wright (2003), where scenarios are divided into C clusters of sizes $S_1, S_2, \ldots, S_C$. The current problem at iteration k is

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T \boldsymbol{x} + \sum_{j=1}^{C} \theta_j \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b}, \\ & D^k \boldsymbol{x} \geq \boldsymbol{d}^k, \\ & E_j^k \boldsymbol{x} + \theta_j \mathbf{1} \geq \boldsymbol{e}_j^k, \quad j = 1, 2, \ldots, C, \\ & \boldsymbol{x} \in \mathbb{R}_+^{n_1}, \boldsymbol{\theta} \in \mathbb{R}^C, \end{aligned} \tag{4.3}$$

where $D^k \boldsymbol{x} \geq \boldsymbol{d}^k$ are the feasibility cuts, $E_j^k \boldsymbol{x} + \theta_j \mathbf{1} \geq \boldsymbol{e}_j^k$ are the optimality cuts.

The computational results reported by Gassmann (1990) and Birge and Louveaux (1988) suggest that the multicut method (with the cluster size of 1) requires in general less iterations to converge than the L-shaped method with aggregated cuts. However the size of the current problem remains smaller in the latter which can result in better overall solution time for problems with large number of scenarios.

To avoid the need of specifying the size of each cluster we provide a simple way of dividing scenarios into clusters of approximately the same size using Algorithm 1. This algorithm takes an input parameter $0 < r \leq 1$ which denotes the cluster size relative to the number of scenarios. For a special case of $r = 0$ we assume $S_i = 1, i = 1, 2, \ldots, C$.

Since all of the decomposition methods considered in this chapter are to some

Algorithm 2: Generic L-shaped method

choose iteration limit $k_{max} \in \mathbb{Z}_+$
choose relative stopping tolerance $\epsilon \in \mathbb{R}_+$
solve the expected value problem to get a solution $\boldsymbol{x}^0$ (initial iterate)
$k \leftarrow 0$, $f^* \leftarrow \infty$
INITIALISE()
while $k < k_{max}$ **do**
  solve the recourse problems (1.3) with $\boldsymbol{x} = \boldsymbol{x}^k$ and compute $f(\boldsymbol{x}^k)$
  if all recourse problems are feasible **then**
    add C optimality cuts
    if $f(\boldsymbol{x}^k) < f^*$ **then**
      $f^* \leftarrow f(\boldsymbol{x}^k)$
      $\boldsymbol{x}^* \leftarrow \boldsymbol{x}^k$
    end if
  else
    add a feasibility cut
  end if
  GET-NEXT-ITERATE()
  $k \leftarrow k + 1$
end while

Here INITIALISE and GET-NEXT-ITERATE are procedures to be defined by specific methods.

extent based on the L-shaped algorithm and therefore have much in common we do not implement them separately. Instead we implement a generic L-shaped algorithm that is extensible through a set of procedures that can be redefined. This is similar to the way some branch and cut frameworks provide a single implementation of the base method which can be extended with user-defined cuts and heuristics. In particular this approach is used in CPLEX (IBM Corp., 2009b) and CBC, a branch and cut solver from the COIN-OR repository (Lougee-Heimer, 2003).

The pseudo-code for this generic L-shaped method is given in Algorithm 2. It can be used to implement various L-shaped based methods such as regularised decomposition. The extensibility is achieved because we are able to redefine the following procedures in alternative (L-shaped) decomposition methods:

- INITIALISE: Perform method-specific initialisation

- GET-NEXT-ITERATE: Compute the next iterate

We assume that these procedures have access to all the variables and parameters in Algorithm 2 and INITIALISE can introduce its own variables accessible by GET-NEXT-ITERATE.

The following algorithms are implemented based on this generic framework:

- The L-shaped method
- The multicut L-shaped algorithm
- L-shaped algorithm with regularisation by the level method
- Trust region method based on l_∞ norm
- Regularised decomposition

When the relative cluster size $r = 1$, which is the default, there is only one cluster of size S resulting in the original L-shaped method of Van Slyke and Wets (1969). $r = 0$ results in the multicut version (Birge and Louveaux, 1988) with each cluster consisting of a single scenario. Intermediate values are also possible, for example if $S = 7$ and $r = \frac{1}{3}$, then scenarios will be divided into 3 clusters of sizes 2, 3 and 2.

Algorithm 3: Pseudo-code of the INITIALISE procedure for the L-shaped method

$f^0 \leftarrow -\infty$

Algorithm 4: Pseudo-code of the GET-NEXT-ITERATE procedure for the L-shaped method

if $f^* - f^k \leq \epsilon|f^*|$ **then**
  stop
end if
solve the current problem (4.3) to get an optimal solution $(\boldsymbol{x}^{k+1}, \boldsymbol{\theta}^{k+1})$ and the optimal objective value f^{k+1}; $\boldsymbol{x}^{k+1}$ is the next iterate
if $f^* - f^{k+1} \leq \epsilon|f^*|$ **then**
  stop
end if

To get the classic L-shaped method the INITIALISE and GET-NEXT-ITERATE procedures are defined as shown in Algorithms 3 and 4 respectively. The INITIALISE procedure is trivial because most of the initialisation is performed in the

base method. The second check of optimality condition in Algorithm 4 ensures that the recourse problems are not solved unnecessarily when the optimality gap becomes acceptable due to increase in the lower bound.

L-shaped method with regularisation by the level method

One of the drawbacks of the L-shaped method is that the values of the first-stage variables may change significantly from iteration to iteration with minor, or even zero, change in the value of the objective function. This effect is due to the linear approximation of the expected recourse function.

Several methods have been proposed that address the above problem. Some of these methods in addition to the sequence of iterates maintain a sequence of stability centres and select the next iterate in a trust region around the current stability centre or use quadratic approximation instead of linear. The methods that use the latter approach include the proximal point algorithm of Rockafellar (1976) and bundle methods of Lemaréchal (1978) and Kiwiel (1985).

The level method of Lemaréchal, Nemirovskii, and Nesterov (1995) is an algorithm for nonsmooth convex optimisation. An inexact version of the level method by Fábián (2000) was applied to the solution of stochastic programming problems in the level decomposition method (Fábián and Szőke, 2007).

We propose a regularisation approach (Zverovich et al., 2010) based on the original level method of Lemaréchal, Nemirovskii, and Nesterov. The main difference of the proposed method from the unregularised L-shaped method is the way the next iterate is selected. Instead of using the solution of the current problem as the next iterate, a projection of the previous iterate on the level set of the current approximation of the objective function is used. Level set is defined as follows:

$$\hat{X} = \left\{ \boldsymbol{x} \in X : \boldsymbol{c}^T \boldsymbol{x} + \sum_{j=1}^{C} \theta_j \leq (1-\lambda) f^k + \lambda f^* \right\}, \tag{4.4}$$

where X is a feasible region of problem (4.3), f^k is the optimal objective value of a piecewise-linear convex function which is an outer approximation of the real objective function at iteration k defined by the current set of cuts, f^* is the upper bound (the best objective value found so far) and $0 < \lambda < 1$ is a parameter.

Computing the projection requires solution of an additional quadratic programming problem at each iteration. As will be shown in Chapter 4.2 despite increased amount of computation per iteration the overall solution time is often reduced when the level regularisation is used due to substantial reduction in the

number of iterations and thus second-stage value function evaluations.

The pseudo-code of the INITIALISE and GET-NEXT-ITERATE procedures for the L-shaped method regularised by the level method is given in Algorithms 5 and 6 respectively.

Algorithm 5: Pseudo-code of the INITIALISE procedure for the L-shaped algorithm with regularisation by the level method

choose $\lambda \in (0, 1)$
$f^0 \leftarrow -\infty$

Algorithm 6: Pseudo-code of the GET-NEXT-ITERATE procedure for the L-shaped algorithm with regularisation by the level method

if $f^* - f^k \leq \epsilon |f^*|$ **then**
  stop
end if
solve the current problem (4.3) to get an optimal solution $(\boldsymbol{x}', \boldsymbol{\theta}')$ and the optimal objective value f^{k+1}.
if $f^* - f^{k+1} \leq \epsilon |f^*|$ **then**
  stop
end if
solve the projection problem:

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{x} - \boldsymbol{x}'\|^2 \\ \text{subject to} \quad & \boldsymbol{c}^T\boldsymbol{x} + \sum_{j=1}^{C} \theta_j \leq (1-\lambda) f^{k+1} + \lambda f^* \\ & A\boldsymbol{x} = \boldsymbol{b}, \\ & D^k\boldsymbol{x} \geq \boldsymbol{d}^k, \\ & E_j^k\boldsymbol{x} + \theta_j \mathbf{1} \geq \boldsymbol{e}_j^k, \quad j = 1, 2, \ldots, C, \\ & \boldsymbol{x} \in \mathbb{R}_+^{n_1}, \boldsymbol{\theta} \in \mathbb{R}^C, \end{aligned} \tag{4.5}$$

let $(\boldsymbol{x}^{k+1}, \boldsymbol{\theta}^{k+1})$ be an optimal solution of the projection problem; then $\boldsymbol{x}^{k+1}$ is the next iterate

The following problem is used to illustrate the L-shaped method and its regularised counterpart.

Figure 4.1: Illustration of the initial iterations of the L-shaped method when solving problem (4.6)

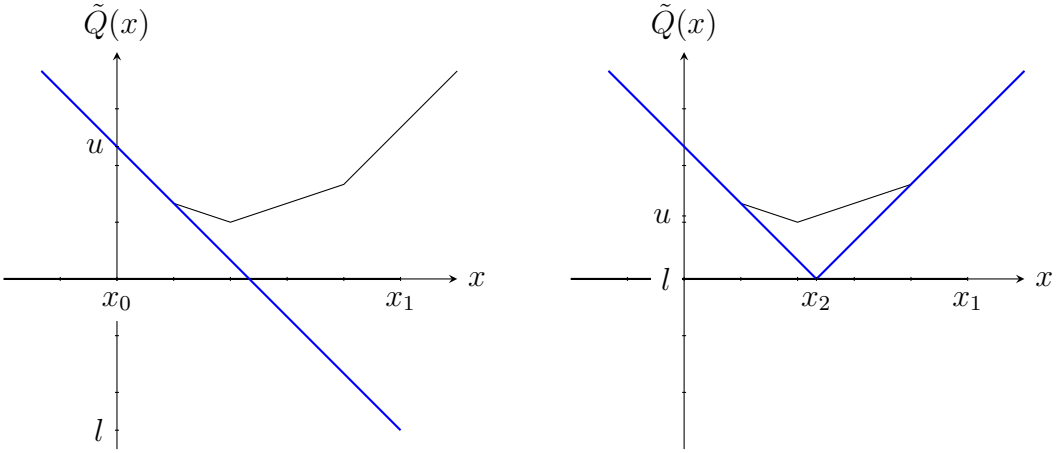
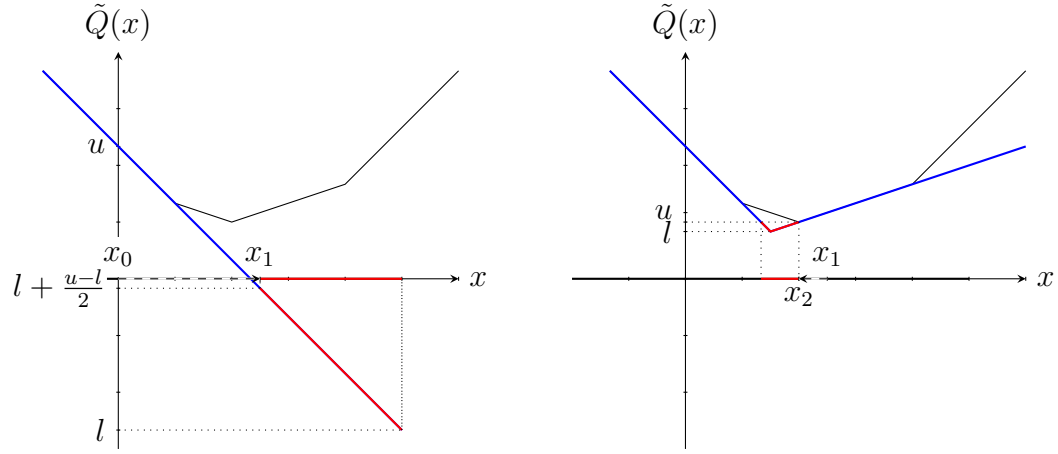


Figure 4.2: Illustration of the initial iterations of the L-shaped method with regularisation by the level method when solving problem (4.6)



$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{3} p_i q(x, \xi_i) \\ &\text{subject to} && x \leq 5, \end{aligned} \tag{4.6}$$

where

$$q(x, \xi) = \begin{cases} \xi - x, & \text{if } x \leq \xi \\ x - \xi, & \text{if } x > \xi \end{cases}$$

and ξ takes on the values 1, 2, and 4, each with probability 1/3.

$\tilde{Q}(x) = \sum_{i=1}^{3} p_i q(x, \xi_i)$ is the expected recourse function.

The initial iterations of the L-shaped method are shown in Figure 4.1. It is clearly seen that the initial iterations are inefficient due to jumps around the feasibility region.

On the other hand the regularised method which is illustrated in Figure 4.2 is less affected by this issue and makes faster progress towards optimality. This is confirmed by the computational results in Section 4.2.

Trust region method based on the infinity norm

Trust region methods construct a sequence of stability centres and select the next iterate from the trust region around the current stability centre. In general a trust region is defined as follows:

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \leq \Delta, \tag{4.7}$$

where $\hat{\boldsymbol{x}}$ is the current stability centre (reference point) and Δ is the radius of a trust region.

Different methods can be obtained by using different norms in (4.7). Also various strategies of adapting the trust region radius Δ can be employed. We implemented the trust region method of Linderoth and Wright (2003) which is based on the infinity norm (l_∞). This method operates on the following current problem with additional bounds that define a hyper-rectangular trust region:

$$\begin{aligned}
\text{minimize} \quad & \boldsymbol{c}^T\boldsymbol{x} + \sum_{j=1}^{C} \theta_j \\
\text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b}, \\
& D^k\boldsymbol{x} \geq \boldsymbol{d}^k, \\
& E_j^k\boldsymbol{x} + \theta_j\mathbf{1} \geq \boldsymbol{e}_j^k, \quad j = 1, 2, \ldots, C, \\
& \boldsymbol{x} \in \mathbb{R}_+^{n_1}, \boldsymbol{\theta} \in \mathbb{R}^C, \\
& \hat{\boldsymbol{x}} - \Delta \leq \boldsymbol{x} \leq \hat{\boldsymbol{x}} + \Delta.
\end{aligned} \tag{4.8}$$

The advantage of this method is that the current problem (4.8) remains linear.

Algorithms 7 and 8 show the pseudo-code of the procedures for the l_∞ trust region method of Linderoth and Wright.

Algorithm 7: Pseudo-code of the INITIALISE procedure for the l_∞ trust region method

---

choose $\xi \in (0, 1/2)$ and maximum trust region radius $\Delta_{hi} \in [1, \infty)$

choose initial radius $\Delta \in [1, \Delta_{hi}]$

counter $\leftarrow 0$

$\hat{f} \leftarrow \infty$

Algorithm 8: Pseudo-code of the GET-NEXT-ITERATE procedure for the l_∞ trust region method

if $\hat{f} < \infty$ **then**
  if $\hat{f} - f(\boldsymbol{x}^k) \geq \xi(\hat{f} - f^k)$ **then**
    if $\hat{f} - f(\boldsymbol{x}^k) \geq 0.5(\hat{f} - f^k)$ **and** $\|\boldsymbol{x}^k - \hat{\boldsymbol{x}}\|_\infty = \Delta$ **then**
      increase the radius:
      $\Delta \leftarrow \min(2\Delta, \Delta_{hi})$
    end if
    set a reference point:
    $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}^k$
    $\hat{f} \leftarrow f(\hat{\boldsymbol{x}})$
    counter $\leftarrow 0$
  else
    $\rho \leftarrow -\min(1, \Delta)(\hat{f} - f(\boldsymbol{x}^k))/(\hat{f} - f^k)$
    if $\rho > 0$ **then**
      counter $\leftarrow$ counter $+ 1$
    end if
    if $\rho > 3$ **or** (counter ≥ 3 **and** $\rho \in (1, 3]$) **then**
      decrease the radius:
      $\Delta \leftarrow \Delta / \min(\rho, 4)$
      counter $\leftarrow 0$
    end if
  end if
else
  set a new reference point:
  $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}^k$
  $\hat{f} \leftarrow f(\hat{\boldsymbol{x}})$
end if
solve the current problem (4.8)
let $(\boldsymbol{x}^{k+1}, \boldsymbol{\theta}^{k+1})$ be an optimal solution of the problem (4.8) and f^{k+1} be its optimal value
if $|\hat{f} - f^{k+1}| \leq \epsilon|\hat{f}|$ **then**
  stop
end if

Regularised decomposition

Regularised decomposition (RD) proposed by Ruszczyński (1986) is an algorithm for minimisation of a sum of piecewise linear convex functions over a convex polyhedron. This method operates on a multicut version of the current problem (4.9) with an additional quadratic term that penalizes deviation from the current stability centre. RD also employs an effective cut reduction strategy to eliminate inactive cuts and therefore keep the size of the current problem manageable even for large number of scenarios.

We implemented deletion of inactive cuts and dynamic adaptation of the penalty parameter σ as described by Ruszczyński and Świętanowski (1997). Algorithms 9 and 10 give the pseudo-code of the INITIALISE and GET-NEXT-ITERATE procedures for regularised decomposition as implemented in our generic framework.

Algorithm 9: Pseudo-code of the INITIALISE procedure for the regularised decomposition method

$\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}^0$
$f^0 \leftarrow \infty$
choose σ and γ

Implementation issues

Preliminary experiments showed that keeping all the cuts in regularised decomposition, while resulting in a smaller number of iterations, increases overall solution time due to current problem becoming much more difficult to solve. At the same time the trust region method based on l_∞ norm is exposed to this issue to a less extent and therefore no cut deletion was done.

The relative stopping tolerance $\epsilon = 10^{-5}$ was used for the L-shaped method with and without regularisation by the level method. The stopping criteria in the trust region algorithm and regularised decomposition are different because these methods do not provide global lower bound. Therefore ϵ was set to a lower value of 10^{-6} for the latter methods.

To speed up the solution of multiple second stage subproblems we used warm start facilities of the underlying LP solver.

Algorithm 10: Pseudo-code of the GET-NEXT-ITERATE procedure for the regularised decomposition method

if $k = 0$ **or** $|f(x^k) - f^k| \leq \epsilon |f(x^k)|$ **then**
  set a new reference point:
  $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}^k$
  $\hat{f} \leftarrow f(\hat{\boldsymbol{x}})$
end if
if $f(\boldsymbol{x}^k) < \infty$ **then**
  if $f(\boldsymbol{x}^k) > \gamma \hat{f} + (1-\gamma) f^k$ **then**
    $\sigma \leftarrow \sigma / 2$
  else if $f(\boldsymbol{x}^k) < (1-\gamma)\hat{f} + \gamma f^k$ **then**
    $\sigma \leftarrow 2\sigma$
  end if
end if
solve the current problem with an additional quadratic term in the objective:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T \boldsymbol{x} + \sum_{j=1}^{C} \theta_j + \frac{1}{2\sigma} \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b}, \\ & D^k \boldsymbol{x} \geq \boldsymbol{d}^k, \\ & E_j^k \boldsymbol{x} + \theta_j \mathbf{1} \geq \boldsymbol{e}_j^k, \quad j = 1, 2, \ldots, C, \\ & \boldsymbol{x} \in \mathbb{R}_+^{n_1}, \boldsymbol{\theta} \in \mathbb{R}^C, \end{aligned} \tag{4.9}$$

let $(\boldsymbol{x}^{k+1}, \boldsymbol{\theta}^{k+1})$ be an optimal solution of the problem (4.9) and
$f^{k+1} = \boldsymbol{c}^T \boldsymbol{x}^{k+1} + \sum_{j=1}^{C} \theta_j^{k+1}$
if $|\hat{f} - f^{k+1}| \leq \epsilon |\hat{f}|$ **then**
  stop
end if
delete constraints that have corresponding dual variables zero in the solution of (4.9), keeping the last C added constraints intact

4.2 Numerical study

Experimental setup

The computational experiments are performed on a Linux machine with 2.4 GHz Intel CORE i5 M520 CPU and 6 GiB of RAM. Deterministic equivalent problems are solved with CPLEX 12.1 dual simplex and barrier optimisers. Crossover to a basic solution is disabled for the barrier optimiser and the number of threads is limited to 1. For other CPLEX options the default values are used.

The times are reported in seconds with the times of reading input files not included. For simplex and IPM the times of constructing deterministic equivalent problems are included though these times only amount to small fractions of the total. CPLEX linear and quadratic programming solver is used to solve the current problems and subproblems in the decomposition methods. All the test problems are presented in SMPS format introduced by Birge et al. (1987).

All solution methods considered in the current study are implemented within the FortSP stochastic solver system (Ellison et al., 2010) which includes the extensible algorithmic framework for creating decomposition-based methods described in the previous section.

We consider the following methods:

- Solution of the DEP with the simplex method (DEP - Simplex),
- Solution of the DEP with IPM (DEP - IPM),
- Benders' decomposition for two stage SP problems (Benders) known as the L-shaped method,
- Benders' decomposition with regularisation by the level method (Level),
- the trust region method based on l_∞ norm (TR),
- regularized decomposition (RD).

The short names in parentheses are used to refer to these methods in the tables and figures.

The first-stage solution of the expected value problem is taken as a starting point for the decomposition methods. The values of the parameters are specified below.

- Benders decomposition with regularisation by the level method:
  $\lambda = 0.5$,

Table 4.1: Sources of test problems

	Source	Reference	Comments
1.	POSTS collection	Holmes (1995)	Two-stage SP problems from the (PO)rtable (S)tochastic programming (T)est (S)et (POSTS)
2.	Slptestset collection	Ariyawansa and Felt (2004)	Two-stage problems from the collection of stochastic LP test problems
3.	Random problems	Kall and Mayer (1998)	Artificial test problems generated with pseudo random stochastic LP problem generator GENSLP
4.	SAMPL problems	König et al. (2007)	Problems instantiated from the SAPHIR gas portfolio planning model formulated in Stochastic AMPL (SAMPL)

- Regularised decomposition:
  $\sigma = 1, \gamma = 0.9$.

- Trust region method based on l_∞ norm:
  $\Delta = 1$ (initial), $\Delta_{hi} = 10^3$ (except for the saphir problems where $\Delta_{hi} = 10^9$), $\xi = 10^{-4}$.

Data sets

We consider test problems which are drawn from four different sources described in Table 4.1. Table 4.2 gives the dimensions of these problems.

Most of the benchmark problems have stochasticity only in the right-hand side (RHS). A notable exception is the SAPHIR family of problems which has random elements both in the RHS and the constraint matrix.

Table 4.2: Dimensions of SP test problems

		Stage 1		Stage 2		Deterministic Equivalent		
Name	Scen	Rows	Cols	Rows	Cols	Rows	Cols	Nonzeros
fxm	6	92	114	238	343	1520	2172	12139
	16	92	114	238	343	3900	5602	31239

Dimensions of test problems (continued)

		Stage 1		Stage 2		Deterministic Equivalent		
Name	Scen	Rows	Cols	Rows	Cols	Rows	Cols	Nonzeros
fxmev	1	92	114	238	343	330	457	2589
pltexpa	6	62	188	104	272	686	1820	3703
	16	62	188	104	272	1726	4540	9233
stormg2	8	185	121	528	1259	4409	10193	27424
	27	185	121	528	1259	14441	34114	90903
	125	185	121	528	1259	66185	157496	418321
	1000	185	121	528	1259	528185	1259121	3341696
airl-first	25	2	4	6	8	152	204	604
airl-second	25	2	4	6	8	152	204	604
airl-randgen	676	2	4	6	8	4058	5412	16228
assets	100	5	13	5	13	505	1313	2621
	37500	5	13	5	13	187505	487513	975021
4node	1	14	52	74	186	88	238	756
	2	14	52	74	186	162	424	1224
	4	14	52	74	186	310	796	2160
	8	14	52	74	186	606	1540	4032
	16	14	52	74	186	1198	3028	7776
	32	14	52	74	186	2382	6004	15264
	64	14	52	74	186	4750	11956	30240
	128	14	52	74	186	9486	23860	60192
	256	14	52	74	186	18958	47668	120096
	512	14	52	74	186	37902	95284	239904
	1024	14	52	74	186	75790	190516	479520
	2048	14	52	74	186	151566	380980	958752
	4096	14	52	74	186	303118	761908	1917216
	8192	14	52	74	186	606222	1523764	3834144
	16384	14	52	74	186	1212430	3047476	7668000
	32768	14	52	74	186	2424846	6094900	15335712

Dimensions of test problems (continued)

		Stage 1		Stage 2		Deterministic Equivalent		
Name	Scen	Rows	Cols	Rows	Cols	Rows	Cols	Nonzeros
4node-base	1	16	52	74	186	90	238	772
	2	16	52	74	186	164	424	1240
	4	16	52	74	186	312	796	2176
	8	16	52	74	186	608	1540	4048
	16	16	52	74	186	1200	3028	7792
	32	16	52	74	186	2384	6004	15280
	64	16	52	74	186	4752	11956	30256
	128	16	52	74	186	9488	23860	60208
	256	16	52	74	186	18960	47668	120112
	512	16	52	74	186	37904	95284	239920
	1024	16	52	74	186	75792	190516	479536
	2048	16	52	74	186	151568	380980	958768
	4096	16	52	74	186	303120	761908	1917232
	8192	16	52	74	186	606224	1523764	3834160
	16384	16	52	74	186	1212432	3047476	7668016
	32768	16	52	74	186	2424848	6094900	15335728
4node-old	32	14	52	74	186	2382	6004	15264
chem	2	38	39	46	41	130	121	289
chem-base	2	38	39	40	41	118	121	277
lands	3	2	4	7	12	23	40	92
lands-blocks	3	2	4	7	12	23	40	92
env-aggr	5	48	49	48	49	288	294	876
env-first	5	48	49	48	49	288	294	876
env-loose	5	48	49	48	49	288	294	876
env	15	48	49	48	49	768	784	2356
	1200	48	49	48	49	57648	58849	177736
	1875	48	49	48	49	90048	91924	277636
	3780	48	49	48	49	181488	185269	559576
	5292	48	49	48	49	254064	259357	783352
	8232	48	49	48	49	395184	403417	1218472
	32928	48	49	48	49	1580592	1613521	4873480
env-diss-aggr	5	48	49	48	49	288	294	876
env-diss-first	5	48	49	48	49	288	294	876
env-diss-loose	5	48	49	48	49	288	294	876

Dimensions of test problems (continued)

		Stage 1		Stage 2		Deterministic Equivalent		
Name	Scen	Rows	Cols	Rows	Cols	Rows	Cols	Nonzeros
	15	48	49	48	49	768	784	2356
	1200	48	49	48	49	57648	58849	177736
	1875	48	49	48	49	90048	91924	277636
env-diss	3780	48	49	48	49	181488	185269	559576
	5292	48	49	48	49	254064	259357	783352
	8232	48	49	48	49	395184	403417	1218472
	32928	48	49	48	49	1580592	1613521	4873480
phone1	1	1	8	23	85	24	93	309
phone	32768	1	8	23	85	753665	2785288	9863176
stocfor1	1	15	15	102	96	117	111	447
stocfor2	64	15	15	102	96	6543	6159	26907
	2000	50	100	25	50	50050	100100	754501
	4000	50	100	25	50	100050	200100	1508501
rand0	6000	50	100	25	50	150050	300100	2262501
	8000	50	100	25	50	200050	400100	3016501
	10000	50	100	25	50	250050	500100	3770501
	2000	100	200	50	100	100100	200200	3006001
	4000	100	200	50	100	200100	400200	6010001
rand1	6000	100	200	50	100	300100	600200	9014001
	8000	100	200	50	100	400100	800200	12018001
	10000	100	200	50	100	500100	1000200	15022001
	2000	150	300	75	150	150150	300300	6758501
	4000	150	300	75	150	300150	600300	13512501
rand2	6000	150	300	75	150	450150	900300	20266501
	8000	150	300	75	150	600150	1200300	27020501
	10000	150	300	75	150	750150	1500300	33774501
	50	32	53	8678	3924	433932	196253	1136753
	100	32	53	8678	3924	867832	392453	2273403
saphir	200	32	53	8678	3924	1735632	784853	4546703
	500	32	53	8678	3924	4339032	1962053	11366603
	1000	32	53	8678	3924	8678032	3924053	22733103

It should be noted that the problems generated with GENSLP do not possess any internal structure inherent in real-world problems. However they are still useful for the purposes of comparing scale-up properties of algorithms.

Computational results

The computational results are presented in Tables 4.3 and 4.4. Iter denotes the number of iterations. For decomposition methods this is the number of master iterations.

There were multiple solver failures on the saphir problems due to numerical difficulties. This is probably due to a very wide range of data values which is inherent in this gas portfolio planning model.

Table 4.3: Performance of DEP solution methods and level-regularised decomposition

		DEP - Simplex		DEP - IPM		Level		Optimal
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Value
fxm	6	0.06	1259	0.05	17	0.15	20	18417.1
	16	0.22	3461	0.13	23	0.15	20	18416.8
fxmev	1	0.01	273	0.01	14	0.13	20	18416.8
pltexpa	6	0.01	324	0.03	14	0.02	1	-9.47935
	16	0.01	801	0.08	16	0.02	1	-9.66331
stormg2	8	0.08	3649	0.25	28	0.16	20	15535200
	27	0.47	12770	2.27	27	0.31	17	15509000
	125	5.10	70177	8.85	57	0.93	17	15512100
	1000	226.70	753739	137.94	114	6.21	21	15802600
airl-first	25	0.01	162	0.01	9	0.03	17	249102
airl-second	25	0.00	145	0.01	11	0.03	17	269665
airl-randgen	676	0.25	4544	0.05	11	0.22	18	250262
assets	100	0.02	494	0.02	17	0.03	1	-723.839
	37500	1046.85	190774	6.37	24	87.55	2	-695.963

Performance of DEP solution methods and level-regularised decomposition (continued)

		DEP - Simplex		DEP - IPM		Level		Optimal
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Value
4node	1	0.01	110	0.01	12	0.06	21	413.388
	2	0.01	196	0.01	14	0.10	42	414.013
	4	0.01	326	0.02	17	0.11	45	416.513
	8	0.03	825	0.05	18	0.10	45	418.513
	16	0.06	1548	0.11	17	0.15	44	423.013
	32	0.16	2948	0.40	15	0.22	51	423.013
	64	0.72	7185	0.44	17	0.36	54	423.013
	128	2.30	12053	0.50	26	0.47	50	423.013
	256	7.69	31745	1.05	30	0.87	48	425.375
	512	57.89	57200	2.35	30	2.12	51	429.963
	1024	293.19	133318	5.28	32	3.95	53	434.112
	2048	1360.60	285017	12.44	36	7.82	49	441.738
	4096	t	-	32.67	46	9.12	46	446.856
	8192	t	-	53.82	45	22.68	55	446.856
	16384	t	-	113.20	46	45.24	52	446.856
	32768	t	-	257.96	48	127.86	62	446.856
4node-base	1	0.01	111	0.01	11	0.04	16	413.388
	2	0.01	196	0.01	14	0.06	29	414.013
	4	0.01	421	0.02	14	0.07	30	414.388
	8	0.03	887	0.04	15	0.10	35	414.688
	16	0.06	1672	0.11	17	0.10	30	414.688
	32	0.15	3318	0.40	15	0.16	37	416.6
	64	0.49	7745	0.36	13	0.22	33	416.6
	128	1.58	17217	0.33	19	0.35	37	416.6
	256	4.42	36201	0.81	23	0.53	31	417.162
	512	22.44	80941	2.20	29	1.45	37	420.293
	1024	141.91	187231	5.21	32	3.33	41	423.05
	2048	694.89	337082	11.12	32	6.13	42	423.763
	4096	t	-	27.03	37	10.60	39	424.753
	8192	t	-	51.29	40	24.99	48	424.775
	16384	t	-	177.81	73	47.31	41	424.775
	32768	t	-	242.91	48	102.29	49	424.775
4node-old	32	0.20	3645	0.49	18	0.09	20	83094.1
chem	2	0.00	29	0.00	11	0.03	15	-13009.2
chem-base	2	0.00	31	0.00	11	0.05	14	-13009.2
lands	3	0.00	21	0.00	9	0.02	10	381.853

Performance of DEP solution methods and level-regularised decomposition (continued)

		DEP - Simplex		DEP - IPM		Level		Optimal
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Value
lands-blocks	3	0.00	21	0.00	9	0.02	10	381.853
env-aggr	5	0.01	117	0.01	12	0.04	16	20478.7
env-first	5	0.01	112	0.01	11	0.02	1	19777.4
env-loose	5	0.01	112	0.01	12	0.02	1	19777.4
	15	0.01	321	0.01	16	0.05	15	22265.3
	1200	1.38	23557	1.44	34	1.73	15	22428.9
	1875	2.90	36567	2.60	34	2.80	15	22447.1
env	3780	11.21	73421	7.38	40	5.47	15	22441
	5292	20.28	102757	12.19	42	7.67	15	22438.4
	8232	62.25	318430	m	-	12.58	15	22439.1
	32928	934.38	1294480	m	-	75.67	15	22439.1
env-diss-aggr	5	0.01	131	0.01	9	0.05	22	15963.9
env-diss-first	5	0.01	122	0.01	9	0.04	12	14794.6
env-diss-loose	5	0.01	122	0.01	9	0.03	5	14794.6
	15	0.01	357	0.02	13	0.10	35	20773.9
	1200	1.96	26158	1.99	50	2.80	35	20808.6
	1875	4.41	40776	3.63	53	4.49	36	20809.3
env-diss	3780	16.94	82363	9.32	57	8.87	36	20794.7
	5292	22.37	113894	16.17	66	12.95	38	20788.6
	8232	70.90	318192	m	-	22.49	41	20799.4
	32928	1369.97	1296010	m	-	112.46	41	20799.4
phone1	1	0.00	19	0.01	8	0.02	1	36.9
phone	32768	t	-	50.91	26	48.23	1	36.9
stocfor1	1	0.00	39	0.01	11	0.03	6	-41132
stocfor2	64	0.12	2067	0.08	17	0.12	9	-39772.4
	2000	373.46	73437	9.41	33	6.10	44	162.146
	4000	1603.25	119712	34.28	62	10.06	32	199.032
rand0	6000	t	-	48.84	60	21.17	51	140.275
	8000	t	-	56.89	49	28.86	50	170.318
	10000	t	-	98.51	71	52.31	71	139.129

Performance of DEP solution methods and level-regularised decomposition (continued)

		DEP - Simplex		DEP - IPM		Level		Optimal
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Value
rand1	2000	t	-	39.97	24	52.70	74	244.159
	4000	t	-	92.71	28	72.30	59	259.346
	6000	t	-	158.24	32	103.00	58	297.563
	8000	t	-	228.68	34	141.81	65	262.451
	10000	t	-	320.10	39	181.98	63	298.638
rand2	2000	t	-	102.61	22	145.22	65	209.151
	4000	t	-	225.71	24	170.08	42	218.247
	6000	t	-	400.52	28	369.35	52	239.721
	8000	t	-	546.98	29	369.01	44	239.158
	10000	t	-	754.52	32	623.59	52	231.706
saphir	50	269.17	84727	n	-	341.86	43	129505000
	100	685.50	152866	n	-	700.44	46	129058000
	200	t	-	549.45	167	t	-	141473000
	500	t	-	t	-	608.48	44	137871000
	1000	t	-	n	-	804.11	46	133036000

Table 4.4: Performance of decomposition methods

		Benders		Level		TR		RD	
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Time	Iter
fxm	6	0.08	25	0.15	20	0.09	22	0.05	5
	16	0.09	25	0.15	20	0.11	22	0.07	5
fxmev	1	0.08	25	0.13	20	0.08	22	0.05	5
pltexpa	6	0.02	1	0.02	1	0.02	1	0.03	1
	16	0.02	1	0.02	1	0.02	1	0.03	1
stormg2	8	0.14	23	0.16	20	0.08	9	0.10	10
	27	0.47	32	0.31	17	0.18	10	0.23	11
	125	1.73	34	0.93	17	0.50	8	0.89	12
	1000	11.56	41	6.21	21	3.38	6	7.30	11
airl-first	25	0.04	16	0.03	17	0.03	6	0.03	10
airl-second	25	0.02	10	0.03	17	0.02	4	0.03	5
airl-randgen	676	0.22	18	0.22	18	0.22	6	0.29	6

Performance of decomposition methods (continued)

		Benders		Level		TR		RD	
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Time	Iter
assets	100	0.02	1	0.03	1	0.03	1	0.02	1
	37500	87.68	2	87.55	2	172.23	2	114.38	1
4node	1	0.03	24	0.06	21	0.03	8	0.03	15
	2	0.04	38	0.10	42	0.02	16	0.05	29
	4	0.04	41	0.11	45	0.03	14	0.05	19
	8	0.07	64	0.10	45	0.03	13	0.05	16
	16	0.11	67	0.15	44	0.04	12	0.05	13
	32	0.23	100	0.22	51	0.05	10	0.07	13
	64	0.27	80	0.36	54	0.08	11	0.12	14
	128	0.39	74	0.47	50	0.15	11	0.19	14
	256	0.95	71	0.87	48	0.20	7	0.29	9
	512	3.72	92	2.12	51	0.46	7	0.62	9
	1024	5.14	70	3.95	53	0.42	3	1.23	10
	2048	11.78	83	7.82	49	1.30	4	1.22	5
	4096	18.46	89	9.12	46	2.79	3	2.03	4
	8192	46.56	106	22.68	55	9.87	3	6.59	4
	16384	99.00	110	45.24	52	38.28	3	27.50	4
	32768	194.68	122	127.86	62	299.85	3	222.61	4
4node-base	1	0.03	31	0.04	16	0.03	21	0.03	14
	2	0.04	44	0.06	29	0.03	19	0.05	19
	4	0.06	58	0.07	30	0.04	20	0.07	34
	8	0.05	47	0.10	35	0.04	19	0.08	28
	16	0.08	56	0.10	30	0.06	21	0.11	28
	32	0.17	63	0.16	37	0.07	13	0.18	22
	64	0.23	61	0.22	33	0.17	19	0.30	21
	128	0.39	65	0.35	37	0.34	19	0.63	23
	256	0.89	66	0.53	31	0.45	11	1.81	26
	512	3.27	84	1.45	37	1.84	14	4.98	29
	1024	9.57	115	3.33	41	5.53	13	9.17	17
	2048	19.72	142	6.13	42	21.82	13	31.08	21
	4096	38.51	174	10.60	39	85.68	12	146.50	18
	8192	133.45	290	24.99	48	354.05	14	t	-
	16384	164.07	175	47.31	41	1430.72	13	t	-
	32768	314.31	191	102.29	49	t	-	t	-
4node-old	32	0.08	30	0.09	20	0.04	7	0.09	10
chem	2	0.04	7	0.03	15	0.03	13	0.04	19
chem-base	2	0.02	6	0.05	14	0.02	13	0.04	22

Performance of decomposition methods (continued)

		Benders		Level		TR		RD	
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Time	Iter
lands	3	0.02	8	0.02	10	0.02	5	0.03	17
lands-blocks	3	0.01	8	0.02	10	0.02	5	0.03	17
env-aggr	5	0.02	3	0.04	16	0.02	3	0.03	5
env-first	5	0.02	1	0.02	1	0.02	1	0.02	1
env-loose	5	0.01	1	0.02	1	0.02	1	0.02	1
	15	0.04	3	0.05	15	0.03	3	0.03	5
	1200	0.34	3	1.73	15	0.48	3	0.76	5
	1875	0.57	3	2.80	15	0.90	3	1.50	5
env	3780	1.26	3	5.47	15	2.48	3	3.79	5
	5292	1.96	3	7.67	15	4.51	3	5.89	5
	8232	3.70	3	12.58	15	10.67	3	12.54	5
	32928	39.88	3	75.67	15	211.90	3	212.05	5
env-diss-aggr	5	0.03	9	0.05	22	0.03	9	0.03	17
env-diss-first	5	0.02	14	0.04	12	0.02	4	0.03	4
env-diss-loose	5	0.03	15	0.03	5	0.02	4	0.02	4
	15	0.05	27	0.10	35	0.05	18	0.07	12
	1200	1.13	24	2.80	35	2.25	18	3.45	19
	1875	2.50	29	4.49	36	5.52	19	4.52	15
env-diss	3780	5.04	29	8.87	36	20.23	19	8.98	11
	5292	8.14	34	12.95	38	40.39	17	17.90	13
	8232	14.21	35	22.49	41	119.88	16	99.19	23
	32928	79.52	35	112.46	41	t	-	t	-
phone1	1	0.02	1	0.02	1	0.02	1	0.02	1
phone	32768	48.34	1	48.23	1	73.45	1	73.75	1
stocfor1	1	0.02	6	0.03	6	0.02	2	0.02	2
stocfor2	64	0.10	7	0.12	9	0.18	14	0.23	18
	2000	10.42	80	6.10	44	30.33	9	93.78	16
	4000	19.97	69	10.06	32	82.75	8	591.45	14
rand0	6000	41.82	108	21.17	51	275.97	9	t	-
	8000	65.51	127	28.86	50	423.51	9	t	-
	10000	153.07	230	52.31	71	871.00	10	t	-

Performance of decomposition methods (continued)

		Benders		Level		TR		RD	
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Time	Iter
	2000	265.14	391	52.70	74	155.81	12	361.54	17
	4000	587.22	502	72.30	59	508.18	11	t	-
rand1	6000	649.58	385	103.00	58	937.74	11	t	-
	8000	917.24	453	141.81	65	1801.43	9	t	-
	10000	1160.62	430	181.98	63	t	-	t	-
	2000	1800.00	818	145.22	65	334.36	12	794.31	17
	4000	1616.56	414	170.08	42	813.49	11	t	-
rand2	6000	t	-	369.35	52	t	-	t	-
	8000	t	-	369.01	44	t	-	t	-
	10000	t	-	623.59	52	t	-	t	-
	50	733.37	128	341.86	43	578.87	110	n	-
	100	1051.89	123	700.44	46	n	-	n	-
saphir	200	t	-	t	-	t	-	n	-
	500	1109.48	122	608.48	44	1283.97	99	n	-
	1000	1444.17	124	804.11	46	n	-	n	-

t - time limit, m - insufficient memory, n - numerical difficulties

Comments on scale-up properties and on accuracy

We perform a set of experiments recording the change in the relative gap between the lower and upper bounds on objective function in the decomposition methods. The results are shown in Figures 4.3 – 4.6. These diagrams show that level regularisation provides consistent reduction of the number of iterations needed to achieve the given precision. There are a few counterexamples, however, such as the env family of problems.

Figure 4.7 illustrates the scale-up properties of the algorithms in terms of the change in the solution time with the number of scenarios on the 4node problems. It shows that Benders' decomposition with the level regularisation scales well at some point overtaking the multicut methods.

The computational results given in the previous section are obtained using the relative stopping tolerance $\epsilon = 10^{-5}$ for the Benders decomposition with and without regularisation by the level method, i.e. the method terminated if $(f^* - f_*)/(|f_*| + 10^{-10}) \leq \epsilon$, where f_* and f^* are, respectively, lower and upper bounds on the value of the objective function. The stopping criteria in

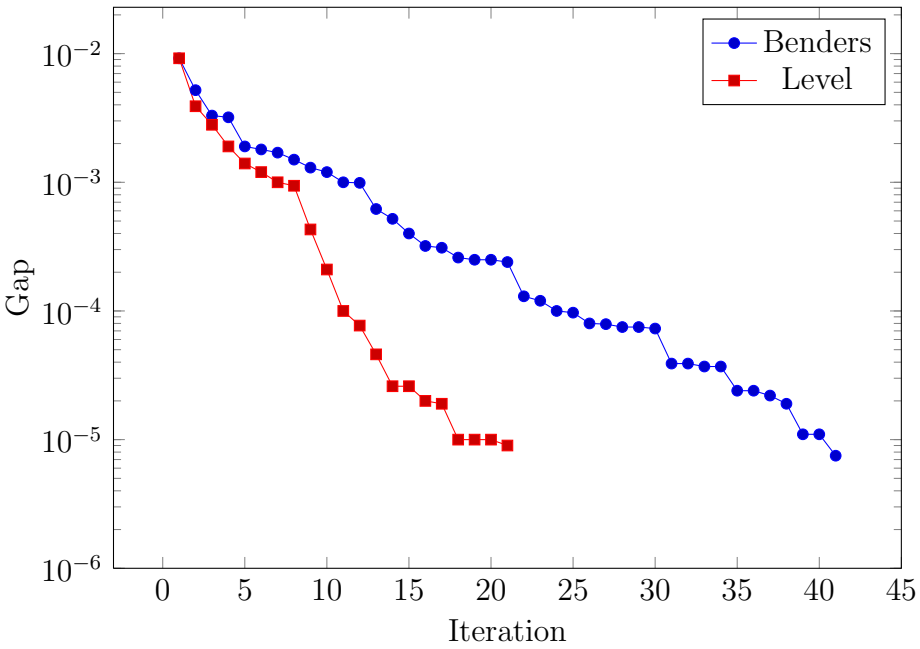


Figure 4.3: Gap between lower and upper bounds for storm-1000 problem

the trust region algorithm and regularised decomposition are different because these methods do not provide global lower bound. Therefore ϵ is set to a lower value of 10^{-6} with the following exceptions that are made to achieve the desirable precision:

- env-diss with 8232 scenarios: $\epsilon = 10^{-10}$ in RD,
- saphir: $\epsilon = 10^{-10}$ in RD and TR.

For CPLEX barrier optimiser the default complementarity tolerance is used as a stopping criterion.

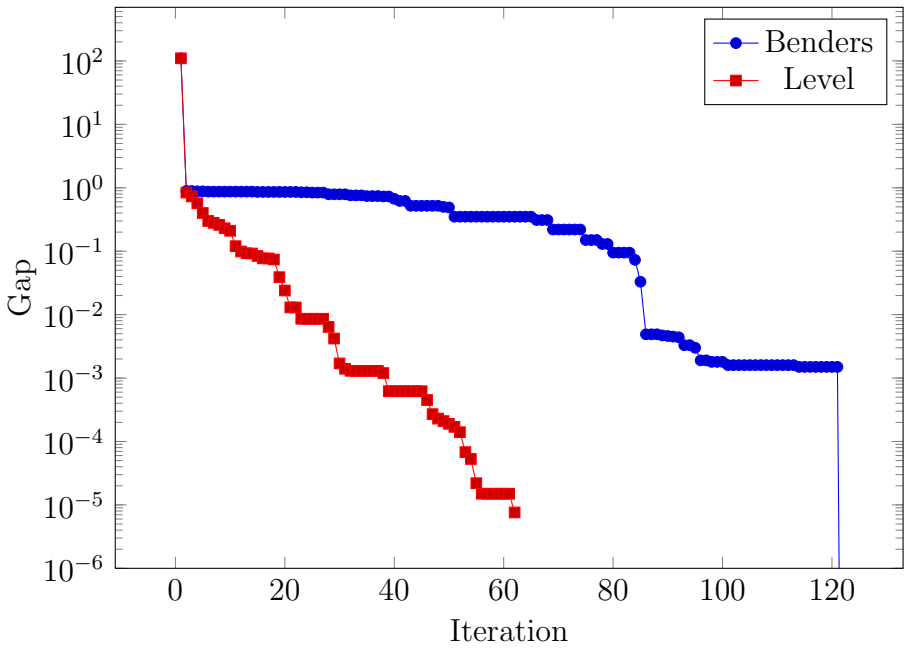


Figure 4.4: Gap between lower and upper bounds for 4node-32768 problem

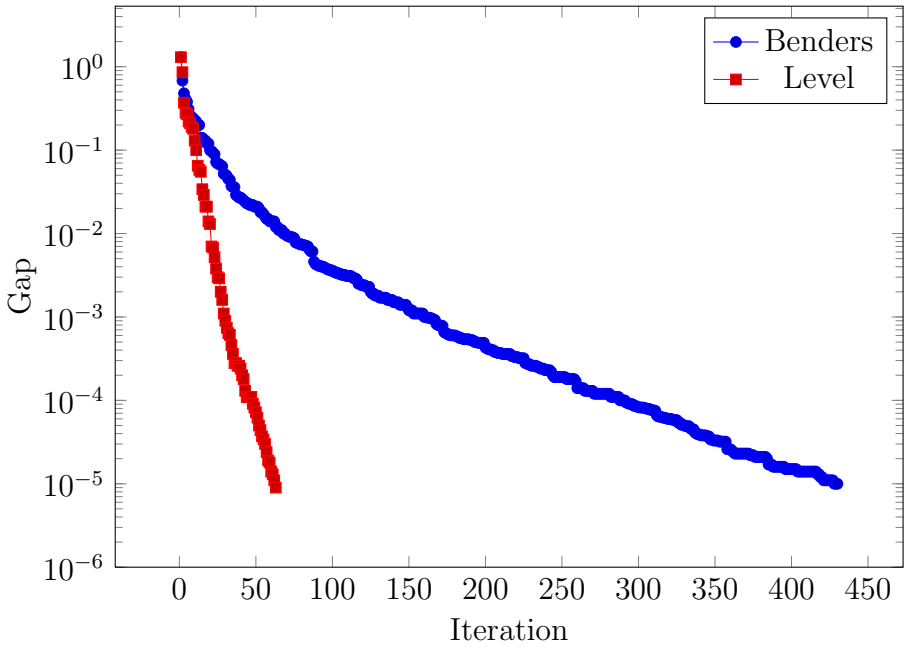


Figure 4.5: Gap between lower and upper bounds for rand1-10000 problem

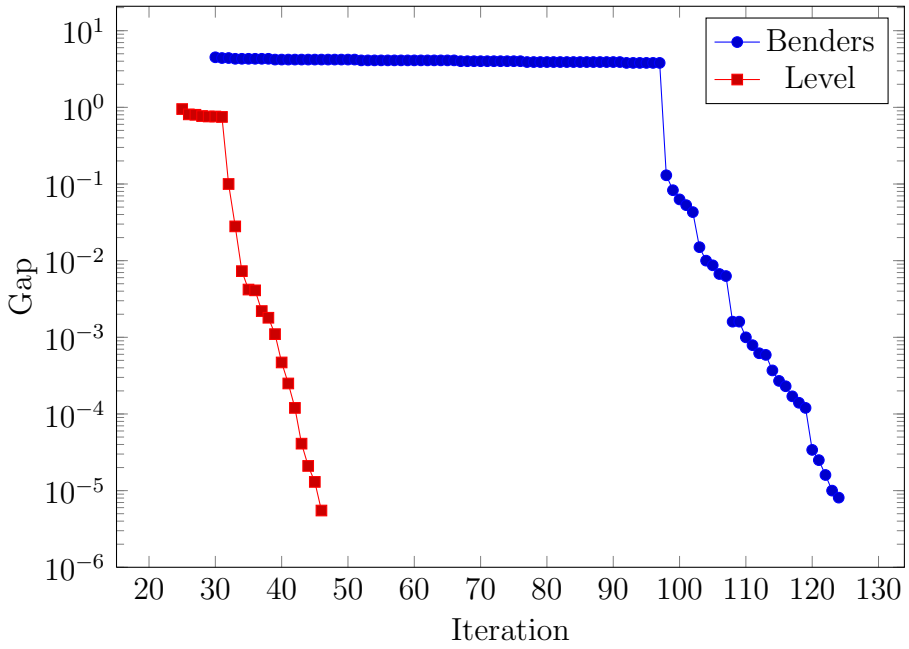


Figure 4.6: Gap between lower and upper bounds for saphir-1000 problem

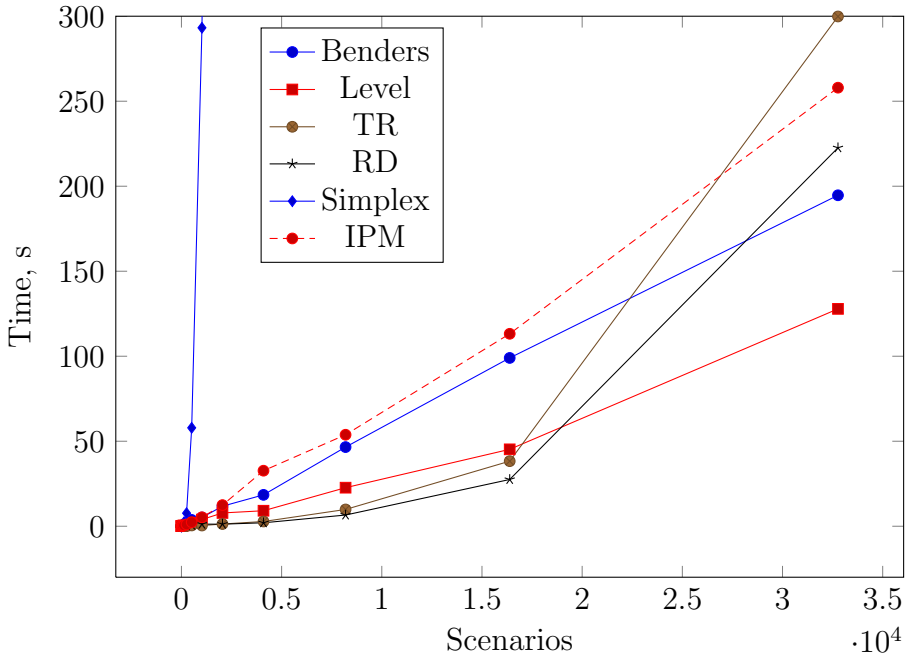


Figure 4.7: Time vs the number of scenarios on the 4node problems

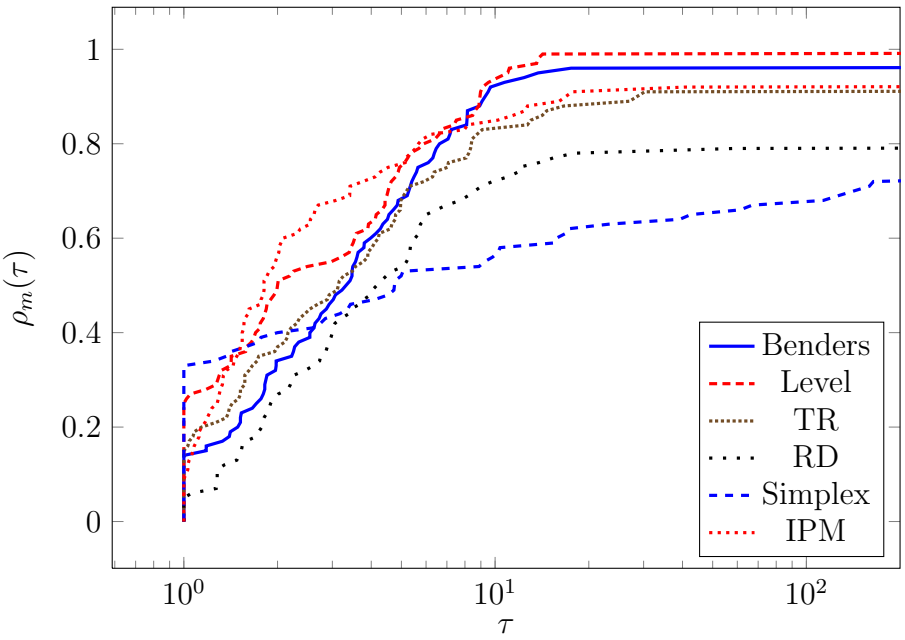


Figure 4.8: Performance profiles

4.3 Performance profiles

Finally we present the results in the form of performance profiles. The performance profile for a solver is defined by Dolan and Moré (2002) as the cumulative distribution function for a performance metric. We use the ratio of the solution time versus the best time as the performance metric. Let P and M be the set of problems and the set of solution methods respectively. We define by $t_{p,m}$ the time of solving problem $p \in P$ with method $m \in M$. For every pair (p, m) we compute performance ratio

$$r_{p,m} = \frac{t_{p,m}}{\min\{t_{p,m} | m \in M\}},$$

If method m failed to solve problem p the formula above is not defined. In this case we set $r_{p,m} := \infty$.

The cumulative distribution function for the performance ratio is defined as follows:

$$\rho_m(\tau) = \frac{|\{p \in P | r_{p,m} \leq \tau\}|}{|P|}$$

We calculated performance profile of each considered method on the whole set of test problems. These profiles are shown in Figure 4.8. The value of $\rho_m(\tau)$ gives the probability that method m solves a problem within a ratio τ of the best solver. For example according to Figure 4.8 the level method is the first in 25% of cases and solved 95% of the problems within a ratio 11 of the best time.

The notable advantages of performance profiles over other approaches to performance comparison are as follows. Firstly, they minimise the influence of a small subset of problems on the benchmarking process. Secondly, there is no need to discard solver failures. Thirdly, performance profiles provide a visualisation of large sets of test results as we have in our case. It should be noted, however, that we still investigated the failures and the cases of unusual performance. This resulted, in particular, in the adjustment of the values of ϵ, Δ_{hi} and ξ for the RD and TR methods and switching to a 64-bit platform with more RAM which is crucial for IPM.

As can be seen from Figure 4.8, Benders' decomposition with regularisation by the level method is robust, as it successfully solves the largest fraction of test problems, and efficient as it compares well with the other methods in terms of performance.

Chapter 5

Solution methods for two-stage stochastic integer programming

5.1 Two-stage stochastic integer programming problem

Two-stage stochastic programming problems with recourse comprise an important class of problems which has a wide range of practical applications (Wallace and Ziemba, 2005). A two-stage SP problem consists of a first-stage (here and now) decision, followed by a realisation of a random vector and a second-stage (recourse) decision.

The focus of this chapter is on problems where some or all of the decision variables are integer. This results in a stochastic integer programming problem that can be formulated as follows:

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} + \mathrm{E}[Q(\boldsymbol{x},\omega)] \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \in \mathbb{Z}_+^{r_1} \times \mathbb{R}_+^{n_1-r_1}, \end{array} \tag{5.1}$$

where the n_1-dimensional vector $\boldsymbol{x}$ represents a first-stage decision partitioned into r_1 integer and $n_1 - r_1$ continuous components, A is a fixed $m_1 \times n_1$ matrix, $\boldsymbol{b} \in \mathbb{R}^{m_1}$ and $\boldsymbol{c} \in \mathbb{R}^{n_1}$ are fixed vectors and $Q(\boldsymbol{x},\omega)$ is the value function of the recourse problem

$$\begin{array}{ll} \text{minimize} & \boldsymbol{q}(\omega)^T\boldsymbol{y} \\ \text{subject to} & T(\omega)\boldsymbol{x} + W(\omega)\boldsymbol{y} = \boldsymbol{h}(\omega), \\ & \boldsymbol{y} \in \mathbb{Z}_+^{r_2} \times \mathbb{R}_+^{n_2-r_2}. \end{array} \tag{5.2}$$

In (5.2) the n_2-dimensional vector $\boldsymbol{y}$ represents a second-stage decision partitioned into r_2 integer and $n_2 - r_2$ continuous components and $\omega \in \Omega$ represents a random event. For a given realisation ω, $T(\omega)$ is a fixed $m_2 \times n_1$ matrix, $W(\omega)$ is a fixed $m_2 \times n_2$ matrix $\boldsymbol{h}(\omega) \in \mathbb{R}^{m_2}$ and $\boldsymbol{q}(\omega) \in \mathbb{R}^{n_2}$ are fixed vectors. We only consider problems with the vector of random parameters (i.e. random components of T, W, $\boldsymbol{h}$ and $\boldsymbol{q}$) having a discrete finite distribution.

Let $f(x)$ denote the objective function in (5.1):

$$f(x) = \boldsymbol{c}^T\boldsymbol{x} + \tilde{Q}(\boldsymbol{x}),$$

where $\tilde{Q}(\boldsymbol{x}) = \mathrm{E}[Q(\boldsymbol{x}, \omega)]$ denotes the expected recourse function.

In this chapter we consider several algorithms for solving stochastic integer programming (SIP) problems. These methods have different additional assumptions such as presence of binary variables in the first stage. These are discussed later in the sections designated to each algorithm.

If the vector of random parameters has S outcomes (scenarios), that is $|\Omega| = S$, with probability of ith outcome p_i then a SIP problem can be formulated as a large-scale MIP problem with a special structure

$$\begin{array}{rlllllll}
\text{minimize} & \boldsymbol{c^T x} & + & p_1 \boldsymbol{q_1}^T \boldsymbol{y_1} & + \ \ldots \ + & p_S \boldsymbol{q_S}^T \boldsymbol{y_S} & & \\
\text{subject to} & A\boldsymbol{x} & & & & & = & \boldsymbol{b}, \\
& T_1\boldsymbol{x} & + & W_1\boldsymbol{y_1} & & & = & \boldsymbol{h_1}, \\
& \vdots & & & \ddots & & & \vdots \\
& T_S\boldsymbol{x} & + & & & W_S\boldsymbol{y_S} & = & \boldsymbol{h_S}, \\
& \boldsymbol{x} \in \mathbb{Z}_+^{r_1} \times \mathbb{R}_+^{n_1 - r_1}, & & & & & & \\
& \boldsymbol{y_s} \in \mathbb{Z}_+^{r_2} \times \mathbb{R}_+^{n_2 - r_2}, \quad s = 1, ..., S. & & & & & &
\end{array}$$

A general-purpose MIP solver can be applied to the solution of this problem.

Stochastic integer programming problems are well known to be computationally challenging. According to the study of Dyer and Stougie (2006) the complexity of SIP problems is primarily determined by the computation of a multidimensional integral. This occurs during the evaluation of the expected recourse function. The second difficulty which is less significant from the theoretical point of view but nevertheless considerable in practice is caused by the integrality restrictions. Unlike the continuous case, the expected recourse function in SIP is non-convex in general and can be discontinuous. The discontinuities appear as a result of the value function of an integer programming problem being lower semi-continuous as shown by Blair and Jeroslow (1982). Even if the integrality restriction is limited to the first stage only and the expected recourse function

is convex applying decomposition or optimising the DEP requires solving MIP problems which are NP-hard (Garey and Johnson, 1979).

Despite the aforementioned difficulties there is a practical need in solving these problems in many application areas such as

- airlift operations scheduling (Midler and Wollmer, 1969),
- batch type chemical plant design (Subrahmanyam et al., 1994),
- telecommunication network planning (Sen et al., 1994),
- cargo transportation scheduling (Mulvey and Ruszczyǹski, 1995),
- semiconductor tool purchase planning (Barahona et al., 2001).

Several more recent examples can be found in Section 5.5 where they are used as benchmark problems.

In this work we bring together the advantages of heuristics from the area of deterministic MIP that lead to feasible solutions and therefore upper bounds quickly with exact solution methods for stochastic integer programming. Such algorithms that combine heuristics and exact methods have been successful in the deterministic context (Ahuja et al., 2002; Danna et al., 2005; Fischetti and Lodi, 2003).

First we introduce the neighbourhood structures into the space of the first-stage decision variables. This allows us to apply heuristics based on variable neighbourhood search (Mladenović and Hansen, 1997). In particular, we use variable neighbourhood decomposition search (Hansen et al., 2001).

We apply variable neighbourhood decomposition search (VNDS) to the solution of stochastic integer programming problems (Lazić, Mitra, Mladenović, and Zverovich, 2010) and use the deterministic equivalent approach to solve the intermediate SIP and SP problems. We call this method VNDS-SIP. For the most difficult instances that were not solved to optimality this heuristic is able to find much better solutions than the DEP solver within the same time limit. Additional experiments performed in the current study show that despite being less efficient for proving optimality, VNDS-SIP leads to good solutions (often the best DEP solutions) quickly.

VNDS-SIP is applicable to SIP problems containing binary variables in the first stage. The second stage can be either continuous or mixed integer. The method exploits the information from the solution of the linear relaxation of the first-stage subproblem to build a sequence of SP and SIP problems which can be

solved using traditional exact methods, for example the L-shaped method (Van Slyke and Wets, 1969) for SP considered in Chapter 4. The intermediate SIP problems are usually easier to solve than the original problem because some of the binary variables are fixed. Also it is not necessary to solve these problems to optimality.

Our approach enhances the original VNDS-SIP heuristic by using the integer L-shaped method of Laporte and Louveaux (1993) to solve the SIP problems constructed during the optimisation process instead of solving their deterministic equivalents with a MIP solver. To distinguish between these two methods we call the first one VNDS-DEP and the second VNDS-ILS. We perform a set of numerical experiments comparing their performance and include the integer L-shaped algorithm itself into consideration.

The structure of this chapter is as follows. In Section 5.2 we give a review of several solution methods for two-stage stochastic integer programming. In particular we discuss the integer L-shaped method of Laporte and Louveaux (1993), an enumeration method of Schultz et al. (1998), the decomposition based branch and bound algorithm of Ahmed and Garcia (2004) and the disjunctive decomposition algorithm of Sen and Higle (2005). The implementation of the integer L-shaped method within the existing branch-and-cut framework is described in Section 5.3. In Section 5.4 we describe two variants of a heuristic method based on variable neighbourhood decomposition search applied to stochastic integer programming. In Section 5.5 we present a computational study comparing the two variants of the heuristic method to the solution of the deterministic equivalent problems with a state-of-the-art mixed integer programming solver and to the integer L-shaped method on a range of benchmark problems from the SIPLIB collection. In this section we also give the performance profiles which provide a summary of the results in a graphical form.

5.2 Review of alternative solution methods for two-stage SIP

The integer L-shaped method

In one of the early papers on solution algorithms for stochastic integer programming Wollmer (1980) observed that both continuous stochastic programming problems and 0-1 MIP problems can be solved with Benders' decomposition.

The author proposed an algorithm which combines these two approaches within an implicit enumeration scheme for solving two-stage SIP problems with binary first-stage and continuous second-stage variables. He successfully applied this algorithm to a network investment problem.

Extending the approach of Wollmer, Laporte and Louveaux (1993) described the integer L-shaped method for solving two-stage SIP problems with complete recourse. Their method is similar to the L-shaped method of Van Slyke and Wets (1969) for stochastic linear programming.

The method operates on the current problem based on the first-stage subproblem with added feasibility and optimality cuts. The current problem at iteration k can be defined as follows:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T\boldsymbol{x} + \theta \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b}, \\ & D_k\boldsymbol{x} \geq \boldsymbol{d_k}, \\ & E_k\boldsymbol{x} + \theta \geq \boldsymbol{e_k}, \\ & \boldsymbol{x} \in \mathbb{R}^{n_1}_+,\ \theta \in \mathbb{R}, \end{aligned} \tag{5.3}$$

where $D_k\boldsymbol{x} \geq \boldsymbol{d_k}$ are the feasibility cuts and $E_k\boldsymbol{x} + \theta \geq \boldsymbol{e_k}$ are the optimality cuts.

The main difference of this algorithm from its continuous counterpart is branching on the first-stage binary variables which results in a branch and cut procedure. As in the L-shaped method the second-stage subproblems are solved only during the evaluation of the expected recourse function when computing the optimality cuts. In this way both decomposition of stages and decomposition of scenarios are achieved.

The integer L-shaped method is applicable to a wide range of SIP problems with binary first stage, continuous second stage and complete fixed recourse. A restricted class of problems with discrete second-stage variables is supported. Additional assumptions are as follows:

- Expected recourse function is computable given the first-stage solution vector. This holds in particular when the random parameters have finite discrete distribution.

- Expected recourse function is bounded from below.

Laporte and Louveaux stated that valid feasibility cuts can be taken from the deterministic context and derive different kinds of optimality cuts. One type of optimality cuts comes from the original L-shaped method. Also the cuts that

provide a valid set of optimality cuts in the presence of discrete second-stage variables are derived.

Finite convergence was proven for the integer L-shaped method in the case of pure binary first stage.

Another extension of the classical L-shaped method was proposed by Carøe and Tind (1998) based on the generalised Benders' decomposition and general duality theory. They showed that the integer L-shaped method of Laporte and Louveaux is a special case of their proposed algorithm.

The method of Carøe and Tind applies to two-stage SIP problems with integer second-stage variables. First stage variables can be continuous or discrete. The random parameters should have discrete distributions with finite support and the continuous relaxation of the second-stage subproblem should be dual feasible.

Unlike some other methods for SIP which require complete or relatively complete recourse generalised L-shaped decomposition can handle infeasible second-stage subproblems.

The underlying concept of the algorithm is the same as the one of the L-shaped method, that is to use dual information from the second-stage subproblems to approximate the expected recourse function. However in case of discrete second-stage variables nonlinear dual price functions have to be considered. It results in nonlinear feasibility and optimality cuts in the master problems.

Carøe and Tind considered two approaches to solving the second-stage subproblems:

1. cutting plane methods,

2. branch and bound.

In the first case the usage of Gomory cutting planes is applied. These cuts can be transformed by introducing auxiliary variables, representing round-up operations with integrality restrictions and incorporated into the current problem as cuts. However this results in a mixed integer problem with a potentially large number of discrete variables making it computationally unattractive.

In the second case, when the branch and bound technique is applied to solve the second-stage subproblems, the master problem becomes a disjunctive programming problem. Therefore this approach is also computationally difficult.

For both cases the finite convergence is proved, provided that a master problem can be solved with a finite method.

Solving SIP problems by enumeration

Schultz et al. (1998) describe an algorithm for solving two-stage SIP problems with integer variables in the second stage. This algorithm is applicable to problems with

- continuous first stage,
- complete integer recourse,
- finite discrete distribution of random parameters,
- fixed rational second-stage constraint matrix,
- dual feasible relaxation of the second-stage problem.

First Schultz et al. prove that at least some optimal first-stage solutions belong to a countable set. It follows from the study of the structural properties of the expected integer recourse function. Moreover, they show that under mild conditions this set can be restricted to a finite one. This result is obtained by considering the intersections of the set of solution candidates and level sets constructed by solving the continuous relaxations of a SIP problem.

Based on the previous result an enumeration scheme for finding an optimal solution is proposed.

Addressing the difficulty of solving multiple second-stage MIP problems a technique based on the Gröbner basis methods from computational algebra is applied. However using this technique is not inherent in the algorithm and other MIP solution methods such as branch and bound can be employed instead.

While computing the Gröbner basis is computationally expensive and therefore impractical for a single MIP problem it can be efficient for solving a family of problems which differ only in the right-hand side. Though using this technique is not essential it justifies the assumption that multiple evaluations of the expected recourse function are possible. This is taken for granted in some other solution algorithms for SIP such as the integer L-shaped method.

Schultz et al. also report the results of comparing their enumeration method with a commercial MIP solver. They show that the former is able to solve more problems to optimality within the specified limits.

The decomposition based branch and bound algorithm

Ahmed and Garcia (2004) address the problem of dynamic capacity acquisition and assignment often arising in supply chain applications. They formulate it as

a two-stage multiperiod SIP problem with mixed-integer first stage, pure binary second stage and a discrete distribution of random parameters. Special care is taken to ensure that the formulation has the complete recourse property.

The authors observe that most existing branch and bound approaches to the solution of stochastic integer programming problems, such as those by Laporte and Louveaux (1993), Carøe and Tind (1998) and others, guarantee finite termination only in the case of pure integer first stage. Based on the work of Ahmed et al. (2004) a decomposition based branch and bound algorithm (DBB) for solving dynamic capacity acquisition and assignment problems is proposed. This algorithm guarantees finite termination even if the first stage contains continuous variables.

The main idea of the DBB algorithm is based on the observation that existing branch and bound methods for SIP partition the first-stage variable space into hyperrectangles while the discontinuities may not be orthogonal to the coordinate axes. In the case of continuous first-stage variables, infinite partitioning of the space may be required. The discontinuities are caused by the fact that the second-stage value function in SIP is not necessarily convex but only lower semi-continuous (Blair and Jeroslow, 1982). To address this issue Ahmed et al. propose a transformation that makes the discontinuities orthogonal to the axes. It is applied to the general two-stage SIP with mixed-integer first stage and pure integer second stage. The problem structure imposed by this transformation is exploited to ensure finiteness of the branch and bound algorithm.

To make the DBB algorithm applicable, the problem should have

- finite discrete distribution of random parameters,
- pure integer second-stage variables,
- non-empty and compact first-stage feasibility set,
- relatively complete fixed recourse,
- integral second-stage constraint matrix for each scenario.

The last requirement can be satisfied by scaling if the matrix elements are rational. Also there exist extensions to the algorithm for the case when the requirements of pure integer second stage and fixed recourse do not hold.

Finally Ahmed and Garcia give computational results of comparing their decomposition based algorithm with the application of a commercial MIP solver (CPLEX) to the solution of deterministic equivalent problems. They apply both

methods to randomly generated instances of the dynamic capacity acquisition and assignment problem. The results suggest that the DBB algorithm compares favorably to the deterministic equivalent approach and the decomposition based algorithm is much less sensitive to the increase in the number of scenarios.

The disjunctive decomposition algorithm

Sen and Higle (2005) consider two-stage SIP problems with focus on binary variables in the second stage and propose a decomposition algorithm for such problems. They call it disjunctive decomposition (D^2) due to the fact that it is to a large extent based on the theory of disjunctive programming (Blair and Jeroslow, 1978; Balas, 1979; Sherali and Shetty, 1980). The key feature of this algorithm is iterative convexification of the second-stage subproblems. Though these convexifications depend on scenario they may have common structure that is exploited in the algorithm.

The requirements imposed on the problems are as follows:

- finite discrete distribution of random parameters,
- pure binary first-stage variables,
- mixed binary second-stage variables,
- compact feasibility set of the first-stage relaxation,
- relatively complete fixed recourse if the first-stage integrality restrictions are not taken into account.

One of the original requirements is that the first-stage feasibility set has to be contained in a unit hypercube; this can be achieved by the appropriate scaling.

Based on the theory of valid inequalities from disjunctive programming Sen and Higle formulate and prove a common cut coefficients (C^3) theorem. This theorem provides the connection between the valid inequalities for the second-stage subproblems under different scenarios given that the SIP problem has fixed recourse. It is related to the earlier work of Carøe (1998) where similar results were obtained in the context of deterministic equivalents of SIP problems.

From the C^3 theorem it follows that the lower bound approximations of the second-stage value functions for different scenarios and first-stage solution vectors differ only in the right-hand sides. However these approximations are not convex in general and Sen and Higle develop convexifications to make a decomposition algorithm such as the L-shaped method of Van Slyke and Wets (1969) applicable.

On the basis of the results of the common cut coefficients theorem the disjunctive decomposition algorithm is developed. It consists of iterative invocation of two steps:

1. Solution of the master problem.
2. Refinement of the convex approximations of the second-stage subproblems and update to the representation of the recourse function.

5.3 A computational framework for the integer L-shaped method

In this section we describe an implementation of the integer L-shaped method within an existing branch and cut framework for mixed integer programming. MIP infrastructure enables reuse of existing heuristics and cuts for the solution of the current problem (5.3). Unlike Laporte and Louveaux (1993) we do not simulate the algorithm by restarting branch and bound after a new iterate is found. Instead the branch and bound process is continued with special care taken to ensure proper fathoming rules which are discussed below. We use the implementation of the integer L-shaped method to improve the performance of the VNDS-SIP heuristic and also compare it to other methods in a computational study in Section 5.5. The algorithm shows good performance on the class of problems where it is applicable.

Unlike usual fathoming rules for MIP, the strategy used in the integer L-shaped method is such that the node is not always fathomed when the integrality restrictions hold. This is due to the addition of the optimality cuts which can cut off an optimal solution of the current problem. We have taken this strategy into account when implementing the algorithm based on existing branch and cut infrastructure.

In our implementation we use continuous L-shaped optimality cuts. Since these do not provide a valid finite set of optimality cuts in general (Laporte and Louveaux, 1993), we consider only the case of continuous second stage.

These aggregated optimality cuts are defined as follows (Birge and Louveaux, 1997):

$$\left(\sum_{s=1}^{S} p_s(\boldsymbol{\pi}_s^*)^T T_s\right) x + \theta \geq \sum_{s=1}^{S} p_s(\boldsymbol{\pi}_s^*)^T h_s,$$

where $\boldsymbol{\pi}_s^*$ is the vector of simplex multipliers associated with an optimal solution of the (continuous) recourse problem corresponding to scenario s:

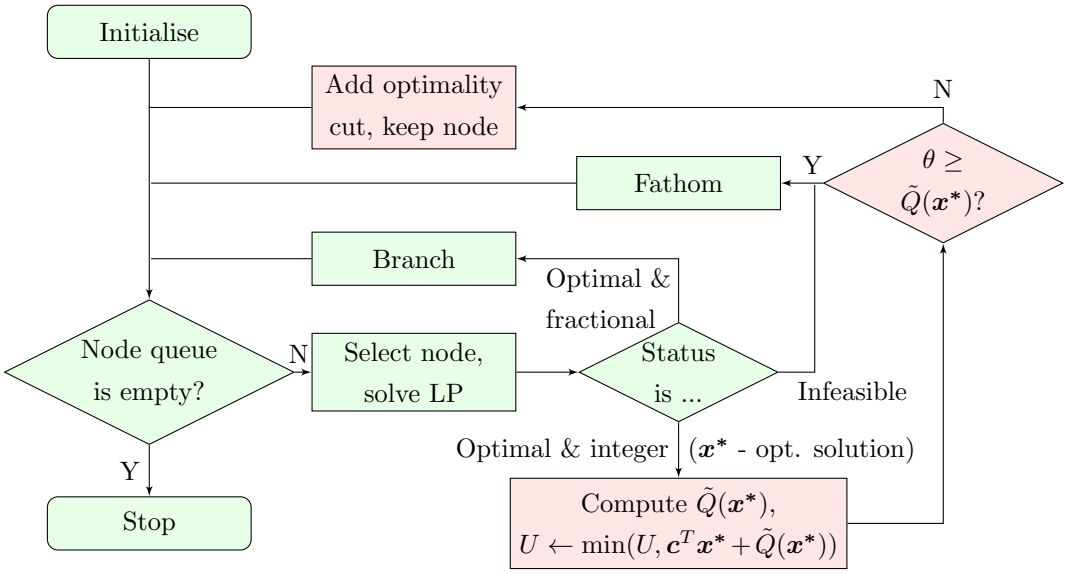


Figure 5.1: Flowchart of the integer L-shaped method within a branch and cut framework

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{q_s}^T \boldsymbol{y} \\ \text{subject to} \quad & W\boldsymbol{y} = \boldsymbol{h_s} - T_s\boldsymbol{x}^*, \\ & \boldsymbol{y} \in \mathbb{R}^{n_2}_+, \end{aligned}$$

where $\boldsymbol{x}^*$ is an optimal solution of the current problem.

We also implemented the L-shaped feasibility cuts that are defined as follows:

$$\left((\boldsymbol{\sigma}_s^*)^T T_s\right) x \geq (\boldsymbol{\sigma}_s^*)^T h_s,$$

where $\boldsymbol{\sigma}_s^*$ is the vector of simplex multipliers associated with an optimal solution of the following problem for scenario s where the recourse problem is infeasible:

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}^T(\boldsymbol{u} + \boldsymbol{v}) \\ \text{subject to} \quad & W\boldsymbol{y} + I(\boldsymbol{u} - \boldsymbol{v}) = \boldsymbol{h_s} - T_s\boldsymbol{x}^*, \\ & \boldsymbol{y} \in \mathbb{R}^{n_2}_+, \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{m_2}_+, \end{aligned}$$

where $\mathbf{1} = (1, 1, \ldots, 1)$ and I is the identity matrix of an appropriate size.

A simplified flowchart of the method is shown in Figure 5.1. In this flowchart U denotes the best upper bound on the value of the objective function initially set to ∞.

To implement the integer L-shaped method we use the branch-and-cut infrastructure of the CPLEX MIP solver (IBM Corp., 2009b) extended through the following callbacks:

- Incumbent callback is called when an integer solution has been found but before this solution replaces the incumbent. Here the recourse problem is

solved for each scenario, the optimality or feasibility cut is constructed and stored.

- Cut callback is called for every node. Here the stored cut if any is added to the problem.
- Branch callback which ensures that the node is not fathomed if there is a pending cut which has not been added to the problem yet.

5.4 Application of variable neighbourhood decomposition search to SIP

Variable neighbourhood search (Hansen and Mladenović, 2001) is a metaheuristic which combines local search with a systematic exploration of the neighbourhoods of the incumbent solution to find better solutions. It has been successfully applied in various areas (Brimberg and Mladenović, 1996; Aouchiche et al., 2006; Dražić et al., 2008).

Variable neighbourhood decomposition search (VNDS) is a two-level variant of the variable neighbourhood search based on decomposition of the problem (Hansen et al., 2001). It was first applied to the 0-1 mixed integer programming by Lazić., Hanafi, Mladenović, and Urošević (2010) and further extended for 0-1 MIP feasibility by Lazić, Hanafi, and Mladenović (2010).

We apply VNDS to the solution of SIP problems and propose a heuristic method for solving this class of problems. We investigate two variants of the VNDS method for stochastic integer programming; these variants use different underlying SIP algorithms for optimising the intermediate problems constructed during the solution process.

Consider the case of mixed binary first stage ($\boldsymbol{x} \in \{0,1\}^{r_1} \times \mathbb{R}_+^{n_1-r_1}$).

To describe the method we first define the distance between two integer feasible first-stage solutions. Let $\boldsymbol{x'}$ and $\boldsymbol{x''}$ be two arbitrary integer feasible solutions of the first-stage subproblem in (5.1). Then the distance between $\boldsymbol{x'}$ and $\boldsymbol{x''}$ is defined as

$$\Delta(\boldsymbol{x'}, \boldsymbol{x''}) = \sum_{j=1}^{r_1} |x'_j - x''_j|. \tag{5.4}$$

If $J \subseteq \{1, 2, \ldots, r_1\}$ then the partial distance between $\boldsymbol{x'}$ and $\boldsymbol{x''}$, relative to J, is defined as

$$\Delta(J, \boldsymbol{x'}, \boldsymbol{x''}) = \sum_{j \in J} |x'_j - x''_j|, \tag{5.5}$$

or, equivalently,

$$\Delta(J, \boldsymbol{x'}, \boldsymbol{x''}) = \sum_{j \in J} x'_j(1 - x''_j) + x''_j(1 - x'_j). \tag{5.6}$$

Based on the above definition of the distance we introduce the neighbourhood structures in the solution space. Let $\mathcal{N}_k, 1 \leq k_{min} \leq k \leq k_{max} \leq n_1$, denote the kth neighbourhood of the solution $\boldsymbol{x'}$, $\boldsymbol{x'} \in X$. The neighbourhood is defined as a set of all solutions lying within the distance k of $\boldsymbol{x'}$:

$$\mathcal{N}_k(\boldsymbol{x'}) = \{\boldsymbol{x''} \in X | \Delta(\boldsymbol{x'}, \boldsymbol{x''}) \leq k\}, \tag{5.7}$$

where X denotes the feasibility region of the first-stage decision variables

$$X = \{\boldsymbol{x} | A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \in \{0,1\}^{r_1} \times \mathbb{R}_+^{n_1 - r_1}\}.$$

Having defined the neighbourhood structures (5.7) it is now possible to use various heuristics based on the variable neighbourhood search principle (Mladenović and Hansen, 1997). In particular we apply the variable neighbourhood decomposition search (Hansen et al., 2001) to the solution of a two-stage SIP problem (5.1)-(5.2).

Numerical experiments have shown the effectiveness of VNDS in finding high quality solutions of difficult MIP test problems. This is the rationale for choosing VNDS among other heuristic methods such as local branching (LB) of Fischetti and Lodi (2003) and relaxation induced neighbourhood search (RINS) of Danna et al. (2005) as well as other variants of VNS metaheuristic such as variable neighbourhood search branching (VNSB) of Hansen et al. (2006). Lazić., Hanafi, Mladenović, and Urošević (2010) performed an extensive comparison of these methods showing favourable performance of VNDS and managed to improve the best known solutions for 8 instances of well-known MIP benchmark problems using a combination of VNDS and an exact solution method.

An important feature of the algorithm is that it operates on the first-stage variables only which means that the distance constraints which are added do not destroy scenario independence. Therefore not only the deterministic equivalent approach can be used to solve the intermediate problems but also decomposition methods. We use this feature later to devise two variants of the heuristic based on different underlying SIP solution methods.

The pseudo code for the VNDS-SIP method is given in Algorithm 11. Note that at each iteration the intermediate SIP problem is not necessarily solved to optimality. In our implementation a time limit is imposed on the solution of each

Algorithm 11: Pseudo-code of variable neighbourhood decomposition search algorithm for two-stage SIP

Given: SIP problem instance P, integer feasible solution $\boldsymbol{x}^*$, stopping tolerance ε
$L \leftarrow -\infty, \quad U \leftarrow f(\boldsymbol{x^*})$
while time limit is not reached **do**
  Solve the LP relaxation of P to obtain the solution $\bar{\boldsymbol{x}}$
  $L \leftarrow f(\bar{\boldsymbol{x}})$
  if $\bar{\boldsymbol{x}} \in \{0,1\}^{r_1} \times \mathbb{R}_+^{n_1 - r_1}$ **then**
    $\boldsymbol{x}^* \leftarrow \bar{\boldsymbol{x}}, \quad U \leftarrow L$
    stop
  end if
  $\Delta_j = |x_j^* - \bar{x}_j|, \quad j = 1, \ldots, r_1$
  Index x_j so that $\Delta_j \leq \Delta_{j+1}, \quad j = 1, \ldots, r_1 - 1$
  $k \leftarrow r_1$
  while time limit is not reached **and** $k \geq 0$ **do**
    $J_k \leftarrow \{1, \ldots, k\}$
    Add constraint $\Delta(J_k, \boldsymbol{x}^*, \boldsymbol{x}) = 0$ to P
    Solve the problem P to obtain the solution $\boldsymbol{x'}$
    if optimal solution found **or** problem is infeasible **then**
      Replace the last added constraint with $\Delta(J_k, \boldsymbol{x}^*, \boldsymbol{x}) \geq 1$
    else
      Delete the last added constraint
    end if
    if $f(\boldsymbol{x'}) < f(\boldsymbol{x^*})$ **then**
      $\boldsymbol{x}^* \leftarrow \boldsymbol{x'}, \quad U \leftarrow f(\boldsymbol{x'})$
      if $|U - L| \leq \varepsilon |U|$ **then**
        stop
      end if
    end if
    $k \leftarrow k - 1$
  end while
end while

of these problems. L and U denote the lower and the upper bound on the value of the objective function respectively. ε denotes the relative stopping tolerance. The starting point x^* can be obtained by solving the problem using the underlying SIP method until the first integer feasible solution is found. This also provides the initial upper bound.

First we solve the continuous relaxation of the SIP problem P. Unlike the intermediate SIP problems the relaxations are solved to optimality. This provides the lower bound on the value of the objective function and also gives the optimal solution $\bar{\boldsymbol{x}}$ which is used to iteratively choose the subsets of variables from the incumbent integer solution $\boldsymbol{x}^*$. The variables are selected according to the distance between their values in the incumbent solution and in the solution of the relaxed problem. These variables are fixed resulting in intermediate problems with smaller number of binaries and therefore often easier to solve.

It is possible to use any SIP solution algorithm for optimising the intermediate problems. We consider two variants of the VNDS heuristic for SIP by using different underlying SIP algorithms. In the first variant, we solve the DEP problems using a general purpose MIP solver. In the second variant we use the integer L-shaped method described in Section 5.2. To distinguish between the two we call the first variant VNDS-DEP, while the second is called VNDS-ILS.

It is easy to see that the constraint $\Delta(J_k, \boldsymbol{x}^*, \boldsymbol{x}) = 0$ in line 15 and the constraint $\Delta(J_k, \boldsymbol{x}^*, \boldsymbol{x}) \geq 1$ in line 18 of Algorithm 11 are linear taking into account that $\boldsymbol{x}^*$ is fixed and using the representation (5.6).

The constraint $\Delta(J_k, \boldsymbol{x}^*, \boldsymbol{x}) = 0$ effectively fixes a subset of the first-stage variables corresponding to J_k. This results in a problem with fewer binary variables which can be usually solved much faster then the original one. In the best case when $J_k = \{1, 2, \ldots, r_1\}$ and there are no second-stage integer variables the resulting problem is a continuous SP and can be solved efficiently using one of the methods discussed in Chapter 4. The number of the first-stage variables to be fixed is changed systematically. This gives a VNDS scheme for two-stage SIP problems.

We observe that VNDS for 0-1 MIP problems of Lazić., Hanafi, Mladenović, and Urošević (2010) can be considered a special case of VNDS-SIP for problems with an empty second stage ($n_2 = 0$).

In this chapter we limit our consideration to SIP problems with binary and possibly continuous variables in the first stage. However the method can be easily extended to make it applicable to problems with general integer variables in the first stage. This can be done by defining the distance between the integer

solutions appropriately.

5.5 Numerical study

We implemented VNDS-DEP, VNDS-ILS and the integer L-shaped method in FortSP (Ellison et al., 2010), a solver system for stochastic programming. The FortSP system already had the ability to construct and solve the deterministic equivalent problems.

The tests are performed on a 64-bit Linux machine with Intel CORE i5 2.4 GHz CPU and 6 GiB of RAM. Deterministic equivalents as well as LP and MIP subproblems in decomposition algorithms are solved with CPLEX 12.2. The relative MIP-gap tolerance of 0.01% and the time limit of 1800 seconds are used in all the methods. If the reported solution time is less than the time limit then the problem has been solved to optimality (up to the given tolerance). Otherwise we report the objective value of the best integer feasible solution found.

The time limit of 150 seconds is used when solving the intermediate SIP problems. An initial integer feasible solution is obtained by solving the SIP problem with the underlying method (DEP or integer L-shaped) until the first such solution is found.

Since parallelisation of the algorithms is out of scope of the current study the thread limit is set to 1 in CPLEX.

Problem set

The benchmark problems that we use in our experiments are listed in Table 5.1. They are available in SIPLIB, a stochastic integer programming test problem library (Ahmed, 2004) in the SMPS format (Birge et al., 1987).

The problem type is given using notation $a/b/c$ where

- a characterises the first-stage variables and is either C (continuous), B (binary) or M (mixed);
- b characterises the second-stage variables and is either C, B or M;
- c specifies the distribution type and is either D (discrete) or C (continuous).

This notation was introduced by Laporte and Louveaux (1993).

The application areas of the test problems are as follows:

- SIZES: product substitution applications,

Table 5.1: SIP problems

Name	Instances	Type	References
DCAP	12	M/B/D	Ahmed and Garcia (2004)
			Ahmed et al. (2004)
SIZES	3	M/M/D	Jorjani et al. (1999)
SSLP	10	B/M/D	Ntaimo and Sen (2005)

- DCAP: dynamic capacity acquisition and assignment arising in supply chain applications,
- SSLP: stochastic server location.

Table 5.2: Dimensions of SIP test problems

		Stage 1		DEP			
Name	Scen	Cols	Ints	Rows	Cols	Nonzeros	Ints
dcap233	200	12	6	3006	5412	11412	5406
	300	12	6	4506	8112	17112	8106
	500	12	6	7506	13512	28512	13506
dcap243	200	12	6	3606	7212	14412	7206
	300	12	6	5406	10812	21612	10806
	500	12	6	9006	18012	36012	18006
dcap332	200	12	6	2406	4812	10212	4806
	300	12	6	3606	7212	15312	7206
	500	12	6	6006	12012	25512	12006
dcap342	200	12	6	2806	6412	13012	6406
	300	12	6	4206	9612	19512	9606
	500	12	6	7006	16012	32512	16006
sizes	3	75	10	124	300	795	40
	5	75	10	186	450	1225	60
	10	75	10	341	825	2300	110
sslp-5-25	50	5	5	1501	6505	12805	6255
	100	5	5	3001	13005	25605	12505

Dimensions of test problems (continued)

		Stage 1		DEP			
Name	Scen	Cols	Ints	Rows	Cols	Nonzeros	Ints
	50	10	10	3001	25510	50460	25010
	100	10	10	6001	51010	100910	50010
sslp-10-50	500	10	10	30001	255010	504510	250010
	1000	10	10	60001	510010	1009010	500010
	2000	10	10	120001	1020010	2018010	1000010
	5	15	15	301	3465	6835	3390
sslp-15-45	10	15	15	601	6915	13655	6765
	15	15	15	901	10365	20475	10140

Results and discussion

In this section we use the following abbreviations to refer to the methods:

- DEP: solution of the deterministic equivalent problems with CPLEX,
- ILS: the integer L-shaped method,
- VNDS-DEP: VNDS heuristic for SIP with intermediate SIP problems solved using the DEP approach,
- VNDS-ILS: VNDS heuristic for SIP with intermediate SIP problems solved with the integer L-shaped method.

We perform two sets of experiments. In the first set we compare the performance of VNDS-DEP with DEP solution on problems containing first-stage binary variables. The results are given in Table 5.3. In this table t_1 denotes the time to reach the best DEP objective value and t_2 denotes the total solution time.

Table 5.3: Performance of the DEP and VNDS-DEP solution methods

		DEP			VNDS-DEP		
Name	Scen	t_1	t_2	Obj	t_1	t_2	Obj
dcap233	200	0.85	1.26	1834.58	2.24	8.93	1834.58
	300	1622.56	T	1644.36	137.00	T	1644.36
	500	3.23	3.23	1737.52	8.05	38.33	1737.52
dcap243	200	5.14	5.15	2322.49	4.48	24.94	2322.52
	300	10.31	10.89	2559.45	16.63	30.91	2559.55
	500	35.45	64.60	2167.36	22.68	63.85	2167.35
dcap332	200	31.83	T	1060.70	25.61	T	1060.70
	300	111.72	125.17	1252.88	82.90	T	1252.88
	500	1674.13	T	1588.81	241.03	T	1588.82
dcap342	200	96.40	101.67	1619.57	93.53	128.00	1619.57
	300	487.29	487.34	2067.71	672.46	T	2067.70
	500	1669.14	T	1904.66	T	T	1905.45
sizes	3	0.58	0.59	224434	1.11	4.73	224434
	5	1.24	2.27	224486	8.38	40.87	224486
	10	1624.38	T	224564	430.74	T	224564
sslp-5-25	50	1.36	2.46	-121.60	0.33	1.05	-121.60
	100	1.11	8.05	-127.37	0.80	2.64	-127.37
sslp-10-50	50	325.63	334.33	-364.64	4.80	T	-364.62
	100	1655.76	T	-354.18	10.15	T	-354.19
	500	1477.25	T	-327.97	49.32	T	-349.14
	1000	1590.45	T	123411	7.67	T	-327.53
	2000	302.32	T	315655	2.59	T	-330.38
sslp-15-45	5	4.91	5.62	-262.40	2.64	58.02	-262.40
	10	10.84	14.52	-260.50	165.29	T	-260.50
	15	102.31	102.32	-253.60	4.92	T	-253.60

t_1 - time to reach the best DEP solution, t_2 - total solution time,

T - time limit reached

According to the results in Table 5.3 VNDS-DEP is not able to reach the best DEP solution only in one case and it returns much better solutions within the

given time limit for 3 most difficult problems. The time to solve the problem to optimality is generally higher for VNDS-DEP and it does not prove optimality more often than the DEP solver. However the time to reach the best DEP solution is generally lower for the heuristic method especially for difficult problems like dcap233 with 300 scenarios. We think that t_1 is a more important performance measure for a heuristic method because its purpose is to get good solutions quickly.

In the second set of experiments we compare the performance of all four methods considered in this study. Since our implementation of the integer L-shaped method requires continuous second stage we relax the second-stage integrality in the DCAP and SSLP problems. The SIZES problems are excluded from the second test run because relaxing second-stage integrality makes them trivial to solve.

The test results for the four methods are given in Table 5.4. The integer L-shaped method shows good performance on all DCAP and most SSLP problems except sslp-15-45 where it is outperformed by other methods. As expected using the integer L-shaped method instead of solving the DEP improved the performance of the VNDS heuristic in most cases.

Table 5.4: Solution times for the SIP methods on problems with relaxed second-stage integrality

		DEP		ILS		VNDS-DEP		VNDS-ILS	
Name	Scen	t_1	t_2	t_1	t_2	t_1	t_2	t_1	t_2
	200	0.07	0.18	0.08	0.08	0.37	0.44	0.27	0.35
dcap233*	300	0.13	0.27	0.12	0.12	0.60	0.70	0.39	0.51
	500	0.35	0.43	0.20	0.20	1.14	1.36	0.63	0.87
	200	0.09	0.18	0.12	0.12	0.32	0.41	0.37	0.46
dcap243*	300	0.15	0.18	0.18	0.18	0.49	0.64	0.49	0.63
	500	0.30	0.36	0.29	0.30	0.88	1.12	0.83	1.10
	200	0.10	0.17	0.11	0.11	0.30	0.35	0.26	0.33
dcap332*	300	0.21	0.25	0.16	0.16	0.53	0.62	0.35	0.45
	500	0.20	0.38	0.24	0.24	0.89	1.05	0.58	0.77
	200	0.10	0.19	0.12	0.12	0.27	0.34	0.33	0.40
dcap342*	300	0.22	0.28	0.22	0.22	0.44	0.56	0.50	0.62
	500	0.33	0.72	0.32	0.32	0.82	3.18	0.82	1.04

Solution times for the SIP methods on problems with relaxed second-stage integrality (continued)

		DEP		ILS		VNDS-DEP		VNDS-ILS	
Name	Scen	t_1	t_2	t_1	t_2	t_1	t_2	t_1	t_2
sslp-5-25*	50	0.61	0.88	0.03	0.03	0.30	1.32	0.16	0.24
	100	2.39	2.75	0.04	0.05	0.79	3.49	0.27	0.50
sslp-10-50*	50	3.55	11.20	0.58	1.25	1.17	28.58	0.54	2.73
	100	28.70	36.84	1.13	2.44	1.57	79.44	0.66	4.14
	500	278.00	1089.80	5.11	11.30	14.47	T	5.70	25.42
	1000	315.28	T	2.46	21.70	33.23	T	12.46	61.25
	2000	1435.25	T	4.87	42.27	70.70	T	28.56	118.19
sslp-15-45*	5	0.14	0.19	1.53	1.65	0.17	1.14	0.14	1.51
	10	0.04	0.68	0.81	2.29	0.26	4.41	0.20	3.23
	15	0.15	0.70	5.12	5.12	0.47	5.97	0.41	4.59

t_1 - time to reach the best DEP solution, t_2 - total solution time,
T - Time limit reached

Table 5.5 gives the best objective values found by each of the method on problems with relaxed second stages within the given time limit. This table only gives the results for the SSLP problems because other problems are solved to optimality and their objective values are the same (up to the relative stopping tolerance).

Both the integer L-shaped method and VNDS-ILS are able to solve all the test problems to optimality. The DEP solver and VNDS-DEP are not able solve two and three most difficult SSLP instances respectively. However even in these cases VNDS-DEP returns solutions with objective values better or equal to those returned by solving the DEP as can be seen from Table 5.5.

Table 5.5: Final objective values returned by the SIP methods on the SSLP problems with relaxed second-stage integrality

Name	Scen	DEP	ILS	VNDS-DEP	VNDS-ILS
sslp-5-25*	50	-121.60	-121.60	-121.60	-121.60
	100	-127.37	-127.37	-127.37	-127.37

Final objective values returned by the SIP methods on the SSLP problems with relaxed second-stage integrality (continued)

Name	Scen	DEP	ILS	VNDS-DEP	VNDS-ILS
	50	-365.44	-365.44	-365.44	-365.44
	100	-354.87	-354.87	-354.87	-354.87
sslp-10-50*	500	-349.92	-349.92	-349.92	-349.92
	1000	-287.36	-352.49	-352.49	-352.49
	2000	-285.81	-348.09	-342.18	-348.09
	5	-265.57	-265.57	-265.57	-265.57
sslp-15-45*	10	-261.90	-261.90	-261.90	-261.90
	15	-254.71	-254.71	-254.71	-254.71

Performance profiling

In this section we present a graphical representation of the above results as performance profiles. Dolan and Moré (2002) defined performance profile as a cumulative distribution function (CDF) for some performance measure. For each problem p and method m we define the performance measure as follows:

$$r_{pm} = \frac{t_{pm}}{\min_{m \in M} t_{pm}}, p \in P, m \in M,$$

where P is a set of test problem, M is a set of solution methods and t_{pm} is the time taken by the method m to reach the best DEP solution when solving the problem p.

Then the cumulative distribution function for r_{pm} is defined as follows:

$$\rho_m(\tau) = \frac{|\{p \in P | r_{pm} \leq \tau\}|}{|P|}$$

We think that the the time to reach the best DEP solution is a more appropriate measure of performance for the VNDS heuristic than the total solution time, because the main purpose and strength of this heuristic is to find good solutions quickly, not to prove optimality.

By plotting the performance profile of each solver we get a clear visual representation that illustrates the performance of each method across the whole set of benchmark problems.

The performance profiles for the DEP and VNDS-DEP on the original test problems are shown in Figure 5.2. They correspond to the results in Table 5.3.

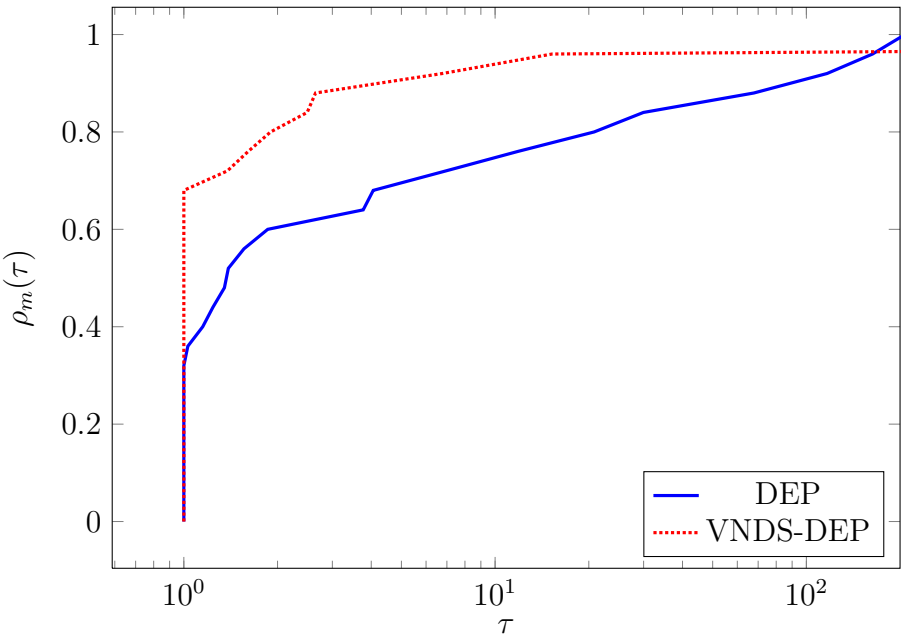


Figure 5.2: Performance profiles for the DEP and VNDS-DEP methods

As can be seen from this diagram VNDS-DEP shows overall better performance being the first in almost 70% of cases. However the right tail of the graph shows that VNDS-DEP was not always able to reach the best DEP solution. This was the case of the dcap342 problem with 500 scenarios.

Figure 5.3 shows the performance profiles for all four methods on the problems with relaxed second-stage integrality. These correspond to the results in Table 5.5. The profile of VNDS-ILS dominates the one of VNDS-DEP confirming our hypothesis that using an alternative solution method may improve performance of the VNDS-SIP heuristic. However, unlike the case with second-stage integer variables, here VNDS based heuristic is often outperformed by the integer L-shaped and even by solving the DEP. Nevertheless VNDS-ILS shows the best worst-case performance solving all the problems within the factor of about 6 of the best method.

Figure 5.4 illustrates the convergence of the VNDS-DEP method. In this experiment a larger time limit of 4 hours was used. It shows that given enough time an optimal solution or a solution very close to the optimal one can be obtained. This example also shows that the lower bound provided by the method can remain quite loose.

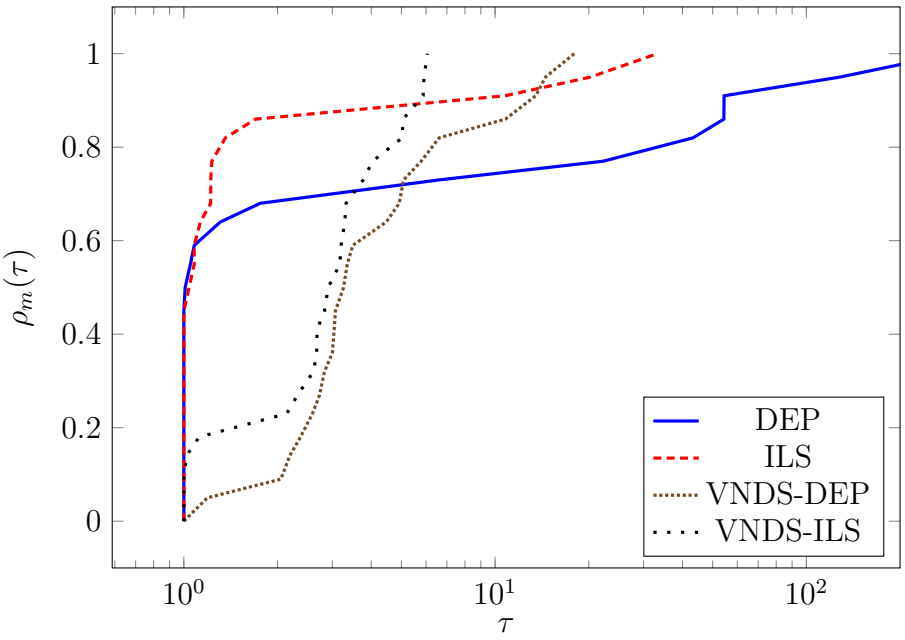


Figure 5.3: Performance profiles for the SIP methods on problems with relaxed second-stage integrality

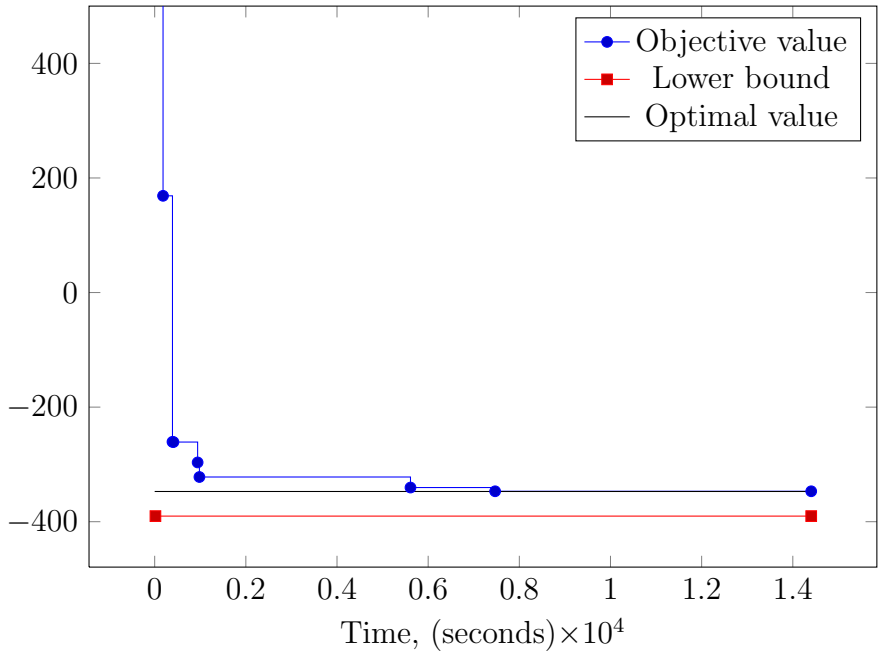


Figure 5.4: Convergence of VNDS-DEP on sslp-10-50 with 2000 scenarios

Chapter 6

Discussion and conclusions

6.1 Summary of findings and contributions

In the first part of our research reported in Chapter 2 we have set out the architecture of an extensible and reusable modelling system for stochastic programming. We have designed and implemented within this system new extensions to the SAMPL modelling language for representing the following important SP constructs:

- chance constraints,
- integrated chance constraints,
- robust optimisation models.

We have found that direct representation of corresponding SP constructs facilitates use of specialised algorithms. In particular we have implemented the cutting-plane method of Klein Haneveld and van der Vlerk (2006) and used it to solve a portfolio planning model with a large number of integrated chance constraints formulated in SAMPL. This has been possible because the SAMPL translator captures the information about the ICCs and passes it to the solver. In Section 3.2 we have described the model and have given some computational results which show huge benefit in terms of performance when using the cutting-plane algorithm compared to solving the DEP.

Also the language extensions allow the modeller to focus on the important aspects of the model, not on the details of how to represent the constructs in the deterministic equivalent form. The representations of alternative robust formulations introduced in Section 2.5 can be rather complex; in our system the transformation to deterministic equivalent representations can be done automatically

by the translator. These extensions to language constructs make a contribution to knowledge of computer-aided modelling for SP and robust optimisation.

In Chapter 3 we have studied a single-stage model with second-order stochastic dominance constraints. We have further investigated the relationships between SSD constraints, conditional value at risk and integrated chance constraints. We have considered an application of this model to the problem of portfolio selection.

While single-stage portfolio models based on SSD choice criterion have been known before (see, for example, Roman et al. (2006)) their applicability was limited because existing solution algorithms could only solve problems with relatively small number of scenarios. We have substantially contributed to making this model more tractable with a new cutting-plane algorithm (Fábián, Mitra, Roman, and Zverovich, 2010) which uses regularisation by the level method of Lemaréchal et al. (1995). As shown in the computational results in Section 3.6 the method scales well with increase in the number of scenarios; this makes solution of practical problems with tens of thousands of scenarios computable within tens of seconds.

We have also compared the return distributions of the optimal portfolios obtained by solving the model of Roman et al. (2006) and the scaled model described in Section 3.3. We have observed that although both models produce SSD efficient portfolios the scaled model gives overall higher outcomes than the model of Roman et al. at the cost of marginally higher risk.

In Chapter 4 we have studied solution methods for two-stage stochastic programming. We have reported the computational framework for decomposition-based SP solution methods. Based on this framework we have implemented the following established solution methods:

- the L-shaped method,
- the multicut L-shaped algorithm,
- trust region method based on l_∞ norm,
- regularised decomposition.

We have also applied the level method to regularisation of the expected recourse function (Zverovich, Fábián, Ellison, and Mitra, 2010).

In Section 4.2 we have reported an extensive computational study in which we compare performance of the above algorithms. In this study we have also included direct application of the Simplex method and IPM to the solution of the

DEP. For the purposes of this study we have used problems from several well-known SP test sets as well as instances of a recent gas portfolio planning model and generated problem instances.

Our empirical computational study has clearly shown that simple use of even the most powerful LP solvers cannot process many practical models in the DEP form especially when the model sizes scale up due to multiple scenarios. The L-shaped method with regularisation by the level method has performed well compared to other solution methods across the entire range of model sizes.

Despite more computations required per iteration it has been often faster than unregularised method due to substantial decrease in the number of iterations. In our experiments regularisation by the level method has also shown better scalability than other regularisation approaches.

In Chapter 5 we have addressed two-stage stochastic integer programming problems. In recent times there has been considerable progress in solution methods and software which can process difficult instances of MIP problems (IBM Corp., 2009b; Gurobi Optimization, 2010; Mittelmann, 2011). Yet many problems in the SIPLIB collection still remain difficult to solve by direct application of a MIP solver to the DEP. We have developed novel heuristic methods for stochastic integer programming (Lazić, Mitra, Mladenović, and Zverovich, 2010) bringing together the advantages of heuristics from the area of deterministic MIP that lead to feasible solutions and therefore upper bounds quickly with exact solution methods for stochastic integer programming. These new methods are based on variable neighbourhood decomposition search which has proven successful in deterministic context.

We have provided a detailed description of our VNDS heuristic method for SIP. A numerical study in Section 5.5 has shown that this method gives much better solutions to the most difficult SSLP problem instances with up to a million binary variables than those obtained by solving the deterministic equivalent problems.

We have also confirmed that the performance of the VNDS heuristic for stochastic integer programming can be improved by the appropriate choice of an underlying SIP solution method. To this end we have implemented the classic integer L-shaped method and used it to solve SIP subproblems. Although the performance of the resulting VNDS-ILS method has often been dominated by other algorithms, our empirical study shows that this method has the best worst case performance and improvement over VNDS-DEP.

The three solution methods, (a) processing of a single-stage problem with second-order stochastic dominance constraints, (b) regularisation by the level method for two-stage SP and (c) method for solving integer SP problems, are novel approaches and each of these makes a contribution to knowledge.

6.2 Suggestions for future research

In our work on the modelling system and language extensions in Chapter 2 we have considered only the case of individual chance constraints and integrated chance constraints. This can be naturally extended to support joint CCs and ICCs. Also the `dist` attribute which we have introduced there could be used to specify distributions of random parameters and extend the scope of the SAMPL modelling language to represent not only scenario-based models, but distribution-based models as well. So far we have only used it for the parameters following the uncertainty model of robust optimisation.

Our translator for the algebraic modelling language the architecture of which is described in Chapter 2 allows embedding in other software and provides application programming interfaces that give access to modelling objects such as variables, objectives and constraints. This opens a lot of possibilities for development of advanced modelling tools. We have shown in Section 2.2 one example of such a tools, an IDE with precise context-sensitive syntax highlighting for AMPL and SAMPL models. However much more can be done in terms of modelling instruments for rapid development and debugging of optimisation models. This includes data visualisation, model analysis and transformation, etc.

In Chapter 3 we have studied single-stage models with second-order stochastic dominance constraints. However we have not suggested any way to directly represent these models in an algebraic modelling language apart from the formulation with integrated chance constraints. Recently some alternative SIP models incorporating risk measures have been proposed and discussed by Escudero (1995), Schultz and Tiedemann (2006), Alonso-Ayuso et al. (2009) and others. A computational comparison of performance of such models in two- and multistage environments is also an interesting direction for future work.

In Chapter 4 we have described a decomposition method with regularisation of the expected recourse function based on the level method. The computational study suggests that this method works well for problems with complete or relatively complete recourse often substantially reducing the number of optimality cuts required to reach an optimal solution. However the number of feasibility

cuts may remain large. Extension of this method for regularisation of feasibility in case of incomplete recourse remains an interesting topic for future research.

In Chapter 5 we have proposed heuristics based on variable neighbourhood decomposition search for two-stage SIP problems. In the future this approach can be extended to multistage problems and compared to recent SIP solution methods.

References

Agnew, N. H., Agnew, R. A., Rasmussen, J., and Smith, K. R. (1969). An application of chance constrained programming to portfolio selection in a casualty insurance firm. *Management Science*, *15*(10), 512–520.

Ahmed, S. (2004). SIPLIB: A Stochastic Integer Programming Test Problem Library, Version 1.0 (December 2004). `http://www2.isye.gatech.edu/~sahmed/siplib/`.

Ahmed, S. and Garcia, R. (2004). Dynamic capacity acquisition and assignment under uncertainty. *Annals of Operations Research*, *124*(1-4), 267–283.

Ahmed, S., Tawarmalani, M., and Sahinidis, N. V. (2004). A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, *100*(2), 355–377.

Ahuja, R. K., Ergun, O., Orlin, J. B., and Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, *123*, 75–102.

Alonso-Ayuso, A., Escudero, F., L., Garín, A., Ortuño, M. T., and Pérez, G. (2003a). An approach for strategic supply chain planning under uncertainty based on stochastic 0-1 programming. *Journal of Global Optimization*, *26*, 97–124.

Alonso-Ayuso, A., Escudero, L. F., and Ortuño, M. T. (2003b). BFC, a branch-and-bound coordination algorithmic framework for solving some types of stochastic pure and mixed 0-1 programs. *European Journal of Operational Research*, *15*, 503–519.

Alonso-Ayuso, A., Escudero, L. F., and Pizarro, C. (2009). On sip algorithms for minimizing the mean-risk function in the multi-period single-source problem under uncertainty. *Annals of Operations Research*, *166*(1), 223–242.

Aouchiche, M., Bonnefoy, J. M., Fidahoussen, A., Caporossi, G., Hansen, P., Hiesse, L., Lacheré, J., and Monhait, A. (2006). Variable neighborhood search for extremal graphs. 14: The AutoGraphiX 2 system. In P. Pardalos, L. Liberti, and N. Maculan (Eds.) *Global Optimization*, vol. 84 of *Nonconvex Optimization and Its Applications*, (pp. 281–310). Springer US.

Ariyawansa, K. A. and Felt, A. J. (2004). On a new collection of stochastic linear programming test problems. *INFORMS Journal on Computing*, *16*(3), 291–299.

Balas, E. (1979). Disjunctive programming. *Annals of Discrete Mathematics*, *5*, 1–20.

Barahona, F., Bermon, S., Gunluk, O., and Hood, S. (2001). Robust capacity planning in semiconductor manufacturing. IBM Research Report RC22196.

Beale, E. M. L. (1955). On minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society, Series B*, *17*, 173–184.

Ben-Tal, A. and Nemirovski, A. (2000). Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, *88*, 411–424.

Ben-Tal, A. and Nemirovski, A. (2002). Robust optimization - methodology and applications. *Mathematical Programming, Series B 92*, (pp. 453–480).

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, *4*, 238–252. Re-publised in *Computational Management Science 2* (2005), 3–19.

Bertsimas, D. and Sim, M. (2004). The price of robustness. *Operations Research*, *52*(1), 35–53.

Bertsimas, D. and Thiele, A. (2006). A robust optimization approach to inventory theory. *Operations Research*, *54*(1), 150–168.

Birge, J. R., Dempster, M. A. H., Gassmann, H. I., Gunn, E. A., King, A. J., and Wallace, S. W. (1987). A standard input format for multiperiod stochastic linear programs. *COAL Newsletter*, *17*, 1–19.

Birge, J. R. and Louveaux, F. V. (1988). A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, *34*, 384–392.

Birge, J. R. and Louveaux, F. V. (1997). *Introduction to Stochastic Programming*. Springer-Verlag, New York.

Blair, C. E. and Jeroslow, R. G. (1978). A converse for disjunctive constraints. *Journal of Optimization Theory and Applications*, *25*(2), 195–206.

Blair, C. E. and Jeroslow, R. G. (1982). The value function of an integer program. *Mathematical Programming*, *23*(1), 237–273.

Bradley, S. P. and Crane, D. B. (1972). A dynamic model for bond portfolio management. *Mathematical Programming*, *19*, 139–151.

Brimberg, J. and Mladenović, N. (1996). A variable neighbourhood algorithm for solving the continuous location-allocation problem. *Mathematical Programming*, *10*, 1–12. Studies in Location Analysis.

Cariño, D. R., Kent, T., Myers, D. H., Stacy, C., Sylvanus, M., Turner, A. L., Watanabe, K., and Ziemba, W. T. (1994). The russell-yasuda kasai model: An asset/liability model for a Japanese insurance company using multistage stochastic programming. *Interfaces*, *24*(1), 29–49.

CARISMA (2010). CARISMA lecture notes on stochastic programming.

Carøe, C. C. (1998). *Decomposition in stochastic integer programming*. Ph.D. thesis, University of Copenhagen, Denmark.

Carøe, C. C. and Tind, J. (1998). L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming*, *83*(3), 451–464.

Charnes, A. and Cooper, W. W. (1959). Chance-constrained programming. *Management Science*, *6*, 73–79. DOI: 10.1287/mnsc.6.1.73.

Colombo, M., Grothey, A., Hogg, J., Woodsend, K., and Gondzio, J. (2009). A structure-conveying modelling language for mathematical and stochastic programming. *Mathematical Programming Computation*, *1*, 223–247.

Consigli, G. and Dempster, M. A. H. (1998). Dynamic stochastic programming for asset-liability management. *Annals of Operations Research*, *81*, 131–162.

Costa, O. L. V. and Paiva, A. C. (2002). Robust portfolio selection using linear-matrix inequalities. *Journal of Economic Dynamics and Control*, *26*(6), 889–909.

Cristóbal, M. P., Escudero, L. F., and Monge, J. F. (2009). On stochastic dynamic programming for solving large-scale planning problems under uncertainty. *Computers and Operations Research*, *36*(8), 2418–2428.

Czyzyk, J., Mesnier, M., and Moré, J. (1998). The neos server. *IEEE Journal on Computational Science and Engineering*, (pp. 68–75).

Danna, E., Rothberg, E., and Pape, C. L. (2005). Exploring relaxation induced neighborhoods to improve mip solution. *Mathematical Programming*, *102*, 71–90.

Dantzig, G. B. (1955). Linear programming under uncertainty. *Management Science*, *1*, 197–206.

Dantzig, G. B. and Madansky, A. (1961). On the solution of two-stage linear programs under uncertainty. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, (pp. 165–176). University of California Press, Berkeley.

Dantzig, G. B. and Wolfe, P. (1960). The decomposition principle for linear programs. *Operations Research*, *8*, 101–111.

Dejax, P. J. and Crainic, T. G. (1987). Survey paper – a review of empty flows and fleet management models in freight transportation. *Transportation Science*, *21*(4), 227–248.

Dentcheva, D. and Ruszczyński, A. (2003). Optimization with stochastic dominance constraints. *SIAM Journal on Optimization*, *14*(2), 548–566.

Dentcheva, D. and Ruszczyński, A. (2006). Portfolio optimization with stochastic dominance constraints. *Journal of Banking & Finance*, *30*, 433–451.

Des Rivières, J. and Wiegand, J. (2004). Eclipse: a platform for integrating development tools. *IBM Systems Journal*, *43*, 371–383.
URL `http://dx.doi.org/10.1147/sj.432.0371`

Di Domenica, N., Lucas, C., Mitra, G., and Valente, P. (2009). Scenario generation for stochastic programming and simulation: a modelling perspective. *IMA Journal of Management Mathematics*, *20*, 1–38.

Dirkse, S. (1998). Stochastic programming using gams. Presented at the Conference on Stochastic Programming, Vancouver, BC, Canada.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, *91*(2), 201–213.

Dormer, A., Vazacopoulos, A., Verma, N., and Tipi, H. (2005). Modeling & solving stochastic programming problems in supply chain management using xpress-sp. In J. Geunes, and P. M. Pardalos (Eds.) *Supply Chain Optimization*, vol. 98 of *Applied Optimization*, (pp. 307–354). Springer US.

Dražić, M., Lavor, C., Maculan, N., and Mladenović, N. (2008). A continuous variable neighborhood search heuristic for finding the three-dimensional structure of a molecule. *European Journal of Operational Research*, *185*(3), 1265–1273.

Dyer, M. and Stougie, L. (2006). Computational complexity of stochastic programming problems. *Mathematical Programming, Series A*, *106*(3), 423–432.

El Ghaoui, L., Oks, M., and Oustry, F. (2003). Worst-case value-at-risk and robust portfolio optimization: A conic programming approach. *Operations Research*, *51*(4), 543–556.

Ellison, E. F. D., Hajian, M., Jones, H., Levkovitz, R., Maros, I., Mitra, G., and Sayers, D. (2008). *FortMP Manual*. Brunel University: London, Numerical Algorithms Group: Oxford. `http://www.optirisk-systems.com/manuals/FortmpManual.pdf`.

Ellison, F., Mitra, G., and Zverovich, V. (2010). *FortSP: A Stochastic Programming Solver*. OptiRisk Systems. `http://www.optirisk-systems.com/manuals/FortspManual.pdf`.

Eppen, G. D., Martin, R. K., and Schrage, L. (1989). OR practice – a scenario approach to capacity planning. *Operations Research*, *37*(4), 517–527.

Ermol'ev, Y. M., Krivets, T. A., and Petuk, V. S. (1976). Planning of shipping empty seaborne containers. *Cybernetics and System Analysis*, *12*(4), 644–646.

Escudero, L. F. (1995). Robust portfolios for mortgage-backed securities. In S. Zenios (Ed.) *Quantitative Methods, Supercomputers and AI in Finance*, (pp. 201–228). UNICOM, London.

Escudero, L. F. (2009). On a mixture of the fix-and-relax coordination and lagrangian substitution schemes for multistage stochastic mixed integer programming. *TOP*, *17*(1), 5–29.

Escudero, L. F., Garín, A., Merino, M., and Pérez, G. (2007a). A two-stage stochastic integer programming approach as a mixture of branch-and-fix coordination and benders decomposition schemes. *Annals of Operations Research*, *152*(1), 395–420.

Escudero, L. F., Garín, A., Merino, M., and Pérez, G. (2007b). The value of the stochastic solution in multistage problems. *TOP*, *15*(1), 48–64.

Escudero, L. F., Garín, A., Merino, M., and Pérez, G. (2009a). Bfc-msmip: an exact branch-and-fix coordination approach for solving multistage stochastic mixed 01 problems. *TOP*, *17*(1), 96–122.

Escudero, L. F., Garín, A., Merino, M., and Pérez, G. (2009b). A general algorithm for solving two-stage stochastic mixed 0-1 first-stage problems. *Journal Computers and Operations Research*, *36*(9), 2590–2600.

Escudero, L. F., Garín, A., Merino, M., and Pérez, G. (2009c). On multistage stochastic integer programming for incorporating logical constraints in asset and liability management under uncertainty. *Computational Management Science*, *6*(3), 307–327.

Escudero, L. F., Garín, A., Merino, M., and Pérez, G. (2010a). An exact algorithm for solving large-scale two-stage stochastic mixed-integer problems: Some theoretical and experimental aspects. *European Journal of Operational Research*, *204*(1), 105–116.

Escudero, L. F., Garín, A., Merino, M., and Pérez, G. (2010b). On bfc-msmip strategies for scenario cluster partitioning, and twin node family branching selection and bounding for multistage stochastic mixed integer programming. *Computers and Operations Research*, *37*(4), 738–753.

Escudero, L. F., Garín, A., Merino, M., and Pérez, G. (2011). An algorithmic framework for solving large scale multistage stochastic mixed 0-1 problems with nonsymmetric scenario trees. *Computers & Operations Research*. Accepted for publication.

Escudero, L. F., Quintana, F. J., and Salmerón, J. (1999). Coro, a modeling and an algorithmic framework for oil supply, transformation and distribution optimization under uncertainty. *European Journal of Operational Research*, (114), 638–656.

Fábián, C. I. (2000). Bundle-type methods for inexact data. *Central European Journal of Operations Research*, *8*, 35–55. Special issue, T. Csendes and T. Rapcsák, eds.

Fábián, C. I., Mitra, G., and Roman, D. (2009). Processing second-order stochastic dominance models using cutting-plane representations. *Mathematical Programming, Series A*. DOI: 10.1007/s10107-009-0326-1.

Fábián, C. I., Mitra, G., Roman, D., and Zverovich, V. (2010). An enhanced model for portfolio choice with ssd criteria: a constructive approach. *Quantitative Finance*. First published on: 11 May 2010.

Fábián, C. I. and Szőke, Z. (2007). Solving two-stage stochastic programming problems with level decomposition. *Computational Management Science*, *4*, 313–353.

Falk, J. E. (1976). Exact solutions of inexact linear programs. *Operations Research*, *24*(4), 783–787.

Ferguson, A. R. and Dantzig, G. B. (1956). The allocation of aircraft to routesan example of linear programming under uncertain demand. *Management Science*, *3*(1), 45–73.

Fischetti, M. and Lodi, A. (2003). Local branching. *Mathematical Programming*, *98*(1-3), 23–47.

Fishburn, P. C. (1964). *Decision and Value Theory*. John Wiley & Sons, New York.

Fishburn, P. C. and Vickson, R. G. (1978). Theoretical foundations of stochastic dominance. In *Stochastic Dominance: An Approach to Decision-Making Under Risk*, (pp. 37–113). D.C. Heath and Company, Lexington, Massachusetts.

Fourer, R. (1996). Proposed new ampl features: Stochastic programming extensions. `http://www.ampl.com/NEW/FUTURE/stoch.html`.

Fourer, R. and Gay, D. M. (1995). Expressing special structures in an algebraic modeling language for mathematical programming. *ORSA Journal on Computing*, (7), 166–190.

Fourer, R., Gay, D. M., and Kernighan, B. W. (2003). *AMPL: a modeling language for mathematical programming*. Pacific Grove, CA ; London: Thomson/Brooks/Cole.

Fourer, R. and Lopes, L. (2009). Stampl: A filtration-oriented modeling tool for multistage stochastic recourse problems. *INFORMS Journal on Computing*, *21*, 242–256.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Company.

Gassmann, H. (1990). MSLiP: a computer code for the multistage stochastic linear programming problem. *Mathematical Programming*, *47*, 407–423.

Gurobi Optimization (2010). *Gurobi Optimizer Reference Manual, Version 4.0*.

Hadar, J. and Russell, W. R. (1969). Rules for ordering uncertain prospects. *The American Economic Review*, *59*, 25–34.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, *130*(3), 449–467.

Hansen, P., Mladenović, N., and Perez-Brito, D. (2001). Variable Neighborhood Decomposition Search. *Journal of Heuristics*, *7*(4), 335–350.

Hansen, P., Mladenović, N., and Urošević, D. (2006). Variable neighborhood search and local branching. *Computers and Operations Research*, *33*, 3034–3045.

Holmes, D. (1995). A (PO)rtable (S)tochastic programming (T)est (S)et (POSTS). Web page: `http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html`.

IBM Corp. (2009a). *IBM ILOG CPLEX V12.1: File formats supported by CPLEX*.

IBM Corp. (2009b). *IBM ILOG CPLEX V12.1: User's Manual for CPLEX*.

Jordan, W. C. and Turnquist, M. A. (1983). A stochastic, dynamic network model for railroad car distribution. *Transportation Science*, *17*(2), 123–145.

Jorjani, S., Scott, C. H., and Woodruff, D. L. (1999). Selection of an optimal subset of sizes. *International Journal of Production Research*, *37*(16), 3697–3710.

Kall, P. and Mayer, J. (1998). On testing SLP codes with SLP-IOR. In *New Trends in Mathematical Programming: Homage to Steven Vajda*, (pp. 115–135). Kluwer Academic Publishers.

Kallberg, J. G., White, R. W., and Ziemba, W. T. (1982). Short term financial planning under uncertainty. *Management Science*, *28*(6), 670–682.

Karmarkar, N. (1984). A new polynomial time algorithm for linear programming. *Combinatorica*, *4*(4), 373–395.

Kiwiel, K. C. (1985). *Methods of descent for nondifferentiable optimization*. Springer-Verlag, Berlin, New York.

Klein Haneveld, W. K. (1986). *Duality in stochastic linear and dynamic programming*. Springer-Verlag, New York.

Klein Haneveld, W. K. and van der Vlerk, M. H. (2006). Integrated chance constraints: reduced forms and an algorithm. *Computational Management Science*, *3*(4), 245–269.

König, D., Suhl, L., and Koberstein, A. (2007). Optimierung des Gasbezugs im liberalisierten Gasmarkt unter Berücksichtigung von Röhren- und Untertagespeichern. In *Sammelband zur VDI Tagung "Optimierung in der Energiewirtschaft" in Leverkusen*.

Künzi-Bay, A. and Mayer, J. (2006). Computational aspects of minimizing conditional value-at-risk. *Computational Management Science*, *3*(1), 3–27.

Kusy, M. I. and Ziemba, W. T. (1986). A bank asset and liability management model. *Operations Research*, *34*(3), 356–376.

Laporte, G. and Louveaux, F. V. (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, *13*, 133–142.

Lazić, J., Hanafi, S., and Mladenović, N. (2010). Relaxation guided diving for 0-1 MIP feasibility. *Discrete Applied Mathematics*. Submitted to special issue devoted to Matheuristics conference, June 28-30, 2010, Vienna, Austria.

Lazić., J., Hanafi, S., Mladenović, N., and Urošević, D. (2010). Variable neighbourhood decomposition search for 0–1 mixed integer programs. *Computers & Operations Research*, *37*(6), 1055–1067.

Lazić, J., Mitra, G., Mladenović, N., and Zverovich, V. (2010). Variable neighbourhood decomposition search for a two-stage stochastic mixed integer programming problem. *Discrete Applied Mathematics*. Submitted to special issue devoted to Matheuristics conference, June 28-30, 2010, Vienna, Austria.

Lemaréchal, C. (1978). Nonsmooth optimization and descent methods. *Research Report* 78-4, IIASA, Laxenburg, Austria.

Lemaréchal, C., Nemirovskii, A., and Nesterov, Y. (1995). New variants of bundle methods. *Mathematical Programming*, *69*, 111–147.

Levy, H. (1992). Stochastic dominance and expected utility: Survey and analysis. *Mathematical Programming*, *38*, 555–593.

Linderoth, J. and Wright, S. (2003). Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, *24*, 207–250.

Linderoth, J. T., Shapiro, A., and Wright, S. J. (2002). The empirical behavior of sampling methods for stochastic programming. Optimization Technical Report 02-01, Computer Science Department, University of Wisconsin-Madison.

Lougee-Heimer, R. (2003). The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, *47*(1), 57–66.

Louveaux, F. V. (1980). A solution method for multistage stochastic programs with recourse with application to an energy investment problem. *Operations Research*, *28*(4), 889902.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, *7*(1), 77–91.

Midler, J. L. and Wollmer, R. D. (1969). Stochastic programming models for scheduling airlift operations. *Naval Research Logistics Quarterly*, *16*, 315–330.

Mitra, G., Di Domenica, N., Birbilis, G., and Valente, P. (2007). Stochastic programming and scenario generation within a simulation framework: An information perspective. *Decision Support Systems*, *42*, 2197–2218.

Mittelmann, H. (2011). Mixed integer linear programming benchmark. `http://plato.asu.edu/ftp/milpf.html`.

Mittelmann, H. D. (1998). Benchmarking interior point LP/QP solvers. *Opt. Meth. Software*, *12*, 655–670.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & OR*, *24*(11), 1097–1100.

Mulvey, J. M. and Ruszczyǹski, A. (1995). A new scenario decomposition method for large-scale stochastic optimization. *Operations Research*, *43*(3), 477–490.

Ntaimo, L. and Sen, S. (2005). The million-variable "march" for stochastic combinatorial optimization. *Journal of Global Optimization*, *32*(3), 385–400.

Ogryczak, W. and Ruszczyński, A. (1999). From stochastic dominance to mean-risk models: Semideviations as risk measures. *European Journal of Operational Research*, *116*, 33–50.

Ogryczak, W. and Ruszczyński, A. (2002). Dual stochastic dominance and related mean-risk models. *SIAM Journal on Optimization*, *13*(1), 60–78.

OptiRisk Systems (2009). SAMPL/SPInE User Manual.

Owen, S. H. and Daskin, M. S. (1998). Strategic facility location: A review. *European Journal Of Operational Research*, *111*(3), 423–447.

Powell, W. B. (1986). A stochastic model of the dynamic vehicle allocation problem. *Transportation Science*, *20*(2), 117–129.

Prékopa, A. (1973). Contributions to the theory of stochastic programming. *Mathematical Programming*, *4*(1), 202–221.

Prékopa, A. (2003). Probabilistic programming. In A. Ruszczynski, and A. Shapiro (Eds.) *Stochastic Programming. Handbooks in Operations Research and Management Science Vol. 10.*, chap. 5, (pp. 267–351). Elsevier, Amsterdam.

Rockafellar, R. T. (1976). Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, *14*, 877–898.

Rockafellar, R. T. and Uryasev, S. (2000). Optimization of conditional value-at-risk. *The Journal of Risk*, *2*(3), 21–41.

Rockafellar, R. T. and Uryasev, S. (2002). Conditional value-at-risk for general loss distributions. *Journal of Banking & Finance*, *26*, 1443–1471.

Roelofs, M. and Bisschop, J. (2010). *AIMMS: The Language Reference*. Paragon Decision Technology. Also available as `http://www.aimms.com/aimms/download/manuals/aimms_ref_printable.pdf`.

Roman, D., Darby-Dowman, K., and Mitra, G. (2006). Portfolio construction based on stochastic dominance and target return distributions. *Mathematical Programming, Series B*, *108*, 541–569.

Roman, D., Mitra, G., and Zverovich, V. (2011). Enhanced indexation based on second-order stochastic dominance. Submitted to Journal of Banking and Finance.

Ross, S. M. (Ed.) (2002). *An Elementary Introduction to Mathematical Finance: Options and other Topics*. Cambridge University Press.

Rudolf, G. and Ruszczyński, A. (2008). Optimization problems with second order stochastic dominance constraints: Duality, compact formulations, and cut generation methods. *SIAM Journal on Optimization*, *19*(3), 1326–1343.

Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, *35*, 309–333.

Ruszczyński, A. (2003). Decomposition methods. In A. Ruszczyński, and A. Shapiro (Eds.) *Stochastic Programming, Handbooks in Operations Research and Management Science*, vol. 10, (pp. 141–211). Elsevier, Amsterdam.

Ruszczyński, A. and Świętanowski, A. (1997). Accelerating the regularized decomposition method for two-stage stochastic linear problems. *European Journal of Operational Research*, *101*, 328–342.

Sadki, M. (2005). AMPL COM Component Library, User's Guide Version 1.6. `http://www.optirisk-systems.com/manuals/AmplComManual.pdf`.

Santoso, T., Ahmed, S., Goetschalckx, M., and Shapiro, A. (2005). A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research*, (167), 96–115.

Schultz, R., Stougie, L., and van der Vlerk, M. H. (1998). Solving stochastic programs with integer recourse by enumeration: a framework using Gröbner basis reductions. *Mathematical Programming*, *83*(2), 229–252.

Schultz, R. and Tiedemann, S. (2006). Conditional value-at-risk in stochastic programs with mixed-integer recourse. *Mathematical Programming*, *105*(2–3), 365–386.

Sen, S., Doverspike, R. D., and Cosares, S. (1994). Network planning with random demand. *Telecommunication Systems*, *3*, 11–30.

Sen, S. and Higle, J. L. (2005). The c^3 theorem and a d^2 algorithm for large scale stochastic mixed-integer programming: Set convexification. *Mathematical Programming*, *104*(1), 1–20.

Sherali, H. D. and Shetty, C. M. (1980). *Optimization with Disjunctive Constraints*. Lecture Notes in Economics and Mathematics. Springer-Verlag, New York.

Soyster, A. L. (1973). Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, *21*(5), 1154–1157.

Subrahmanyam, S., Pekny, J. F., and Reklaitis, G. V. (1994). Design of batch chemical plants under market uncertainty. *Ind. Eng. Chem. Res.*, *33*, 2688–2701.

Suganuma, T., Ogasawara, T., Takeuchi, M., Yasue, T., Kawahito, M., Ishizaki, K., Komatsu, H., and Nakatani, T. (2000). Overview of the IBM Java just-in-time compiler. *IBM Systems Journal*, *39*(1), 175–193.

Valente, C., Mitra, G., Sadki, M., and Fourer, R. (2009). Extending algebraic modelling languages for stochastic programming. *Informs Journal on Computing*, *21*(1), 107–122.

Valente, P., Mitra, G., and Poojari, C. (2005). A stochastic programming integrated environment (SPInE). In S. W. Wallace, and W. T. Ziemba (Eds.) *MPS/SIAM Series on Optimisation: Applications of Stochastic Programming*, chap. 8, (pp. 115–136). Society for Industrial and Applied Mathematic.

Van de Panne, C. and Popp, W. (1963). Minimum-cost cattle feed under probabilistic protein constraints. *Management Science*, *9*(3), 405–430.

Van der Vlerk, M. H. (2003). Integrated chance constraints in an alm model for pension funds. Department of Econometrics & OR, University of Groningen.

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

Wallace, S. W. and Fleten, S.-E. (2003). Stochastic programming models in energy. In A. Ruszczynski, and A. Shapiro (Eds.) *Stochastic Programming*, vol. 10 of *Handbooks in Operations Research and Management Science*, (pp. 637 – 677). Elsevier.

Wallace, S. W. and Ziemba, W. T. (Eds.) (2005). *Applications of Stochastic Programming*. Society for Industrial and Applied Mathematic.

Wets, R. J. B. (1974). Stochastic programs with fixed recourse: The equivalent deterministic program. *SIAM Review*, *16*, 309–339.

Whitmore, G. A. and Findlay, M. C. (Eds.) (1978). *Stochastic Dominance: An Approach to Decision-Making Under Risk*. D.C. Heath and Company, Lexington, Massachusetts.

Wollmer, R. D. (1980). Two stage linear programming under uncertainty with 0-1 integer first stage variables. *Mathematical Programming*, *19*(1), 279–288.

Zverovich, V., Fábián, C. I., Ellison, F., and Mitra, G. (2010). A computational study of a solver system for processing two-stage stochastic linear programming problems. Submitted to Mathematical Programming Computation, currently under the second set of revisions.