UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Solving large-scale two-stage stochastic optimization problems by specialized interior point methods

Paula de la Lama Zubirán

Solving Large-Scale Two-Stage Stochastic Optimization Problems by Specialized Interior Point Method

Author:
Paula de la Lama Zubirán

A thesis presented for the degree of
Doctor of Philosophy

Supervisor: **Dr. Jordi Castro Pérez**

Department of Statistics and Operations Research
Universitat Politècnica de Catalunya - BarcelonaTech
Barcelona

2020

I dedicate this thesis to my parents,
to my husband, and my little fairy Laila

Contents

List of Figures

List of Tables

Acknowledgments

First I would like to thank my thesis director, Dr. Jordi Castro. Our path toward this dissertation was, not only about research, but it was also about personal and cultural learning. He teaches by example (always working in his office or at home); he shared his work without restrictions and his guidance helped me find good results in our experiments. On a personal level, communication was not always easy, but in the end, we even laughed together. Thank you very much.

There are not enough words to thank my parents. Thanks, Dad for being a tireless worker in this academic world. Your motivation towards life makes me always keep working. Mom, it is indescribable how you have guided us all on the path to success –millions of thanks. I admire and love you very much.

I thank my husband Daniel, together we achieved this thesis. From the beginning, your unconditional help and support have been the foundation for this achievement. Many more triumphs await us.

I would also want to thank my brother, even at a distance, and with little communication, I will always count on your love and support. You showed me that mathematics is not so difficult.

I am grateful to my colleagues and friends in office C201. Dani, for the healthy fun. Diana, for her support in the programming section, it was a privilege to have you as a partner and teacher. Gloria, for her wise company and the hiking trips. Jeaneth, Carlos, Xavi, and Vicky, thank you for your advice and good gossip. I love you all.

Finally, this research would not have been possible without the assistance of CONACyT and the UAM-I (Universidad Autónoma Metropolitana-Iztapalapa) for the support all this time.

Abstract

Two-stage stochastic optimization models give rise to very large linear problems (LP). Several approaches have been devised for efficiently solving them, among which are interior-point methods (IPM). However, using IPM, the linking columns that are associated with first-stage decisions cause excessive fill-ins for the solutions of the normal equations, thus making the procedure computationally expensive. We have taken a step forward on the road to a better solution by reformulating the LP through a variable splitting technique which has significantly reduced the solution time.

This work presents a specialized IPM that first applies variable splitting, then exploits the structure of the deterministic equivalent formulation of the stochastic problem. The specialized IPM works with an algorithm that combines Cholesky factorization and preconditioned conjugate gradients for solving the normal equations when computing the Newton direction for stochastic optimization problems in which the first-stages variables are large enough. Our specialized approach outperforms standard IPM.

This work provides the computational results of two stochastic problems from the literature: (1) a supply chain system and (2) capacity expansion in an electric system. Both linear and quadratic formulations were used to obtain instances of up to 39 million variables and six million constraints. When used in these applications, the computational results show that our procedure is more efficient than alternative state-of-the-art IP implementations (e.g., CPLEX) and other specialized methods for stochastic optimization.

Chapter 1

Introduction

1.1 Motivation and objectives

Nowadays, technology and the ability to generate and store large information has grown significantly. The necessity for analyzing all this data in a short amount of time for making important decisions is an indispensable tool for companies to survive in this competitive and globalized world. The discipline dealing with the application of advanced analytical methods for making those better decisions is called Operations Research. For many years, an increasing number of companies around the world have applied this kind of tool. Besides, science continuously improves the way to solve these problems, and models resemble more and more to real situations.

Stochastic optimization is a branch of Operations Research that deals with optimization problems that involve uncertainty. These models can describe a wide range of applications in real-life situations. Some examples include sales projections, prices or any unknown random element (usually, but not necessarily, in the future). If a decision-maker wishes to take into account a large number of possibilities situations, the model will be very extensive. Every possible situation is called a scenario, and the likelihood of each scenario is considered known in advance [13].

An excellent example of these problems is the classical newsvendor (or newsboy) problem, general class, found in *two-stage stochastic linear programs with recourse* literature [11]. In this problem, a news vendor goes to a publisher every morning and buys x number of newspapers at a price of c per paper. He can only carry u number of papers. The vendor then has to walk along the streets to sell as many newspapers as possible. Any unsold newspaper can then be returned to the publisher at a return price of r. The model helps the newsvendor decide how many newspapers to buy every morning. Given that the demand for newspapers varies over days, it is described by a random variable demand (ξ).

This problem illustrates the two-stage stochastic model, which encloses two types of variables. *First-stage variables*, which are the decisions made before the outcome of the random events, when the period begins; in the newsvendor problem is the buying decision x. This decision is made before any information is given about the demand. Then there are *second-stage variables* that are decided after the outcome of uncertain events are known, in the example out the demand [13]. When the demand is known in advance, then profit, or the so-called second-stage problem, represented at the end of the sales period of a given edition, can be computed.

One way to work with these problems is the extensive form of the model, where all scenarios are incorporated explicitly. This approach entails a vast number of constraints and/or variables. Also, they have a special structure with separable blocks. When the problem challenges the current state-of-the-art implementation of optimization methods and has a special structure is called a Large scale problem. These instances enable the uses of a reformulation technique combined with specialized methods, algorithms, and advanced computing power that allows us to find the solution in a reasonable amount of time.

The implementation of optimization methods with large scale problems has a few remaining challenges, most of them because of the amount rounding 25 million variables and 300 thousand constraints [18]. Problems that previously were considered intractable, from a computational point of view, now become amenable to practical mathematical-programming solutions.

Furthermore, the two most typical structures on a large scale problems are "linking variables" (dual angular block structure) or "linking constraints" (primal angular block structure) [14]. One classical methods of resolution is Benders decomposition for dual block angular [7]. Other methodology, the interior-point method (IPM), has also proved to be useful for working with these kinds of problems [43], [80].

A specialized interior-point algorithm (implemented in the `BlockIP` package) solves the normal equations using a method that combines Cholesky factorizations for the diagonal blocks and preconditioned conjugate gradient (PCG) for the linking constraints [18]. The idea for an applied a preconditioned conjugate gradient for this kind of model was presented by Gondzio [39] with some problems in the fast convergence. Our preconditioned conjugate has proved to be efficient when the linking constraints are sparse [15]. Moreover, a reformulation technique from [59], which is called *splitting formulation* makes the linking constraints simple and sparse.

The approach we propose, in this study, aims to solve efficiently two-stage stochastic problems. The `BlockIP` solver and the *splitting formulation* technique were used for making the linkings constraints very sparse. Although the solution does not work efficiently for every problem, this study demonstrates that in problems with large enough first-stage variables, this procedure outperforms the alternative state-of-the-art IPM implementations. The comparative was made for two kinds of problems with different instances, and the performances were measured in terms of CPU times.

The focus of this thesis is the efficient solution of large scale two-stage stochastic problems containing primal block angular structure, using the `BlockIP` solver and the *splitting formulation* technique for making the linkings constraints really sparse. As for the case of numerical experiments, publicly available instances were used to compare the performance against recent results already presented in the literature. This solution does not work efficiently for every problem. However, two examples (namely, a supply chain system and capacity expansion in an electric system) to demonstrate that in problems with large enough first-stages variables, this procedure outperforms the alternative state-of-the-art IPM implementations. The comparative was made in two kinds of problems with different instances, and the performances were measured in terms of CPU times.

The remainder of this document is organized as follows. In Chapter 2, the state of the art is divided into four parts. The first describes large-scale problems with their different structures (Section 2.2). In the second place, Section 2.3, the formulation of the Two-Stage Stochastic model (TSSP), with its extensive form is explained (Primal and Dual structure). Third, in Section 2.4 some solution methods are mentioned; focusing on the Interior-Point method Primal-Dual path-following algorithm. Finally, the last part Section 2.5, discusses difficulties with the interior-point method for stochastic optimization with two-stages problems.

The next Chapter 3, a specialized primal-dual path-following algorithm adapted for primal block-angular problems called `BlockIP` is presented. `BlockIP` works with a primal angular problem described in Section 3.2. Then, Section 3.3 explains how it solves the normal equations with a combination of Cholesky and Preconditioned Conjugate Gradient (PCG) for computing directions in an interactive optimization process when it searches for the next point. Also, in Section B, the general implementation of BIP shows how it can be used and, in the end in Section 3.4, the integration of `BlockIP` with the splitting formulation in TSSP, showing the structure of the resulting E matrix.

Chapter 4 for Results, discusses briefly the stochastic instances, that are available in the literature 4.2 —moreover, details of the applications that help us manage their information and applied `BlockIP`. Then, the results of the performances in comparison with the CPLEX barrier algorithm, as well as the results of the quadratic version of these problems (Subsection 4.2.4 and 4.2.5) are presented, Also, other specialized solvers were tested in Subsection 4.2.6. Following the instances that can be found in the literature, the creation of two new instances is carried out in Section 4.3 with large first-stage variables. Next, their outcomes of the performance against the CPLEX barrier algorithm, in linear and quadratic versions (Subsection 4.3.3 and 4.3.4) are shown. Finally, the conclusions of this research can be found at Chapter 5.

1.2 Contributions

As a part of the contributions of this study, a new methodology for solving Large-scale two-stage stochastic problems with a full splitting technique from Lustig et al. [58], and using the solver `BlockIP` from Castro [18] where the advantage of Preconditioned Conjugate Gradient (PCG) is derived by combining Cholesky's approach for computing directions. Therefore the new structure of the normal equations is considered as an advantage for computing the solution.

Here, the solution of both linear and quadratic formulations up to 39 million variables and six million constraints were solved more efficient than alternative state-of-the-art IP implementations.

1.2.1 Article and presentation

This research was presented in the 30th European Conference on Operational Research (EURO 2019), in Dublin, Ireland from the 23rd to the 26th of June, 2019, co-authored with Prof. Jordi Castro, with the name of *A new interior-point approach for large two-stage stochastic problems.*

Later, a paper was accepted for publication in *Optimization Methods and Software*, titled "A new interior-point approach for large separable convex quadratic two-stage stochastic problems".

Chapter 2

State of the art

2.1 Introduction

In this chapter, a research map to place this thesis in the field of mathematical optimization and computer science is presented. This chapter starts by explaining large scale concepts for establishing the relevance of this work. Second, the mathematical model of two-stage stochastic optimization is described. This model manages decisions in the present (before a random event), taking into consideration possible situations after the random event, which helps decision-makers. Additionally, within this topic, the matrix constraint structure of the model was analyzed, which is an essential part of this research.

Next, we follow with an overview of the bases of different methods used to solve these kinds of problems and the preliminaries of the actual method used in this work, i.e., the interior-point method (IPM). As part of this section, a description of the implementation of IPM with a primal-dual path-following algorithm is included. Moreover, to close the section, some ideas of how IPM deals with the structure of the constraint matrix of stochastic problems.

2.2 Large-Scale Problems

Large-scale problems generally contain a special structure of the pattern of zero and nonzero coefficients in the constraints. This feature then derived in efficient specialized algorithms [14]. When some variables do not appear in common constraints, these variables are independent, so the problem can be separated into smaller or independent subproblems. Hence, the subproblems can be gathered, analyzed, stored, and solve separately and simultaneously. Other cases are the *nearly* independent subsystems as illustrated by the next three structures in Figure 2.1.

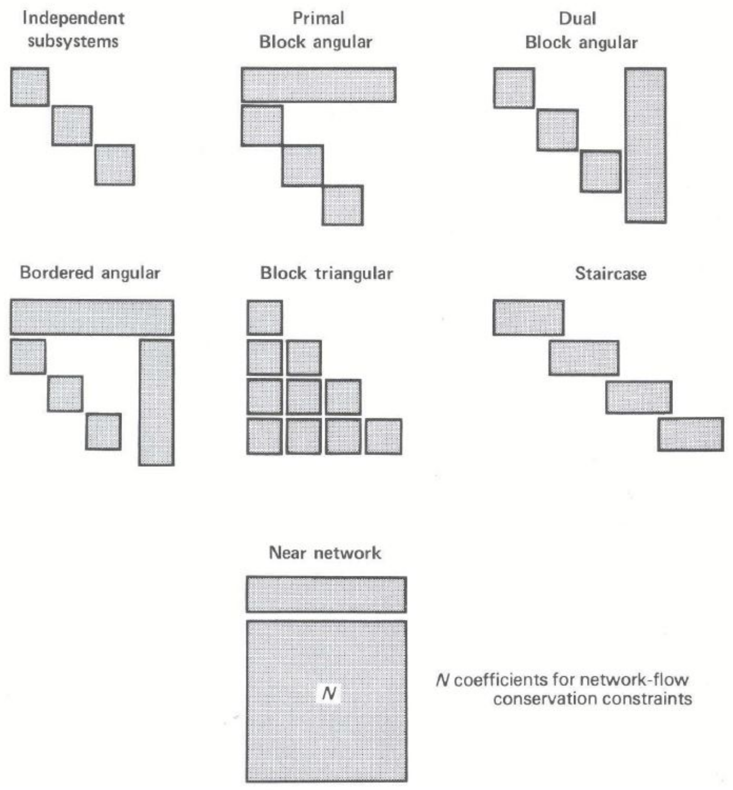


Figure 2.1: Structure of Large-Scale Problems [14]

In the primal block angular structure, subsystem variables that are sharing common resources are *linkings constraints* because they interact with all the constraints. The dual block angular structure has variables that have common activities. The bordered angular system generalizes these models by combining the linking constraints and the common variables. For multistage optimization, the staircases and block triangular structures of Figure 2.1 are frequent. In staircases, some activities are linked by two periods. Finally, in the block triangular case, the decisions in each period affect resource allocation in future periods.

2.3 Two-Stage Stochastic Optimization Model

The general idea behind a stochastic formulation is to choose some initial decisions that want to be optimized, taking into account the expected value of future recourse actions. The problem can be expressed as a two-stage or multistage model.

This research focuses on the two-stage stochastic problems (TSSP) with either linear or convex quadratic objective functions. This formulation computes some initial (first-stage) decisions that must be made before the occurrence of uncertainty, taking into account the expected value of future recourse (second-stage) actions.

The *first-stage problem* refers to so-called "here-and-now" decisions, and the *second-stage problem* has the "wait-and-see" decisions, which represent the recourse actions. It deals with the uncertainty by analyzing possible outcomes (different scenarios that consider random events –more than one–) [11].

The convex quadratic two-stage stochastic problem can be formulated in standard form as

$$\begin{array}{ll} \min\limits_{x} & c^\top x + \frac{1}{2}x^\top F x + \mathcal{Q}(x) \\ \text{s. to} & Mx = b \\ & u_x \geq x \geq 0, \end{array} \tag{2.1}$$

where

$$\mathcal{Q}(x) = E_\xi[Q(x,\xi)] \quad \text{and} \quad \begin{array}{rl} Q(x,\xi) = \min\limits_{y} & q_\xi^\top y + \frac{1}{2}y^\top G_\xi y \\ \text{s. to} & Wy = h_\xi - T_\xi x \\ & u_y \geq y \geq 0. \end{array} \tag{2.2}$$

The variables $x \in \mathbb{R}^{n_x}$ and $y \in \mathbb{R}^{n_y}$ are, respectively, the first- and second-stage decisions. The number of first- and second-stage constraints are, respectively, m_x and m_y. The first-stage vectors and matrices are $c \in \mathbb{R}^{n_x}$, $F \in \mathbb{R}^{n_x \times n_x}$, $b \in \mathbb{R}^{m_x}$, $M \in \mathbb{R}^{m_x \times n_x}$, where F is symmetric and positive definite. $\mathcal{Q}(x)$, known as the recourse function, is the future average cost of our second-stage decisions, for all scenarios (i.e., for all realizations of ξ). In the second-stage, $q_\xi \in \mathbb{R}^{n_y}$, $G_\xi \in \mathbb{R}^{n_y \times n_y}$, $W \in \mathbb{R}^{m_y \times n_y}$, $T_\xi \in \mathbb{R}^{m_y \times n_x}$, and $h_\xi \in \mathbb{R}^{m_y}$; G_ξ is symmetric and positive definite. The stochastic random vector is comprised of q_ξ, G_ξ, h_ξ and T_ξ. From now on, the notation will be simplified by eliminating the subscript ξ.

2.3.1 Extensive form

For some particular problems, a closed-form solution can be obtained for $Q(x,\xi) = q^\top y^* + \frac{1}{2}{y^*}^\top G y^*$, where y^* is the optimum of the second-stage future decisions. In these cases, it is possible to compute $\mathcal{Q}(x) = E_\xi[Q(x,\xi)]$, which allows for the solution of (2.1) only in terms of the first-stage decisions.

In general, however, no closed-form exists for $Q(x,\xi)$, which forces us to use the extensive form of the stochastic problem. For this purpose, let us consider that ξ is a discrete random variable of k values $\xi_1,\ldots,\xi_k$ with probabilities $p_1,\ldots,p_k$ (if ξ is continuous, it must be previously discretized). Each particular value ξ_i, $i = 1,\ldots,k$ is usually known as a scenario. Next, second-stage variables and constraints are replicated for each scenario (i.e., $y_i, i = 1, \dots, k$ for variables), combining problems (2.1) and (2.2) into a single one, as follows:

$$\begin{array}{rl} \min\limits_{x,y_i} & c^\top x + \frac{1}{2} x^\top F x + \sum\limits_{i=1}^{k} p_i \left(q_i^\top y_i + \frac{1}{2} y_i^\top G_i y_i \right) \\ \text{s. to} & Mx = b \\ & 0 \le x \le u \\ & \left. \begin{array}{l} T_i x + W y_i = h_i \\ 0 \le y_i \le u_i \end{array} \right\} i = 1, \dots, k. \end{array} \tag{2.3}$$

The dual formulation of this problem would be:

$$\begin{array}{rl} \max\limits_{z_i,\mu,x} & b^\top \lambda + \sum\limits_{i=1}^{k} h_i^T \lambda_i - \frac{1}{2} \left(x^T F x + \sum\limits_{i=1}^{k} p_i y_i^\top G_i y_i \right) - u^T \mu_u - \sum\limits_{i=1}^{k} u_i^T \mu_{u_i} \\ \text{s. to} & M^\top \lambda + \sum_{i=1}^{k} T_i^\top \lambda_i - F x + \mu - \mu_u = c \quad i = 1, \dots, k. \\ & \left. \begin{array}{l} W^\top \lambda_i - p_i G y_i + \mu_i - \mu_{u_i} = p_i q_i \\ \mu, \mu_i, \mu_u, \mu_{u_i} \ge 0 \end{array} \right\} i = 1, \dots, k, \end{array} \tag{2.4}$$

where $\lambda \in \mathbb{R}^{m_x}$ and $\lambda_i \in \mathbb{R}^{m_y}$ are the Lagrange multipliers of the equality constraints, μ and μ_i are the multipliers of the lower bounds of x and y_i, and μ_u and μ_{u_i} are the multipliers of the upper bounds. F and G_i are symmetric and positive semidefinite [26] [71].

For real problems, (2.3) and (2.4) can be very large and needs to be solved by specialized procedures that exploit the particular problem structure [11, Ch. 5–8], [54, 72].

2.3.1.1 The constraint structure of the two-stage problem

Good matrix management is essential for making efficient procedures for solving two-stage stochastic problems [57]. Particularly, the constraints matrix in (2.3) shows the following dual block-angular structure:

$$A = \begin{array}{c} \begin{array}{ccccc} x & y_1 & y_2 & \cdots & y_k \end{array} \\ \left[\begin{array}{ccccc} M & & & & \\ T_1 & W & & & \\ T_2 & & W & & \\ \vdots & & & \ddots & \\ T_k & & & & W \end{array} \right] \end{array}, \tag{2.5}$$

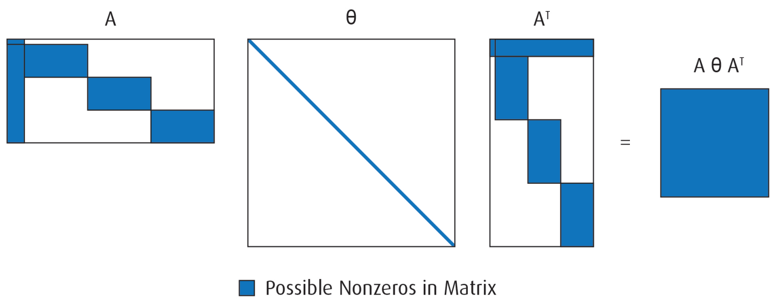


Figure 2.2: Multiplication of $A\Theta T^T$ [59]

where T_{ξ_i} in (2.3) has been renamed as T_i to simplify the notation.

The dual problem of the TSSP in the extensive form (2.4) has a constraint matrix (2.6) that can be classified as a primal block angular structure. For matching with Figure 2.1, the linkings constraints must move to the uppermost of the constraints.

$$A = \begin{array}{c} \begin{array}{ccccc} z & z_1 & z_2 & \cdots & z_k \end{array} \\ \left[\begin{array}{ccccc} & W^T & & & \\ & & W^T & & \\ & & & W^T & \\ \vdots & & & \ddots & \\ M & T_1 & T_2 & \dots & T_k \end{array}\right] \end{array}, \tag{2.6}$$

where every block contains all corresponding variables per block.

The advantage of the dual block-angular structure (2.5) is the reduction of the fill-in matrices in an IPM. As shown in Figure 2.2, the linking columns associated with the first-stage variables gives rise to large fill-in when computing the Cholesky factorization of the normal equations $PA\Theta A^\top P^\top$ in an IPM—with Θ being a positive diagonal matrix and P a row permutation matrix. We note that F and G_ξ in (2.3) must be diagonal matrices if Θ is assumed to be diagonal; indeed, F and G_ξ are considered diagonal in the specialized IPM of Chapter 3 for reasons of efficiency. Solving the normal equations system constitutes the main computational burden on any IPM. The solution time critically depends upon preserving the sparsity in matrix A [59] [71].

2.4 Solution methods

Several algorithms have been developed for solving optimization problems that take advantage of the special structure. One classical method of solution is from

Benders [7] for dual block angular, then extended to stochastic optimization as L-shaped method by Van Slyke and Wets [77] and explained in [11]. Later, many techniques were developed looking for an efficient and fast solution. This challenging area now include the large-scale data.

Our technique use an interior-point method based in [51]. It has been discussed for stochastic optimization by Birge and Qi [12] and Lustig et al. [59], among others.

In this section, the author presents the general idea about the L-shaped method and others examples of the currents techniques. Next, for our research, the interior-point method is described with the Primal-Dual version with the path-following algorithm, mostly from [69], [50], and [38]. The last part, this study worked based on the interior-point methods in stochastic optimization by Andrzej Ruszczynski [72] and Birge and Holmes [10].

2.4.1 Review of different approaches

The *L-shaped* is a well-known method in which the basic idea is to approximate the nonlinear term in the objective function $\mathcal{Q}$ (*the recourse function*) by using an outer linearisation. This term involves a solution of all second-stage recourse linear programs. This approach creates a master problem with this term in x (first-stage variables), but the recourse function is only evaluated as a subproblem [11]. For this approach, the *deterministic equivalent problem* or *extensive form* (Subsection 2.3.1), and the structure in Figure 2.1 (Dual block angular) is used.

Another technique is the Lagrangean Decomposition (LD) that solves two-stage problems. It focuses on using Lagrangian relaxation to obtain good bounds. A solution for problems with integer variables could be found in [29]. Here, Escudero et al. solve the optimization instances with a splitting variable in the Deterministic Equivalent Model 2.3.1. More implementations can be found in [28] for multistage problems.

Regularization is another technique that can be implemented. FortSP is a commercial solver that implements variants of Benders decomposition with different kinds of regularization methods applied in the expected resource, the trust-region or feasibility issues reported by Zverovich et al. [81]. For example, one of these approaches (expected resources) is called Level Method by Lemaréchal et al. [55]. It is an iterative method that uses level sets for model function for regularization. When the two-stage stochastic problems have a network recourse the method improves considerably. For every n (number of variables) steps add a new accurate digit in the estimate of the optimum [21]. For more details refer to [27].

The Primal-Dual column generation technique (PDCGM) is implemented by Gondzio et al. [41] [40]. This method finds a suboptimal well-centered dual solution. Assisted by a primal-dual IPM and adjusting dynamically the tolerance to solve the restricted master problems. The PDCGM was tested in large-scale instances as two-stage stochastic optimization problems with a good solution (number of iterations and CPU times). For the results of the applications with a small number of first-stage variables [40].

Another software package, called DSP, is an algorithm that follows the dual decomposition method with cuts to tighten Lagrangian subproblems for problems with not necessarily have relatively complete recourse [53]. It can deal with integer variables and is applied in two-stage stochastic programs. This technique includes an interior-point cutting-plane method for solving the Lagrangian master problem. Some results can be seen in Section 4.2.6 through *Neos Server* [67].

Furthermore, there is another branch of optimization methods that work with heuristic algorithms to reach a good solution in a short time.

2.4.2 Interior-point method

The approach that is presented in this research uses an interior-point method (IPM) with a primal-dual path-following algorithm. To achieve a better understanding of the methodology, the IPM is formally described.

The name IPM is based precisely on the points that are generated by these algorithms are inside the feasible region (Figure 2.3). With these methods, we could

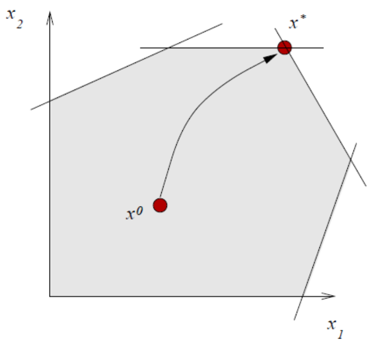


Figure 2.3: Interior-point method [17]

find the solution by a *Linear search.* Here the algorithms started with either feasible or infeasible, interior-point $x^0 \in \mathbb{R}^n$. Then generate an iterative sequence of points until reach or converge to the optimal (meets the conditions of optimal). In this way, we would obtain each new point from the previous one with the next process;

$$x^{k+1} = x^k + \alpha^k \Delta x^k, \quad \alpha^k \geq 0 \tag{2.7}$$

where $\Delta x^k \in \mathbb{R}^n$ and represents the movement direction, while α^k is the step length.

There has been a substantial number of contributions since Karmarkar's method [51]. These variants can be divided into four general methods [24] such as:

1. Projective. A projective transformation in a new space of variables proposed by Karmarkar [51].
2. Affine scaling. In this method, a transformation of the original problem by a similar scaling of the variables was created by Dikin in 1967 [25].
3. Path-following. Consisting of small-step methods such as one accomplished by Gill et al. [36]
4. Affine potential reduction. This procedure search for the reduction of a potential function. Proposed by Anstreicher and Bosch [1].

These methods can be applied to the primal, dual, or both approaches. The primal algorithm maintains primal feasibility while iterating toward dual feasibility and reducing the duality gap. Conversely, in the dual algorithm, the dual feasibility is maintained and then iterate toward primal feasibility. The primal-dual algorithm, in the theoretical versions, maintains both primal and dual feasibility while iterating to reduce the duality gap [3] [57].

The primal-dual has been used very efficiently on large problems. Hence these methods are the focus of this section with a description of the *primal-dual Path-following* algorithm. It must be recalled that the practical implementations have an infeasible starting point and infeasible iterations [69] as `BlockIP` that is explained in Chapter 3.

2.4.2.1 Primal-Dual Methods

In this section, we will examine a general primal-dual convex quadratic programming method with the primal (P) representation as:

$$\begin{aligned} (P) \quad \min \quad & c^T x + \frac{1}{2} x^T Q x \\ \text{so t.} \quad & Ax = b, \\ & 0 \leq x \leq u \end{aligned}$$

Where A is an $m \times n$ full row rank matrix, Q is an $n \times n$ positive semidefinite matrix, c, x, and u are vectors in $\mathbb{R}^n$ and b, is a vector in $\mathbb{R}^m$. When $Q = 0$, the problem becomes lineal. The solution associated with both problems P and D must satisfy the Karush-Kuhn-Tucker conditions (KKT) that are associated with the following equations:

$$\begin{aligned} A^T\lambda + z - v - Qx &= c, && (2.8a)\\ Ax &= b, && (2.8b)\\ XZe &= \mu e, && (2.8c)\\ SVe &= \mu e, && (2.8d)\\ (x, s, z, v) &\geq 0 && (2.8e) \end{aligned}$$

Where $\lambda \in \mathbb{R}^m$ are the Lagrange multipliers of the equality constraints, z and $v \in \mathbb{R}^n$ are the Lagrange multipliers for the lower and upper bounds; X, Z, S, and V are diagonal matrices in $\mathbb{R}^n$ of their respective vectors when $s = u - x$ and e is the vector of ones. μ is the current duality measure.

The parameter μ is gradually driven to zero throughout the iterations. A family of solutions for each μ is known as a *central path* and when $\mu \to 0$ the solution converges to the optimum. The algorithm terminates when $\mu \leq \epsilon$ for some ϵ. This procedure is called a primal-dual path-following interior-point method.

In the case of path-following primal-dual interior-point methods, the KKT equations (2.8) can be considered as a mapping H.

$$H(x, \lambda, z, v; \mu) = \begin{bmatrix} -Qx + A^T\lambda + z - v - c \\ Ax - b \\ XSe - \mu e \\ SVe - \mu e \end{bmatrix} = 0, \qquad (x, z, v) \geq 0, \tag{2.9}$$

With $(x, z, v) \geq 0$, however, $(x^k, \lambda^k, z^k, v^k)$ are strictly greater than zero at every iteration, hence the name Interior-Points Methods and usually apply the Newton method to solve the KKT equations (2.8). Therefore, compute the Newton direction and make one step in this direction before reducing the barrier parameter μ.

For finding the search direction $(\Delta x, \Delta\lambda, \Delta z, \Delta v)$ the following system can be solved with Newton's method for non-linear problems.

$$J(x, \lambda, s) \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta z \\ \Delta v \end{bmatrix} = -H(x, \lambda, s) \tag{2.10}$$

Where J is the Jacobian of H. So is computed by solving the following system of linear equations:

$$\begin{bmatrix} A & 0 & 0 & 0 \\ -Q & A^T & I_n & -I_n \\ Z & 0 & X & 0 \\ -V & 0 & 0 & S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta z \\ \Delta v \end{bmatrix} = \begin{bmatrix} b - Ax \\ c + Qx - A^T\lambda - z + v \\ \sigma\mu e - XZe \\ \sigma\mu e - SVe \end{bmatrix} \tag{2.11}$$

Where I_n denotes an identity matrix of dimension n, and σ is a centering parameter.

The next iteration will be:

$$(x, \lambda, z, v) + \alpha(\Delta x, \Delta\lambda, \Delta z, \Delta v) \tag{2.12}$$

Where α is called the *Step Length* and is conveniently chosen to keep the interior point condition $(x, \lambda, z, v) > 0$.

The choices of the centering parameter σ and the step length α are crucial to the performance of the method. Different techniques give rise to a wide variety of algorithms. In the primal-dual path-following interior-point algorithm, the μ parameter is reduced at each iteration.

2.4.2.2 Primal-Dual path-following algorithm

The general framework for such methods is as follows:

step 0. Find $(x^0, \lambda^0, z^0, v^0)$ interior point with $(X^0, Z^0, V^0) > 0$;
Let $k := 0$

step 1. Choose $\sigma_k \in [0, 1]$ and solve for the direction. (normally $\sigma = .01$)

$$\begin{bmatrix} A & 0 & 0 & 0 \\ -Q & A^T & I_n & -I_n \\ Z & 0 & X & 0 \\ -V & 0 & 0 & S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta z \\ \Delta v \end{bmatrix} = \begin{bmatrix} b - Ax \\ c + Qx - A^T\lambda - z + v \\ \sigma\mu e - XZe \\ \sigma\mu e - SVe \end{bmatrix}$$

where $\mu_k = \sigma \frac{\textit{duality gap}}{n}$; (*gap dual*, distance to the current point between primal and dual functions) *step 2.* Set

$$(x^{k+1}, \lambda^{k+1}, z^{k+1}, v^{k+1}) = (x^k, \lambda^k, z^k, v^k) + \alpha_k(\Delta x^k, \Delta\lambda^k, \Delta z^k, \Delta v^k), \qquad (2.13)$$

choosing α_k so that $(x^{k+1}, z^{k+1}, v^{k+1}) > 0$

If $\frac{||\xi_p^k||}{1+||b||} \leq \epsilon_p, \frac{||\xi_d^k||}{1+||c||} \leq \epsilon_d$ and $\frac{(x^k)^T s^k/n}{1+|c^T x^k + 1/2(x^k)^T Q x^k|} \leq \epsilon_o$,

stop; else return to step 1 with $k := k+1$.

$\epsilon_p, \epsilon_d, \epsilon_o$ primal feasibility, dual feasibility and optimality tolerance

Step Lengths

One aspect for knowing how to move from one point to the next one is the step lengths α as in (2.12), There are two general methods for calculating this step. One is the *Short-step* that iterates in a narrow neighborhood of central path. The second is called the *Long-Step* and iterate in a wide neighborhood of central path. In practice, the second one is more efficient and of more frequent use.

The general idea for the Long-Step is to move the maximum movement possible in x, z, and v variables without violating nonnegativity. To ensure this condition, a step length of slightly less than this maximum (not greater than 1).

One approach for choosing α is $\alpha = min(\alpha_\tau^{pri}, \alpha_\tau^{dual})$, where

$$\alpha_\tau^{pri} = max\{\alpha \in (0,1] : x + \alpha\Delta x \geq (1-\tau)x\}, \qquad (2.14a)$$

$$\alpha_\tau^{dual} = max\{\alpha \in (0,1] : (z,v) + \alpha(\Delta z, \Delta v) \geq (1-\tau)(z,v)\} \qquad (2.14b)$$

The parameter $\tau \in (0,1)$ controls how far from the maximum step it can moves.

Another approach used is the maximum step length such that $(x, z, v) \geq 0$, and later it is reduced by parameter $\rho \in [0.95; 0.99995]$.

Computing the direction

To get an efficient method, the system (2.11) for the search direction must solve efficiently in every iteration, since it is the most expensive computational step. Typically, in most applications, this coefficient matrix is large, sparse, and has a special structure, thence a compact system could be reformulated with symmetric nonsingular coefficient matrices.

Let us rewrite the system (2.11) with r_b, r_c and r_{xs} notation:

$$\begin{bmatrix} A & 0 & 0 & 0 \\ -Q & A^T & I_n & -I_n \\ Z & 0 & X & 0 \\ -V & 0 & 0 & S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta z \\ \Delta v \end{bmatrix} = \begin{bmatrix} -r_b \\ -r_c \\ -r_{xs} \\ -r_{sv} \end{bmatrix} \tag{2.15}$$

Where $r_b = Ax - b$, $r_c = -Qx + A^T\lambda + z - v - c$, $r_{xs} = XSe - \sigma\mu e$ and $r_{sv} = SVe - \sigma\mu e$. Moreover, in practical implementations, r_{xs} and r_{sv} may enclose a predictor, corrector, and centering contribution that has a significant effect on the performance. When no much progress can be made along one direction, σ is the centering parameter; a larger value of σ will ensure that the next iterate is more centered, so in the next point, a long step will be possible. Further details can be observed in [61].

The reformulation creates a more compact system. Since x, z, and v are strictly positive, the diagonal matrices X, Z, and V are nonsingular. The first step is to isolate Δz and Δv (2.16). Then replaced them in the second block of equations in (2.15).

$$\begin{aligned} \Delta_z &= X^{-1}r_{xz} - X^{-1}Z\Delta x \\ \Delta_v &= S^{-1}r_{sv} + S^{-1}V\Delta x \end{aligned} \tag{2.16}$$

The result is known as the *augmented system* (2.17)

$$\begin{bmatrix} -\Theta^{-1} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -r_c + S^{-1}r_{sv} - X^{-1}r_{xz} \\ -r_b \end{bmatrix} \tag{2.17}$$

Where $\theta = (Q + ZX^{-1} + VS^{-1})^{-1}$. The next step solves (2.17), First Δx is found in the first block of the augmented system (2.18b). Then replace it in the second block of equations (2.18a). Hence the *normal equations* (2.18) are obtained.

$$(A\Theta A^T)\Delta\lambda = r_b + A\Theta r, \tag{2.18a}$$

$$\Delta x = \Theta(A^T\Delta\lambda - r) \tag{2.18b}$$

where $r = r_c + S_{-1}r_{sv} - X_{-1}r_{xz}$

For the factorization of $A\Theta A^T$ in (2.18a), usually, the Cholesky algorithm is implemented. Then perform a triangular substitution to solve the resulting sparse factors and obtain $\Delta\lambda$ in (2.18a). Δx is recovered from (2.18b) and go on to the equations before (2.16) accordingly. In cases when $A\Theta A^T$ is ill-conditioned of singular, some modifications to the Cholesky algorithm are needed.

Nevertheless, if normal-equations (2.18) contain A matrix with any dense columns, the factorization of $A\Theta A^T$ will be dense; as a result, the implementation may be complicated and slower. Therefore, the fill-in columns are usually avoided. Unfortunately, the structure of stochastic linear models can lead to quite dense systems, limiting the use of the interior-points method for the solution. On account of that situation, some methods for improving the efficiency of solving the linear system associated with (two-stage) stochastic problems have been developed [10].

2.5 IPM for Stochastic Optimization

The optimization problems of the two-stage extensive form has a dual block angular structure (2.5) that potentially has many dense columns. Also, the density of the matrix ($A\Theta A^T$) largely depends on the number of dense columns contained in this original matrix A [10]. To see this, let $\Theta_k \in \mathbb{R}^{m_k \times m_k}$ be defined by Θ_k, $s = 0, \ldots, K$. Suppose further that $T_K = T$. Then solving the system requires a factorization of

$$A\Theta A^T = \begin{bmatrix} M & & & \\ T & W & & \\ \vdots & & \ddots & \\ T & & & W \end{bmatrix} \begin{bmatrix} \Theta_0 & & & \\ & \Theta_1 & & \\ & & \ddots & \\ & & & \Theta_K \end{bmatrix} \begin{bmatrix} M^T & T^T & \ldots & T^T \\ & W^T & & \\ & & \ddots & \\ & & & W^T \end{bmatrix}$$

$$= \begin{bmatrix} M\Theta_0 M^T & M\Theta_0 T^T & M\Theta_0 T^T & \ldots & M\Theta_0 T^T \\ T\Theta_0 M^T & T\Theta_0 T^T + W\Theta_1 W^T & T\Theta_0 T^T & \ldots & T\Theta_0 T^T \\ T\Theta_0 M^T & T\Theta_0 T^T & T\Theta_0 T^T + W\Theta_2 W^T & \ldots & T\Theta_0 T^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ T\Theta_0 M^T & T\Theta_0 T^T & T\Theta_0 T^T & \ldots & T\Theta_0 T^T + W\Theta_K W^T \end{bmatrix} \qquad (2.19)$$

Noticeably, the first-stage variables (T columns) creates a dense matrix to factorize. There are many ways to reduce fill-in and improve solutions times. Of some of the methods proposed for finding a solution, four of them will be mentioned in these sections, but the focus will be on the first one, the *splitting formulation* because it is one of the main ideas in this research.

Thus, the first method is called *splitting formulation* or *Reformulation of the program*. It is created by Lustig et al. [59]. Here, the dense column associated with the first-stage decisions x is replicated. This means that $x = x_i$ or $x_i = x_{i+1}$, $i = 1, ..., k$ constraints are added (partial or full splitting respectively). These constraints satisfy the nonanticipativity requirement (invariant across scenarios).

Although, the problem increases its size, the performance can be faster. Furthermore, the effectiveness of this method depends on the size and forms of the first-stage coefficient matrices. For more details, please refer to Subsection 2.5.1.

The second method generated by Arantes and Birge [2] proposes a reformulation of the primal problem in the polyhedral inequality form. This method, consider that $A^T \Theta A$ is sparser than $A \Theta A^T$, therefore, it has a more efficient performance. It is later proved in [9] that both matrices could be interchangeable. Also, it is showed that solving the dual of the extensive form approach with a matrix B of the form (2.6), may be larger than the coefficient matrix of the primal, the matrix $B \Theta B^T$ enables an efficient Cholesky factorization. However, if W is very large or dense, solving $B \Theta B^T$ may not be as efficient.

$$B \Theta B^T = \begin{bmatrix} M^T \Theta_0 M + \sum_{k=1}^{K} T^T \Theta_s T & T^T \Theta_1 W & \dots & T^T \Theta_K W \\ W^T \Theta_1 T & W \Theta_1 W^T & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ W^T \Theta_K T & 0 & 0 & W \Theta_K W^T \end{bmatrix} \tag{2.20}$$

Another method is called the *Schur complement* described in [59] where the matrix A is partitioned into a sparse part and a dense part as follows:

$$A = [A_d | A_k] \tag{2.21}$$

In the two-stage stochastic programs are:

$$A_d = \begin{bmatrix} M \\ T_1 \\ \vdots \\ T_K \end{bmatrix}, \qquad A_k = \begin{bmatrix} W & & \\ & \ddots & \\ & & W \end{bmatrix} \tag{2.22}$$

which involves solving a small, dense matrix derived from a larger sparse matrix. In the case of a two-stage dual block angular program, A_d contains the first-stage variables, and A_k is the columns of the second-stage. So $A \Theta A^T = A_k \Theta_k A_k^T + A_d \Theta_d A_d^T$. In general, for the sparse part, Cholesky's method is used, and for the dense part, the *Schur's complement* is applied. When $A_d \Theta_d A_d^T$ remains sparse, and the numbers of the dense columns are small; this method could be efficient. The downside is that as the number of dense columns grows large, the effort needed to solve the Schur complement drastically increases.

Another approach is an explicit factorization of the dual block angular programs proposed by Birge and Qi ([12]). Here the matrix $A \Theta A^T$ may be written as the sum of a block diagonal matrix Θ and the product of two similar matrices U and V *Sherman-Morrison-Woodbury formula* may be used to find the inverse of the matrix.

$$U = \begin{bmatrix} M & I \\ T_1 & 0 \\ \vdots & \vdots \\ T_K & 0 \end{bmatrix}, \qquad V = \begin{bmatrix} M & -I \\ T_1 & 0 \\ \vdots & \vdots \\ T_K & 0 \end{bmatrix} \tag{2.23}$$

The last method described in this section is the *augmented system* (2.17), that solves Δx and $\Delta\lambda$ together by computing a sparse factorization for the entire coefficient matrix. This technique allows more freedom in the pivot sequence. In the case of two-stage stochastic problems the cost of factoring the indefinite matrix could be lower than the cost of factoring the positive semi-definite matrix in the normal equations (2.18a) [22]. This is because that A allows the following partition:

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \tag{2.24}$$

Where A_1 is sparse with no dense columns or rows, represent by the second-stage decision variables, A_2 are the first-stage decision variables with more density of A_1, and A_3 is empty [62] [60] (details in Symmetric quasi-definite matrices [78]). One variant proposes by Mészáros is a dynamical factorization on the augmented system with 1×1 pivots and without requiring the quasi-definite property. This attempt proves that the pivot sequence is stable and very competitive.

2.5.1 Splitting formulation

This method [59] split up the first-stage decisions x for reducing the fill-in created by the dense column associated with them. Therefore, The linking columns ensure that these x variables are feasible under each future scenario, so the second-stage decisions $y_i, i = 1, \ldots, k$, are depending on both the first-stage variables and the realization of stochastic events.This technique has been tested in general interior-point methods with good results [22].

Two variants were attempted: full and partial splitting. The full version split all first-stage variables across all scenarios, creating a staircase structure of all the linking constraints inside `BlockIP`. While in partial splitting only the dense first-stage variables were replicated. Thus, we will make use of full splitting, which is more appropriate than partial splitting for the specialized interior-point algorithm in this work because the structure of the linking constraints causes more highly sparse (but also much larger) matrices. This is different from general interior-point solvers, where partial splitting was shown to be superior in [59]. Figure 2.4 shows the effect of splitting on the normal equations matrix. We can compare the effects without splitting in Figure 2.2

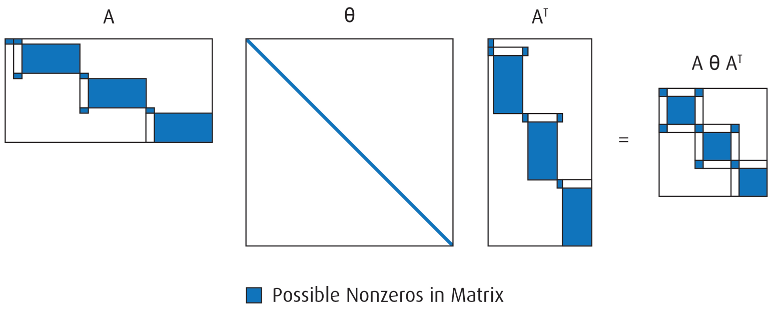


Figure 2.4: Multiplication of $A\Theta T^T$ with splitting [59]

Replicating the first-stage variables for each scenario one obtains the following equivalent extensive form formulation:

$$\begin{aligned}
\min_{x_i,y_i} \quad & c^\top x_1 + \frac{1}{2}x_1^\top F x_1 + \sum_{i=1}^{k} p_i \left(q_{\xi_i}^\top y_i + \frac{1}{2} y_i^\top G_{\xi_i} y_i \right) \\
\text{s. to} \quad & M x_1 = b \\
& u_x \geq x_1 \geq 0 \\
& \left.\begin{aligned} T_{\xi_i} x_i + W y_i &= h_{\xi_i} \\ u_y \geq y_i &\geq 0 \end{aligned}\right\} i = 1, \ldots, k \\
& \left.\begin{aligned} x_1 - x_i &= 0 \\ u_x \geq x_i &\geq 0 \end{aligned}\right\} i = 2, \ldots, k.
\end{aligned} \tag{2.25}$$

Reordering columns by the number of scenarios, the constraint matrix in (2.25) can be written as:

$$A_{(2.25)} = \begin{array}{c} \begin{array}{cccccccc} x_1 & y_1 & x_2 & y_2 & \cdots & x_k & y_k & \end{array} \\ \left[\begin{array}{cccccccc} M & & & & & & & \\ T_1 & W & & & & & & \\ & & T_2 & W & & & & \\ & & & & \ddots & & & \\ & & & & & T_k & W & \\ I & & -I & & & & & \\ \vdots & & & & \ddots & & & \\ I & & & & & -I & & \end{array}\right] \end{array}. \tag{2.26}$$

Alternative formulations can be obtained by different linking constraints that force the same values for copies of the first-stage variables. For instance, another equivalent problem is obtained by replacing the last group of constraints in (2.25) with $x_i = x_{i+1}, i = 1, \ldots, k-1$, thus obtaining

$$\begin{aligned}
\min_{x_i, y_i} \quad & c^\top x_1 + \frac{1}{2} x_1^\top F x_1 + \sum_{i=1}^{k} p_i \left(q_{\xi_i}^\top y_i + \frac{1}{2} y_i^\top G_{\xi_i} y_i \right) \\
\text{s. to} \quad & M x_1 = b \\
& u_x \geq x_1 \geq 0 \\
& \left.\begin{array}{l} T_{\xi_i} x_i + W y_i = h_{\xi_i} \\ u_y \geq y_i \geq 0 \end{array}\right\} i = 1, \ldots, k \\
& \left.\begin{array}{l} x_i - x_{i+1} = 0 \\ u_x \geq x_i \geq 0 \end{array}\right\} i = 1, \ldots, k-1.
\end{aligned} \tag{2.27}$$

The constraints matrix of formulation (2.27) is

$$A_{(2.27)} = \begin{array}{c} \begin{array}{cccccccccccc} x_1 & y_1 & x_2 & y_2 & x_3 & y_3 & \cdots & x_{k-1} & y_{k-1} & x_k & y_k \end{array} \\ \left[\begin{array}{ccccccccccc}
M & & & & & & & & & & \\
T_1 & W & & & & & & & & & \\
 & & T_2 & W & & & & & & & \\
 & & & & T_3 & W & & & & & \\
 & & & & & & \ddots & & & & \\
 & & & & & & & T_{k-1} & W & & \\
 & & & & & & & & & T_k & W \\
I & & -I & & & & & & & & \\
 & & I & & -I & & & & & & \\
 & & & & & & \ddots & & & & \\
 & & & & & & & I & & -I &
\end{array}\right] \end{array}. \tag{2.28}$$

Although (2.25) and (2.27) are equivalent, the latter is computationally more efficient for the specialized IPM that will be used, as will be shown in Subsection 3.4.2.

Both constraint matrices (2.26) and (2.28) have a primal block-angular structure with $(k-1)n_x$ very sparse linking constraints. The linear optimization problems

(2.25) and (2.27) match the following general block-angular formulation:

$$\begin{aligned}
\min_{x^1,\dots,x^k} \quad & \sum_{i=1}^{k} \left({c^i}^\top x^i + \frac{1}{2} {x^i}^\top Q^i x^i \right) \\
\text{s. to} \quad & \begin{bmatrix} N_1 & & & & \\ & N_2 & & & \\ & & \ddots & & \\ & & & N_k & \\ R_1 & R_2 & \dots & R_k & I \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^k \\ x^0 \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^k \\ b^0 \end{bmatrix} \\
& 0 \leq x^i \leq u^i \quad i = 1, \dots, k \\
& 0 \leq x^0 \leq b^0.
\end{aligned} \tag{2.29}$$

For both formulations (2.25) and (2.27) $x^i, u^i \in \mathbb{R}^{n_x+n_y}$ are, respectively, $(x_i^\top \;\; y_i^\top)^\top$ and $(u_x^\top \;\; u_y^\top)^\top$ $i = 1, \dots, k$; x^0 are the slacks of the linking constraints (they are zero since linking constraints are equalities in the splitting formulation of two-stage stochastic problems). The linear cost vectors $c^i \in \mathbb{R}^{n_x+n_y}$ are defined as $c^1 = (c^\top \;\; p_1 q_{\xi_1}^\top)^\top$, whereas $c^i = (0 \;\; p_i q_{\xi_i}^\top)^\top$ for $i = 2, \dots, k$. The quadratic cost matrices $Q^i \in \mathbb{R}^{(n_x+n_y)\times(n_x+n_y)}$ are $Q^1 = \begin{bmatrix} F & \\ & G_{\xi_1} \end{bmatrix}$, and $Q^i = \begin{bmatrix} \mathbf{0} & \\ & G_{\xi_i} \end{bmatrix}$ for $i = 2, \dots, k$; we assume Q^i are diagonal matrices. The right-hand-side terms are $b^1 \in \mathbb{R}^{m_x+m_y} = (b^\top \;\; h_{\xi_1}^\top)^\top$, while $b^i \in \mathbb{R}^{m_y} = h_{\xi_i}$ $i = 2, \dots, k$, and $b^0 \in \mathbb{R}^{(k-1)n_x} = 0$. As for the diagonal blocks, $N_1 \in \mathbb{R}^{(m_x+m_y)\times(n_x+n_y)}$ is $\begin{bmatrix} M & \\ T_1 & W \end{bmatrix}$, and $N_i \in \mathbb{R}^{m_y\times(n_x+n_y)}$, are $[T_i \;\; W]$, $i = 2, \dots, k$.

Linking constraints for (2.25) are defined by

$$R_1 = \begin{bmatrix} I_1 & 0_1^y \\ \vdots & \vdots \\ I_{k-1} & 0_{k-1}^y \end{bmatrix} \qquad R_i = \begin{bmatrix} 0_1^x & 0_1^y \\ \vdots & \vdots \\ 0_{i-2}^x & 0_{i-2}^y \\ -I_{i-1} & 0_{i-1}^y \\ 0_i^x & 0_i^y \\ \vdots & \vdots \\ 0_{k-1}^x & 0_{k-1}^y \end{bmatrix} i = 2, \dots, k, \tag{2.30}$$

where $0_i^x, I_i \in \mathbb{R}^{n_x\times n_x}$, $0_i^y \in \mathbb{R}^{n_x\times n_y}$, and the subindex denotes the block row position.

For (2.27) the linking constraints are defined by

$$R_1 = \begin{bmatrix} I_1 & 0^y_1 \\ 0^x_2 & 0^y_2 \\ \vdots & \vdots \\ 0^x_{k-1} & 0^y_{k-1} \end{bmatrix} \qquad R_i = \begin{bmatrix} 0^x_1 & 0^y_1 \\ \vdots & \vdots \\ 0^x_{i-2} & 0^y_{i-2} \\ -I_{i-1} & 0^y_{i-1} \\ I_i & 0^y_i \\ 0^x_{i+1} & 0^y_{i+1} \\ \vdots & \vdots \\ 0^x_{k-1} & 0^y_{k-1} \end{bmatrix} \qquad R_k = \begin{bmatrix} 0^x_1 & 0^y_1 \\ \vdots & \vdots \\ 0^x_{k-2} & 0^y_{k-2} \\ -I_{k-1} & 0^y_{k-1} \end{bmatrix}, \tag{2.31}$$

$$i = 2, \ldots, k-1,$$

where I_i, 0^x_i and 0^y_i have the same dimensions as above. We will denote as m_i and n_i the number of rows and columns of each diagonal block N_i, and by l the number of linking constraints. The next section outlines the specialized interior-point approach for the efficient solution of (2.29).

Chapter 3

Specialized interior point

3.1 Introduction

In this chapter, we sketch the specialized IPM, implemented in the `BlockIP` package used in this research. It is based on a primal-dual path-following algorithm [80]. It is written in C++ in about 17000 lines of code, created by Castro [18]. This procedure was initially suggested for multicommodity flow problems [15] and later extended to primal block-angular problems [16]. For this research, the extensive form 2.3.1 with the splitting technique 2.5.1 was used. This formulation has a primal block angular structure illustrated in Section3.2. This the format required by `BlockIP`.

Then, the matrix for $A\Theta A^\top$ factorization that is key in the performance of `BlockIP` is described. Also, in Section 3.3, the solution of the normal equations in the `BlockIP` package is illustrated by a sensible combination of Cholesky factorizations [68] and an iterative preconditioned conjugate gradient (PCG). Besides, some details of how the solver works are added. Along with the dual constraint matrix of the quadratic models because we will need it to compare the performances between the splitting idea or without it in that kind of problem.

Finally in Section 3.4, the structure of the E matrix inside the $A\Theta A^\top$ factorization created by using the splitting technique (partial and Full) with the TSSPs shows how the efficient solutions are reached with the specialized interior point method, `BlockIP`. In Appendix B, general features of `BlockIP` are explained.

3.2 The primal block angular problem

The problems in our work have a primal block angular structure, as Figure 2.1. However, the linking constraints are arranged below, like the constraint matrices

(2.26) and (2.28). `BlockIP` utilizes this structure as an input. Hence the model that is needed has the next form:

$$
\begin{array}{ll}
\min & \displaystyle\sum_{i=0}^{k} f_i(x^i) \\
\text{s. to} & \begin{bmatrix} N_1 & & & \\ & \ddots & & \\ & & N_k & \\ L_1 & \dots & L_k & I \end{bmatrix} \begin{bmatrix} x^1 \\ \vdots \\ x^k \\ x^0 \end{bmatrix} = \begin{bmatrix} b^1 \\ \vdots \\ b^k \\ b^0 \end{bmatrix} \\
 & 0 \leq x^i \leq u^i \quad i = 0, \dots, k.
\end{array}
\tag{3.1}
$$

Matrices $N_i \in \mathbb{R}^{m_i \times n_i}$ and $L_i \in \mathbb{R}^{l \times n_i}$, $i = 1, \dots, k$ respectively define the block and linking constraints, k being the number of blocks. $x^i \in \mathbb{R}^{n_i}, i = 1, \dots, k$, are vectors of the variables for each block. Whereas $x^0 \in \mathbb{R}^l$ are the slacks of the linking constraints. Vector $b^i \in \mathbb{R}^{m_i}, i = 1, \dots, k$ is the right-hand-side for each block of constraints and $b^0 \in \mathbb{R}^l$ is for the linking constraints. The upper bounds for each block of variables are defined by $u^i, i = 0, \dots, k$. These are of the form $b^0 - u^0 \leq \sum_{i=1}^k L_i x^i \leq b^0$. Equality linking constraints can be defined by setting $u^0 \approx 0$.

Functions $f_i : \mathbb{R}^{n_i} \to \mathbb{R}$, $i = 0, \dots, k$, are assumed to be convex. Although the specialized IPM to be described is valid for any f_i, here the quadratic function is considered (Linear when $Q = 0$), that is $f_i(x^i) = c^{i^T} x^i + \frac{1}{2} x^{i^T} Q_i x^i$ Where $c^i \in \mathbb{R}^{n_i}$ and $F_i \in \mathbb{R}^{n_i \times n_i}$, $i = 1, \dots, k$, are the linear and quadratic cost for each group of variables. For the slacks, $c^k \in \mathbb{R}^l$ and $Q_k \in \mathbb{R}^{l \times l}$ are considered. Also, it has been restricted to the separable cases where Q_i, $i = 0, \dots, k$ are positive semidefinite diagonal matrices.

The standard problem (3.1) is an optimization problem with $m = \sum_{i=1}^k m_i + l$ constraints and $n = \sum_{i=1}^k n_i + l$ variables.

Considering the problem (3.1) with a quadratic objective function $f_i(x^i)$ and the upper bounds u^i where the inequality $x \leq u$ then $w \in \mathbb{R}^n$ its the vector of Lagrange multiplier, $z = u - x$. W and $Z \in \mathbb{R}^{n \times n}$ the diagonal matrix created with w and z. Hence Θ becomes. (details from 2.4.2.2)

$$\Theta = (Q + ZX^{-1} + WS^{-1})^{-1} \tag{3.2}$$

Since the objective function is separable, Θ is an easily computable diagonal matrix. Exploiting the structure of the constraint matrix A from (3.1), and appropriately partitioning Θ, as follows

$$A = \begin{bmatrix} N_1 & & & \\ & \ddots & & \\ & & N_k & \\ L_1 & \dots & L_k & I \end{bmatrix} \qquad \Theta = \begin{bmatrix} \Theta_1 & & & \\ & \ddots & & \\ & & \Theta_k & \\ & & & \Theta_0 \end{bmatrix},$$

the matrix of system (2.18a) can be rewrite as

$$A\Theta A^\top = \left[\begin{array}{ccc|c} N_1\Theta_1N_1^\top & & & N_1\Theta_1L_1^\top \\ & \ddots & & \vdots \\ & & N_k\Theta_kN_k^\top & N_k\Theta_kL_k^\top \\ \hline L_1\Theta_1N_1^\top & \dots & L_k\Theta_kN_k^\top & \Theta_0 + \sum_{i=1}^k L_i\Theta_iL_i^\top \end{array}\right] = \begin{bmatrix} B & C \\ C^\top & E \end{bmatrix}, \tag{3.3}$$

$B \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$ where $\tilde{n} = \sum_{i=1}^k n_i$, $C \in \mathbb{R}^{\tilde{n}\times l}$ and $E \in \mathbb{R}^{s\times s}$ being the blocks of $A\Theta A^\top$, and Θ_i, $i = 0,\dots,k$, the submatrices of Θ associated with the $k+1$ groups of variables in (3.1). Denoting by g the properly right-hand-side of (2.18a), and appropriately partitioning g and $\Delta\lambda$, the normal equations (2.18a) can be written as

$$\begin{bmatrix} B & C \\ C^T & E \end{bmatrix} \begin{bmatrix} \Delta\lambda_1 \\ \Delta\lambda_2 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}. \tag{3.4}$$

3.3 Solving the normal equations with `BlockIP`

Solving the system (3.4), $\Delta\lambda_1$ can be isolated from the first group of equations (3.5). Then substituting it in the second set of equations, (3.6) is obtained.

$$B\Delta\lambda_1 = (g_1 - C\Delta\lambda_2). \tag{3.5}$$

$$(E - C^TB^{-1}C)\Delta\lambda_2 = (g_2 - C^TB^{-1}g_1) \tag{3.6}$$

In `BlockIP`, the system (3.5) is solved by Cholesky factorization for each diagonal block $N_i\Theta^iN_i^T, i = 1,\dots,k$ of B. The next system (3.6) with the *Schur complement*

$$S = E - C^TB^{-1}C, \tag{3.7}$$

of dimension l (number of linking constraints) is calculated by a preconditioned conjugate gradient (PCG). A good preconditioner is essential for the efficient solution of (3.6). The preconditioner obtained in [15] can be applied to any primal block-angular problem [16] and relies on the Schur complement ($S = E - C^T B^{-1} C$) is a P-*regular splitting* where S is symmetric and positive definite, E is nonsingular and $E + C^T B^{-1} C$ is positive definite (more details in [8]). Therefore (3.8) is guarantees

$$\rho(E^{-1}(C^\top B^{-1} C)) < 1, \tag{3.8}$$

To simplify the notation, $\rho(E^{-1}(C^\top B^{-1} C)$ will be referred to as ρ and denotes *the spectral radius* of a matrix (i.e., the maximum absolute eigenvalue). The inverse of the Schur complement $E - C^\top B^{-1} C$ can be computed as the following infinite power series

$$(E - C^T B^{-1} C)^{-1} = \left(\sum_{i=0}^{\infty} (E^{-1}(C^T B^{-1} C))^i \right) E^{-1}. \tag{3.9}$$

This inverse (3.9) is based in the Neuman's series preconditioner (see [15] for a proof)

The preconditioner, an approximation of $(E - C^T B^{-1} C)^{-1}$, is thus obtained by truncating the infinite power series (3.9) at some term ϕ. The larger ϕ, the better the approximation of the inverse. For example

$$\begin{array}{lcll} M^{-1} & = & E^{-1} & \text{if } \phi = 0, \\ M^{-1} & = & (I + E^{-1}(C^T B^{-1} C))E^{-1} & \text{if } \phi = 1. \end{array} \tag{3.10}$$

In addition, any extra term in the series means an additional linear system solution with matrix B. Efficient implementations of this matrix-vector ($C^T B^{-1} C$) products for particular N_i and L_i, $i = 1, \ldots, k$, matrices can significantly speed the computational efficiency.

When ρ is not too close to 1, the contribution of higher-order terms of the Neumann series can be neglected, and just a few terms (i.e., a small ϕ) are enough for a good preconditioner (explained in the next Subsection 3.3.1). In our problems, $\phi = 0$ has been used for all the computational results. However, note that even for $\phi = 0$, one system with matrix E needs to be solved at each PCG iteration. Therefore, the efficient solution to systems with E is indispensable for the performance of the method. In Subsections 3.4.1 and 3.4.2 we analyze the structure and efficient factorization of E for both formulations (2.25) and (2.27).

3.3.1 The spectral radius

From (3.9), the quality of the preconditioner depends on the spectral radius ρ [15]. A suitable preconditioner is given by $\rho \in [0, 1)$. The farther from 1, the closer to $(E - C^T B^{-1} C)^{-1}$. In practice, ρ comes closer to 1 as the solution is close to the optimum, this means that it needs more iterations. An improvement of the preconditioner by a quadratic regularization (can be seen in [20] and [18]). The `BlockIP` solver implements two types of regularization. The proximal point and a quadratic regularization. Both add a different term in the standard logarithmic barrier function of the problem (3.11)

$$B(x, \mu) \triangleq f(x) + \mu \left(-\sum_{i=1}^{n} \ln x_i - \sum_{i=1}^{n} \ln(u_i - x_i) \right) \tag{3.11}$$

The proximal point adds $\frac{1}{2}(x - \overline{x})^T Q_P (x - \overline{x})$. Where Q_P is a diagonal positive definite matrix, and $\overline{x}$ the current point. The quadratic regularization adds $\frac{1}{2} x^T Q_{R^x}$; where Q_R is a diagonal positive semidefinite matrix. This term is controlled by μ that tend to zero (more details in [18]).

Those terms make changes in the dual feasibility of KKT conditions and matrix $Theta$. In practice, the quadratic regularization preferred because it gives a better approximation to the original Θ.

Since the preconditioner is used at each iteration of PCG for the solution of system $(E - C^\top B^{-1} C)z = r$ (for some vectors z and r), increasing ϕ by one means solving an additional system with matrices B and E at each PCG iteration. Therefore, even though it is problem-dependent, we can consider as a rule of thumb $\phi = 0$ or $\phi = 1$ are reasonable choices.

3.3.2 Tolerance reduction factors

One of the important parameters for the efficient solution of (3.6) is the tolerance requested to the PCG solution. This tolerance is dynamically updated at each interior-point iteration i as $\epsilon_i = \max\{\beta\epsilon_{i-1}, \min_\epsilon\}$, where ϵ_0 is the initial tolerance (by default 10^{-2} for linear problems, and 10^{-3} for quadratic ones), $\min_\epsilon$ is the minimum allowed tolerance (by default is 10^{-8}), and β is a tolerance reduction factor at each interior-point iteration (by default 0.95). Unless otherwise stated, the above default values were used in the computational results of Chapter 4

3.3.3 Dual constraint matrix of the quadratic model

Consider the following general primal and dual quadratic optimization models based in [6] and [38]:

$$
\begin{array}{lll}
 & \textit{Primal} & \\
\min & c^T x + \frac{1}{2} x^T Q x & \\
\text{s.to} & Ax = b & \\
 & x \geq 0 \equiv -x \leq 0, &
\end{array}
\qquad
\begin{array}{ll}
\textit{Dual} & \\
\max\limits_{y,\mu,x} & b^T y - \frac{1}{2} x^T Q x \\
 & A^T y - Qx + \mu = c \\
 & y\, free, \mu \geq 0
\end{array}
\tag{3.12}
$$

Here, $A \in \mathbb{R}^{m \times n}$ has full row rank $m \leq n, Q \in \mathbb{R}^{n \times n}$ is symmetric and positive semidefinite matrix, $x, \mu, c \in \mathbb{R}^n$ and $y, b \in \mathbb{R}^m$. Therefore, the objective function is still convex.

The constraints matrix for the dual stochastic quadratic problem $A^T \lambda - Qx + \mu = c$ in (3.12) has to be reordered, for our purposes, to suit according to the formulation in blocks.

We consider the variable vector as $x^i = (x^i, \mu^i)$. Then, the dual variables are $(y^1, y^2, \ldots, y^k, y^0, x^0, x^1, x^2, \ldots, x^k)$. Here x^0 was for the variables of first-stage. The next step is to rearrange the vector as: $(y^1, x^1 y^2, x^2, \ldots, y^k, x^k, y^0, x^0)$. The Q_i $i = 0, \ldots k$ is the corresponding scenario-related matrices in Q. Therefore, the new constraint matrix is as follows:

$$
A_{new}^T = \begin{bmatrix}
W_1^T y^1 & -Q_1 x^1 & & & & & & \\
 & & W_2^T y^2 & -Q_2 x^2 & & & & \\
 & & & & \ddots & & & \\
 & & & & & W_k^T y^k & -Q_k x^k & \\
 & & & & & & & \\
M^T L_1^T y^1 & & L_2^T y^2 & & \ldots & L_k^T y^k & & -Q_0 x^0
\end{bmatrix}
\tag{3.13}
$$

3.4 Two-Stage Stochastic Optimization with Full Splitting in `BlockIP`

For working with `BlockIP` the problem has to be formulated as a standard form (3.1) with a primal block angular structure (2.6), explained before in Subsection (2.3.1.1). The two-stage stochastic problems have a dual block angular structure (2.5), however, for avoiding the dense column, the full splitting formulation is needed (2.5.1) giving us a matrix of primal block angular structure (2.28) that is just as our software required. For computing, easily, the linking constraints (2.31), a new function *splitting* was incorporated in `BlockIP`.

To find the solution for our stochastic problem with `BlockIP` (3), the normal equations (2.18) has to be solved (explained in Section 3.3). Taking back the structure of $A\Theta A^T$ (matrix 3.3 in Section 3.2) with an appropriate partition on Θ, the matrix may be written as:

$$A\Theta A^T = \begin{bmatrix} B & C \\ C^\top & E \end{bmatrix} \tag{3.14}$$

Now, the linking constraints in E have a special structure.

3.4.1 Structure of E for formulation (2.25)

Θ_i is made of two diagonal submatrices, Θ_i^x and Θ_i^y, which are, respectively, related to first- and second-stage variables x_i and y_i. Since $E = \Theta_0 + \sum_{i=1}^k R_i\Theta_i R_i^\top$, and using (2.30), the structure of E is given by

$$E = \Theta_0 + \begin{bmatrix} \Theta_1^x + \Theta_2^x & \Theta_1^x & \dots & \Theta_1^x & \Theta_1^x \\ \Theta_1^x & \Theta_1^x + \Theta_3^x & \dots & \Theta_1^x & \Theta_1^x \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \Theta_1^x & \Theta_1^x & \dots & \Theta_1^x + \Theta_{k-1}^x & \Theta_1^x \\ \Theta_1^x & \Theta_1^x & \dots & \Theta_1^x & \Theta_1^x + \Theta_k^x \end{bmatrix}. \tag{3.15}$$

E in (3.15) is a symmetric positive definite matrix and it is also partially banded, with only $2(k-2)+1$ non-zero diagonals: $E_{ij} > 0$ if $|i-j|$ is a multiple of n_x and $E_{ij} = 0$ elsewhere. This is a particular case of the general band matrix (see, e.g., [37, Ch. 4]). Figure 3.1.a shows the structure of (3.15) for a problem with $n_x = 50$ first-stage variables and $k = 4$ scenarios, as well as $2(k-2)+1 = 5$ non-zero diagonals and 450 non-zero elements.

It is not difficult to prove that either the $LDL^\top$ or Cholesky factorizations preserve the sparsity (zero fill-in) of (3.15); this guarantees an efficient factorization. However, we will omit such details. Instead, our focus in the next subsection will be on the E matrix of formulation (2.27), which is a more efficient alternative.

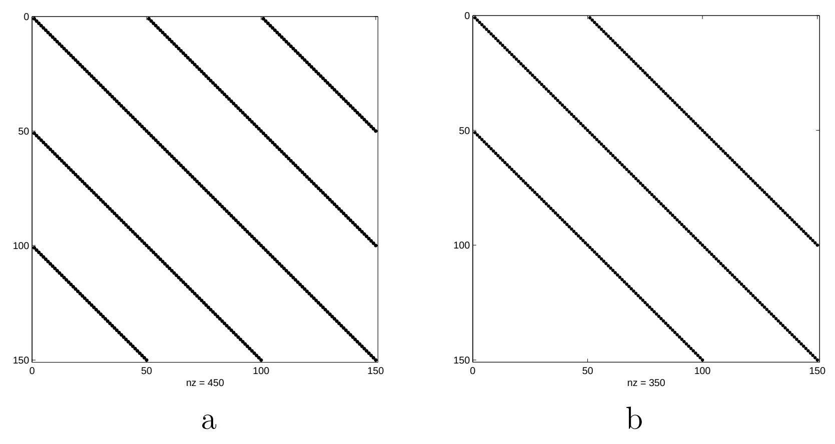


Figure 3.1: Problem with $n_x = 50$ first-stage variables and $k = 4$ scenarios: a) Structure of (3.15) for formulation (2.25); b) Structure of (3.16) for formulation (2.27)

3.4.2 Structure of E for formulation (2.27)

From the definition of E in (3.3) and those of L_i in (2.31), and considering as above that Θ_i is made of Θ_s^x and Θ_i^y, by block multiplication we get

$$E = \begin{bmatrix} \Theta_1^x + \Theta_2^x & -\Theta_2^x & & & \\ -\Theta_2^x & \Theta_2^x + \Theta_3^x & -\Theta_3^x & & \\ & \ddots & \ddots & \ddots & \\ & & -\Theta_{k-2}^x & \Theta_{k-2}^x + \Theta_{k-1}^x & -\Theta_{k-1}^x \\ & & & -\Theta_{k-1}^x & \Theta_{k-1}^x + \Theta_k^x \end{bmatrix}. \qquad (3.16)$$

Figure 3.1.b shows the structure of (3.16) for the same problem in Subsection 3.4.1, reformulated as in (2.27). The fundamental difference between (3.15) and (3.16) is that the latter has only three non-zero diagonals, independently of the number of scenarios. In practice, this leads to the efficient solution of systems $Ez = r$ (for some z and r).

Matrix (3.16) is a symmetric positive definite "n_x-shifted tridiagonal matrix", a generalization of a tridiagonal matrix where the superdiagonal (nonzero diagonal above the main diagonal) and subdiagonal (nonzero diagonal below the main nonzero diagonal) are shifted n_x positions from the main diagonal, i.e., elements (i, j) are non-zero only if $|i-j|$ is either 0 or n_x. Matrices with such a structure can be efficiently factorized with zero fill-in by extending a standard factorization for tridiagonal matrices. This particular factorization has been added to the `BlockIP` solver.

Chapter 4

Results

4.1 Introduction

In this chapter, the problem formulation (2.27) was implemented and solved using the `BlockIP` package [16, 18], which is based on the specialized IPM of Chapter 3.

The results are divided into two groups. The first one are the instances proposed in [4], [44] and [34]. They are publicly available in SMPS format [35] and have been used to measure performance in several solvers, such as in [40] and [81]. They are up to 2,400,000 restrictions and 8,000,000 variables (instance 4nodeBase32768). Also, The first-stage only reaches to 144 variables (instance pltexpA2). The outcomes that came out of these problems gave us the guidelines of the second group.

In Section 4.2, the first group is explained. We describe the SMPS format, including its different varieties. Also, we implemented an application, called readSTO, which brings these files into the same style SCENARIO (4.2.1.3). Then with the aid of *Stochastic Modeling Interface* (SMI), we convert them into the equivalent deterministic form (2.3.1).

The second group (4.3) is composed of two available problems: (1) Supply Chain Problem and (2) Power generation problem. They were modified according to the preliminary results. These problems were programmed to be able to change the amount of input data and thus create different instances by increasing the number of variables. The experiments reached up to 6,000,000 restrictions and 38,000,000 variables

One way to evaluate the efficiency of an algorithm is through the comparison of the *computing time*. This is a term used in the complexity theory for quantifying the performance of an interactive algorithm [57]. We used this measure to compare the performance with the CPLEX barrier algorithm mainly, which is an optimization commercial software package by IBM.

4.2 Applications with a small number of first-stage variables

For this research, we chose Two-Stage Stochastic problems (TSSP) from the literature. Most of the problems in this study came from [4], and their data can be download from [30]; as for the rest, they can be found at [44].

These problems were developed in a real context. However, they were adapted for research purposes. This collection is used to compare the efficiency of algorithms between scientists. A general description for the family of the problems is as follows

1. **A Network model for asset or liability management**, Mulvey and Vladimirou (1991) [66] and Mulvey and Ruszczynski (1995) [65]. (linear stochastic problem)
   Description: The management of assets/liabilities can be treated as a network problem, where asset categories are represented by nodes, and transactions are represented by arcs. The purchase or sale of an asset usually is fixed and associated with deterministic costs, while the return on investment from one stage to the next is usually unknown. There are five nodes in each stage: checking, savings, certificate of deposit, cash, and loans. Random coefficients are found in the objective, left, and right-hand-side of the second-stage.

2. **Electrical investment planning**, Louveaux and Smeers (1988) [56] (linear stochastic problem)
   Description: Consider the challenge of planning investments in the electricity generation industry. While the model is generally multistage, the specific example given is a two-stage model, with three random realizations for a random variable in the right-hand-side of stage two.

3. **Cargo network scheduling**, Mulvey and Ruszczynski (1995) [65] (mixed-integer linear or nonlinear stochastic problem)
   Description: This problem is about scheduling cargo transportation. The flight schedule is completely determined in stage one, and the amount of cargo to be shipped is uncertain. In this case, the recourse actions are used to determine which cargo is to be put in which flight. In transshipment, getting cargo from node m to node n using more than one flight or more than one route is allowed. When making a transshipment, cargo must be unloaded at some intermediate node, so then it may be loaded onto a different ship (route) going through the same node. Any undelivered cargo results in a penalty, increasing the costs of the operations. In this case, random variables appear on the right-hand side only. All flights have two tracks, that is, the airport of origin and the three airports in each flight.

4. **Telecommunication network planning**, Sen, Doverspike, and Cosares (1994) [74] (mixed-integer linear stochastic problem)

Description: Providing private lines of telecommunication to customers. Large corporations used such service between its business locations for high speed and secure data transmission. Private lines are generally used for a much longer duration than public ones, and they generally can carry more volume per connection. A manager of such a network must be continually making predictions in which to base decisions on where and how much to increase capacity. For this problem formulation, the "how much" is decided beforehand within budget limits. In this way, data expansion is not penalized. The goal here is to minimize unserved requests while staying within budget. These networks are usually very interconnected so that for any point-to-point demand pair, there is usually more than one route that may service the demand. Each route is made of one or more direct links. The resulting model is a two-stage network model with a stochastic (right-hand-side) demand variable in the second-stage.

5. **Airlift operations scheduling**, Midler and Wollmer (1969) [63] (mixed-integer linear stochastic problem)
   Description: In scheduling monthly airlift operations, we can predict the demand for specific routes. A random variable can represent the actual requirements. Aircraft of several different types are available for service. Each of these types of aircraft has its restriction on the number of flight hours available during the month. The recourse actions available include allowing available flight time to go unused, switching aircraft from one route to another, and buying commercial flights. Each of these has its associated cost, depending on the type of aircraft involved.

6. **Forest planning**, H. Gassmann (1989) [33] (multistage, linear stochastic problem)
   Description: This involves the decision about choosing the sections and time to harvest the forest. The requirements are linked to the age of the trees and the likelihood of the remaining trees to be destroyed by a fire. The forest is divided into groups, depending on those features. Depending on whether one tree survives from fire or it is harvested, it will then be incorporated into another group. The burned areas will be immediately replanted. This objective function maximizes the value of timber, both cut and remaining.

7. **Design if batch chemical plants**, Subrahmanyam, Pekny, and Reklaitis (1994) [76] (multistage, mixed linear stochastic problem)
   Description: Design of a chemical plant that wants to satisfy future demand. Decisions are how many plants to build, when what type, and how to operate. Every plant might have different tasks, one or more tasks at the same time. Material constraints include inventory, production, consumption, sales, and purchasing effects.

8. **Energy and environmental planning**, Fragnière (1995) [32] (multistage,

linear stochastic problem)
Description: Assistance in planning energy supply infrastructure and policies. The problem considers emissions of greenhouse gases, capacity expansion, demand, and production. Many different technologies supply energy. The capacity of each technology was installed before or after the beginning of the optimization. The objective is to minimize capital and operating costs.

Some of these problems are mixed; that is, they have also integer variables. In this investigation, these kinds of problems are relaxed because `BlockIP` uses only real numbers. The next Table 4.1 summarizes the family problems displaying the size of the first-stage, second-stage variables, and the maximum number of scenarios in the instances.

Family	**Original Source**	**1st Stage**	**2st Stage**	**Scenarios** (up to)
Assets	Mulvey and Ruszcznski (1995)	13	13	32,768
Electricity	Louveaux and Smeers (1988)	4	12	3
Cargo	Mulvey and Ruszcznski (1995)	52	202	32,768
Phone	Sen et al. (1994)	8	84	32,768
Airlift	Midler and Wollmer (1969)	4	8	676
Forest	Gassmann (1989)	15	96	64
Chem	Pekny and Reklaitis (1994)	39	41	4
Environ	Fragnière (1995)	49	49	5

Table 4.1: Family of problems collection

4.2.1 Format

In the literature, this kind of instance is found in the SMPS format, which makes use of three text files: core, time, and stochastic. These files are column-oriented, and everything (variables, constraints, and others) gets a name [35] and its value. In the next subsections (4.2.1.1, 4.2.1.2, and 4.2.1.3), examples of the different files are given based on the *Electrical investment planning* problem from [4]. The name of the files (instance) is *LandS* each.

4.2.1.1 Core file

The core file defines the deterministic model created by the first-stage problem and the first scenario of the second-stage. The core file may be in the usual MPS format [46]. An example of the power generation problem (LandS.core) can be found in Appendix C.

4.2.1.2 Time file

The time file describes the structure of the problem specifying the number of stages. The file records include the first row and column of each stage. For this research, only two-stage problems were analyzed. Therefore, as in the example LandS.time that is given in Figure 4.1, we can read that the first variable of the first-stage $X1$ and the first variable of the second-stage $Y11$. To carry out a correct reading of both files, variables inside the core file must be ordered. In this file, the first label is TIME. Then, the name of the instance (LandS). It terminates with the ENDATA too.

```
TIME          LandS
PERIODS
    X1        MINCAP                    PERIOD1
    Y11       OPLIM1                    PERIOD2
ENDATA
```

Figure 4.1: LandS.time file

4.2.1.3 Stochastic file

Finally, the stochastic (stoch, sto) file gives the stochastic data. There are many different ways to present this information, such as Scenarios, Node, INDEP (independent discrete distribution or by Blocks) or Networks. The ultimate goal is to produce an event tree, as is shown in Figure 4.2. In this work, all the instances were INDEP (both cases).

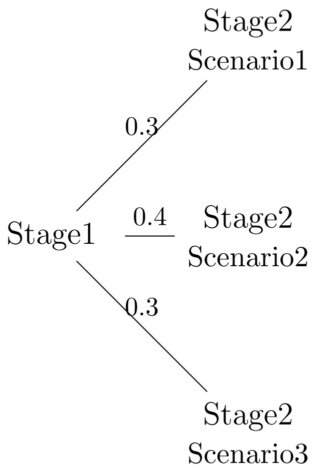


Figure 4.2: Event tree

The file has first the word STOCH that refers to a stochastic file, then the name of the instance problem (LandS). In the next line, it has an identifier to explain which kind of file is and the probability distribution, DISCRETE or CONTINUOUS. The discrete distribution was the only type found for this work. To describe the different kinds of files, we also use the example problem (Lands). Other features of the stoch file include linear and quadratic penalties for violating a stochastic constraint, probabilistic constraints and objectives, and integrated chance constraints.

For using `BlockIP`, we need the format of the deterministic equivalent (extensive form), that is, a plain MPS file where all the data is included in the same file. For expanding the problem from the three files into this unique MPS file the SMI package (Stochastic Modeling Interface) [75] was used. This solver needs the Scenario file as an input for the stoch file, so other kinds of stoch files must transform into Scenarios files (explained later in Section 4.2.3).

SCENARIO

The example below (Figure 4.3) defines a SCENARIO file with three scenarios. Here, each record marked 'SC' denotes the start of a new scenario, its probability, and the stage the branch occurs (second-stage). The following data are the information on that scenario, which is different from the other scenarios. For instance, in the example file, all the data are the same for all scenarios except for the only stochastic data: the parameter in the right-hand side of the constraint DEMAND1 'RIGHT DEMAND1'. For scenario, 'SCENO1' has 0.3 probability of occurring with the value 3. In the next scenario, 'SCENO2' has 0.4 probability, and the possible value will be 5. The last 'SCENO3' has the 0.3 probability with the value 7. Note that the sum of the probability of all scenarios must be one.

```
STOCH          LandS
SCENARIO         DISCRETE                  REPLACE
 SC SCEN01     PERIOD2   0.3     PERIOD2
    RIGHT      DEMAND1   3.0
 SC SCEN02     PERIOD2   0.4     PERIOD2
    RIGHT      DEMAND1   5.0
 SC SCEN03     PERIOD2   0.3     PERIOD2
    RIGHT      DEMAND1   7.0
ENDATA
```

Figure 4.3: LandS.stoch. file (scenario)

INDEP

For these files the ID word is INDEP and has all the information by line, i.e. each stochastic element which differs from the rest of second-stage comes with their identifiers (first two columns), the different values (third column), the corresponding branch for the stage, and their probability. Figure 4.4 shows an example file.

```
STOCH          LandS
INDEP          DISCRETE
    RIGHT      DEMAND1   3.0             PERIOD2   0.3
    RIGHT      DEMAND1   5.0             PERIOD2   0.4
    RIGHT      DEMAND1   7.0             PERIOD2   0.3
ENDATA
```

Figure 4.4: LandS.stoch (indep)

BLOCKS

The last kind of file used in this research is BLOCKS, which has independent distribution ordered by blocks. Each block marked 'BL' denotes a different scenario with the assigned probability and the branch. Data below the blocks came with the identifier and the values that will be changed. The file example is illustrated in Figure 4.5.

```
STOCH          LandS
BLOCKS         DISCRETE
 BL BLOCK1     PERIOD2   0.3
    RIGHT      DEMAND1   3.0
 BL BLOCK1     PERIOD2   0.4
    RIGHT      DEMAND1   5.0
 BL BLOCK1     PERIOD2   0.3
    RIGHT      DEMAND1   7.0
ENDATA
```

Figure 4.5: LandS.stoch (Blocks)

In some cases, the file can combine INDEP and BLOCKS style; the first incoming is the INDEP data, and later comes the information by the BLOCKS. Not all software, that reads SMPS can manage all kinds of stochastic files. In this research, all the instances are INDEP, BLOCKS, and a combination of both, so a program was created in C++ to read those files and elaborate a new SCENARIO-style file. The program is named READSTO. The SCENARIO file can be used in the interface SMI (see below in Section 4.2.3) to expand the problem into a big MPS format.

4.2.2 readSto application

This program was created to receive the stoch file that can be written as INDEP, BLOCKS or a mix of both. The output is to create a SCENARIO style, that is, the input file that is needed for the SMI package (described in Section 4.2.3). First, *readSto* reads all the data classifying the parameters and probabilities. Then, it calculates the probability for each scenario and reorders all the parameters, which are going to change in the expansive form. The following algorithm summarizes it:

```
Input= INDEP,BLOCKS;
Ouput= SCENARIO:
if INDEP
    calculate probability
else                                                        (4.1)
    copy probability
reorder;
list parameters
```

After applying all instances thought *readSto* application, they are ready for SMI software. The next step is to execute the SMI that helps us creating the deterministic equivalent in a plain MPS format for `BlockIP`.

4.2.3 Stochastic Modeling Interface

For this research, the SMI software was used with some modifications. SMI stands for Stochastic Modeling Interface [75] is a project from the Computational Infrastructure for Operations Research (COIN-OR), which is an open-source software published by IBM and writing in C++ [52]. Currently, SMI supports the implementation of a Stochastic MPS (SMPS) reader. Here, the three SMPS files (core, time, stoch) are scanned. However, it only considers a SCENARIO-style file with discrete random variables. Then it generates the deterministic equivalent problem (Section 2.3.1) in a file with the format of MPS.

The program was tailored by the professor Antonio Frangioni from the Operations Research Group at the Universitá di Pisa, (Italy) in 2005. Therefore, we have to updated and reinstate his changes. This hacked version has a function `setAnnotStream` that makes an annotation file that identifies the first-stage and every scenario of the deterministic equivalent MPS format. Figure 4.6 illustrated the LandS.annot file.

```
num_stages= 2; num_scenarios= 3;
info_nodes= [
1   1   1   2   5   3   2   17  10  2   29  17  ];

rows_with_slacks =[
1
2
];
```

Figure 4.6: LandS.annot file

Thus, SMI creates an extensive form in a new MPS file and the annotation file.

4.2.4 CPLEX Barrier vs `BlockIP`

For testing the performance of the proposed specialized interior-point method, an initial comparison was made using the state-of-the-art IBM ILOG CPLEX (v.12.7) barrier optimizer [47]. This decision was made based on the fact that both algorithms are IPM.

Unlike `BlockIP`, CPLEX computes directions by Cholesky factorizations, instead of a combination of Cholesky and PCG. After finding a solution in the interior point space with a tolerance of ϵ, by default, the process called a crossover method which moves from an optimal point to an optimal vertex. This process uses the simplex method to find a basic solution [57].

In this work, CPLEX executions were made without crossover for a fair comparison with `BlockIP`. Also, the setting of a standard barrier algorithm was chosen and the same tolerance was used in both solvers. As we already mentioned in Subsection 4.2.4.1, CPLEX also was used with one thread for all the performances and we executed it in the same server.

In these experiments, for `BlockIP` the problems were modeled using the full splitting formulation (2.27), whereas for CPLEX the dual of the extensive form (2.4) was formulated, in an attempt to avoid dense columns due to the first-stage variables.

To get the solution, we solve all the instances with the linear objective function, as we found in the literature. Besides, we create a new version for each instance by adding quadratic terms in all the variables. We measured the performance also in theses problems. All the experiments were carried out on a Fujitsu Primergy RX2540 M1 4X server with two 2.6 GHz Intel Xeon E5-2690v3 CPUs (48 cores) and 192 Gigabytes of RAM, under a GNU/Linux operating system (OpenSuse 13.2), without the exploitation of multithreading capabilities.

4.2.4.1 MPS2BIP application

For working with `BlockIP` the MPS2BIP application was created for suiting the MPS file form the deterministic equivalent into the input data of `BlockIP`. Also, adding the Full Splitting formulation (illustrated in Section 2.5.1).

MPS2BIP has as inputs the expanded MPS file, which includes the first-stage and all the scenarios of the second-stage, and the annot file with the information of where every scenario starts.

After taking into account all the information, the constraint matrix was reordered (2.27) considering the full splitting constraints (2.31); this means, that now the problem has a primal block-angular structure (3.2) with $(S-1)n_x$ very sparse linking constraints with only identity and minus identity matrices.

4.2.4.2 Results for instances with small first-stage variables

The measure used to compare the performance of the proposed specialized interior-point method with CPLEX (barrier optimizer) was the time computed for solving each instance, without the time of reading the problem.

Sometimes the PCG could fail when the matrix is detected positive semidefinite. In this case, `BlockIP` may switch to Cholesky factorization. From our initial 74 instances, mention at the beginning of the section, 16 cases were detected like this. In our experiments, we decided to stop the execution when this happened, otherwise, for these large-scale instances the solution by Cholesky could be computationally expensive. One thing that helped us manage these problems was replacing very large values (1.0e128) with a smaller exact "infinity" where BlockIP might handle better.

Table 4.2 shows a sample of the results with the instances referred to above. The first column indicates the name of each instance; next, the number of variables in the first-stage (named $1st$); columns k, m, and n refer to, respectively the number of scenarios, constraints, and variables. Then, the number of interior-point iterations and CPU time (seconds) for both solvers are shown. In the case of `BlockIP`, the PCG iterations are also reported. The last column refers to the rate between the time of CPLEX barrier and `BlockIP` (cb/BIP). We considered an initial tolerance of $\epsilon_0 = 10^{-2}$ for the PCG solutions (which is the default in `BlockIP` for linear problems). The optimality tolerances for `BlockIP` and CPLEX were set to 10^{-3} (smaller values could be difficult to achieve by `BlockIP` due to the approximate solution of the Newton direction by PCG). The results of Table 4.2 indicate that `BlockIP` is slower in these kinds of instances where, although with a large number of scenarios, the number of first-stage variables is small. In two cases, CPLEX sends us a message: Out of memory.

					BlockIP			CPLEX		
Instance	$1st$	k	m	n	Iter	CPU	PCG	Iter	CPU	cb/BIP
4node2	52	2	162	508	24	0.00	126	12	0.00	0.00
4node4	52	4	310	1008	27	0.01	241	16	0.01	1
4node8	52	8	606	2008	32	0.04	450	13	0.02	0.5
4node16	52	16	1198	4008	35	0.14	785	13	0.06	0.43
4node16old	52	32	2382	8008	43	0.33	1313	17	0.31	0.94
4node128	52	128	9486	32008	35	0.97	1167	25	0.30	0.31
4node256	52	256	18958	64008	38	1.69	1074	25	0.48	0.28
4node512	52	512	37902	128008	43	3.18	1068	29	1.13	0.36
4node1024	52	1024	75790	256008	43	6.65	1094	30	2.57	0.39
4node2048	52	2048	151566	512008	52	46.44	3908	36	6.56	0.14
4node4096	52	4096	303118	1024008	49	76.06	2938	45	17.30	0.23
4node16384	52	16384	1212430	4096008	59	522.10	5131	43	57.89	0.11
4node32768	52	32768	2424846	8192008	69	1470.91	7194	43	120.19	0.08
4nodeBase2	52	2	164	508	21	0.00	120	10	0.00	0.00
4nodeBase4	52	4	312	1008	25	0.02	195	13	0.02	1
4nodeBase8	52	8	608	2008	28	0.02	238	11	0.02	1
4nodeBase16	52	16	1200	4008	37	0.05	424	14	0.07	1.4
4nodeBase16old	52	32	2384	8008	29	0.12	313	12	0.26	2.17
4nodeBase128	52	128	9488	32008	40	0.84	861	15	0.17	0.20
4nodeBase256	52	256	18960	64008	43	1.67	1071	22	0.43	0.26
4nodeBase512	52	512	37904	128008	52	3.63	1104	24	0.97	0.27
4nodeBase1024	52	1024	75792	256008	62	9.99	1662	25	2.25	0.23
4nodeBase2048	52	2048	151568	512008	72	26.62	1893	31	5.90	0.22
4nodeBase4096	52	4096	303120	1024008	70	62.13	2132	34	14.32	0.23
4nodeBase16384	52	16384	1212432	4096008	110	570.13	5274	46	59.55	0.10
4nodeBase32768	52	32768	2424848	8192008	103	785.99	3354	31	88.51	0.11
AIRL_first	4	25	152	402	12	0.00	121	7	0.00	0.00
AIRL_second	4	25	152	402	11	0.00	89	5	0.00	0.00
AIRL_randgen	4	676	4058	10818	17	0.09	120	11	0.04	0.44
assets_100	13	100	505	2600	50	0.05	602	0	0.01	0.2
assets_37500	13	37500	187505	975000	32	17.39	461	17	2.70	0.16
chemBase	39	2	118	226	29	0.00	272	11	0.00	0.00
chem	39	2	130	238	15	0.00	118	10	0.00	0.00
env_15	49	15	768	2046	21	0.02	78	14	0.01	0.5
env_1200	49	1200	57648	160836	51	4.22	749	0	0.56	0.13
env_1875	49	1875	90048	251286	64	9.09	1096	0	1.17	0.13
env_3780	49	3780	181488	506556	75	24.45	1237	0	2.70	0.11
env_5292	49	5292	254064	709164	92	43.23	1541	0	4.80	0.11
env_8232	49	8232	395184	1103124	125	99.47	2316	0	10.06	0.10
env_32928	49	32928	1580592	4412388	155	804.61	3753	Out	of memory	
env_aggr	49	5	288	706	16	0.00	44	11	0.00	0.00
env_first	49	5	288	706	18	0.00	45	10	0.00	0.00
env_loose	49	5	288	706	13	0.00	43	10	0.00	0.00
envDiss15	49	15	768	2046	26	0.03	150	12	0.01	0.33
envDiss1200	49	1200	57648	160836	66	7.67	1689	40	0.91	0.12
envDiss1875	49	1875	90048	251286	78	14.13	1928	42	1.63	0.12
envDiss3780	49	3780	181488	506556	104	50.15	2985	44	4.36	0.09
envDiss5292	49	5292	254064	709164	130	92.52	3786	54	7.62	0.08
envDiss8232	49	8232	395184	1103124	135	154.36	4066	62	15.02	0.10
envDiss32928	49	32928	1580592	4412388	200	1580.19	10897	Out	of memory	
envDiss_aggr	49	5	288	706	17	0.01	67	8	0.00	0.00
envDiss_loose	49	5	288	706	18	0.00	60	8	0.00	0.00
LandS	4	3	23	62	9	0.00	41	6	0.00	0.00
pltexpA2_6	188	6	686	2760	60	0.05	492	10	0.01	0.2
phone	8	32768	753665	3309569	21	38.02	255	23	26.00	0.68

Table 4.2: Results of linear instances with `BlockIP` and CPLEX barrier

For the sake of completeness, the dual simplex method in CPLEX [48] was tested. It was run in the same conditions as the CPLEX Barrier. In Table 4.3, the results of the performance of this algorithm are presented with a $5 \cdot 10^{-2}$ optimality gap.

This table shows, the name of each instance, the number of iteration, and the CPU time in seconds; at the last column, the rate of time, CPLEX Dual over BlockIP (from Table 4.2). In these results, it can be appreciated that BlockIP is, in general, faster than CPLEX Dual. Also, it can be noticed that for these kinds of problems CPLEX Barrier is more competitive.

In addition, the primal simplex algorithm from CPLEX was also executed; however, it was too slow in the medium and large instances. As a result, the CPU times were omitted in the results.

4.2.5 Quadratic benchmark for the small first-stage instances

From the above linear instances, we created a set of quadratic stochastic problems by adding convex separable quadratic costs to the first- and second-stage variables; these quadratic terms were synthetic and of the same order as the linear costs. Table 4.4 reports the characteristics of these quadratic instances and the results obtained with BlockIP and CPLEX with the algorithm Barrier. In these executions, the initial PCG tolerance was set to $\epsilon_0 = 10^{-3}$ (the default value in BlockIP for quadratic problems). The optimality tolerances used for BlockIP and CPLEX were $5{\cdot}10^{-2}$. The specialized IPM in BlockIP is known to be more efficient for quadratic than for linear instances [20]. Then, as expected, the performance of BlockIP improved in these quadratic stochastic instances. However, it was still outperformed by CPLEX.

	CPLEX Dual		
Instance	Iter	CPU	cd/BIP
4node2	197	0.00	0.00
4node4	394	0.01	1.00
4node8	764	0.02	0.50
4node16	1514	0.08	0.57
4node16old	3409	0.13	0.39
4node128	12847	1.47	1.52
4node256	25547	3.94	2.33
4node512	62970	25.99	8.17
4node1024	112829	70.50	10.60
4node2048	250327	395.93	8.53
4node4096	444252	1637.46	21.53
4node16384	2435581	31776.71	60.86
4node32768	4045956	164203.33	111.63
4nodeBase2	208	0.01	1.11
4nodeBase4	431	0.01	0.50
4nodeBase8	840	0.02	1.00
4nodeBase16	1743	0.06	1.20
4nodeBase16old	3325	0.09	0.75
4nodeBase128	16432	0.91	1.08
4nodeBase256	36462	2.85	1.71
4nodeBase512	78874	10.62	2.93
4nodeBase1024	177091	52.15	5.22
4nodeBase2048	348293	175.67	6.60
4nodeBase4096	706291	983.03	15.82
4nodeBase16384	2951060	30222.37	53.01
4nodeBase32768	5831983	219262.22	278.96
AIRL_first	162-75	0.00	0.00
AIRL_second	145-75	0.00	0.00
AIRL_randgen	4497-2028	0.12	1.33
assets_100	224	0.00	0.00
assets_37500	43749	31.36	1.80
chemBase	31	0.00	0.00
chem	29	0.00	0.00
env_15	321-16	0.01	0.50
env_1200	23557	0.72	0.17
env_1875	36567	1.29	0.14
env_3780	73421	4.92	0.20
env_5292	102673	7.73	0.18
env_8232	158775	18.41	0.19
env_32928	1260613	202.78	0.25
env_aggr	117-6	0.00	0.00
env_first	112-6	0.00	0.00
env_loose	112-6	0.00	0.00
envDiss15	356	0.00	0.00
envDiss1200	26063	0.95	0.12
envDiss1875	41203	1.94	0.14
envDiss3780	83491	7.88	0.16
envDiss5292	117359	13.89	0.15
envDiss8232	178885	25.80	0.17
envDiss32928	1289763	268.37	0.17
envDiss_aggr	131	0.00	0.00
envDiss_loose	122	0.00	0.00
LandS	21	0.00	0.00
pltexpA2_6	no conv		
phone	972239	7495.43	197.14

Table 4.3: Performance of the benchmark set of instances with CPLEX Dual algorithm

					BlockIP			CPLEX		
Instance	$1st$	k	m	n	Iter	CPU	PCG	Iter	CPU	cb/BIP
4node2	52	2	162	508	20	0.00	139	8	0.00	0.00
4node4	52	4	310	1008	22	0.00	199	7	0.01	1.00
4node8	52	8	606	2008	24	0.03	251	7	0.02	0.54
4node16	52	16	1198	4008	25	0.08	302	7	0.05	0.63
4node16old	52	32	2382	8008	19	0.15	480	17	0.31	2.07
4node128	52	128	9486	32008	29	0.61	534	7	0.12	0.20
4node256	52	256	18958	64008	31	1.25	583	7	0.22	0.18
4node512	52	512	37902	128008	31	2.99	754	7	0.44	0.15
4node1024	52	1024	75790	256008	33	6.36	859	7	1.12	0.18
4node2048	52	2048	151566	512008	33	15.51	894	7	3.98	0.26
4node4096	52	4096	303118	1024008	31	36.60	984	7	6.81	0.19
4node16384	52	16384	1212430	4096008	31	137.87	1219	8	14.29	0.10
4node32768	52	32768	2424846	8192008	32	300.99	1305	Out	of memory	
4nodeBase2	52	2	164	508	20	0.00	151	8	0.00	0.00
4nodeBase4	52	4	312	1008	22	0.01	202	9	0.01	1.00
4nodeBase8	52	8	608	2008	24	0.04	294	7	0.02	0.50
4nodeBase16	52	16	1200	4008	27	0.09	370	7	0.06	0.67
4nodeBase16old	52	32	2384	8008	18	0.14	370	18	0.33	2.36
4nodeBase128	52	128	9488	32008	31	0.85	817	11	0.14	0.16
4nodeBase256	52	256	18960	64008	32	1.50	833	11	0.29	0.19
4nodeBase512	52	512	37904	128008	31	3.10	924	11	0.65	0.21
4nodeBase1024	52	1024	75792	256008	30	6.75	1012	12	1.22	0.18
4nodeBase2048	52	2048	151568	512008	31	19.49	1277	12	3.01	0.15
4nodeBase4096	52	4096	303120	1024008	33	46.44	1459	12	8.09	0.17
4nodeBase16384	52	16384	1212432	4096008	35	213.29	1979	Out	of memory	
4nodeBase32768	52	32768	2424848	8192008	37	472.27	2168	Out	of memory	
AIRL_first	4	25	152	402	11	0.00	173	10	0.00	0.00
AIRL_second	4	25	152	402	11	0.00	210	11	0.00	0.00
AIRL_randgen	4	676	4058	10818	17	1.09	3512	16	0.05	0.05
assets_100	13	100	505	2600	31	0.05	327	4	0.00	0.00
assets_37500	13	37500	187505	975000	34	30.08	881	14	1.75	0.06
chemBase	39	2	118	226	13	0.00	67	9	0.00	0.00
chem	39	2	130	238	8	0.00	49	5	0.00	0.00
env_15	49	15	768	2046	14	0.01	46	13	0.02	2.00
env_1200	49	1200	57648	160836	14	1.52	209	18	0.66	0.43
env_1875	49	1875	90048	251286	15	2.24	261	17	1.11	0.50
env_3780	49	3780	181488	506556	17	8.50	357	18	2.62	0.31
env_5292	49	5292	254064	709164	17	11.54	355	20	3.16	0.27
env_8232	49	8232	395184	1103124	17	19.21	388	18	4.62	0.24
env_32928	49	32928	1580592	4412388	20	91.85	568	19	17.59	0.19
env_aggr	49	5	288	706	11	0.00	27	8	0.00	0.00
env_first	49	5	288	706	12	0.00	33	7	0.00	0.00
env_loose	49	5	288	706	10	0.00	30	6	0.00	0.00
envDiss15	49	15	768	2046	14	0.01	46	13	0.02	2.00
envDiss1200	49	1200	57648	160836	14	1.44	209	18	0.66	0.46
envDiss1875	49	1875	90048	251286	15	2.72	261	17	1.10	0.40
envDiss3780	49	3780	181488	506556	17	7.90	331	18	2.84	0.36
envDiss5292	49	5292	254064	709164	17	11.25	346	20	3.26	0.29
envDiss8232	49	8232	395184	1103124	17	19.44	393	18	4.60	0.24
envDiss32928	49	32928	1580592	4412388	20	100.45	638	19	18.16	0.18
envDiss_aggr	49	5	288	706	11	0.00	28	8	0.00	0.00
envDiss_first	49	5	288	706	12	0.00	33	7	0.01	1.00
envDiss_loose	49	5	288	706	10	0.00	29	9	0.00	0.00
fxm2_16	114	16	3900	9045	200	0.73	311	10	0.07	0.10
LandS	4	3	23	62	9	0.00	48	6	0.00	0.00
pltexpA2_16	188	16	1726	7360	45	0.13	303	10	0.08	0.62
pltexpA2_6	188	6	686	2760	54	0.03	241	10	0.02	0.67
stocfor2	15	64	6543	10182	12	0.26	452	13	0.10	0.38
LandS	4	3	23	62	9	0.00	48	6	0.00	0.00
phone	8	32768	753665	3309569	13	21.95	169	18	12.88	0.59

Table 4.4: Results of quadratic instances with `BlockIP` and CPLEX barrier

4.2.6 Comparison with other specialized solvers for stochastic optimization

In addition to the CPLEX general-purpose solver, we explored other specialized methods for two-stage stochastic problems, namely: Benders decomposition, and Benders decomposition with regularization by the level set method, as implemented in the FortSP stochastic solver [81]; the primal-dual column generation approach of [40]; and the dual decomposition approach (with an interior-point cutting-plane generator) implemented in the DSP (Decomposition for Structured Programming) stochastic solver [53]. Comparing `BlockIP` with these other approaches is not straightforward since the solvers of [81] and [40] are not freely available; the DSP solver had to be used remotely from the *Neos Server* [23] (we did not succeed in installing it locally due to its many dependencies with third-party software); and the four hardware (i.e., those of [81], [40], the Neos Server, and our work) were different, so only an indirect comparison can be performed.

Table 4.5 shows the results obtained with `BlockIP` and two of the algorithms in the DSP package (namely, a Benders decomposition and a dual decomposition) for a subset of the instances of Table 4.2. CPU times for DSP were provided by the Neos Server [67], so the comparison between our approach and DSP should be done with caution. When the instance exceeded the maximum memory allotted by the Neos Server to a job, the solution was not found; this is marked with "—" in Table 4.5. From the times reported, DSP seems not to be competitive with our approach, especially for the largest instances. A possible explanation might be that DSP was mainly designed for mixed-integer stochastic problems.

Concerning the other two specialized methods, and according to the information provided by their authors, the processors used in [40] and [81] were, in single-thread mode, respectively a 27% and an 11% faster (in terms of Mflops, millions of floating-point operations per second) than the one used in our work.

The approach of [40] outperformed in general all the methods tested in [81], from the results in those papers. Therefore we will focus on the primal-dual column generation method of [40]. This approach required (in the hardware used in [40]) for the 10 instances of Table 4.2, respectively, 0.03, 0.03, 0.11, 0.56, 10.83, 37.97, 0.05, 0.68, 5.86 and 0.76 CPU seconds. Using the 27% correction factor between processors, if we had run the approach of [40] in our hardware, the CPU times would have been, approximately, 0.04, 0.04, 0.15, 0.76, 14.77, 51.78, 0.07, 0.93, 7.99 and 1.03. We observe from Table 4.2 that the performance of CPLEX was very similar in those instances (excluding the last two, where the approach of [40] was significantly faster—especially for the last instance). Therefore, although being a general-purpose solver, CPLEX can be considered a good candidate for benchmarking (this is also consistent with the results of [81], where the CPLEX barrier ranked among the best three algorithms for stochastic optimization). Also, CPLEX can solve quadratic instances, while the approaches of [40] and [81] dealt

					BlockIP			DSP	
Instance	1*st*	*k*	*m*	*n*	Iter	CPU	PCG	Algorithm	CPU
AIRL_first	4	25	152	402	12	0.00	121	Benders	0.8
								Dual	6.6
pltexpA2	188	6	686	2760	60	0.05	492	Benders	0.1
								Dual	0.1
LandS	4	3	23	62	9	**0.00**	41	Benders	0
								Dual	0.3
4node128	52	128	9486	32008	35	**0.97**	1167	Benders	3.7
								Dual	16529.1
4node4096	52	4096	303118	1024008	49	**76.06**	2938	Benders	1535.1
								Dual	—
env_1200	49	1200	57648	160836	51	**4.22**	749	Benders	—
								Dual	—
env_32928	49	32928	1580592	4412388	155	**804.61**	3753	Benders	—
								Dual	—
phone	8	32768	753665	3309569	21	38.02	255	Benders	—
								Dual	—

Table 4.5: Results of linear instances with `BlockIP` and DSP

only with linear problems. CPLEX will thus be the solver used to test our approach in the next Section 4.3, for the solution of more difficult instances provided by two particular applications.

4.2.7 Comparison with other IPMs that solve the augmented system

From the outputs provided by the barrier CPLEX algorithm, it apparently solves, as `BlockIP`, the normal equations, which may suffer of fill-in. Therefore, for a fair comparison, an IPM solving the symmetric indefinite augmented system (2.17) —whose $LDL^\top$ factorization is less affected by fill-in—should also be considered. In addition, nowadays there are available some very efficient sparse $LDL^\top$ solvers, such as the MA57 routine of the Harwell Subroutine Library (HSL) [45].

Table 4.6 provides the number of iterations and CPU time with the package Ipopt (v. 3.9.3) [79], which solves the augmented system using MA57, for the same instances used in previous Table 4.5. Optimality tolerances for Ipopt were adjusted to mimic as much as possible those of `BlockIP` and CPLEX (since Ipopt can deal with nonlinear problems, the potential number of tolerances to tune is much larger than for `BlockIP` and CPLEX). The average CPU times per iteration for `BlockIP`, CPLEX and Ipopt are also reported in Table 4.6. These ratios are more informative than the total CPU time, since Ipopt is not tailored for linear problems and then it might require more IPM iterations. We remind that the CPU time is dominated by

					Ipopt		CPU/Iter		
Instance	$1st$	k	m	n	Iter	CPU	`BlockIP`	CPLEX	Ipopt
LandS	4	3	23	62	16	0.01	0.00	0.00	0.00
AIRL_first	4	25	152	402	64	0.06	0.00	0.00	0.00
pltexpA2	188	6	686	2760	33	0.12	0.00	0.00	0.00
4node128	52	128	9486	32008	27	1.51	0.03	0.01	0.06
4node4096	52	4096	303118	1024008	52	213.78	1.55	0.38	4.11
4node16384	52	16384	1212430	4096008	44	1867.92	8.85	1.35	42.45
env_15	49	15	768	2046	99	0.20	0.00	0.00	0.00
env_1875	49	1875	90048	251286	140	19.15	0.14	0.05	0.14
envDiss8232	49	8232	395184	1103124	171	135.93	1.14	0.24	0.79
phone	8	32768	753665	3309569	19	166.58	1.81	1.13	8.77

Table 4.6: Results of linear instances with Ipopt, and CPU time per iteration for `BlockIP`, CPLEX and Ipopt

the solution of either the normal equations (in `BlockIP` and CPLEX) or augmented system (in Ipopt). We clearly observe that in these instances with only a few first-stage variables, CPLEX is the fastest per iteration, followed by `BlockIP`, and the solution of the augmented system appears as the less competitive option. The results of the below Section 4.3.3.3 will show that, for larger and more difficult instances (in particular, for instances of the two particular applications), Ipopt may be more efficient than CPLEX (in terms of time per iteration), but it is never competitive against our specialized approach.

4.3 Computational results for two particular applications

The instances of Tables 4.2 and 4.4 have a small number of first-stage linking variables, so they are loosely coupled among the different scenarios. For this reason, they can be considered "not too difficult", and specially tailored for decomposition algorithms. In order to get more difficult instances, we modified two problems from the literature: (1) a stochastic supply chain problem based on [5], which will be described in Subsection 4.3.1; (2) and instance *LandS* of Section 4.2, which corresponds to a stochastic electricity generation problem [4] and it is explained below in Subsection 4.3.2. The implementations developed for these two problems allowed us to manage the number of scenarios and the number of first- (mainly) and second-stage variables. Both problems were solved with CPLEX and `BlockIP`. For the solution with `BlockIP`, they were modeled with the splitting formulation (2.27), thus obtaining the structure of the constraints (2.28). For CPLEX we considered two formulations: the full splitting formulation (2.27) (as for `BlockIP`); and the dual of the original extensive form (2.4) (whose constraints matrix is (2.6), which is the transpose of (2.5) for linear problems), such that linking variables are converted into linking constraints which, in principle, avoid the presence of dense columns, thus potentially increasing the performance of the CPLEX barrier algorithm.

4.3.1 The stochastic supply chain problem

This problem was created from the models in [5] and [73]. The original model considered a supply chain network $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. The set $\mathcal{N}$ consists of suppliers $\mathcal{S}$, potential processing facilities $\mathcal{P}$ and customer centers $\mathcal{C}$, i.e., $\mathcal{N} = \mathcal{S} \cup \mathcal{P} \cup \mathcal{C}$. The decisions for first-stage are associated with a binary variable x_i, such that $x_i = 1$ if processing facility i is built, and 0 otherwise. The second-stage refers to tactical decisions on routing the flow of some product from suppliers to customers. Variable y_{ij} denotes the flow of the products from a node i to a node j in the network, where $(ij) \in \mathcal{A}$, and z_j denotes a shortfall of the product at customer j, meaning that it is impossible to meet the demand. The random vector ξ includes the demands, supplies, and costs (of processing, transportation, and shortage).

In our model, modifications were made to expand the problem, especially the first-stage variables. First, the number of nodes for each set could be chosen (this is provided as input data), thus allowing to increase the number of first-decision variables as well as the number of scenarios for expanding the problem as much as is necessary. Also, the capacity decision u_{ij} for arcs $(ij) \in \mathcal{A}$ was included, where u_{ij} cannot be greater than M. This capacity could not be exceeded during transportation operations. Finally, we added the nodes of suppliers $\mathcal{S}$ to the decision on whether or not to build the facility. Furthermore, all binary variables x_i were relaxed, i.e., $0 \le x_i \le 1$.

The resulting stochastic supply chain design problem is formulated as follows:

$$\begin{array}{lll} \min\limits_{x,u} & \sum\limits_{i\in\mathcal{S}\cup\mathcal{P}} c_i x_i + \sum\limits_{(ij)\in\mathcal{A}} f_{ij}u_{ij} + \mathcal{Q}(x,u) & \\ \text{s. to} & 0 \le x_i \le 1 \quad \forall i \in \mathcal{S}\cup\mathcal{P} & \\ & 0 \le u_{ij} \le M \quad \forall (ij) \in \mathcal{A} & \end{array} \qquad \text{and} \qquad \mathcal{Q}(x,u) = E_\xi[Q(x,u,\xi)], \tag{4.2}$$

where $Q(x, u, \xi)$ is the optimal value of the following problem:

$$
\begin{aligned}
Q(x,u,\xi) = \min_{y,z} \quad & \sum_{(ij)\in\mathcal{A}} q_{ij}y_{ij} + \sum_{j\in\mathcal{C}} h_j z_j && && \text{(4.3a)}\\
\text{s. to} \quad & \sum_{i\in\mathcal{S}} y_{ij} - \sum_{l\in C} y_{jl} = 0 && \forall j \in \mathcal{P} && \text{(4.3b)}\\
& \sum_{i\in\mathcal{P}} y_{ij} + z_j \geq d_j && \forall j \in \mathcal{C} && \text{(4.3c)}\\
& \sum_{j\in\mathcal{P}} y_{ij} \leq s_i x_i && \forall i \in \mathcal{S} && \text{(4.3d)}\\
& \sum_{i\in\mathcal{S}} y_{ij} \leq m_j x_j && \forall j \in \mathcal{P} && \text{(4.3e)}\\
& 0 \leq y_{ij} \leq u_{ij} && \forall (ij) \in \mathcal{A} && \text{(4.3f)}\\
& z_j \geq 0 && \forall j \in \mathcal{C}, && \text{(4.3g)}
\end{aligned}
$$

whose parameters are:

- c_i is the investment cost for supply and building facility i.
- f_{ij} is the cost-per-unit of capacity on the arc ij.
- q_{ij} represents the per-unit cost of transporting the product on arc ij.
- h_j is the per-unit penalty incurred for failing to meet the demand of the product at the customer center j.
- M denotes the maximum capacity per arc.
- d_j is the demand of customer j.
- s_i is the maximum supply capacity of supplier i.
- m_j is the maximum processing capacity of processing plant j.

The first-stage problem (4.2) consists of choosing the design variables x_i and u_{ij}. The objective function minimizes the total investment for nodes $\mathcal{S}$ and $\mathcal{P}$, arc capacities u_{ij}, and the expected value of the second-stage. Binary variables x_i are relaxed, and the upper bound M is set for all arc capacities.

The second-stage problem (4.3) involves the processing and the transportation of the product, where the objective function (4.3a) minimizes the transportation and shortage costs. The first set of constraints (4.3b) enforces the flow conservation across each processing node j. The next group of constraints (4.3c) requires that the total flow for the customer j plus its shortfall should be at least the demand d_j. Constraints (4.3d) ensure that the total flow from a supplier node i should not exceed the supply capacity s_i at that node if it is built. The set of constraints (4.3e) establishes that the number of units to the process should not be greater than the capacity m_j of facility j if it is built. If facility $i \in \mathcal{S}$ is not built, the above two groups of constraint will force all flow variables $y_{ij} = 0$ for all $i \in \mathcal{S}$. Finally, constraints (4.3f) and (4.3g) are the upper and lower bounds.

4.3.1.1 Test instances

The original model in [5] considered a supply chain with four suppliers (A, B, C, and D), four possible processing facilities (E, F, G, and H), and three customer centers (L, M, and N). This supply network is depicted in Figure 4.7. The product considered in [5] was uniform-quality wine in bulk (raw material). Four scenarios were considered (boom, good, fair and poor), with different probabilities associated with each one. It should be stressed that in [5] the supplies, transportation costs, and shortage costs were considered as deterministic parameters for the extensive form (or deterministic equivalent) formulation.

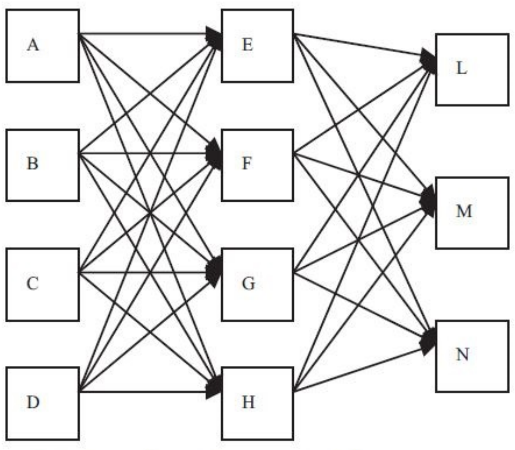


Figure 4.7: The network of the wine company [5]

We extended the original model in [5] in several ways. First, all the parameters of the problem were created utilizing a random generator considering different ranges of values. For instance, the range of investment costs for building each processing (bottling) plant was between 250,000 and 350,000. The costs per unit of arc capacities were between 1000 and 1,000,000. The unit production costs and market demands under each scenario were in the range of 500–750 and 550–800, respectively. The costs of transporting bulk wine from the suppliers to the processing facilities, and bottled wine from the processing facilities to the clients were, respectively, around 150 and 450. The unit storage costs at each distribution center were in the range of 10–16 thousand units. Furthermore, the maximum amount of bulk wine that can be shipped from the suppliers was between 250 and 350 units.

Next, we increased the number of first-stage variables by considering 10 suppliers, 10 processing facilities and 50 customers. This resulted in a problem with 620 first-stage variables, used as the base case from which the several instances in below Sections 4.3.3.1 and 4.3.4.1 are obtained by replicating the number of scenarios.

4.3.2 The stochastic power generation problem

This problem (based on [56]) consists of finding the optimal investment over various types of power plants to satisfy uncertain electricity demand. A two-period model with a set of $\mathcal{M}$ operating modes and a set of $\mathcal{T}$ different technologies is considered. The modes are a discrete representation of the true load-duration demand curve, where each mode is a rectangular approximation of a part of this curve. It is necessary to consider a large number of modes for a good approximation to reality. Several power plants use one of the available technologies. See [4, 11] for more details.

Technology $i \in \mathcal{T}$ has an associated investment cost c_i (a multiple of €/Mw), and a production cost q_i (a multiple of €/Mwh). Operating modes are defined by its duration t_j (hours) and demand d_j (Mw), for $j \in \mathcal{M}$; the demand $d_1 = \xi$ of the first mode is the stochastic parameter of this problem. First-stage decisions are the capacities x_i (Mw) invested in each technology $i \in \mathcal{T}$. Second-stage variables are the capacities y_{ij} effectively operated in each mode $j \in \mathcal{M}$, for each technology $i \in \mathcal{T}$. We assume technologies are always available, so they can be operated full-time.

The model formulation is the following:

$$\begin{aligned} \min_x \quad & \sum_{i\in\mathcal{T}} c_i x_i + \mathcal{Q}(x) \\ \text{s. to} \quad & \sum_{i\in\mathcal{T}} x_i \geq C \\ & \sum_{i\in\mathcal{T}} c_i x_i \leq B \\ & x_i \geq 0 \quad i \in \mathcal{T} \end{aligned} \qquad \text{and} \qquad \mathcal{Q}(x) = E_\xi[Q(x,\xi)], \tag{4.4}$$

where

$$\begin{aligned} Q(x,\xi) = \min_y \quad & \sum_{i\in\mathcal{T}} \sum_{j\in\mathcal{M}} q_i t_j y_{ij} && \text{(4.5a)} \\ \text{s. to} \quad & \sum_{j\in\mathcal{M}} y_{ij} \leq x_i \quad i \in \mathcal{T} && \text{(4.5b)} \\ & \sum_{i\in\mathcal{T}} y_{ij} = \xi \qquad j \in \{1\} && \text{(4.5c)} \\ & \sum_{i\in\mathcal{T}} y_{ij} = d_j \quad j \in \mathcal{M} \setminus \{1\} && \text{(4.5d)} \\ & y_{ij} \geq 0 \qquad i \in \mathcal{T}, j \in \mathcal{M}, && \text{(4.5e)} \end{aligned}$$

B and C are, respectively, a budget limit and a minimum capacity to be provided in the first-stage.

The first-stage problem (4.4) considers the investment cost for each technology, subject to a minimum total capacity and maximum budget constraints. The objective function (4.5a) of the second-stage problem consists of minimizing the operational costs for each technology effectively used in every mode. The constraints (4.5b) impose that available capacities cannot be exceeded, for each technology. The other two sets of constraints (4.5c)–(4.5d) impose demand satisfaction for every mode; the first constraint is for the first mode, whose demand is stochastic.

4.3.2.1 Test instances

The instance presented in [4] and [11], considered $m = 3$ operating modes and $n = 4$ available technologies, along with demands $d = (\xi, 2, 3)$ (that is, the demand of the first operating mode is stochastic) and load durations $t = (10, 6, 1)$. The three scenarios had the values of $\xi = (3, 5, 7)$ and $(0.3, 0.4, 0.3)$ probabilities, respectively. The investment costs were $c = (10, 7, 16, 6)$ for each technology, with production costs $q = (4, 4.5, 3.2, 5.5)$. The budget keeps all investments below $B = 120$ and minimum capacity is $C = 12$.

The previous basic instance was extended by increasing the number of technologies available (which is the same as the number of first-stage constraints), existing operating modes and scenarios for every instance. The rest of the parameters were randomly generated from the values of the basic instance. For instance, the budget and minimum capacity varied according to the number of technologies. The random demand took values between 2 and 10, the investment costs c in the range 2–20, q between 2 and 6, and t went up to 10.

Next, we began to increase the number of technologies (that is, first-stage decisions) until finding a considerable number of first-stage variables, as is shown in the Table 4.7. In these instances, parameters in `BlockIP` are default values.

			BlockIP			CPLEX barrier No Splitting		
1st	m	n	Iter	CPU	PCG	Iter	CPU	cb/BIP
4	31	70	26	0.00	87	14	0.00	0
9	56	155	43	0.00	223	19	0.00	0
20	111	342	26	0.00	74	20	0.01	2.97
100	511	1702	38	0.02	94	23	0.03	1.34
200	1011	3402	37	0.04	88	26	0.07	1.71
300	1511	5102	60	0.08	140	29	0.10	1.20

Table 4.7: Increasing the first-stage variables in Power Generation Problem

			BlockIP			CPLEX barrier no splitting		CPLEX barrier with splitting	
k	m	n	Iter	CPU	PCG	Iter	CPU	Iter	CPU
200	259380	511380	140	28.18	2747	19	149.76	21	109.34
400	519380	1023380	174	59.36	2504	25	424.54	30	262.32
600	779380	1535380	155	77.32	2076	32	616.63	29	250.59
800	1039380	2047380	192	128.81	2565	19	566.50	31	342.79
1000	1299380	2559380	200	312.07	6295	23	843.60	32	428.56

Table 4.8: Results for supply chain problem up to 1000 scenarios

The advantage of BlockIP can be appreciated since early performances (e.g., 20 variables in the first-stage). We decided to increase up to 600 (as it will be seen in below sections of results); in this case, rather than technologies, first-stage decisions could be considered as different power plants.

4.3.3 Results for linear instances

Next two subsections present results for the linear instances of the supply chain and power generation problem. Those instances were obtained by increasing the number of first-stage variables and/or scenarios in the base instances described in previous sections. In these runs, the optimality gap was set to 10^{-6} for both BlockIP and CPLEX, unless otherwise stated. For BlockIP, the initial tolerance and the tolerance reduction factor of the PCG were, respectively, $\epsilon_0 = 10^{-2}$ and $\beta = 0.95$ (which are BlockIP default values), unless otherwise stated.

4.3.3.1 Linear instances of the stochastic supply chain problem

From the base instance of 10 suppliers, 10 processing facilities and 50 customers—with a total of 620 first-stage variables—described in above Section 4.3.1.1, we generated a first set of stochastic instances with 200, 400, 600, 8000 and 1000 scenarios. Table 4.8 shows the results obtained with BlockIP (splitting formulation), and CPLEX (for both the with and without splitting formulations). The meaning of the columns is the same as in previous tables. These instances resulted to be difficult for BlockIP, as the spectral radius ρ was close to one in the early iterations. Therefore, the optimality gap was set to 10^{-5} for both BlockIP and CPLEX. The PCG tolerance reduction factor in BlockIP was also set to $\beta = 1$ (that is, the PCG tolerance was not reduced at each interior-point iteration). From Table 4.8 it is clearly observed that BlockIP was more efficient than CPLEX either with or without full splitting.

Table 4.9 reports results for a second set of larger instances with some scenar-

			`BlockIP`			CPLEX barrier no splitting		CPLEX barrier with splitting	
k	m	n	Iter	CPU	PCG	Iter	CPU	Iter	CPU
2000	2599380	5119380	48	48.34	204	11	647.42	20	627.82
3000	3899380	7679380	43	67.41	197	10	916.60	18	1208.98
4000	5199380	10239380	86	159.94	275	10	1242.50	18	2112.53
5000	6499380	12799380	81	214.08	379	10	1572.92	19	3678.25

Table 4.9: Results for supply chain problem up to 5000 scenarios

ios between 2000 and 5000; the largest instance has 6.4M constraints and 12.8M variables. The optimality tolerance was increased to 10^{-3} for both `BlockIP` and CPLEX barrier, which can be considered a reasonable choice for stochastic models with a large number of scenarios, and at the same time, it allowed us to avoid the last interior-point iterations where the spectral radius ρ usually tends to one. From Table 4.9 we see that `BlockIP` clearly outperformed both CPLEX variants(with or without full splitting). It is also worth noting that, unlike in Table 4.8, the CPLEX runs for the model with splitting were much slower than those without full splitting. This fact will be even more evident in Table 4.14 for the quadratic instances of the supply chain problem. This is due to the aggressive preprocessing applied by CPLEX to the splitting model, which removes a significant number of constraints and variables at the expense of modifying the matrix structure and significantly increasing the fill-in of the factorizations (some numbers will be provided below for the results of Table 4.14 for quadratic instances).

As summarized in Figure 4.8 (which plots the solution time against the number of scenarios for the instances of Tables 4.8 and 4.9), `BlockIP` always outperformed CPLEX, and one could infer that the difference between them will increase if the number of scenarios grows higher.

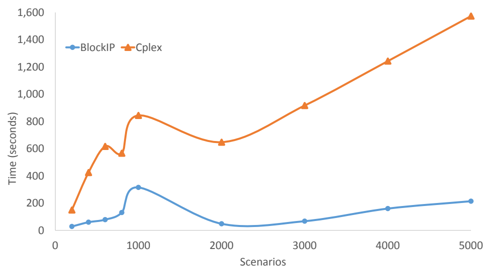


Figure 4.8: `BlockIP` vs CPLEX for linear supply chain problem (CPLEX without splitting)

4.3.3.2 Linear instances of the stochastic power generation problem

We generated a preliminary set of instances where the first-stage variables (the number of available technologies) were increased up to 600, the number of modes of electricity demand was set at 10, and the number of scenarios at 100. The default tolerances were used (i.e., optimality tolerance was 10^{-6}, $\epsilon_0 = 10^{-2}$ and $\beta = 0.95$). The performance of `BlockIP` was very promising, as it is shown in Table 4.10.

			BlockIP			CPLEX barrier No Splitting		
1st	m	n	Iter	CPU	PCG	Iter	CPU	cp/bip
400	80602	519602	83	7.14	559	23	87.50	12.25
450	90552	584552	85	8.20	533	20	86.42	10.52
500	100502	649502	81	9.02	470	23	110.02	12.18
550	110452	714452	90	12.45	714	28	139.01	11.16
600	120402	779402	85	12.00	491	19	108.76	9.05

Table 4.10: Increasing the number of the first-stage variables in power generation problem

Encouraged by the previous results, we worked on the 600 first-stage variables instance while increasing the number of scenarios to 1000. Default tolerances were also used. The results are given in Table 4.11, which shows that the performance of `BlockIP` is much better than the one of CPLEX either with or without full splitting. It is seen that CPLEX with either model required large executions in those instances. For this reason, the second set of even larger instances obtained by considering some scenarios between 2000 and 5000 only ran with `BlockIP`. The results are provided in Table 4.12. `BlockIP` was very efficient, being able to solve the largest case of 6M constraints and 39M variables in less than 18 minutes. It is also worth remarking that in those largest instances CPLEX ran out of memory (it exhausted the available 192 Gigabytes of RAM).

			`BlockIP`			CPLEX barrier no splitting		CPLEX barrier with splitting	
k	m	n	Iter	CPU	PCG	Iter	CPU	Iter	CPU
200	241402	1559402	89	30.07	703	22	589.79	33	1309.60
400	483402	3119402	107	66.73	790	126	17633.54	40	10547.41
600	725402	4679402	79	81.93	691	87	54178.15	41	33770.16
800	967402	6239402	106	145.09	1036	20	45467.37	40	73447.25
1000	1209402	7799402	84	145.06	761	117	309381.56	44	167873.29

Table 4.11: Results for power generation problem up to 1000 scenarios

			BlockIP		
k	m	n	Iter	CPU	PCG
2000	2419402	15599402	101	359.41	1074
3000	3629402	23399402	116	660.73	1204
4000	4839402	31199402	108	825.09	1193
5000	6049402	38999402	107	1027.31	1199

Table 4.12: Results for power generation problem up to 5000 scenarios

The plot in Figure 4.9 (which reports the solution time against the number of scenarios) helps better appreciate that `BlockIP` is by far faster than CPLEX in those instances.

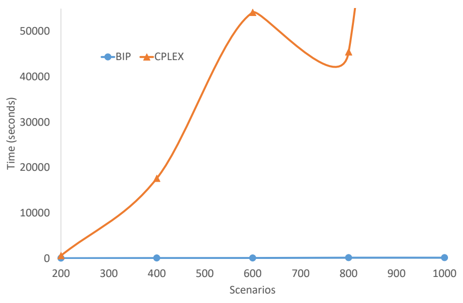


Figure 4.9: `BlockIP` vs CPLEX for linear power generation problem (CPLEX with splitting)

4.3.3.3 Normal equations vs augmented system

As it was advanced in previous Section 4.2.7, Table 4.13 reports the results with Ipopt (which solves the augmented system using the direct solver implemented in the HSL routine MA57), for the supply chain and electricity generation instances of 200 and 400 scenarios (that is, the first two and last two rows of Table 4.13 correspond, respectively, to the first two instances of Tables 4.8 and 4.11). These results show that Ipopt was outperformed by both CPLEX and `BlockIP` in the supply chain instances, but it was more efficient than CPLEX for the electricity generation problems. In some instances, however, Ipopt provided a suboptimal solution, unlike `BlockIP` and CPLEX which converged to the same optimal solution; a possible explanation is that Ipopt is tailored for nonlinear problems. Table 4.13 also provides the CPU time per iteration, which is more informative in order to compare how efficient is the computation of the direction with either the augmented system with a direct solver (Ipopt), normal equations with a direct solver (CPLEX), or normal equations combining direct and PCG solvers (`BlockIP`). The

		Ipopt no splitting		Ipopt with splitting		CPU/Iter				
						BlockIP	CPLEX		Ipopt	
Problem	k	Iter	CPU	Iter	CPU		no	with	no	with
Sup. Chain	200	107	212.2	84	3477.4	0.2	7.9	5.2	1.9	41.4
	400	121	789.0	85	20546.2	0.3	17.0	8.7	6.5	241.7
Elect. Gen.	200	36	121.7	57	627.8	0.3	26.8	39.6	3.4	11.0
	400	29	211.2	35	2050.2	0.6	139.9	263.6	7.3	58.6

Table 4.13: Results for supply chain and electricity generation instances with 200 and 400 scenarios with Ipopt, and CPU time per iteration for `BlockIP`, CPLEX and Ipopt

conclusions obtained from these results are: (i) if the augmented system is solved by a direct solver (Ipopt), the formulation without splitting should always be used; (ii) if normal equations are solved by a direct solver (CPLEX), the best formulation (either with or without splitting) depends of the problem structure; (iii) solving the normal equations by the specialized approach (that is, combining Cholesky and PCG) is the best option.

4.3.4 Results for quadratic instances

From the previous linear instances, we created a set of quadratic cases by adding convex separable quadratic costs to the first- and second-stage variables. The quadratic terms were synthetic and of the same order as the linear costs. The interest of testing quadratic instances relies on the fact that `BlockIP` is known to be faster for quadratic problems than for linear ones since quadratic terms tend to reduce the spectral radius ρ (see [20] for details). In all the runs, the optimality tolerance for `BlockIP` and CPLEX was 10^{-6}. For `BlockIP` the initial PCG tolerance and reduction factors were $\epsilon_0 = 10^{-3}$ and $\beta = 0.95$, which are its default values for quadratic problems. As for the linear instances, the results with `BlockIP` were computed with the splitting formulation (2.27), while for CPLEX both the splitting formulation (2.27) and the (no splitting) dual of the extensive form (2.3) were considered (note that in the latter case the constraints matrix contains both the transpose of (2.5) and the terms associated to quadratic costs).

4.3.4.1 Quadratic instances of the stochastic supply chain problem

As for the linear instances, we considered the base case of 620 first-stage variables and increased the number of scenarios up to 1000. The results can be seen in Table 4.14. Clearly, `BlockIP` outperformed both CPLEX runs (with and without full splitting). It is also observed that CPLEX is much less efficient with splitting in those instances.

			`BlockIP`			CPLEX barrier no splitting		CPLEX barrier with splitting	
k	m	n	Iter	CPU	PCG	Iter	CPU	Iter	CPU
200	259380	511380	89	26.08	2774	17	132.12	15	1299.24
400	519380	1023380	146	78.34	3952	18	338.61	16	10960.21
600	779380	1535380	112	108.95	3703	20	403.64	15	36211.61
800	1039380	2047380	126	210.26	5661	19	548.12	16	102556.88
1000	1299380	2559380	129	289.83	6048	21	788.77	14	173633.19

Table 4.14: Results for supply chain problem up to 1000 scenarios

After analyzing the CPLEX outputs, we concluded this is due to the large fill-in created to the reduced problem after the aggressive preprocessing applied by CPLEX.

For example, for the 1000 scenarios case of Table 4.14, CPLEX without splitting has after preprocessing a reduced problem with 650620 rows, 1331240 columns, and 3621240 nonzeros, but the number of nonzeros in the lower triangle of $AA^\top$ is 22100000, and the number of nonzeros in the factor (after reordering) is 346453439. For the splitting model of the same instance, these numbers are 680000 rows, 650620 columns, and 2970000 nonzeros (then same order as for the no-splitting case), but the number of nonzeros in the lower triangle of $AA^\top$ is 312500000 (14 times higher), and the number of nonzeros in the factor (after reordering) is 10855849207 (31 times higher). This explains the poor performance of the splitting model in some cases.

Next, we raised the number of scenarios up to 5000. The results are shown in Table 4.15. CPLEX was not executed with the splitting model for these instances to avoid large CPU times. `BlockIP` is also faster than CPLEX. In short, the differences in this quadratic problem can also be seen in Figure 4.10, where `BlockIP` remains always below the time of CPLEX, even in small instances.

			`BlockIP`			CPLEX barrier no splitting	
k	m	n	Iter	CPU	PCG	Iter	CPU
2000	2599380	5119380	170	889.21	9538	22	1199.15
3000	3899380	7679380	193	1052.82	6834	21	1701.36
4000	5199380	10239380	147	1054.70	5206	20	2307.52
5000	6499380	12799380	149	1701.24	5282	19	2680.00

Table 4.15: Results for supply chain problem up to 5000 scenarios

			`BlockIP`			CPLEX barrier no splitting		CPLEX barrier with splitting	
k	m	n	Iter	CPU	PCG	Iter	CPU	Iter	CPU
200	41902	259902	43	5.45	1760	71	336.45	65	10.02
400	83902	519902	42	11.19	1911	66	652.94	61	18.13
600	125902	779902	43	14.77	1471	67	996.91	69	37.18
800	167902	1039902	46	22.42	1575	73	1447.08	62	48.01
1000	209902	1299902	56	191.98	13604	72	1784.58	62	63.80
2000	419902	2599902	58	258.42	9037	73	3610.31	64	138.01
3000	629902	3899902	60	153.12	3205	74	5544.63	67	211.03
4000	839902	5199902	59	219.14	3493	78	7714.95	67	282.93
5000	1049902	6499902	61	227.20	2738	77	9599.25	67	363.00
10000	2099902	12999902	59	457.10	2509	75	19681.28	70	764.49

Table 4.16: Results for power generation problem up to 10000 scenarios

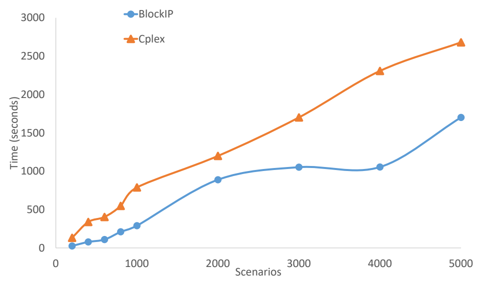


Figure 4.10: `BlockIP` vs CPLEX for quadratic supply chain problems (CPLEX without splitting)

4.3.4.2 Quadratic instances of the stochastic power generation problem

We considered as the base case the instance with 100 first-stage variables, such that the number of scenarios can be largely increased without obtaining an out-of-memory error by CPLEX (like it happened for linear instances and the 600 first-stage variables base case). Table 4.16 presents the results for up to 10000 scenarios (`BlockIP` vs CPLEX barrier with or without full splitting).

In these executions the model with full splitting was the most efficient variant for CPLEX; anyway, it was still outperformed by `BlockIP` except for the instances with 1000 and 2000 scenarios. In those two instances, `BlockIP` required many PCG iterations and this explains its poor behavior. The good results obtained in most cases with `BlockIP` are evident in Figure 4.11.

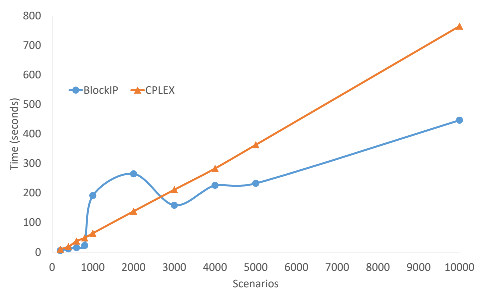


Figure 4.11: `BlockIP` vs CPLEX for quadratic power generation problem (CPLEX with splitting)

Chapter 5

Conclusions

5.1 Conclusion

In this research, we explored the performance of a specialized interior-point algorithm for block-angular problems (implemented in the `BlockIP` package) for two-stage stochastic optimization while using a splitting technique. The computational experiments used small- and large-scale instances to test both linear and quadratic objective functions. The performance of `BlockIP` was compared against alternative state-of-the-art interior-point (CPLEX barrier) and other specialized solvers for stochastic optimization.

The achievements are the following:

- The contribution applied a new method for solving the two-stage stochastic optimization models with the splitting technique using `BlockIP`. Where new functions were added to the BlockIP optimization package (Splitting).
- The results proved that `BlockIP` is competitive, mainly when the instances have a large number of first-stage variables.
- However, for the list of (small to medium) problems in [4], `BlockIP` was outperformed by the other approaches. These instances has few variables in the first-stage and the large-scale comes increasing the number of scenarios.

This Ph.D. thesis has potential in areas like computing efficient algorithms, and in a socio-economic environment helping the decision-makers who need to analyze many first-stage variables (present decisions), with a large number of future scenarios, and in a very short time. This gives them and their organization an advantage to survive the wild and competitive world. Besides, it should open up a new frontier in the knowledge of optimization techniques with efficient algorithms. Since solves large-scale *difficult instances* mean more realistic problems and better projections for the future.

5.2 Accomplishments

As a result of this thesis, an article was submitted to *Optimization Methods and Software*, titled A new interior-point approach for large two-stage stochastic problems (GOMS-2020-0067), currently under review.

Besides, this work was presented in the 30th European Conference on Operational Research (EURO 2019), in Dublin, Ireland from the 23rd to the 26th of June, 2019, co-authored with Prof. Jordi Castro, with the name of *A new interior-point approach for large two-stage stochastic problems*.

5.3 Future directions

Although the technique shows a big difference in the performance in terms of the CPU times for some problems, it was not enough for all the cases. Therefore future work should include:

- Testing our new instances in specialized solvers with the same computer for a better comparison.
- Create new instances for different applications to spread the potential of these results. The generator GENSLP could be used [49] and some others could include existing problems with the relaxation of integer variables.
- Testing multi-stage problems, which have a primal block-angular structure in each block with `BlockIP` and the splitting technique.
- The applications could include integer variables

Appendix A

2-stoch-prog-IPM

Download

The 2-stoch-prog-IPM.zip is available in Jordi Castro's web page `http://www-eio.upc.es/~jcastro/`. Section *Data*

Requirement

For making the comparison you will need the CPLEX software application.

Installation

The application runs on Linux. The 2-stoch-prog-IPM.zip file must be unzip. For unzip the file from the terminal, the command used is:

```
> unzip 2-stoch-prog-IPM.zip
```

The next step is to compile `BlockIP`, from the terminal, inside the directory 2-stoch-prog-IPM, the commands are:

```
> make clean
```

```
> make
```

In this step `BlockIP` will be ready to work, with all the associated problems.

Content

2-stoch-prog-IPM application has all the folders of the two general instances that are created to measure the performance of `BlockIP` and CPLEX with large variables in the first-stage. Also, the `BlockIP` library.

- Supply Chain

- Electric

The directories can be divided into executable folders and the ones for the results.

In the first group, we can find two subgroups, one for each type of problem:

- Supply Chain
  - SCPrimal. It is the primal linear version.
  - SCDual. It is the linear dual version of the problem.
  - SCPQuad for the primal quadratic version.
  - SCDQuad for the dual quadratic version

And the second subgroup is:

- power generation
  - ElPrimal. It is the primal linear version of this problem (Electric)
  - ElDual, with the linear dual version of the problem (ElectricDual)
  - ElPQuad for the primal Quadratic version
  - ElDQuad for the dual Quadratic version

Inside each folder, we can find a namefile.C file with the code in C++ that generate and solve the problem with BlockIP. Each problem required some input data, which generate a specific instance. These values can be given directly or using the `Comp` script (explained later) from the Results folders. Beside, each folder contains the Makefile file.

Executing directly

After compiling `BlockIP`, you need to be inside the directory that you want to run. For example

```
> cd SCPrimal
```

Then, execute with some parameters: <number of Supplies> <number Pfailities> <number of clients> and <number of blocks>. Before this data you need the `-SupplyChainEsc` flag. For example to run an instance with 4 suppliers, 3 processing facilities, 3 Clients and 3 scenarios you need the following command:

```
> ./SCPrimal -SupplyChainEsc 4 3 3 3
```

For the group of Electric (power generation problem), the `-ElectricEsc` flag is need, for example:

```
> cd ElPrimal
> ./Electric -ElectricEsc 2 20 10 10
```

This command runs the instance with 2 stages, 20 number of technologies, 10 different modes, and 10 scenarios. The problems must have more than 2 scenarios. (more details below)

The second group, Results, are folders to manage the outcomes. Here, each folder organizes the results with the aid of a scripts file for each problem:

- Supply Chain
  - SCResults, for the linear performances.
  - SCQResults, for the quadratics problems.
- Electric
  - ElResults, for linear experiments.
  - ElQResults, for quadratic instances.

Inside each one of those folders, we can find the script with all the commands that help you to create new instances and make the comparison with CPLEX. Also, the folders with the outcomes.

For example, in SCResults we can find the next folders:

- ResultsPrimal. For the outcomes of BlockIP.
- ResultDual. For generate the problem dual and create the MPS file for CPLEX.
- ResultsMPS. For storage the MPS file resulting from BlockIP and then read it by CPLEX.
- ResultsCplexBarrier. Here we save all the results of the CPLEX barrier algorithm.
- ResultsCplexDual. Only store files if comparisons with dual CPLEX are activated.

The scripts inside SCResults are:

- Comp. Create different instances and help you to make a comparison between BlockIP and CPLEX. Then, send the outcomes to the corresponding folder of results.
- ScriptCplexBarrier.txt with the commands to runs CPLEX with the barrier algorithm (baropt).
- ScriptCplexDual.txt with the commands to runs CPLEX with the Dual algorithm.

Modifying the size of the instances

The instances can be manipulated in order to increase, mainly, the first-stage variables.

For creating the Supply Chain instances, you must edit the `Comp` script. Here, we have four variables.

- S1. It is the number of suppliers that we what to consider
- P1. It is the number of process facilities
- C1. The number of clients and
- B1. The number of scenarios.

For the power generation problem, we have four variables too.

- S1. The number of stages, always 2.
- P1. The number of technologies
- C1. The different modes
- B1. The number of scenarios.

How to generate and solve new instances

For creating a new set of instances, for example, of Supply Chain with various scenarios.

- Step 1. Go inside SCResults folder
- Step 2. Open the Comp script
- Step 3. Check the files and directions for the outcomes files: ResultPrimal, ResultsMPS, and ResultsCplexBarrier (ResultsCplexDual, only if is activated)
- Step 4. Edit the variables (S1, P1, C1) for creating the specific instance. For B1 variable, that refers to the number of scenarios, you can use the for loop with the different numbers.
- Step 5. Execute the Comp file from SCResults.
- Step 6. Look for the outcomes inside their corresponding directories.

Appendix B

General Implementations of `BlockIP`

General Implementation

The solver `BlockIP` has been implemented in C++. It is around 17000 lines of source code, aside from the external package for Cholesky factorization [68], The `BlockIP` can be obtained for research purposes from `http://www-eio.upc.edu/~jcastro/BlockIP.html`. The distribution contains a reference manual and a small example [18].

Additional features of the package `BlockIP` are described more exhaustive in [20] and [18] :

- `BlockIP` may solve linear, quadratic and convex linearly constrained block angular optimization problems.
- Problems can be read in four different formats:
  1. BlockIP callable library, which create matrices and vectors as is shown in Figure B.1.
  2. `BlockIP` file, which has the matrices and vector in a format of sparse matrices (i, j, a_{ij}). (i, j), denotes the rows, and the columns and A_{ij} value (Figure B.2).
  3. The MPS format explained in Chapter 4, and
  4. SML (structure conveying modeling language) from AMPL [31].
- It implements two types of directions: The standard Newton direction and the second-order heuristic direction.
- If the system (3.6) becomes more ill-conditioned closer to the optimal, and PCG gives inaccurate solutions, `BlockIP` switches to Cholesky for (3.4).

- The solver has four different preconditioners for increasing efficiency.

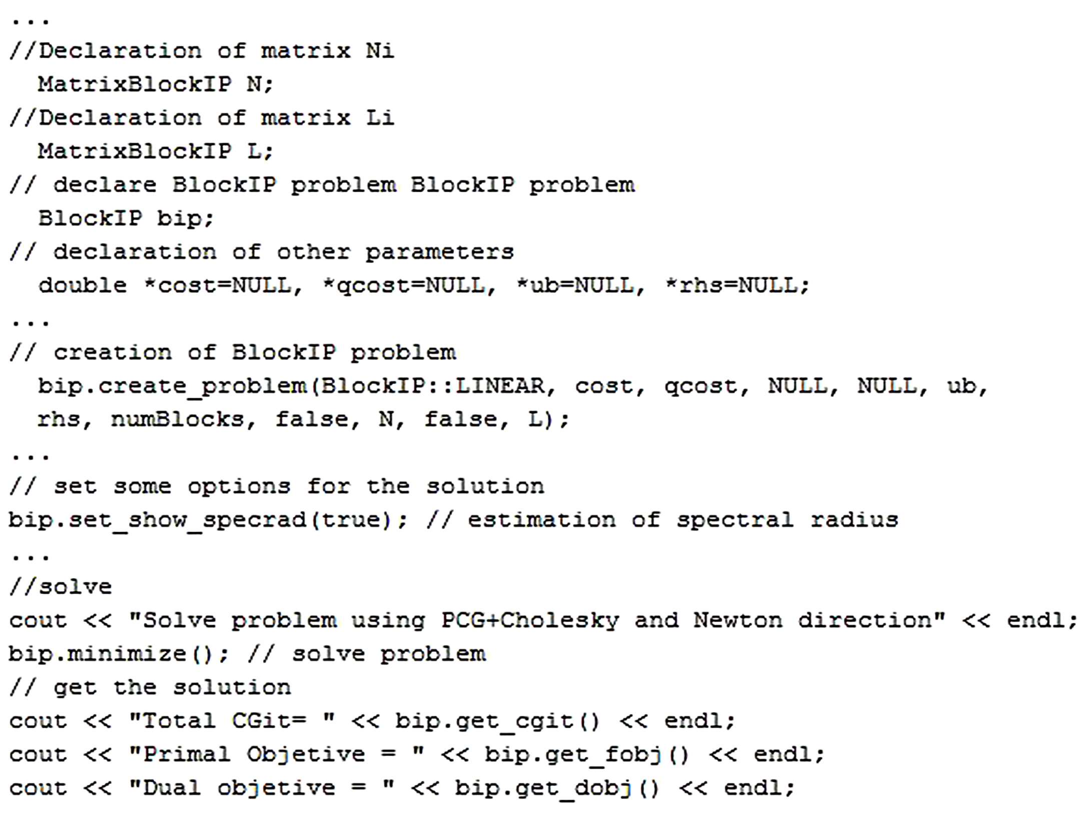
- `BlockIP` can remove inactive mutual capacity constraints to reduce the dimension of the Schur complement, thus lowering the computational effort. In this regard a linking constraint i ($\leq$) could be removed using the complementary condition $\lambda^i s^i = 0$, the following conditions are satisfied [15] [42]:

  1. The optimality gap is small.

  2. Its primal linking slack s^i is far enough from 0 ($s^i \gg 0$).

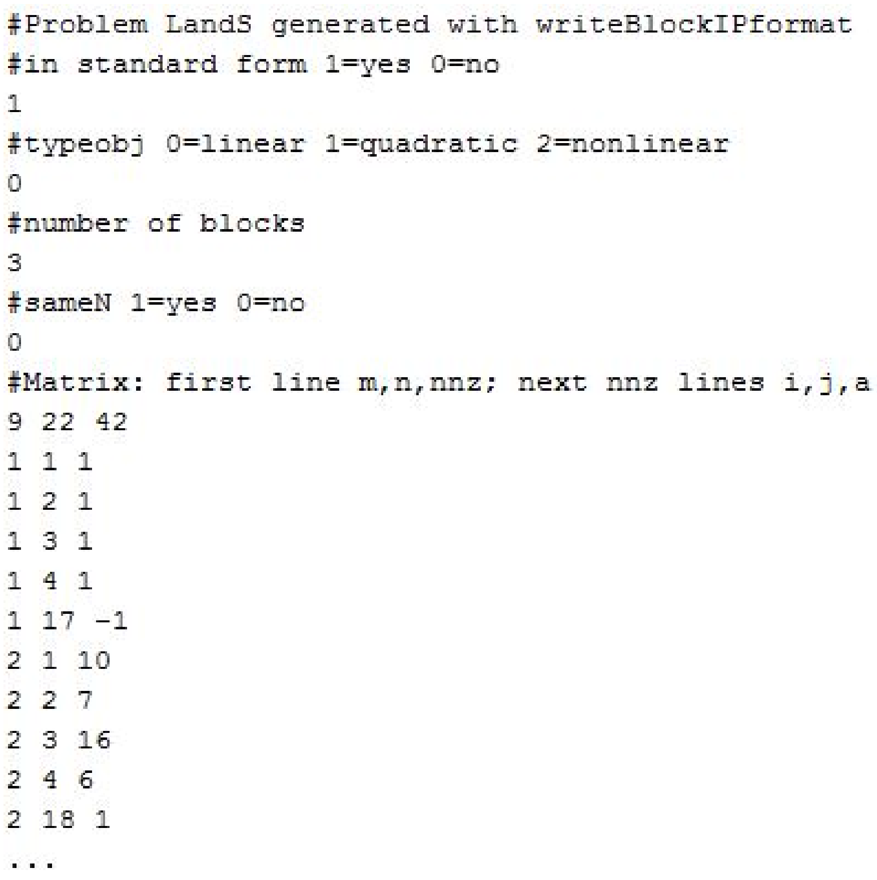
  3. Its Lagrange multiplier λ^i is close to 0.

- The stopping criteria are based in [70], which one guarantees better $\Delta\lambda_2$ directions as we get closer to the solution. By default, `BlockIP` uses a tolerance of $\epsilon = 10^{-2}$ for lineal problems and $\epsilon = 10^{-3}$ for quadratic instances. (it is proven that the preconditioner is more effective in the last instances [19])



```
...
//Declaration of matrix Ni
  MatrixBlockIP N;
//Declaration of matrix Li
  MatrixBlockIP L;
// declare BlockIP problem BlockIP problem
  BlockIP bip;
// declaration of other parameters
  double *cost=NULL, *qcost=NULL, *ub=NULL, *rhs=NULL;
...
// creation of BlockIP problem
  bip.create_problem(BlockIP::LINEAR, cost, qcost, NULL, NULL, ub,
  rhs, numBlocks, false, N, false, L);
...
// set some options for the solution
bip.set_show_specrad(true); // estimation of spectral radius
...
//solve
cout << "Solve problem using PCG+Cholesky and Newton direction" << endl;
bip.minimize(); // solve problem
// get the solution
cout << "Total CGit= " << bip.get_cgit() << endl;
cout << "Primal Objetive = " << bip.get_fobj() << endl;
cout << "Dual objetive = " << bip.get_dobj() << endl;
```

Figure B.1: Code from BlockIP callable library



```
#Problem LandS generated with writeBlockIPformat
#in standard form 1=yes 0=no
1
#typeobj 0=linear 1=quadratic 2=nonlinear
0
#number of blocks
3
#sameN 1=yes 0=no
0
#Matrix: first line m,n,nnz; next nnz lines i,j,a
9 22 42
1 1 1
1 2 1
1 3 1
1 4 1
1 17 -1
2 1 10
2 2 7
2 3 16
2 4 6
2 18 1
...
```

Figure B.2: BlockIP file format generated by writeBlockIPformat (Matlab)

Details of `BlockIP` -Stored sparse matrices-

This kind of algorithms, large scale, handle the sparse matrices with the column-wise and row-wise method. This storage the nonzero values of the sparse matrix in one-dimensional three arrays and they do not store any unnecessary zero elements.

For example, in the column-wise matrix *inicola* is the index to the first element of each column and is of length $n+1$ where n is the number of columns, *irowa* denotes the row position for each element and a is its value.

$$\begin{bmatrix} 0 & a & 0 & 0 \\ b & c & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & e & 0 & f \end{bmatrix} \Rightarrow \begin{array}{rcl} inicola & = & (0,1,4,5,6) \\ irowa & = & (1,0,1,3,2,3) \\ a & = & (b,a,c,e,d,f) \end{array}$$

On the other hand, the row-wise matrix is stored by *inirowa* for the index to the first element of each row, and with the length of $m+1$ where m is the number of rows, *icola* denotes the column position for each element.

$$\begin{bmatrix} 0 & a & 0 & 0 \\ b & c & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & e & 0 & f \end{bmatrix} \Rightarrow \begin{array}{rcl} inirowa & = & (0,1,3,4,5) \\ icola & = & (1,0,1,2,1,3) \\ a & = & (a,b,c,d,e,f) \end{array}$$

Appendix C

MPS Format

For a better understanding of the MPS format, here is an example of the power generation problem (LandS.core) written first in lp-format (C.1), this format is most commonly used in optimization models:

$$
\begin{aligned}
\min \quad & 10X_1 + 7X_2 + 16X_3 + 6X_4 \\
& + 40Y_{11} + 24Y_{12} + 4Y_{13} + 45Y_{21} + 27Y_{22} + 4.5Y_{23} \\
& + 32Y_{31} + 19.2Y_{32} + 3.2Y_{33} + 55Y_{41} + 33Y_{42} + 5.5Y_{43} \\
\text{subject to:} & \\
& X1 + X2 + X3 + X4 >= 12 \\
& 10X1 + 7X2 + 16X3 + 6X4 <= 120 \\
& -X1 + Y11 + Y12 + Y13 <= 0 \\
& -X2 + Y21 + Y22 + Y23 <= 0 \\
& -X3 + Y31 + Y32 + Y33 <= 0 \\
& -X4 + Y41 + Y42 + Y43 <= 0 \\
& Y11 + Y21 + Y31 + Y41 = 1 \\
& Y12 + Y22 + Y32 + Y42 = 3 \\
& Y13 + Y23 + Y33 + Y43 = 2 \\
& Xi \geq 0 \qquad i = 1, ..., 4 \\
& Yij \geq 0 \qquad i = 1, ..., 4 \qquad j = 1, ..., 3
\end{aligned}
\tag{C.1}
$$

Now, a file with the MPS format of the same problem is shown in Figure C.1.

```
NAME          LandS
ROWS
 N  OBJ
 G  MINCAP
 L  BUDGET
 L  OPLIM1
 L  OPLIM2
 L  OPLIM3
 L  OPLIM4
 E  DEMAND1
 E  DEMAND2
 E  DEMAND3
COLUMNS
    X1        OBJ       10.0           MINCAP    1.0
    X1        BUDGET    10.0           OPLIM1    -1.0
    X2        OBJ       7.0            MINCAP    1.0
    X2        BUDGET    7.0            OPLIM2    -1.0
    X3        OBJ       16.0           MINCAP    1.0
    X3        BUDGET    16.0           OPLIM3    -1.0
    X4        OBJ       6.0            MINCAP    1.0
    X4        BUDGET    6.0            OPLIM4    -1.0
    Y11       OBJ       40.0           OPLIM1    1.0
    Y11       DEMAND1   1.0
    Y12       OBJ       24.0           OPLIM1    1.0
    Y12       DEMAND2   1.0
    Y13       OBJ       4.0            OPLIM1    1.0
    Y13       DEMAND3   1.0
    Y21       OBJ       45.0           OPLIM2    1.0
    Y21       DEMAND1   1.0
    Y22       OBJ       27.0           OPLIM2    1.0
    Y22       DEMAND2   1.0
    Y23       OBJ       4.5            OPLIM2    1.0
    Y23       DEMAND3   1.0
    Y31       OBJ       32.0           OPLIM3    1.0
    Y31       DEMAND1   1.0
    Y32       OBJ       19.2           OPLIM3    1.0
    Y32       DEMAND2   1.0
    Y33       OBJ       3.2            OPLIM3    1.0
    Y33       DEMAND3   1.0
    Y41       OBJ       55.0           OPLIM4    1.0
    Y41       DEMAND1   1.0
    Y42       OBJ       33.0           OPLIM4    1.0
    Y42       DEMAND2   1.0
    Y43       OBJ       5.5            OPLIM4    1.0
    Y43       DEMAND3   1.0
RHS           RIGHT
    RIGHT     MINCAP    12.0
    RIGHT     BUDGET    120.0
    RIGHT     DEMAND1   1.0
    RIGHT     DEMAND2   3.0
    RIGHT     DEMAND3   2.0
ENDATA
```

The MPS format was initially introduced by IBM [46] to express linear and integer programs in a standard way. The format is a fixed column format where the files must be separated by white space (blank, tab, etc.). This file has different sections explained below [64].

The first label that can find in the file is NAME. Then, it has the specific name of the instance problem (LandS). The ROWS set defines the names of all the constraints, comes with an identifier: E for equality rows, L for less-than (<=) rows, G for greater-than (>=) rows, and N for the objective function.

The COLUMNS section has the coefficients of A-matrix. All entries for a given column must be placed consecutively. The rows not mentioned are implied to have a coefficient of zero.

The RHS section allows right-hand-side vectors to be defined. In cases when this value is zero, the number is omitted.

The optional BOUNDS section allows placing upper and lower bounds. Variables not mentioned in the set are taken to be non-negative. A type UP bound of means an upper bound and LO are for the lower bound. An FX (”fixed”) bound type represents the variable with upper and lower bounds equal to a single value. An FR (”free”) bound type defines the variable that has neither lower nor upper bounds. MI means minus infinity (lower bound $= -\infty$). PL Plus infinity (upper bound $= \infty$).

There is another optional section called RANGES. It denotes the limits for the constraints that are between two values. The final card must be ENDATA, and it is the last entry of the file.

Bibliography

[1] K.M. Anstreicher and R.A. Bosch. "Long steps in an O (n3L) algorithm for linear programming". *Mathematical Programming* 54.1-3 (1992), pp. 251–265.

[2] J. Arantes and J.R. Birge. "Matrix structure and interior point methods in stochastic programming". *Presentation at Fifth Int. Stochastic Programming Conf., Ann Arbor, MI* (1989).

[3] A. Arbel. *Exploring Interior-Point Linear Programming: Algorithms and Software.* mit Press, (1993).

[4] K.A. Ariyawansa and A.J. Felt. "On a new collection of stochastic linear programming test problems". *INFORMS Journal on Computing* 16.3 (2004), pp. 291–299.

[5] A. Azaron et al. "A multi-objective stochastic programming approach for supply chain design considering risk". *International Journal of Production Economics* 116.1 (2008), pp. 129–138.

[6] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear programming: theory and algorithms.* John Wiley & Sons, (2013).

[7] J.F. Benders. "Partitioning procedures for solving mixed-variables programming problems". *Numerische mathematik* 4.1 (1962), pp. 238–252.

[8] M. Benzi. "Splittings of symmetric matrices and a question of Ortega". *Linear Algebra and its Applications* 429.10 (2008), pp. 2340–2343.

[9] J.R. Birge, R.M. Freund, and R. Vanderbei. "Prior reduced fill-in in solving equations in interior point algorithms". *Operations Research Letters* 11.4 (1992), pp. 195–198.

[10] J.R. Birge and D.F. Holmes. "Efficient solution of two-stage stochastic linear programs using interior point methods". *Computational Optimization and Applications* 1.3 (1992), pp. 245–276.

[11] J.R. Birge and F. Louveaux. "Introduction to stochastic programming". *Springer Science & Business Media* (2011).

[12] J.R. Birge and L. Qi. "Computing block-angular Karmarkar projections with applications to stochastic programming". *Management Science* 34.12 (1988), pp. 1472–1479.

[13] J. Bisschop. *AIMMS optimization modeling.* Paragon Decision Technology B.V, (2006).

[14] S.P. Bradley, A.C. Hax, and T.L. Magnanti. "Applied mathematical programming". *Addison Wesley* (1977).

[15] J. Castro. "A specialized interior-point algorithm for multicommodity network flows". *SIAM journal on Optimization* 10.3 (2000), pp. 852–877.

[16] J. Castro. "An interior-point approach for primal block-angular problems". *Computational optimization and Applications* 36.2-3 (2007), pp. 195–219.

[17] J. Castro. *An introduction to the affine scaling algorithm for linear programming.* Research Report DR 2000/02. (2000).

[18] J. Castro. "Interior-point solver for convex separable block-angular problems". *Optimization Methods and Software* 31.1 (2016), pp. 88–109.

[19] J. Castro. "Solving quadratic multicommodity problems through an interior-point algorithm". *IFIP Conference on System Modeling and Optimization.* Springer. (2001), pp. 199–212.

[20] J. Castro and J. Cuesta. "Quadratic regularizations in an interior-point method for primal block-angular problems". *Mathematical programming* 130.2 (2011), pp. 415–445.

[21] I.F. Csaba and S Zoltán. "Solving two-stage stochastic programming problems with level decomposition". *Computational Management Science* 4.4 (2007), pp. 313–353.

[22] J. Czyzyk, R. Fourer, and S. Mehrotra. "A study of the augmented system and column-splitting approaches for solving two-stage stochastic linear programs by interior-point methods". *ORSA Journal on Computing* 7.4 (1995), pp. 474–490.

[23] J. Czyzyk, M.P. Mesnier, and J.J. Moré. "The NEOS server". *IEEE Computational Science and Engineering* 5.3 (1998), pp. 68–75.

[24] D. Den Hertog and C. Roos. "A survey of search directions in interior point methods for linear programming". *Mathematical Programming* 52.1-3 (1991), pp. 481–509.

[25] I.I. Dikin. "Iterative solution of problems of linear and quadratic programming". *Doklady Akademii Nauk.* Vol. 174. 4. Russian Academy of Sciences. (1967), pp. 747–748.

[26] W.S. Dorn. "Duality in quadratic programming". *Quarterly of Applied Mathematics* 18.2 (1960), pp. 155–162.

[27] E.F.D. Elison, G. Miltra, and V Zverovich. *FortSP: a stochastic programming solver. OptiRisk Systems.* Available in: `https://optirisk-systems.com/wp-content/uploads/2018/05/FortspManual.pdf`. [Accessed on 2020-06-22].

[28] L.F. Escudero, M.A. Garín, and A. Unzueta. “Scenario cluster Lagrangean decomposition for risk averse in multistage stochastic optimization”. *Computers & Operations Research* 85 (2017), pp. 154–171.

[29] L.F. Escudero et al. “Lagrangian decomposition for large-scale two-stage stochastic mixed 0-1 problems”. *Top* 20.2 (2012), pp. 347–374.

[30] A.J. Felt. *Test-Problem Collection for Stochastic Linear Programming.* Available in: `https://www4.uwsp.edu/math/afelt/slptestset/download.html`. [Accessed on 2019-03-16].

[31] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: a modeling language for mathematical programming.* Vol. 1. Scientific Press San Francisco, (1993).

[32] E. Fragniere. “Choix énergétiques et environnementaux pour le Canton de Genève”. No. 412. PhD thesis. Universitè de Genève, (1995).

[33] H.I. Gassmann. “Optimal harvest of a forest in the presence of uncertainty”. *Canadian Journal of Forest Research* 19.10 (1989), pp. 1267–1274.

[34] H.I. Gassmann. *stocfor1, stocfor2, stocfor3.* Available in: `http://www.netlib.org/lp/data/`. [Accessed on 2018-11-15].

[35] H.I. Gassmann and B. Kristjánsson. “The SMPS format explained”. *IMA Journal of Management Mathematics* 19.4 (2008), pp. 347–377.

[36] P.E. Gill et al. “On projected Newton barrier methods for linear programming and an equivalence to Karmarkar’s projective method”. *Mathematical programming* 36.2 (1986), pp. 183–209.

[37] G.H Golub and C.F. Van Loan. *Matrix computations.* Johns Hopkins University press, (2012).

[38] J. Gondzio. “Interior point methods 25 years later”. *European Journal of Operational Research* 218.3 (2012), pp. 587–601.

[39] J. Gondzio. *”Preconditioned conjugate gradients in an interior point method for two-stage stochastic programming”.* Working Paper WP-94-130, IIASA International Institute for Applied Systems Analysis, Laxemburg, Austria.(1994).

[40] J. Gondzio, P. González-Brevis, and P. Munari. “Large-scale optimization with the primal-dual column generation method”. *Mathematical Programming Computation* 8.1 (2016), pp. 47–82.

[41] J. Gondzio, P. González-Brevis, and P. Munari. “New developments in the primal–dual column generation technique”. *European Journal of Operational Research* 224.1 (2013), pp. 41–51.

[42] J. Gondzio and A. Grothey. “Exploiting structure in parallel implementation of interior point methods for optimization”. *Computational Management Science* 6.2 (2009), pp. 135–160.

[43] J. Gondzio and A. Grothey. “Solving non-linear portfolio optimization problems with the primal-dual interior point method”. *European Journal of Operational Research* 181.3 (2007), pp. 1019–1029.

[44] D. Holmes. *A (PO)rtable (S)tochastic programming (T)est (S)et (POSTS).* Available in: `http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html`. (1995) [Accessed on 2019-03-16].

[45] HSL. *A collection of Fortran codes for large scale scientific computation.* Available in: `http://www.hsl.rl.ac.uk`. [Accessed 2020-10-06].

[46] IBM, ILOG S.A. *ILOG CPLEX 10.0 file formats.* Available in: `https://www.lix.polytechnique.fr/.../cplex/reffileformatscplex.pdf`. [Accessed on 2019-06-07].

[47] IBM Knowledge Center. *Barrier optimizer.* Available in: `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.2/ilog.odms.cplex.help/CPLEX/UsrMan/topics/cont_optim/qp/10_change_terms.html`. [Accessed on 2018-02-22].

[48] IBM Knowledge Center. *Dual simplex optimizer.* Available in: `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.cplex.help/CPLEX/UsrMan/topics/cont_optim/simplex/06_simplex_dual.html`. [Accessed on 2019-06-17].

[49] P. Kall and J Mayer. "On testing SLP codes with SLP-IOR". *New trends in mathematical programming.* Vol. 13. (1998). Springer Boston. MA, pp. 115–135.

[50] P. Kall, J. Mayer, et al. *Stochastic linear programming.* Vol. 7. Springer, (1976).

[51] N. Karmarkar. "A new polynomial-time algorithm for linear programming". *Proceedings of the sixteenth annual ACM symposium on Theory of computing.* ACM. (1984), pp. 302–311.

[52] M. Kaut and M. Kopa. "COIN-OR tools for stochastic programming". *On Selected Software for Stochastic Programming* (2008), pp. 88–116.

[53] K. Kibaek and V.M. Zavala. "Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs". *Mathematical Programming Computation* 10.2 (2018), pp. 225–266.

[54] W.K. Klein Haneveld and M.H. Van der Vlerk. "Stochastic integer programming: General models and algorithms". *Annals of Operations Research* 85 (1999), pp. 39–57.

[55] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov. "New variants of bundle methods". *Mathematical programming* 69.1-3 (1995), pp. 111–147.

[56] F.V. Louveaux and Y. Smeers. "Optimal Investments for Electricity Generateion: A Stochastic Model and A Test Problem". *Numerical Techniques for Stochastic Optimization* (1988), pp. 445–453.

[57] D.G. Luenberger and Y Ye. *Linear and nonlinear programming.* Springer, (2008).

[58] I.J. Lustig, R.E. Marsten, and D.F. Shanno. "Interior point methods for linear programming: Computational state of the art". *ORSA Journal on Computing* 6.1 (1994), pp. 1–14.

[59] I.J Lustig, J.M. Mulvey, and T.J. Carpenter. "Formulating two-stage stochastic programs for interior point methods". *Operations Research* 39.5 (1991), pp. 757–770.

[60] I. Maros and C. Mészáros. "The role of the augmented system in interior point methods". *European Journal of Operational Research* 107.3 (1998), pp. 720–736.

[61] S. Mehrotra. "On the implementation of a primal-dual interior point method". *SIAM Journal on optimization* 2.4 (1992), pp. 575–601.

[62] C. Mészáros. "The augmented system variant of IPMs in two-stage stochastic linear programming computation". *European journal of operational research* 101.2 (1997), pp. 317–327.

[63] J.L. Midler and R.D. Wollmer. "Stochastic programming models for scheduling airlift operations". *Naval Research Logistics Quarterly* 16.3 (1969), pp. 315–330.

[64] *MPS file format.* Available in: `http://lpsolve.sourceforge.net/5.5/mps-format.htm`. [Accessed on 2017-06-09].

[65] J.M. Mulvey and A. Ruszczyński. "A new scenario decomposition method for large-scale stochastic optimization". *Operations research* 43.3 (1995), pp. 477–490.

[66] J.M. Mulvey and H. Vladimirou. "Applying the progressive hedging algorithm to stochastic generalized networks". *Annals of Operations Research* 31.1 (1991), pp. 399–424.

[67] NEOS Server: State-of-the-Art Solvers for Numerical Optimization. *Wisconsin Institute for Discovery.* Available in: `https://neos-server.org/neos/`. [Accessed on 2019-06-17].

[68] E.G. Ng and B.W. Peyton. "Block sparse Cholesky algorithms on advanced uniprocessor computers". *SIAM Journal on Scientific Computing* 14.5 (1993), pp. 1034–1056.

[69] J. Nocedal and S.J. Wright. *Numerical optimization.* Springer Science & Business Media, (2006).

[70] M. Resende and G. Veiga. "An implementation of the dual affine scaling algorithm for minimum-cost flow on bipartite uncapacitated networks". *SIAM Journal on Optimization* 3.3 (1993), pp. 516–537.

[71] R.T. Rockafellar and R.J.-B Wets. "A Lagrangian finite generation technique for solving linear-quadratic problems in stochastic programming". *Stochastic Programming 84 Part II.* Springer, 1986, pp. 63–93.

[72] A. Ruszczynski. “Interior point methods in stochastic programming”. *International Institute of Applied Systems analysis* (1993).

[73] T. Santoso et al. “A stochastic programming approach for supply chain network design under uncertainty”. *European Journal of Operational Research* 167.1 (2005), pp. 96–115.

[74] S. Sen, R.D. Doverspike, and S. Cosares. “Network planning with random demand”. *Telecommunication systems* 3.1 (1994), pp. 11–30.

[75] SMI coingor.org. *Stochastic Modeling Interface.* Available in: `https://projects.coin-or.org/Smi`. [Accessed on 2019-06-06].

[76] S. Subrahmanyam, J.F. Pekny, and G.V. Reklaitis. “Design of batch chemical plants under market uncertainty”. *Industrial & Engineering Chemistry Research* 33.11 (1994), pp. 2688–2701.

[77] R. Van Slyke and R. Wets. “L-shaped linear programs with applications to optimal control and stochastic programming”. *SIAM Journal on Applied Mathematics* 17.4 (1969), pp. 638–663.

[78] R.J. Vanderbei. “Symmetric quasidefinite matrices”. *SIAM Journal on Optimization* 5.1 (1995), pp. 100–113.

[79] A. Wächter and L.T. Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. *Mathematical programming* 106.1 (2006), pp. 25–57.

[80] S.J. Wright. *Primal-dual interior-point methods.* Vol. 54. Siam, (1997).

[81] V. Zverovich et al. “A computational study of a solver system for processing two-stage stochastic LPs with enhanced Benders decomposition”. *Mathematical Programming Computation* 4.3 (2012), pp. 211–238.